4th Dimension

ランゲージリファレンス

Windows® and Mac OS®



4D

4th Dimension ランゲージリファレンス

Version 2004.2 for Windows® and Mac OS®

Copyright© 1994 - 2006 4D SA
All rights reserved.

このマニュアルに記載されている事項は、将来予告なしに変更されることがあり、いかなる変更に関しても4D SAは一切の責任を負いかねます。このマニュアルで説明されるソフトウェアは、本製品に同梱のLicense Agreement（使用許諾契約書）のもとでのみ使用することができます。

ソフトウェアおよびマニュアルの一部または全部を、ライセンス保持者がこの契約条件を許諾した上での個人使用目的以外に、いかなる目的であれ、電子的、機械的、またどのような形であっても、無断で複製、配布することはできません。

4th Dimension、4D Server、4D、4Dロゴ、およびその他の4D製品の名称は、4D SAの商標または登録商標です。

MicrosoftとWindowsはMicrosoft Corporation社の登録商標です。

Apple, Macintosh, Mac, Power Macintosh, Laser Writer, Image Writer, ResEdit, QuickTimeはApple Computer Inc.の登録商標または商標です。

その他、記載されている会社名，製品名は、各社の登録商標または商標です。

注意

このソフトウェアの使用に際し、本製品に同梱のLicense Agreement（使用許諾契約書）に同意する必要があります。ソフトウェアを使用する前に、License Agreementを注意深くお読みください。

目次

**第1章　はじめに ・・・

配列 ……74
サブテーブル ……74
未定義 ……74
データタイプの変換 ……75
定数 ……75
定義済定数 ……75
リテラル定数 ……77
変数 ……78
変数の作成 ……79
変数にデータを代入する ……80
ローカル変数、プロセス変数、インタープロセス変数 …80
フォーム上のオブジェクトと変数 ……82
システム変数 ……82
システム変数 ……83
ポインタ ……85
ポインタの使用例 ……86
ボタンに対するポインタの使用 ……87
テーブルに対するポインタの使用 ……88
フィールドに対するポインタの使用 ……88
ポインタ変数を使用する ……88
配列要素に対するポインタの使用 ……89
配列に対するポインタの使用 ……89
ポインタ配列の使用 ……90
ポインタを使用したボタンの設定 ……90
メソッドに対するポインタの受け渡し ……92
ポインタに対するポインタ ……93
識別子 ……94
テーブル ……94
フィールド ……94
サブテーブル ……95
サブフィールド ……95
インタープロセス変数 ……96
プロセス変数 ……96
ローカル変数 ……96
配列 ……96
フォーム ……99
メソッド ……100
プラグインコマンド（プラグイン、関数、エリア） ….101
セット ……101
命名セレクション ……103
プロセス ……104

名前規則のまとめ ……105
名前が重複する場合 ……105
制御フロー ……106
If...Else...End if構造 ……107
Case of...Else...End case構造 ……108
While...End whileループ ……110
Repeat...Untilループ ……111
For...End forループ ……111
メソッド ……116
メソッドの種類 ……117
メソッドの例と用語 ……120
プロジェクトメソッド ……122
メニューメソッド ……123
サブルーチン（メソッドから起動されるメソッド） ……123
サブルーチンへ渡す引数（パラメータ） ……125
関数としてのサブルーチン ……127
再帰プロジェクトメソッド ……127

第3章　4D環境コマンド ……131

Application type ……131
Version type ……132
Application version ……133
Compiled application ……135
Application file ……136
Structure file ……137
Data file ……139
Is data file locked ……141
Get 4D folder ……142
DATA SEGMENT LIST ……145
ADD DATA SEGMENT ……147
FLUSH BUFFERS ……148
OPEN DATA FILE ……149
CREATE DATA FILE ……150
QUIT 4D ……151
SELECT LOG FILE ……153
GET SERIAL INFORMATION ……155
Is license available ……157
OPEN 4D PREFERENCES ……159

第4章　配列コマンド……163

配列 ……163
配列を作成する ……164
配列とフォームオブジェクト ……166
例：ドロップダウンリストを作成する ……167
配列のサイズを取得する ……171
配列の要素を並べ替える ……171
要素の追加と削除 ……171
配列内のクリックの処理 : 選択した要素のテスト ……171
選択した要素を設定する ……172
配列内の値を検索する ……173
コンボボックスの扱い ……174
グループ化されたスクロールエリア ……175
配列と4D言語 ……177
ローカル配列、プロセス配列、インタープロセス配列 ‥177
配列の引数（パラメータ）としての受け渡し ……179
配列の他の配列への割り当て ……179
配列とポインタ ……179
配列の要素ゼロを使用する ……181
2次元配列 ……182
配列とメモリ ……184
ARRAY INTEGER ……186
ARRAY LONGINT ……187
ARRAY REAL ……188
ARRAY STRING ……189
ARRAY TEXT ……191
ARRAY DATE ……192
ARRAY BOOLEAN ……193
ARRAY PICTURE ……195
ARRAY POINTER ……197
Size of array ……199
SORT ARRAY ……200
MULTI SORT ARRAY ……202
Find in array ……205
Count in array ……207
INSERT ELEMENT ……208
DELETE ELEMENT ……209
APPEND TO ARRAY ……210

COPY ARRAY ・・・

BLOB to text ····· 276
INSERT IN BLOB ····· 279
DELETE FROM BLOB ····· 280
COPY BLOB ····· 281
ENCRYPT BLOB ····· 282
DECRYPT BLOB ····· 288

第7章　ブールコマンド ····· 289

True ····· 290
False ····· 290
Not ····· 291

第8章　クリップボードコマンド ····· 293

APPEND TO CLIPBOARD ····· 293
CLEAR CLIPBOARD ····· 300
GET CLIPBOARD ····· 301
GET PICTURE FROM CLIPBOARD ····· 303
Get text from clipboard ····· 304
SET PICTURE TO CLIPBOARD ····· 306
SET TEXT TO CLIPBOARD ····· 307
Test clipboard ····· 308

第9章　通信コマンド ····· 311

SET CHANNEL ····· 311
SET TIMEOUT ····· 316
USE ASCII MAP ····· 317
GET SERIAL PORT MAPPING ····· 319
SEND PACKET ····· 320
RECEIVE PACKET ····· 322
RECEIVE BUFFER ····· 325
SEND VARIABLE ····· 327
RECEIVE VARIABLE ····· 328
SEND RECORD ····· 329
RECEIVE RECORD ····· 330

第10章　コンパイラコマンド ····· 335

コンパイラコマンドの使用 ····· 338
変数のデータタイプ ····· 338
コンパイラコマンドをどこに記述するか ····· 345

目次

タイプ設定ガイド ・・・

IDLE ・・・

「シンタックスエラー」ウインドウ ……449
デバッガ ……450
「実行コントロール」ツールバーボタン ……452
「実行コントロール」ツールバーについて ……455
「デバッグ」ウインドウの各エリア ……456
「ウォッチ」エリア ……456
「メソッド連鎖」エリア ……462
「カスタム・ウォッチ」エリア ……463
新しい式の挿入 ……464
「カスタム・ウォッチ」エリアの「コンテキスト」メニュー ……464
「ソースコード」エリア ……466
プログラムカウンタ ……467
デバッガのブレークポイントの設定 ……468
ブレークポイント ……470
ブレークポイントの編集 ……470
ブレークリスト ……472
コマンドの中断（コマンドのキャッチ） ……474
デバッガのショートカット ……479

第15章 ドラッグ&ドロップコマンド……483
ドラッグ&ドロップ ……483
ドラッグ可能およびドロップ可能なオブジェクトプロパティ ……483
ドラッグ&ドロップのユーザインタフェース処理 ……484
ドラッグ&ドロップのコマンド ……489
Drop position ……491
DRAG AND DROP PROPERTIES ……492

第16章 入力制御コマンド……499
ACCEPT ……499
CANCEL ……500
Keystroke ……502
FILTER KEYSTROKE ……507
GOTO AREA ……513
REJECT ……514
EDIT ITEM ……516

第17章 外部データソース……519
ODBC LOGIN ……519
ODBC LOGOUT ……521

ODBC SET OPTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・522
ODBC GET OPTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・523
ODBC EXECUTE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・524
ODBC End selection ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・526
ODBC LOAD RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・527
ODBC CANCEL LOAD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・528
ODBC SET PARAMETER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・529
ODBC GET LAST ERROR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・532
ODBC IMPORT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・533
ODBC EXPORT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・535

第18章　フォームイベント関数・・・・・・・・・・・・・・・・・・・・・・・・537

Form event ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・537
　　イベントとメソッド ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・540
Before ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・558
During ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・558
After ・・559
In header ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・559
In break ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・560
In footer ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・560
Activated ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・561
Deactivated ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・562
Outside call ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・562
Get edited text ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・563
SET TIMER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・565
Right click ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・567
Contextual click ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・568

第19章　フォームページコマンド・・・・・・・・・・・・・・・・・・・・・・・569

GET FORM PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・569
SET FORM SIZE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・571
SET FORM HORIZONTAL RESIZING ・・・・・・・・・・・・・・・・・・・・・・・・575
SET FORM VERTICAL RESIZING ・・・・・・・・・・・・・・・・・・・・・・・・・576
GET FORM OBJECTS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・577
GOTO PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・578
FIRST PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・580
LAST PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・580
NEXT PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・581

PREVIOUS PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・581
Current form page ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・582
INPUT FORM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・584
OUTPUT FORM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・586

第20章　グラフコマンド・・・・・・・・・・・・・・・・・・・・・・・・・・・・589

GRAPH ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・590
GRAPH SETTINGS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・595
GRAPH TABLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・597

第21章　フォーミュラコマンド・・・・・・・・・・・・・・・・・・・・・・・・599

SET ALLOWED METHODS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・599
GET ALLOWED METHODS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・600
EDIT FORMULA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・601

第22章　階層リストコマンド・・・・・・・・・・・・・・・・・・・・・・・・・・603

Load list ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・603
SAVE LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・605
New list ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・606
Copy list ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・607
CLEAR LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・608
Count list items ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・610
Is a list ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・612
REDRAW LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・613
SET LIST PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・614
GET LIST PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・623
SORT LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・625
APPEND TO LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・628
INSERT LIST ITEM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・635
SET LIST ITEM PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・636
GET LIST ITEM PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・639
List item position ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・641
List item parent ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・642
DELETE LIST ITEM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・644
GET LIST ITEM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・645
SET LIST ITEM ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・647
Selected list items ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・649
SELECT LIST ITEM BY POSITION ・・・・・・・・・・・・・・・・・・・・・・・・652

SELECT LIST ITEM BY REFERENCE ・・・・・・・・・・・・・・・・・655

第23章　データ読み込みとデータ書き出しコマンド・・・・・657

IMPORT TEXT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・657
EXPORT TEXT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・659
IMPORT SYLK ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・661
EXPORT SYLK ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・663
IMPORT DIF ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・665
EXPORT DIF ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・667
IMPORT DATA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・669
EXPORT DATA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・671

第24章　割り込みコマンド・・・・・・・・・・・・・・・・・・・・・・・・673

ON EVENT CALL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・673
Method called on event ・・・・・・・・・・・・・・・・・・・・・・・・・・・677
FILTER EVENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・678
ON ERR CALL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・679
Method called on error ・・・・・・・・・・・・・・・・・・・・・・・・・・・683
ABORT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・684

第25章　ランゲージコマンド・・・・・・・・・・・・・・・・・・・・・・・・687

Count parameters ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・687
Type ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・689
Self ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・692
RESOLVE POINTER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・693
Nil ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・695
Is a variable ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・696
Get pointer ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・697
EXECUTE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・698
Command name ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・699
Current method name ・・・・・・・・・・・・・・・・・・・・・・・・・・・・702
TRACE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・703
NO TRACE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・705

第26章　リストボックスコマンド・・・・・・・・・・・・・・・・・・・・・707

リストボックスの管理 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・707
作成および値の管理 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・708
オブジェクト、カラム、ヘッダ ・・・・・・・・・・・・・・・・・・・・・・・708
リストボックスとランゲージ ・・・・・・・・・・・・・・・・・・・・・・・・710

並び替えの管理 ……711
選択項目の管理 ……712
INSERT LISTBOX COLUMN ……713
DELETE LISTBOX COLUMN ……715
Get number of listbox columns ……716
SORT LISTBOX COLUMNS ……717
SET LISTBOX COLUMN WIDTH ……718
Get listbox column width ……719
MOVED LISTBOX COLUMN NUMBER ……720
SELECT LISTBOX ROW ……721
INSERT LISTBOX ROW ……722
DELETE LISTBOX ROW ……723
Get number of listbox rows ……724
SET LISTBOX ROWS HEIGHT ……725
Get listbox rows height ……726
MOVED LISTBOX ROW NUMBER ……727
GET LISTBOX ARRAYS ……728
Get listbox information ……730
SHOW LISTBOX GRID ……731
SET LISTBOX GRID COLOR ……732
SHOW LISTBOX SCROLLBAR ……733

第27章　算術関数……735

Abs ……735
Int ……736
Dec ……737
Round ……738
Trunc ……739
Random ……740
Mod ……741
Square root ……742
Log ……743
Exp ……744
Sin ……745
Cos ……746
Tan ……746
Arctan ……747
SET REAL COMPARISON LEVEL ……748

実数の表示について ……750
Euro converter ……752

第28章　メニュー……755

メニューの管理 ……755
メニューバー ……755
メニュー番号とメニュー項目番号 ……756
関連メニューバー ……756
プログラムからメニュー項目を修正する ……757
MENU BAR ……759
HIDE MENU BAR ……761
SHOW MENU BAR ……762
SET ABOUT ……763
Menu selected ……764
Count menus ……766
Count menu items ……766
Get menu title ……767
Get menu item ……767
SET MENU ITEM ……768
Get menu item style ……769
SET MENU ITEM STYLE ……770
Get menu item mark ……771
SET MENU ITEM MARK ……772
Get menu item key ……773
SET MENU ITEM KEY ……774
DISABLE MENU ITEM ……775
ENABLE MENU ITEM ……776
APPEND MENU ITEM ……777
INSERT MENU ITEM ……779
DELETE MENU ITEM ……780

第29章　メッセージコマンド……781

MESSAGES ON ……781
MESSAGES OFF ……781
ALERT ……783
CONFIRM ……785
Request ……787
MESSAGE ……790

GOTO XY ……794
DISPLAY NOTIFICATION ……796

第30章　命名セレクションコマンド……797

命名セレクションとセット ……798
COPY NAMED SELECTION ……799
CUT NAMED SELECTION ……801
USE NAMED SELECTION ……802
CLEAR NAMED SELECTION ……803
CREATE SELECTION FROM ARRAY ……804

第31章　オブジェクトプロパティコマンド……805

FONT ……807
FONT SIZE ……808
FONT STYLE ……809
ENABLE BUTTON ……811
DISABLE BUTTON ……813
BUTTON TEXT ……815
Get format ……817
SET FORMAT ……819
SET FILTER ……828
SET CHOICE LIST ……830
SET ENTERABLE ……831
SET VISIBLE ……833
SET COLOR ……836
SET RGB COLORS ……838
GET OBJECT RECT ……842
MOVE OBJECT ……843
BEST OBJECT SIZE ……845
Get alignment ……847
SET ALIGNMENT ……849

第32章　リスト（統計）上の関数……851

Sum ……853
Average ……854
Min ……855
Max ……856
Std deviation ……857

Variance ……858
Sum squares ……859

第33章 演算子……861

演算子 ……861
文字列演算子 ……862
数値演算子 ……862
日付演算子 ……863
時間演算子 ……863
比較演算子 ……864
論理演算子 ……868
ピクチャ演算子 ……870
ビットワイズ演算子 ……871

第34章 印刷コマンド……875

PRINT LABEL ……875
PRINT SELECTION ……879
Print form ……881
PAGE BREAK ……884
PRINT RECORD ……885
Printing page ……887
PRINTERS LIST ……888
SET CURRENT PRINTER ……889
Get current printer ……890
BREAK LEVEL ……891
SET PRINT OPTION ……892
GET PRINT OPTION ……896
PRINT OPTION VALUES ……898
ACCUMULATE ……900
Subtotal ……901
Level ……903
PAGE SETUP ……905
Get print marker ……907
PRINT SETTINGS ……908
SET PRINT PREVIEW ……909
SET PRINT MARKER ……910
GET PRINTABLE MARGIN ……915
SET PRINTABLE MARGIN ……917

GET PRINTABLE AREA ・・・

第38章　プロセスコマンド・・・・・・・・・・・・・・・・・・・・・・・・・・・967
プロセス ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・967
プロセスの作成と消去 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・968
プロセスの要素 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・968
ユーザプロセス ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・970
4th Dimensionによって作成されるプロセス ・・・・・・・・・・・・970
グローバルプロセスとローカルプロセス ・・・・・・・・・・・・・971
プロセス間のレコードのロック ・・・・・・・・・・・・・・・・・・・971
New process ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・972
Execute on server ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・975
DELAY PROCESS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・980
PAUSE PROCESS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・981
RESUME PROCESS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・982
Process aborted ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・983
Current process ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・984
Process state ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・985
PROCESS PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・987
Process number ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・990
Count users ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・992
Count tasks ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・992
Count user processes ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・993
EXECUTE ON CLIENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・994
REGISTER CLIENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・996
UNREGISTER CLIENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・999
GET REGISTERED CLIENTS ・・・・・・・・・・・・・・・・・・・・・・・・・・1000

第39章　クエリーコマンド・・・・・・・・・・・・・・・・・・・・・・・・・1001
QUERY BY EXAMPLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1001
QUERY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1003
QUERY SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1009
QUERY BY FORMULA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1011
QUERY SELECTION BY FORMULA ・・・・・・・・・・・・・・・・・・・・・1013
QUERY WITH ARRAY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1014
SET QUERY DESTINATION ・・・・・・・・・・・・・・・・・・・・・・・・・・1015
SET QUERY LIMIT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1021
Find index key ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1023
ORDER BY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1024
ORDER BY FORMULA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1029

第40章　クイックレポートコマンド ・・・・・・・・・・・・・・・・1031

QR REPORT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1031
QR REPORT TO BLOB ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1035
QR BLOB TO REPORT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1036
QR New offscreen area ・・・・・・・・・・・・・・・・・・・・・・・・・・・1037
QR DELETE OFFSCREEN AREA ・・・・・・・・・・・・・・・・・・・・・・・・1037
QR SET DESTINATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1038
QR GET DESTINATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1040
QR SET DOCUMENT PROPERTY ・・・・・・・・・・・・・・・・・・・・・・・・1041
QR Get document property ・・・・・・・・・・・・・・・・・・・・・・・・・1042
QR SET REPORT KIND ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1043
QR Get report kind ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1044
QR SET AREA PROPERTY ・・・・・・・・・・・・・・・・・・・・・・・・・・・1045
QR Get area property ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1046
QR SET REPORT TABLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1047
QR Get report table ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1048
QR SET TEXT PROPERTY ・・・・・・・・・・・・・・・・・・・・・・・・・・・1049
QR Get text property ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1051
QR RUN ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1053
QR EXECUTE COMMAND ・・・・・・・・・・・・・・・・・・・・・・・・・・・1054
QR Get command status ・・・・・・・・・・・・・・・・・・・・・・・・・・・1055
QR ON COMMAND ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1056
QR Find column ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1057
QR SET SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1058
QR GET SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1059
QR SET HEADER AND FOOTER ・・・・・・・・・・・・・・・・・・・・・・・・1060
QR GET HEADER AND FOOTER ・・・・・・・・・・・・・・・・・・・・・・・・1062
QR SET BORDERS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1064
QR GET BORDERS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1066
QR SET INFO COLUMN ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1068
QR GET INFO COLUMN ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1071
QR SET INFO ROW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1074
QR Get info row ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1075
QR SET SORTS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1076
QR GET SORTS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1077
QR SET TOTALS DATA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1078
QR GET TOTALS DATA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1081

QR SET TOTALS SPACING ・・・・・・・・・・・・・・・・・・・・・・・・1083
QR GET TOTALS SPACING ・・・・・・・・・・・・・・・・・・・・・・・1084
QR INSERT COLUMN ・・・・・・・・・・・・・・・・・・・・・・・・・・1085
QR Get drop column ・・・・・・・・・・・・・・・・・・・・・・・・・・1086
QR Count columns ・・・・・・・・・・・・・・・・・・・・・・・・・・・1087
QR DELETE COLUMN ・・・・・・・・・・・・・・・・・・・・・・・・・・1088
QR SET HTML TEMPLATE ・・・・・・・・・・・・・・・・・・・・・・・1089
QR Get HTML template ・・・・・・・・・・・・・・・・・・・・・・・・1091

第41章　レコードロックコマンド ・・・・・・・・・・・・・・・・・・1093

レコードのロック ・・・・・・・・・・・・・・・・・・・・・・・・・・・1093
ロックされたレコード ・・・・・・・・・・・・・・・・・・・・・・・・1094
リードオンリー状態とリードライト状態 ・・・・・・・・・・・・・1094
レコードのロード、更新、アンロード ・・・・・・・・・・・・・・1096
アンロックされたレコードをロードするためのループ処理 ・・1097
マルチユーザやマルチプロセスモードでのコマンドの使用 ・・1099
READ WRITE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1101
READ ONLY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1102
Read only state ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1103
LOAD RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1104
UNLOAD RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・1106
Locked ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1107
LOCKED ATTRIBUTES ・・・・・・・・・・・・・・・・・・・・・・・・・1108

第42章　レコードコマンド ・・・・・・・・・・・・・・・・・・・・・・1109

DISPLAY RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・1109
CREATE RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1110
DUPLICATE RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・1111
Is new record ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1112
Modified record ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1113
Is record loaded ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1114
SAVE RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1115
DELETE RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1117
Records in table ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1118
Record number ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1119
GOTO RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1120
Sequence number ・・・・・・・・・・・・・・・・・・・・・・・・・・・1121
レコードに付けられた番号の使用 ・・・・・・・・・・・・・・・・・・1123
レコードに付けられた番号の例 ・・・・・・・・・・・・・・・・・・・1124

PUSH RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1127
POP RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1128
レコードスタックの使用 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1129

第43章　リレートコマンド ・・・・・・・・・・・・・・・・・・・・・・・・1131

リレート ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1131
コマンドを使用したテーブルの自動リレート ・・・・・・・・・1131
テーブルリレートを実行するコマンド ・・・・・・・・・・・・・・1133
SET AUTOMATIC RELATIONS ・・・・・・・・・・・・・・・・・・・・・・1134
GET AUTOMATIC RELATIONS ・・・・・・・・・・・・・・・・・・・・・・1135
SET FIELD RELATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1136
GET FIELD RELATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1138
RELATE ONE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1142
RELATE MANY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1146
CREATE RELATED ONE ・・・・・・・・・・・・・・・・・・・・・・・・・・・1148
SAVE RELATED ONE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1149
OLD RELATED ONE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1150
OLD RELATED MANY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1151
RELATE ONE SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・1152
RELATE MANY SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・1153

第44章　リソースコマンド ・・・・・・・・・・・・・・・・・・・・・・・・1155

リソースの概念 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1155
データフォークとリソースフォーク ・・・・・・・・・・・・・・・・・1155
リソースファイル ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1155
リソースファイル連鎖 ・・・・・・・・・・・・・・・・・・・・・・・・・・1156
リソースタイプ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1158
リソース名とリソースID番号 ・・・・・・・・・・・・・・・・・・・・・・1159
リソースプロパティ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1162
リソース内容の取り扱い ・・・・・・・・・・・・・・・・・・・・・・・・・1162
4Dコマンドとリソース ・・・・・・・・・・・・・・・・・・・・・・・・・・1163
リソースと4D Insider：例題 ・・・・・・・・・・・・・・・・・・・・・・・1163
Open resource file ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1170
Create resource file ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1174
CLOSE RESOURCE FILE ・・・・・・・・・・・・・・・・・・・・・・・・・・1177
RESOURCE TYPE LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1178
RESOURCE LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1180
STRING LIST TO ARRAY ・・・・・・・・・・・・・・・・・・・・・・・・・・1182
ARRAY TO STRING LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・1183

Get indexed string ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1186
Get string resource ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1187
SET STRING RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1188
Get text resource ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1189
SET TEXT RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1190
GET PICTURE RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1191
SET PICTURE RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1192
GET ICON RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1193
GET RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1195
SET RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1197
Get resource name ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1200
SET RESOURCE NAME ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1202
Get resource properties ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1203
SET RESOURCE PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・1204
DELETE RESOURCE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1208
Get component resource ID ・・・・・・・・・・・・・・・・・・・・・・・・・・・1211

第45章　暗号化プロトコル ・・・・・・・・・・・・・・・・・・・・・・・・1213

GENERATE ENCRYPTION KEYPAIR ・・・・・・・・・・・・・・・・・・・・・・・1213
GENERATE CERTIFICATE REQUEST ・・・・・・・・・・・・・・・・・・・・・・・1215

第46章　カレントセレクションコマンド ・・・・・・・・・・・・・・・・1219

ALL RECORDS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1219
Records in selection ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1220
DELETE SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1221
Selected record number ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1223
GOTO SELECTED RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・1224
FIRST RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1226
Displayed line number ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1227
NEXT RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1228
LAST RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1229
PREVIOUS RECORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1230
Before selection ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1231
End selection ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1233
DISPLAY SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1235
MODIFY SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1239
APPLY TO SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1240
REDUCE SELECTION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1242

SCAN INDEX ・・・

Lowercase ・・・

QUERY SUBRECORDS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1347

第51章　システムドキュメントコマンド ・・・・・・・・・・・・1349

システムドキュメントの概要 ・・・・・・・・・・・・・・・・・・・・・・・・1349
ドキュメントファイルのタイプとクリエータ ・・・・・・・・・1351
DocRef：ドキュメント参照番号 ・・・・・・・・・・・・・・・・・・・・1351
I／Oエラーの処理 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1352
Documentシステム変数 ・・・・・・・・・・・・・・・・・・・・・・・・・・1352
ドキュメント名またはドキュメントパス名の指定 ・・・・・1352
ディスク上にあるドキュメントを操作する場合に便利なプロジェクトメソッド ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1353
Document type ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1357
SET DOCUMENT TYPE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1358
Document creator ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1359
SET DOCUMENT CREATOR ・・・・・・・・・・・・・・・・・・・・・・・・1359
Open document ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1360
Create document ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1363
Append document ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1365
CLOSE DOCUMENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1367
COPY DOCUMENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1368
MOVE DOCUMENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1369
DELETE DOCUMENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1370
Select document ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1371
Test path name ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1375
SHOW ON DISK ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1376
CREATE FOLDER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1378
Select folder ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1379
DELETE FOLDER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1382
CREATE ALIAS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1383
RESOLVE ALIAS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1385
VOLUME LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1386
VOLUME ATTRIBUTES ・・・・・・・・・・・・・・・・・・・・・・・・・・・1387
FOLDER LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1390
DOCUMENT LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1391
MAP FILE TYPES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1392
GET DOCUMENT PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・1394
SET DOCUMENT PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・1400
GET DOCUMENT ICON ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1401

Get document size ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1402
SET DOCUMENT SIZE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1403
Get document position ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1403
SET DOCUMENT POSITION ・・・・・・・・・・・・・・・・・・・・・・・・・・1404

第52章　システム環境コマンド ・・・・・・・・・・・・・・・・・・・・1405

Screen height ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1405
Screen width ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1406
Count screens ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1406
SCREEN COORDINATES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1407
SCREEN DEPTH ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1408
SET SCREEN DEPTH ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1410
Menu bar screen ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1410
Menu bar height ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1411
FONT LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1411
Font name ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1412
Font number ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1413
PLATFORM PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1414
System folder ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1418
Temporary folder ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1419
Current machine ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1420
Current machine owner ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1421
Gestalt ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1422
LOG EVENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1423
SET ENVIRONMENT VARIABLE ・・・・・・・・・・・・・・・・・・・・・・・・・1425
LAUNCH EXTERNAL PROCESS ・・・・・・・・・・・・・・・・・・・・・・・・・1426

第53章　テーブルコマンド ・・・・・・・・・・・・・・・・・・・・・・・1429

DEFAULT TABLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1429
Current default table ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1431
Current form table ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1432

第54章　ツールバーコマンド ・・・・・・・・・・・・・・・・・・・・・・1435

HIDE TOOL BAR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1435
SHOW TOOL BAR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1435

第55章　ツールコマンド ・・・・・・・・・・・・・・・・・・・・・・・・1437

BUILD APPLICATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1437
ENCODE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1439

DECODE ・・・

SET FIELD TITLES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1490
GET FIELD TITLES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1492
Shift down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1493
Caps lock down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1494
Windows Ctrl down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1494
Windows Alt down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1495
Macintosh command down ・・・・・・・・・・・・・・・・・・・・・・・・・・・1496
Macintosh option down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1497
Macintosh control down ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1498
GET MOUSE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1499
Pop up menu ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1500
POST KEY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1503
POST CLICK ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1504
POST EVENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1505
SCROLL LINES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1507
GET HIGHLIGHT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1508
HIGHLIGHT TEXT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1510
SET CURSOR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1511
Focus object ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1512
REDRAW ・・1513
INVERT BACKGROUND ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1514

第６０章　ユーザ&グループコマンド ・・・・・・・・・・・・・・・・１５１５

EDIT ACCESS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1515
CHANGE CURRENT USER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1516
Validate password ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1518
CHANGE PASSWORD ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1519
Current user ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1520
Get default user ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1521
User in group ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1522
DELETE USER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1523
Is user deleted ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1524
GET USER LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1525
GET USER PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1526
USERS TO BLOB ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1528
BLOB TO USERS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1529
Set user properties ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1531
GET GROUP LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1534

GET GROUP PROPERTIES ・・・・・・・・・・・・・・・・・・・・・・・・・1535
Set group properties ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1537
CHANGE LICENSE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1539
SET PLUGIN ACCESS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1540
Get plugin access ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1541
GET PLUGIN LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1542

第61章　変数コマンド ・・・・・・・・・・・・・・・・・・・・・・・・・・・1543

SAVE VARIABLES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1543
LOAD VARIABLES ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1545
CLEAR VARIABLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1546
Undefined ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1548

第62章　Webサーバコマンド ・・・・・・・・・・・・・・・・・・・・・・・1551

概要 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1551
Webサーバー設定と接続管理 ・・・・・・・・・・・・・・・・・・・・・・・・1556
Web上への4Dデータベース公開条件 ・・・・・・・・・・・・・・・・・1556
Mac OS XにおけるWebサーバの設定 ・・・・・・・・・・・・・・・・・1557
4D Webサーバの開始 ・・・・・・・・・・・・・・・・・・・・・・・・・・1560
Webサーバの検証 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1561
Web上に公開された4Dデータベースへの接続 ・・・・・・・・・1562
Webプロセスの管理 ・・・・・・・・・・・・・・・・・・・・・・・・・・1564
「Web接続」プロセス ・・・・・・・・・・・・・・・・・・・・・・・・・1566
Webサービス：入門編 ・・・・・・・・・・・・・・・・・・・・・・・・・・・1568
接続セキュリティ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1576
Webアクセス用のパスワードマネージメントシステム ・1576
一般Webユーザ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1580
デフォルトHTMLルートフォルダを定義する ・・・・・・・・・・1581
4DACTION、4D METHOD、4D SCRIPTを利用する ・・・・1583
「On Web Authentication」データベースメソッド ・・・・・・1584
「On Web Connection」データベースメソッド ・・・・・・・・・1589
On Web Connectionデータベースメソッドの呼び出し ・・・1592
HTMLオブジェクトと4Dオブジェクトのバインド ・・・・・1597
ダイナミックな値の送信 ・・・・・・・・・・・・・・・・・・・・・・・1598
ダイナミックな値の受信 ・・・・・・・・・・・・・・・・・・・・・・・1601
COMPILER_WEBプロジェクトメソッド ・・・・・・・・・・・・・1606
HTMLオブジェクトと4D変数とのバインド設定・イメージマッピング ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1607
URLとフォームアクション ・・・・・・・・・・・・・・・・・・・・・・・・1608
URL 4D ACTION/ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1609

URL 4DMETHOD/ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1613
URL 4DCGI/<アクション> ・・・・・・・・・・・・・・・・・・・・・・・・・1614
URLでコールした4Dメソッドに渡されたテキスト引数 ・1615
4D HTMLタグ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1617
4D HTMLタグについて ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1618
Webサーバセッテイング ・・・・・・・・・・・・・・・・・・・・・・・・・・・1627
設定ページ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1627
詳細ページ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1633
Webプロセスの最大数を定義する ・・・・・・・・・・・・・・・・・・・・1634
Javascriptを入力制御に使用する ・・・・・・・・・・・・・・・・・・・・1635
オプションページ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1636
拡張ASCII文字を直接送る ・・・・・・・・・・・・・・・・・・・・・・・・・1636
文字セット ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1637
4D Connectを経由した4D WebSTARの接続を許可する ・・1638
Web互換性 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1639
ブラケットの代りに4DVARコメントを使用する ・・・・・・・1639
新しいコンテキスト参照モードを使用 ・・・・・・・・・・・・・・・1640
Webサイトに関する情報 ・・・・・・・・・・・・・・・・・・・・・・・・・・・1641
Webサーバ管理URL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1641
接続ログファイル ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1642
ランタイムエクスプローラの情報 ・・・・・・・・・・・・・・・・・・・・1644
コンテキストモードの使用 ・・・・・・・・・・・・・・・・・・・・・・・・・1645
非コンテキストモードとコンテキストモード ・・・・・・・・・1645
Web接続のコンテキスト管理：コンテキストモード ・・・1649
HTMLの自動変換 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1655
HTMLの埋め込み ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1662
ファイル参照とURL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1662
SSLプロトコルの使用 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1664
XMLとWMLのサポート ・・・・・・・・・・・・・・・・・・・・・・・・・・・1669
CGIの使用 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1671
4D WebサーバからCGIを実行する ・・・・・・・・・・・・・・・・・・・1671
CGIを使用した4D Webサーバ呼び出し（Windowsのみ） ・・・1676
START WEB SERVER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1681
STOP WEB SERVER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1682
SET WEB TIMEOUT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1682
SET HTML ROOT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1683
SET WEB DISPLAY LIMIT ・・・・・・・・・・・・・・・・・・・・・・・・・1684
SET HOME PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1687
SEND HTML FILE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1688

SEND HTML BLOB ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1691
SEND HTML TEXT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1694
PROCESS HTML TAGS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1695
GET WEB FORM VARIABLES ・・・・・・・・・・・・・・・・・・・・・・・・・・1697
Web Context ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1699
SET HTTP HEADER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1700
GET HTTP HEADER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1702
GET HTTP BODY ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1705
SEND HTTP RAW DATA ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1707
SEND HTTP REDIRECT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1710
WEB CACHE STATISTICS ・・・・・・・・・・・・・・・・・・・・・・・・・・・1712
Secured Web connection ・・・・・・・・・・・・・・・・・・・・・・・・・・・1713
SET CGI EXECUTABLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1714
OPEN WEB URL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1716

第63章　Webサービス（クライアント）コマンド ・・・・1719

Webサービス（クライアント） ・・・・・・・・・・・・・・・・・・・・・・・・1719
SET WEB SERVICE PARAMETER ・・・・・・・・・・・・・・・・・・・・・・・1720
SET WEB SERVICE OPTION ・・・・・・・・・・・・・・・・・・・・・・・・・・1722
CALL WEB SERVICE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1724
GET WEB SERVICE RESULT ・・・・・・・・・・・・・・・・・・・・・・・・・・1729
AUTHENTICATE WEB SERVICE ・・・・・・・・・・・・・・・・・・・・・・・・1731
Get Web Service error info ・・・・・・・・・・・・・・・・・・・・・・・・・・1732

第64章　Webサービス（サーバ）コマンド ・・・・・・・・・1735

Webサービス（サーバ） ・・・・・・・・・・・・・・・・・・・・・・・・・・・1735
SOAP DECLARATION ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1736
SEND SOAP FAULT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1739
Is SOAP request ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1740
Get SOAP info ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1741

第65章　ウインドウコマンド ・・・・・・・・・・・・・・・・・・・・・・1743

ウインドウについて ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1743
ウインドウのタイプ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1744
Open window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1752
Open external window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1756
CLOSE WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1758
ERASE WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1759

REDRAW WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1760
DRAG WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1761
HIDE WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1763
SHOW WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1764
MAXIMIZE WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1765
MINIMIZE WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1767
Get window title ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1768
SET WINDOW TITLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1769
WINDOW LIST ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1770
Window kind ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1772
Window process ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1773
GET WINDOW RECT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1774
SET WINDOW RECT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1775
Frontmost window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1777
Next window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1777
Find window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1778
Open form window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1779
Current form window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1782
RESIZE FORM WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1783

第66章　XMLコマンド ・・・・・・・・・・・・・・・・・・・・・・・・・・1785

XMLについて ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1785
DOMコマンドとSAXコマンド ・・・・・・・・・・・・・・・・・・・・・・・1785
XPathノーテーションの使用（DOM） ・・・・・・・・・・・・・・・・・1786
XSL Transformationsの管理 ・・・・・・・・・・・・・・・・・・・・・・・1787
用語 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1789
DOM Create XML Ref ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1791
DOM SET XML OPTIONS ・・・・・・・・・・・・・・・・・・・・・・・・・・・1793
DOM Parse XML source ・・・・・・・・・・・・・・・・・・・・・・・・・・・1794
DOM Parse XML variable ・・・・・・・・・・・・・・・・・・・・・・・・・・1796
DOM Create XML element ・・・・・・・・・・・・・・・・・・・・・・・・・・1798
DOM Find XML element ・・・・・・・・・・・・・・・・・・・・・・・・・・・1801
DOM Count XML elements ・・・・・・・・・・・・・・・・・・・・・・・・・・1803
DOM Get XML element ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1804
DOM Get parent XML element ・・・・・・・・・・・・・・・・・・・・・・・・1805
DOM Get First Child XML element ・・・・・・・・・・・・・・・・・・・・・1806
DOM Get next sibling XML element ・・・・・・・・・・・・・・・・・・・・・1808
DOM Get previous sibling XML element ・・・・・・・・・・・・・・・・・・1810

DOM Get last child XML element ・・・・・・・・・・・・・・・・・・・・・・1811
DOM SET XML ELEMENT NAME ・・・・・・・・・・・・・・・・・・・・・1812
DOM GET XML ELEMENT NAME ・・・・・・・・・・・・・・・・・・・・1814
DOM SET XML ELEMENT VALUE ・・・・・・・・・・・・・・・・・・・1815
DOM GET XML ELEMENT VALUE ・・・・・・・・・・・・・・・・・・・1816
DOM REMOVE XML ELEMENT ・・・・・・・・・・・・・・・・・・・・・1817
DOM Count XML attributes ・・・・・・・・・・・・・・・・・・・・・・・・1818
DOM SET XML ATTRIBUTE ・・・・・・・・・・・・・・・・・・・・・・1820
DOM GET XML ATTRIBUTE BY INDEX ・・・・・・・・・・・・・・・1821
DOM GET XML ATTRIBUTE BY NAME ・・・・・・・・・・・・・・・1822
DOM Get XML information XML ・・・・・・・・・・・・・・・・・・・・・1824
DOM EXPORT TO FILE ・・・・・・・・・・・・・・・・・・・・・・・・・・1825
DOM EXPORT TO VAR ・・・・・・・・・・・・・・・・・・・・・・・・・・1826
DOM CLOSE XML ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1827
SAX SET XML OPTIONS ・・・・・・・・・・・・・・・・・・・・・・・・1828
SAX GET XML DOCUMENT VALUES ・・・・・・・・・・・・・・・・1829
SAX ADD XML DOCTYPE ・・・・・・・・・・・・・・・・・・・・・・・・1830
SAX Get XML node ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1831
SAX ADD XML COMMENT ・・・・・・・・・・・・・・・・・・・・・・・1832
SAX GET XML COMMENT ・・・・・・・・・・・・・・・・・・・・・・・1833
SAX OPEN XML ELEMENT ・・・・・・・・・・・・・・・・・・・・・・・1834
SAX OPEN XML ELEMENT ARRAYS ・・・・・・・・・・・・・・・・1835
SAX CLOSE XML ELEMENT ・・・・・・・・・・・・・・・・・・・・・・1837
SAX GET XML ELEMENT ・・・・・・・・・・・・・・・・・・・・・・・・1838
SAX ADD XML ELEMENT VALUE ・・・・・・・・・・・・・・・・・・1840
SAX GET XML ELEMENT VALUE ・・・・・・・・・・・・・・・・・・1841
SAX ADD XML CDATA ・・・・・・・・・・・・・・・・・・・・・・・・・・1842
SAX GET XML CDATA ・・・・・・・・・・・・・・・・・・・・・・・・・・1844
SAX ADD PROCESSING INSTRUCTION ・・・・・・・・・・・・・・1845
SAX GET XML PROCESSING INSTRUCTION ・・・・・・・・・・1846
SAX GET XML ENTITY ・・・・・・・・・・・・・・・・・・・・・・・・・1847
GET XML ERROR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1848
APPLY XSLT TRANSFORMATION ・・・・・・・・・・・・・・・・・・1849
SET XSLT PARAMETER ・・・・・・・・・・・・・・・・・・・・・・・・・1851
GET XSLT ERROR ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1853

第67章　エラーメッセージ ・・・・・・・・・・・・・・・・・・・・・・・・1855

シンタックスエラー ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1855

データベースエンジンエラー ・・・・・・・・・・・・・・・・・・・・・・・・1859
ネットワークエラーコード ・・・・・・・・・・・・・・・・・・・・・・・・1864
File Managerのエラー ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1865
Memory Managerのエラー ・・・・・・・・・・・・・・・・・・・・・・・・・1867
プリントエラーコード ・・・・・・・・・・・・・・・・・・・・・・・・・・・1868
Resource Managerのエラー ・・・・・・・・・・・・・・・・・・・・・・・・1869
Macintosh SANE NaN Errorのコード ・・・・・・・・・・・・・・・・・・1870
Sound Managerのエラー ・・・・・・・・・・・・・・・・・・・・・・・・・・1871
シリアルポートのエラーコード ・・・・・・・・・・・・・・・・・・・・・・1872
Mac OSシステムエラー ・・・・・・・・・・・・・・・・・・・・・・・・・・・1873

第68章　ASCIIコード・・・・・・・・・・・・・・・・・・・・・・・・・・・・1875

ASCIIコード0から127 ・・・・・・・・・・・・・・・・・・・・・・・・・・・1875
拡張ASCIIコード（128から255） ・・・・・・・・・・・・・・・・・・・・1878
ASCIIコードと4th Dimensionの理解 ・・・・・・・・・・・・・・・・・1883
ファンクションキー ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1884

定義済み定数・・1887

4D Environment ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1887
ASCII Codes ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1888
Backupと修復 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1890
BLOB ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1891
Clipboard ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1892
Colors ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1893
Communications ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1894
Database Engine ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1895
Database Events ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1896
Database Parameters ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1897
Date Display Formats ・・・・・・・・・・・・・・・・・・・・・・・・・・・1898
Days and Months ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1899
Dictionaries ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1900
Euro currencies ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1901
Events(Modifiers) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1902
Events(What) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1903
Expressions ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1904
External data source ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1905
Field and Variable Types ・・・・・・・・・・・・・・・・・・・・・・・・・1906
Find window ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1907

Font Styles ····· 1908
Form area ····· 1909
Form Events ····· 1910
Form options ····· 1911
Function Keys ····· 1912
Hierarchical Lists ····· 1913
Is license available ····· 1914
ISO Latin Character Entities ····· 1915
List box ····· 1917
Math ····· 1918
Object alignment ····· 1919
Open form window ····· 1920
Open window ····· 1921
Picture Compression ····· 1922
Picture Display Formats ····· 1923
Platform Interfaces ····· 1924
Platform Properties ····· 1925
Print options ····· 1926
Process state ····· 1927
Process Type ····· 1928
QR Area Properties ····· 1929
QR Borders ····· 1930
QR Commands ····· 1931
QR Document Properties ····· 1933
QR Operators ····· 1934
QR Output Destination ····· 1935
QR Report Types ····· 1936
QR Rows for Properties ····· 1937
QR Text Properties ····· 1938
Query Destinations ····· 1939
Relations ····· 1940
Resources Properties ····· 1941
SCREEN DEPTH ····· 1942
SET RGB COLOR ····· 1943
Standard System Signatures ····· 1944
System Documents ····· 1945
System Folder ····· 1946

TCP Port Numbers ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1947
Time Display Formats ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1948
Web Services(Client) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1949
Web Services(Server) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1950
Window kind ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1951
Windows Log Events ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1952
XML ・・1953

**コマンド索引・・・・・・・・・・・・・・・・・・・

概要

4th Dimensionは、総数500以上にもおよぶ独自のプログラミング言語を持つ強力なリレーショナルデータベースです。この4th Dimensionプログラミング言語は、簡単な計算処理から複雑なカスタムユーザインタフェースの作成に至るまで、さまざまな業務で用いることができます。例えば、4th Dimensionプログラミング言語を使用すれば、次のようなことが可能になります。

■「ユーザ」モードで使用できる任意エディタのプログラミングによるアクセス

■ データベース情報からのレポートやグラフ、ラベルの作成と印刷

■ 他のデバイスとの通信

■ ドキュメント管理

■ 4th Dimensionと別のデータベース間におけるデータ読み込みと書き出し

■ 別の言語で作成したプログラムを4th Dimensionプログラミング言語の中に組み込むこと

4th Dimensionプログラミング言語は、柔軟性と能力を備え、あらゆるレベルのユーザや開発者がさまざまな情報管理業務を達成するための理想的なツールです。初心者ユーザでも計算処理を素早く行うことができます。またプログラミング経験がなくても、ある程度コンピュータの知識を持っているユーザであれば、自分のデータベースをカスタマイズすることができます。一方、熟練した開発者であれば、4th Dimensionの強力なプログラミング言語を使用して、ファイル転送や通信などの高度な機能をデータベースに組み込むことができます。他の言語でプログラミング経験がある開発者は、その独自のコマンドを4th Dimensionに追加することができます。

4th Dimensionプログラミング言語は、4th Dimensionのプラグインがアプリケーションに組み込まれると増加します。それぞれのプラグインは、専用のコマンドを持っています。

マニュアル全般について

4th Dimensionと4D Serverの両方の機能については、下記のマニュアルで説明しています。4D Serverの専門的な機能の説明は、4D Serverパッケージに含まれている『4D Serverリファレンス』の中でのみ説明されています。

『4th Dimensionランゲージリファレンス』は、4th Dimension言語を記述する際のリファレンスガイドです。4th Dimension言語を使ってデータベースをカスタマイズする方法を学習する時に、このマニュアルを利用してください。

『4th Dimensionデザインリファレンス』は、「デザイン」モード内で有効な操作を詳しく説明した「デザイン」モードのリファレンスガイドです。パッケージ内の他の解説書とともにご利用してください。

『4th Dimensionユーザリファレンス』は、「ユーザ」モードに関する全情報を提供します。「ユーザ」モードとは、データを登録したり操作したりするデータベースやレイアウトを使用するモードです。

『4th Dimensionクイックスタート』は、実践演習をこなしながら、4th Dimensionのデータベースを作成および使用していきます。例題は、4th Dimensionの概念と機能を熟知できるように、簡単な体験学習方式になっています。

『4D Serverリファレンス』は、4D Serverをインストールする、または4D Serverを使ってマルチユーザデータベースを管理する際のリファレンスガイドです。このマニュアルは、4D Serverパッケージの中にのみ含まれています。

『4th Dimensionインストールガイド』は、4th Dimensionをインストールするための手引書です。

その他に、バージョンアップ等で新しく追加／修正された情報はオンラインドキュメントで提供されています。このドキュメントは、4th Dimensionをインストールする際にハードディスク上にインストールされます。

このマニュアルについて

この『4th Dimensionランゲージリファレンス』マニュアルは、4th Dimensionのプログラミング言語を詳細に説明したものです。このマニュアルでは、テーブル、フィールド、フォームなどの用語を既に習得していることを前提としています。このマニュアルを進む前に次のようなことを行ってください。

■ 『4th Dimensionクイックスタート』マニュアルを使って、例題のデータベースを実行してみる。

■ 自分自身でデータベースを作成してみる。必要に応じて、『4th Dimensionデザインリファレンス』マニュアルを参照する。

■ 「ユーザ」モードで作成したデータベースを使用してみる。「ユーザ」モードに関する詳細は『4th Dimensionユーザリファレンス』を参照する。

表記方法について

このマニュアルでは、複数の表記規則が使用されています。

■ 4th Dimensionの「メソッド」エディタにあわせ、コマンドは特定の文字を用いてすべて大文字で表記します（例：**OPEN DOCUMENT**）。関数（値を返すコマンド）は、1桁目は大文字で記述し、その後は小文字で記述します（例：**Change string**）。

■ コマンドの構文において、“{ }”（中括弧）は任意の引数を表わします。例えば、**SET DEFAULT CENTURY(世紀{; ピボット年度})**という構文では、コマンドを呼び出す際に引数＜ピボット年度＞が省略される可能性があることを意味します。

■ また、コマンドの構文において、“¦”はいずれか1つを選択するということを示します。例えば、**Table(テーブル番号 ¦ ポインタ)**という構文では、この関数がテーブル番号またはポインタのいずれかを引数として受け入れることを示します。

■ 本マニュアルの特定の例題において、スペースの不足により1行のコードが次行に継続されている場合があります。しかし、実際にはこれらの例題を改行しないで1行のコードとして記述してください。

注：このような注意書きは、重要な情報に対して注意を促しています。

警告：このような警告は、データが失われる可能性のある状況に対して注意を促しています。

また、このマニュアルはメソッドやコマンドを識別するために、次のような表記規則を使用します。

■ メソッドの例やコマンドは、次のように表記します。
例：Piece of Code

■ メソッドの例では、コマンドは太字（ボールド）で表記します。
例：**ADD RECORD**

■ 値を返さないコマンドは、すべて大文字で表記します。
例：**DEFAULT FILE**

■ 値を返すコマンド（関数）は、頭文字のみ大文字で表記します。
例：**Records in file**

■ プロジェクトメソッドは、斜体（イタリック）で表記します。
例：*My Proc*

■ プラグインは、太字斜体（ボールドイタリック）で表記します。
例：***My External***

■ コマンドの引数は、＜＞で囲んで表記します。
例：＜テーブル＞

イントロダクション

この章では、4th Dimensionのプログラミング言語を紹介します。以下のような事柄について説明します。

■ プログラミング言語とは何か、また何ができるのか

■ メソッドの使用方法

■ 4th Dimensionを使ったアプリケーション開発

ここでは、一般的な概要についてのみ述べることにして、詳細については章の終わりで説明します。

プログラミング言語とは何か

4th Dimensionのプログラミング言語は、英会話に似ています。つまり、概念を表す、情報を与える、または指示するためのコミュニケーション手段の役割をします。会話体の言語と同じように4th Dimensionも独自の語彙、文法、構文を持っています。プログラミング言語は、4th Dimensionに対してデータベースとデータの処理方法や管理方法を指示するために使用します。

プログラミング言語についてすべてを知る必要はありません。これは、話すために単語をすべて知る必要がないことと同じです。実際に、少ない語彙しかなくても雄弁に話すことができるように、プログラミング言語をすべて知らなくても立派なデータベースを作成することができます。開発するために知る必要があるのは、プログラミング言語のごく一部だけです。あとは必要に応じて習得すれば、それで十分です。

プログラミング言語の使用目的

最初は、4th Dimensionのプログラミング言語をほとんど必要とすることはありません。あらかじめ、「デザイン」モードと「ユーザ」モードにプログラミングを必要としない柔軟性のあるツールが用意されているため、さまざまなデータ管理処理を実現することができます。データ入力、検索、ソート、レポート等の、基本的な処理を簡単に実行するのはもちろんのこと、データチェック、データ入力支援、グラフやラベル作成等の多くの機能をプログラミング言語を用いずに使用することができます。

それでは、どうしてプログラミング言語が必要なのでしょうか？それには、いくつかの理由があります。例えば、以下のような場合にプログラミング言語が必要になります。

■ **「反復処理の自動化」**：データ修正、複雑なレポートの作成、一連の操作の自動実行

■ **「ユーザインタフェースの管理」**：ウインドウ管理、メニュー管理、フォーム管理、インタフェースオブジェクト管理

■「上級機能によるデータ管理の実行」：トランザクション処理、複雑なデータチェック、マルチユーザ管理、セット操作、命名セレクションの操作

■「コンピュータの管理」：シリアルポート通信、ドキュメント管理、エラー管理

■「データベースの作成」：「カスタム」モードで使用するカスタマイズされたデータベースの作成

■「内蔵4D Webサービスへの機能の追加」：4Dがフォームから自動的に変換するページに加えて、ダイナミックHTMLページの作成

4th Dimensionのプログラミング言語は、データベース処理を制御するためのものです。「ユーザ」モードは強力な生成ツールを備えていますが、必要に応じて4th Dimensionのプログラミング言語を使用することにより、データベースをカスタマイズすることができます。

データ制御の実施

4th Dimensionは、強力かつ柔軟にデータを制御します。このプログラミング言語は初心者にも簡単に使用することができます。また、アプリケーション開発者の高度な要求にも十分に対応することができます。データベース内蔵の制御からカスタマイズされたデータベースへスムーズに移行することもできます。

4th Dimensionのコマンドは、「ユーザ」モードで同じ操作をほとんど実行することができます。例えば、**QUERY**コマンドを使用すると「クエリ」エディタが表示されますが、これは「ユーザ」モードの「クエリ」メニューから「検索」メニューを選択することと同じです。また、この**QUERY**コマンドは特定のデータを検索することもできます。例えば、**QUERY**（[従業員] ; [従業員]名前="山田"）はデータベースの中から“山田”という名前のすべての人を検索します。

このプログラミング言語は非常に強力です。1つのコマンドで、従来のコンピュータ言語の何百あるいは何千ものステップに相当するものもあります。また、柔軟性に富み、強力であると同時にコマンドは平易な英語名を使用しているため非常に扱いやすくなっています。例えば、新しいレコードを追加するためには**ADD RECORD**コマンドを使用するだけでよいのです。

この言語は、ほとんどの標準的なデータ処理を簡単に実行できるように設計されています。レコードの追加やテーブルのソート、データの検索等の操作は単純で直接的なコマンドで指定できます。

従来のコンピュータ言語との相違点

従来のコンピュータ言語に馴れ親しんでいる方は、本章を参照してください。それ以外の方は、この章を読み飛ばしても構いません。

4th Dimensionのプログラミング言語は従来のコンピュータ言語とは異なります。この言語は、今日のコンピュータで使用できる最も先進的で柔軟性のある言語の1つです。4th Dimensionのプログラミング言語は、その言語自体で機能するように設計されています。

従来の言語を使用して開発を行う場合、まず広範な計画を立てる必要があります。そして計画の立案も開発の重要な工程の1つとなります。しかし、4th Dimensionはデータベースのあらゆる部分でプログラミング言語をいつでも使用することができます。例えば、フォームにオブジェクトメソッドを追加し、その後で1つまたは2つのメソッドを追加することができます。またデータがより高度になった場合でも、メニューから制御するプロジェクトメソッドを追加することもできます。つまり、4th Dimensionのプログラミング言語は、他の多くのデータベースのようにプログラミング言語が“すべてか、あるいは全くないか”ではなく、必要最低限のプログラミング言語を使用するだけでいいのです。

従来の言語の構文では、オブジェクトを定義または事前に宣言（定義）しなければなりませんでした。しかし、4th Dimensionではオブジェクトを作成し、それを使用するだけで構いません。4th Dimensionは自動的にオブジェクトを管理します。例えば、ボタンを使用するためには、ボタンをフォーム上に作成し、それを指定します。ユーザがボタンをクリックした時点で、そのことをメソッドに自動的に通知します。

従来の言語では、コマンドの使用を固定化したり限定する等して融通性に欠ける点が多いのに対して、4th Dimensionのプログラミング言語は優れたユーザインタフェースを実現しています。

メソッドはプログラミング言語の入口

メソッドは、4th Dimensionに処理を実行させるための一連の命令文です。メソッド内の各行を、“ステートメント”と呼びます。各ステートメントは、プログラミング言語の一部分（コマンド等）で構成されます。

ここでは、既に『はじめよう4D』を通読し、オブジェクトメソッドやプロジェクトメソッドを作成し使用した経験があることを前提に説明を行います。

4th Dimensionで使用するメソッドには、オブジェクトメソッド、フォームメソッド、プロジェクトメソッドの3種類があります。

■「オブジェクトメソッド」：フォームオブジェクトを制御するために使用する短いメソッド

■「フォームメソッド」：フォームの表示を管理するメソッド

■「トリガ」：データベースの規則を強制するためのメソッド

■「プロジェクトメソッド」：データベース全体で使用できるメソッド

■「データベースメソッド」：データベースのオープンやクローズのとき、またはWebブラウザがインターネットおよびイントラネット上でWebサーバとして発行されているデータベースに接続するときに、初期化や特別な動作を実行するメソッド

次の節では、各メソッドの紹介とデータベースを自動化する方法について説明します。すべてのメソッドタイプの詳細は、このマニュアルの第3章で紹介します。

オブジェクトメソッド入門

動作を実行できるフォーム（すなわち、アクティブオブジェクト）は、それに関連するメソッドを持つことができます。オブジェクトメソッドは、データ入力時や印刷時にアクティブオブジェクトの監視や管理を行います。オブジェクトをコピーして貼り付けると、オブジェクトメソッドがそのアクティブオブジェクトとともにコピーされます。これにより、作成したオブジェクトの再利用可能なライブラリを作成することができます。オブジェクトメソッドはまさに必要な場合のみ制御を行います。

オブジェクトメソッドは、データベースへの入り口であるユーザインタフェースを管理するための主要ツールです。ユーザインタフェースは、コンピュータがユーザと通信するための手順と規則から成り立っています。その最終目的は、データベースのユーザインタフェースをできるだけ簡単で使いやすくすることです。ユーザインタフェースは、コンピュータとのやり取りを快適にし、ユーザがそれを楽しめるように、または気にならないようにする必要があります。

フォームには、以下の2つの基本的なタイプのアクティブオブジェクトがあります。

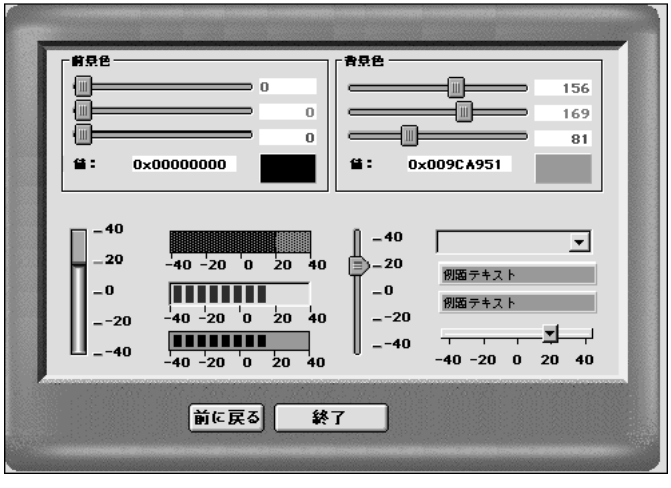
■ フィールドやサブフィールド等のデータの入力、表示、格納のためのアクティブオブジェクト

■ 入力エリア、ボタン、スクロールエリア、階層化リスト、メーター等のコントロールのためのアクティブオブジェクト

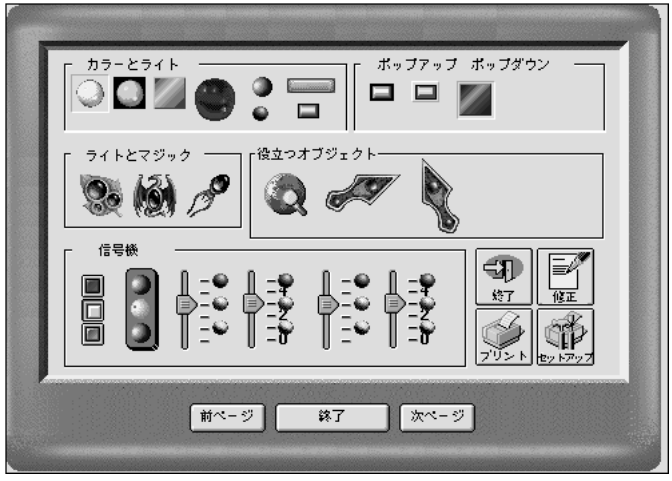
4th Dimensionでは、以下のようなクラシックフォームを作成することができます。



また、以下のような複数のグラフィックコントロールを使用してフォームを作成することができます。



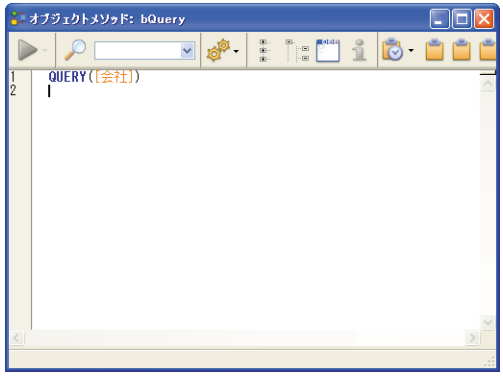
想像力が許す限りのグラフィックを使用したフォームを作成することもできます。



作成するスタイルがどのような形式であろうと、アクティブオブジェクトはすべてデータ入力エリアの範囲チェックや入力フィルタ、コントロール、メニュー、ボタン用の自動動作等の内蔵の支援機能を持っています。オブジェクトメソッドを追加する前には、必ずこれらの支援機能を使用します。内蔵の支援機能はメソッドと同様に、アクティブオブジェクトに関連付けられ、それがアクティブなのはアクティブオブジェクトが使用されるときだけです。通常、内蔵の支援機能とオブジェクトメソッドを組み合わせて使用して、ユーザインタフェースを制御します。

データ入力に使用するオブジェクトメソッドは、フィールドまたは変数に対して特定の処理を実行します。このオブジェクトメソッドは、データの属性チェック、フォーマット編集、計算処理、別テーブルのデータ取得等を行います。もちろん、これらの処理の一部は、4th Dimensionから提供されているデータ入力時のオブジェクトツールで実行することもできます。しかし、複雑な処理が必要な場合には、オブジェクトメソッドを使用します。このツールに関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

オブジェクトメソッドは、制御に使用するボタン等のオブジェクトにも作成することができます。これらのオブジェクトは、データベースを使用する上でとても重要です。これには、レコード間の移動や別フォームの使用、データの追加、修正、削除等を実行するボタンがあります。また、これらのオブジェクトは、データベースの使用を簡単にし、それを習得するために必要な時間を減少させます。ボタンにも、データ入力時と同様に4th Dimension内に組み込まれているツールの使用が可能です。オブジェクトメソッドを作成する前に、これらの組み込みツールを使用してください。オブジェクトメソッドを使用すれば、4th Dimension内に組み込まれていない動作を実現することができます。例えば、以下の図は、クリックされた時点で「クエリ」エディタを表示するボタンのオブジェクトメソッドを示しています。



オブジェクトメソッドに精通してくると、オブジェクトメソッドを持つオブジェクトのライブラリを作成すると便利なことに気付きます。これらのオブジェクトやオブジェクトメソッドをフォーム、テーブル、データベース間でコピーしたり、貼り付けることができます。また、必要な場合に使用できるように、スクラップブックにそれらを保存することもできます。

フォームメソッドを使ってフォームを制御する

オブジェクトメソッドがフォームオブジェクトに付随するのと同様に、フォームメソッドはフォームに付随します。1つのフォームには、1つのフォームメソッドしか持つことができません。

フォームはデータを表示します。フォームを使用することで、魅力的で使いやすいデータ入力画面やレポートを作成することができます。フォームメソッドはデータの入出力におけるフォームをチェックし、制御します。

フォームメソッドは、オブジェクトメソッドよりも上位のレベルでフォームを管理します。オブジェクトメソッドは属しているオブジェクトが使用された場合にのみ実行されます。フォームメソッドは、フォーム上のオブジェクト全体を制御するために使用します。つまり、オブジェクトメソッドが実行されるたびに、フォームメソッドも実行されます。

フォームは非常に多くの用途に用いられるため、フォームには、フォームが使用されている状況をチェックするための“フォームイベント”が用意されています。このフォームイベントは、いくつかのフェーズに分かれています。それぞれのフェーズはフォームの状況が変わると発生します。フォームイベントに関する詳細は、後の章で説明します。

トリガを使ってデータベースの規則を制御する

トリガはテーブルに設定されるため、テーブルメソッドとも呼ばれます。トリガは、テーブルのレコードを操作（追加、削除、修正、ロード）するたび、4Dデータベースエンジンによって自動的に起動されます。トリガはデータベースのレコードに対する「違法な」操作を防止することができます。例えば、請求書発行システムでは、請求書発行先の顧客を指定しない請求書を追加できないようにすることができます。トリガはテーブル上の操作を制限する非常に強力なツールであると同時に、偶発的なデータの損失や不正な変更を防止します。最初は簡単なトリガを作成しておいて、後で複雑にしていくこともできます。

トリガについての詳細は、後述の「トリガコマンド」を参照してください。

データベース全体でプロジェクトメソッドを使用する

特定のフォームに付随するフォームメソッドや、特定のオブジェクトに付随するオブジェクトメソッドと異なり、プロジェクトメソッドはデータベースのどこからでも使用することができます。また、プロジェクトメソッドは、繰り返して使用したり、別のメソッドの中から呼び出して使用することもできます。同じ処理を繰り返し実行する必要がある場合でも、それぞれに対して同一のメソッドを書く必要はありません。

プロジェクトメソッドは、別のプロジェクトメソッドやオブジェクトメソッド、フォームメソッドから必要な場所で随時呼び出されます。呼び出されたプロジェクトメソッドは、呼び出した場所にメソッド全体を書き込んだ時と同じように動作します。別のメソッドから呼び出されるプロジェクトメソッドのことを“サブルーチン”と呼びます。

プロジェクトメソッドを使用するもう1つの方法として、メニューへの割り当てがあります。メニューに割り当てられたメソッドは、メニューを選択した時に実行されます。メニューは、プロジェクトメソッドを呼び出すものと考えることができます。

データベースメソッドによる作業セッションの取り扱い

フォームにイベントが発生したときに、オブジェクトメソッドとフォームメソッドが起動するのと同様に、作業セッションイベントの発生時に起動されるデータベースに関連付けるメソッドがあります。これがデータベースメソッドです。例えば、データベースをオープンするたびに、作業セッション全体で使用する変数をいくつか初期化したい場合があります。これには、データベースをオープンしたときに4Dが自動的に実行する「On Startup」データベースメソッドを使用します。

データベースメソッドについての詳細は、後述の「データベースメソッド」の節を参照してください。

データベースを作成する

データベースを作成することは、プログラミング言語や4th Dimension内に用意されている内蔵ツールを使用してデータベースをカスタマイズするということです。

つまり、データベースを作成するだけで、プログラミング言語の第一歩を踏み出したことになります。データベースのテーブル、フィールド、フォーム、オブジェクト、メニューの各部分は、プログラミング言語と密接に関連しています。プログラミング言語は、データベースのこれらの部分について、すべて“熟知”しています。

プログラミング言語を初めて使用する場面は、おそらくデータ入力を制御するために、フォームオブジェクトやオブジェクトメソッドを作成する場合でしょう。そして、フォームの表示を制御するために、フォームメソッドを作成します。大規模なデータベースの場合、データベースを完全にカスタマイズするには、プロジェクトメソッドとメニューバーを作成する必要があります。

4th Dimensionは、その他についても非常に柔軟性があります。データベースの作成に、決まった方法があるわけではありません。自分自身に合った方法で作成することができます。もちろん、以下のようないくつかの一般的なパターンはあります。

■「実行」：「デザイン」モードで設計したものを実行する。

■「テスト」：設計したものを「ユーザ」モードでテストしたり、カスタマイズしたデータベースを使ってみる。

■「使用」：カスタマイズしたデータベースが完成したら、「カスタム」モードで使ってみる。

■「訂正」：エラーを見つけたら、「デザイン」モードに戻って、そのエラーを訂正する。

データベース作成に、特別な支援ツールが用意されています。これらは4th Dimension内部に組み込まれており、必要になるまでは隠れていますが、プログラミング言語の使用に熟知するにつれて、これらのツールが開発プロセスを容易にしてくれることがわかってきます。例えば、「メソッド」エディタは入力エラーを検知したり、シンタックスのネストレベル（入れ子の階層）を表示します。また、インタプリタ（4th Dimension内部に組み込まれたプログラミング言語の実行制御システム）はシンタックス中のエラーを取り出し、どこが誤りかを示します。デバッガは、メソッド中のエラーを見つけるために、実行中のメソッドをリスト表示します。

アプリケーションを構築する

これで、データ入力、検索、ソート、レポート等のデータベースの一般的な使用方法について理解されたと思います。これらの処理は、4th Dimension内に組み込まれているメニューとエディタを使用して「ユーザ」モードで実行してきました。

データベースを使用するにつれて、繰り返して実行する一連の処理があることが明らかになります。例えば、個人連絡先のデータベースにおいて、データを変更するたびに、仕事上の知人の検索、名字でソート、特定のレポートを印刷すると仮定します。この場合、これらの処理はとても単純ですが、この処理を20回以上も実行するとなると、とても大変です。また、数週間データベースを使用しなかった場合には、レポートを作成する手順を克明に憶えているとも限りません。しかし、メソッドを作成すれば、1つのメニューを選択するだけで、一連の処理を自動的に実行してくれるためレポートを作成するための手順を忘れていても大丈夫です。

アプリケーションは、データベースを自動化します。アプリケーションは、データベースを使用するユーザのニーズに合ったカスタムメニューや、特殊な実行処理用のメニューを持つことができます。アプリケーションは、データベースの要素（ストラクチャ、フォーム、オブジェクトメソッド、メソッド、メニュー、パスワード等）から構成されます。

アプリケーションには、人の名前を入力して、レポートを印刷するだけの単純なものから、請求・在庫管理システムといった複雑なものまであります。アプリケーションの使用に関しては特に制限事項のようなものはありませんが、一般的にデータベースは、「ユーザ」モードで使用するものからカスタムメニューで完全に管理する大規模なものまでさまざまです。

アプリケーションの開発は、ユーザの思うままに複雑にも簡単にもすることができます。

4Dアプリケーションを構築する

アプリケーションは、特定のニーズを満たすために設計されたデータベースです。アプリケーションには、円滑に利用できるよう特別に作成されたユーザインタフェースがあります。あるアプリケーションが実行する作業は、その目的に合致したものに限定されます。4th Dimensionを使用すると、アプリケーション作成は従来のプログラミングよりも速やかに、かつ容易に行うことができます。4th Dimensionは、次に示すようなさまざまなアプリケーション作成に利用可能です。

■ 請求書システム

■ 在庫管理システム

■ 会計システム

■ 給与システム

■ 人事システム

■ 顧客管理システム

■ インターネットやイントラネット経由による共有データベース

これらすべてのシステムを一つのアプリケーションに納めることもできます。このようなアプリケーションは、データベースの代表的な利用例です。さらに、4th Dimensionのツールを使用すると、次のような画期的なアプリケーションを作成することができます。

■ 文書管理システム

■ 画像管理システム

■ カタログ発行アプリケーション

■ シリアルデバイス制御／監視システム

■ 電子メールシステム（E-メール）

■ マルチユーザスケジューリングシステム

■ メニューリスト、ビデオコレクション、音楽コレクション等の一覧

通常、アプリケーションは、「ユーザ」モードで使用するデータベースから始まります。カスタマイズするにつれて、データベースがアプリケーションへと“進化”していきます。アプリケーションの異なる点は、データベース管理に必要となるシステムがユーザの目に触れないというところです。データベース管理は自動化され、ユーザはメニューを利用して特定の作業を実行します。

4th Dimensionデータベースを「ユーザ」モードで使用する場合、結果を実現するために必要となる手順を知っておかなければなりません。アプリケーションの「カスタム」モードを使用するには、「ユーザ」モードでの自動化されているすべての事柄を管理する必要があります。

■ **テーブル操作：**ユーザは「テーブル／フォーム選択」ダイアログボックスおよび「テーブルリスト」ウインドウを利用することができません。メニューコマンドとメソッドを使用して、テーブル操作を制御することができます。

■ **メニュー：**「カスタム」モードには、「終了」メニューコマンドが納められたデフォルトの「ファイル」メニュー、「編集」メニュー、「ヘルプ」メニュー（Windowsのみ）またはアプリケーションメニュー（Macintoshのみ）だけが存在します。アプリケーションでさらにメニューが必要となる場合、4Dメソッドを用いてこれらのメニューを作成し、管理しなくてはなりません。

■ **エディタ：**「カスタム」モードでは、「クエリ」や「並び替え」等のエディタを自動的に利用することはできません。これらのエディタを「カスタム」モードで使用したい場合、4Dメソッドを用いて呼び出す必要があります。

次の節では例題を用いて、プログラミング言語によってデータベースの使用を自動化する方法を紹介します。

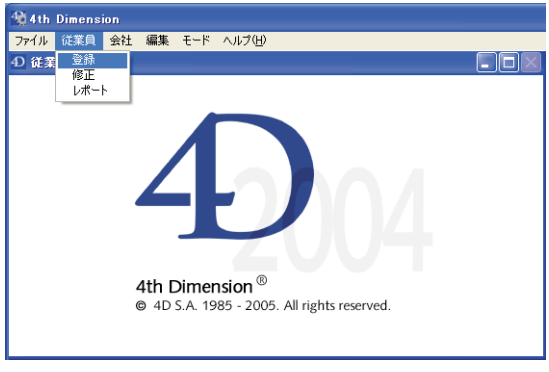
カスタムメニュー：例題

カスタムメニューは、アプリケーションにおける主要なインタフェースです。このメニューにより、ユーザはより簡単にデータベースの習得や利用を行えるようになります。カスタムメニューは、「メニュー」エディタを使用して、各メニューコマンド（メニュー項目とも呼ばれる）にメソッドや自動アクションを関連付けるだけで簡単に作成することができます。

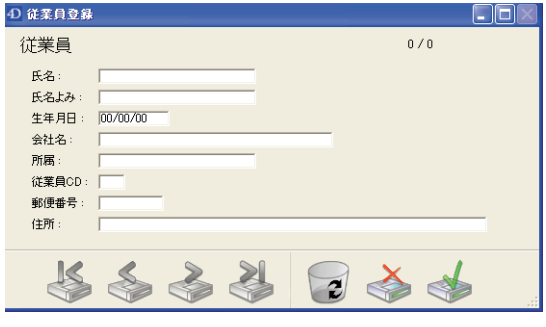
「ユーザの視点」の節では、ユーザがメニューコマンドを選択した後の事象について説明します。次の節「メニューの舞台裏」では、この動作を実現するための設計について説明します。例題はシンプルですが、カスタムメニューを用いることにより、データベースの使用や習得がいかに楽になるかが理解できます。「ユーザ」モードの“一般的な”ツールやメニューコマンドではなく、ユーザはそれぞれのニーズに合った事柄だけに注目することができます。

ユーザの視点

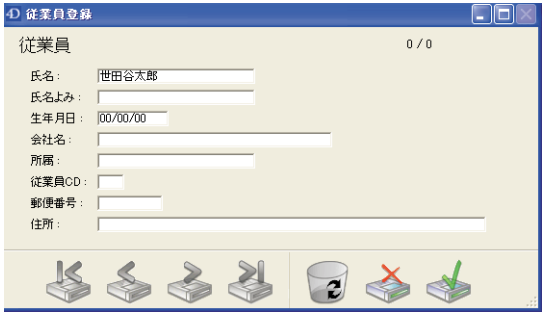
ユーザは「従業員」メニューから「登録」というメニュー項目を選択し、新しい従業員をデータベースに追加します。



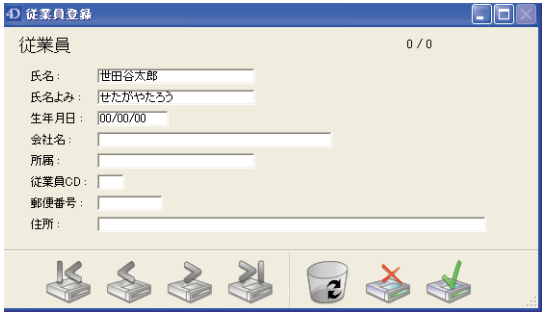
従業員テーブルの入力フォームが表示されます。



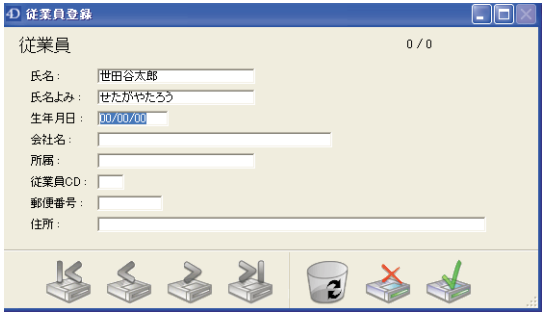
ユーザは従業員の名前を入力し、タブで次のフィールドへ移動します。



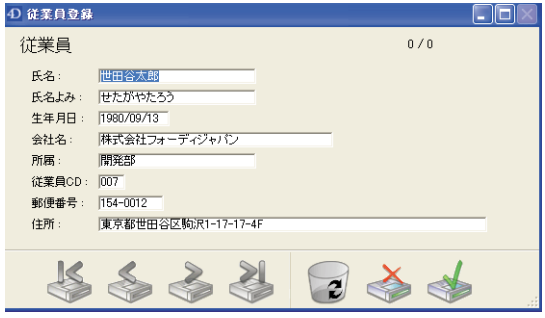
ユーザは従業員のよみがなを入力します。



ユーザはタブで次のフィールドへ移動します。



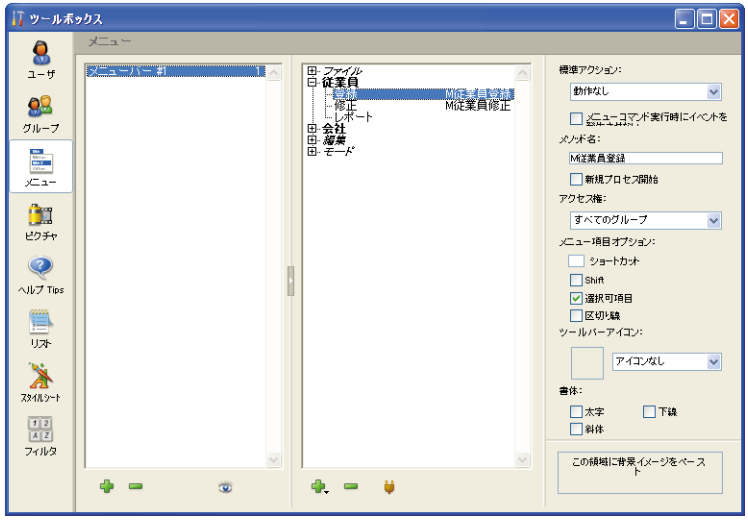
ユーザはレコードの入力を終了し、「登録」ボタンをクリックします（通常は、ボタンバー上の最後のボタン）。



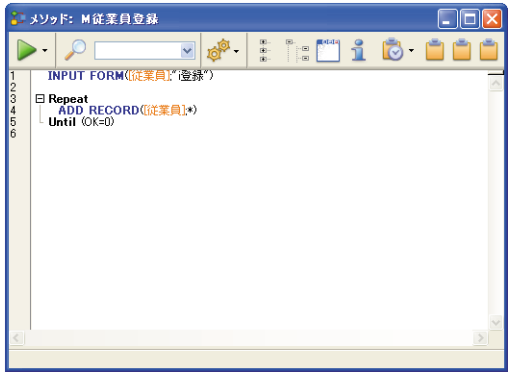
空のレコードが表示されるので、ユーザは「キャンセル」ボタン（“X”印が付いたボタン）をクリックして“データ入力ループ”を終了します。再びメニューバーが表示されます。

メニューの舞台裏

メニューバーは、「デザイン」モードの「メニューバー」エディタを使用して作成します。



「登録」メニュー項目には、“M従業員登録”という名前のプロジェクトメソッドが関連付けられています。このメソッドは、「デザイン」モードの「メソッド」エディタを使用して作成されています。



ユーザがこのメニュー項目を選択すると、メソッド“M従業員登録”が実行されます。

```
Repeat
        ADD RECORD([従業員])
Until(OK=0)
```

ADD RECORDコマンドが指定された「**Repeat**...**Until**」ループは、「ユーザ」モードにおける「新規レコード」メニュー項目と同様の動作を行います。このループではユーザに入力フォームが表示され、新規レコードを追加することができます。ユーザがレコードを保存すると、新たに空のレコードが表示されます。この**ADD RECORD**ループは、ユーザが「キャンセル」ボタンをクリックするまで実行を続けます。

レコードを入力すると、以下の作業が行われます。

レコードが入力され、ユーザが次のレコードで「キャンセル」ボタンをクリックすると、システム変数OKが0に設定されて**ADD RECORD**ループが終了します。

次に実行するステートメントが存在しないため、“M従業員登録”メソッドは実行を終了し、制御がメニューバーに戻ります。

「ユーザ」モードで実行される自動タスクとアクションの比較

あるタスクを「ユーザ」モードで実行する方法と、プログラミング言語で同じタスクを実行する方法とを比較してみましょう。実行するタスクは共通のものです。

■ レコードグループの検索

■ 並び替え

■ レポートの印刷

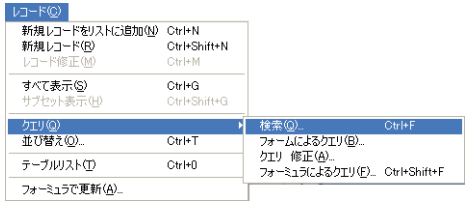
次の節の「ユーザモードにおけるデータベースの使用」では、「ユーザ」モードで実行されるタスクを示します。

その次の節、「カスタムモードにおける組み込みエディタの使用」では、同じタスクがアプリケーションで実行される様子を示します。

これらのメソッドはともに同じタスクを実行しますが、2番目の節の手順はプログラミング言語を使用して自動化されている点に注目してください。

ユーザモードにおけるデータベースの使用

ユーザは、「レコード」メニューから「クエリ＞検索...」を選択します。

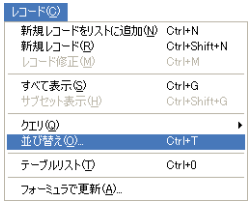


「クエリ」エディタが表示されます。

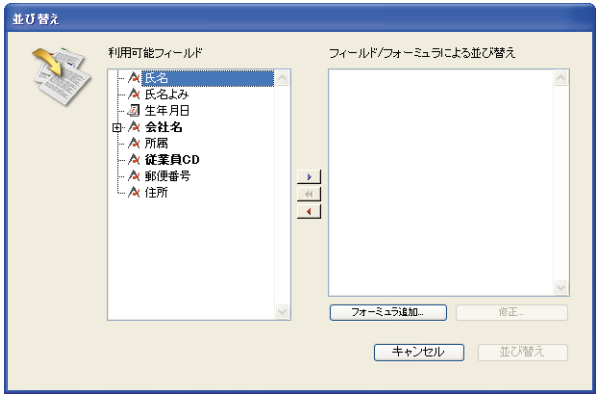


ユーザが検索条件を入力して「クエリ」ボタンをクリックすると、検索が実行されます。

ユーザは「レコード」メニューから「並び替え...」を選択します。



「並び替え」エディタが表示されます。



ユーザが並び替え条件を入力して「並び替え」ボタンをクリックすると、並び替えが実行されます。

この後、レコードを印刷するには、さらに次の手順が必要となります。

■ ユーザが「ファイル」メニューから「プリント...」を選択します。

■ いずれのフォームを印刷するかをユーザが認識する必要があるため、「プリント」ダイアログボックスが表示されます。

■ 印刷用ダイアログボックスが表示されます。ユーザが各設定を選択すると、レポートが印刷されます。

「カスタム」モードにおける組み込みエディタの使用

前述の作業が「カスタム」モードではどのように行われるかを検証しましょう。

ユーザは、「従業員」メニューから「レポート」を選択します。

すでにこの時点で、ユーザは最初の手順がクエリであることを認識する必要がないため、アプリケーションを使用する方がより簡単になります。

メニューコマンドには“M従業員印刷”というメソッドが関連付けられています。このメソッドは次の通りです。

```
QUERY([従業員])
ORDER BY([従業員])
OUTPUT FORM([従業員] ;"印刷")
PRINT SELECTION([従業員])
```

最初の行が実行されます。

```
QUERY([従業員])
```

「クエリ」エディタが表示されます。



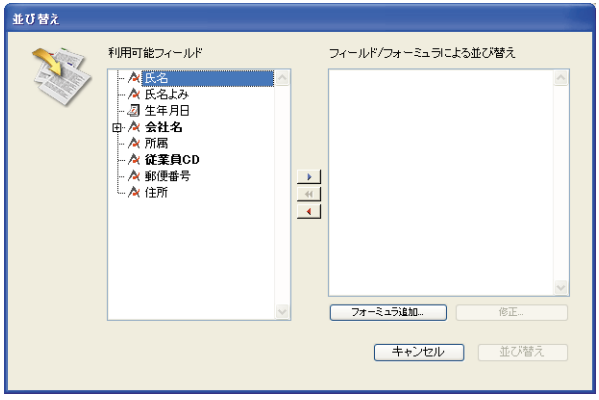
ユーザが検索条件を入力し「クエリ」ボタンをクリックすると、検索が実行されます。

次に、メソッド“M従業員印刷”の2行目が実行されます。

```
ORDER BY([従業員])
```

ユーザは次の手順がレコードの並び替えであることを認識する必要がない点に注目してください。

「並び替え」エディタが表示されます。



ユーザが並び替え条件を入力し「並び替え」ボタンをクリックすると、並び替えが実行されます。

次に、メソッド“M従業員印刷”の3行目が実行されます。

```
OUTPUT FORM([従業員] ; "印刷")
```

ここでもユーザは、次に実行することを知る必要がありません。このメソッドがすべて処理します。

最後に、メソッド“M従業員印刷”の最終行が実行されます。

```
PRINT SELECTION([従業員])
```

印刷用ダイアログボックスが表示されます。ユーザが各設定を選択すると、レポートが印刷されます。

アプリケーションをさらに自動化する

前述の例題で使用したものと同じコマンドを使用し、データベースの自動化をさらに進めることができます。

新しくなった“M従業員印刷”メソッドを見てみましょう。

ユーザは、「従業員」メニューから「レポート」を選択します。メニューコマンドには“M従業員印刷”というメソッドが関連付けられています。このメソッドは次の通りです。

```
QUERY([従業員]; [従業員]会社="刀根エンタープライズ")
ORDER BY([従業員];[従業員]名字;>;[従業員]名前;>)
OUTPUT FORM([従業員];"印刷")
PRINT SELECTION([従業員];*)
```

最初の行が実行されます。

```
QUERY([従業員]; [従業員]会社="刀根エンタープライズ")
```

「クエリ」エディタは表示されません。その代わりに、**QUERY**コマンドにより検索条件が指定され実行されます。ユーザは何も行う必要はありません。

メソッド“M従業員印刷2”の2行目が実行されます。

```
ORDER BY([従業員];[従業員]名字;>;[従業員]名前;>)
```

「並べ替え」エディタは表示されず、並び替えが即座に実行されます。ここでもユーザによる処理は必要ありません。

メソッド“M従業員印刷2”の最終行が実行されます。

```
OUTPUT FORM([従業員];"印刷")
PRINT SELECTION([従業員];*)
```

印刷用ダイアログボックスは表示されません。**PRINT SELECTION**コマンドは、任意の引数であるアスタリスク（*）を受け入れ、これによりコマンドはレポートフォームの作成時点に指定された印刷設定を使用し、レポートが印刷されます。

このように自動化をさらに進めることにより、ユーザは3つのダイアログボックスでオプションを入力する必要がなくなりました。これらの利点は次の通りです。

- **クエリが自動的に実行される：**ユーザが指定すると、クエリの作成時に誤った条件を選択する可能性があります。
- **並び替えが自動的に実行される：**ユーザが指定すると、並び替えの定義時に誤った条件を選択する可能性があります。
- **印刷が自動的に実行される：**ユーザが指定すると、誤ったフォームを選択して印刷する可能性があります。

4Dアプリケーション開発用ツール

4Dアプリケーションの開発を進める上で、最初は気付かなかった数多くの機能を発見することでしょう。また、他のツールやプラグインを4D開発環境に追加し、標準の4Dの機能を高めることができます。

ツールと4Dプラグイン

4Dアプリケーションの機能を向上させる各種ツールならびにプラグインを提供しています。

■ 4D Insider：このツールを使用すると、作成した4th Dimensionデータベースの相互参照を行うことができます。また、メソッドや変数、コマンド、プラグイン、ストラクチャ、リスト、フォームの表示や印刷のために使用することも可能です。相互参照機能により、これらのオブジェクトがそれぞれ、データベース内のいずれの箇所で使用されているかを知ることができます。さらに、テーブル、フォーム、メソッド、メニューバー、リスト、パッケージ、スタイル等のオブジェクトをデータベース間で移動することができます。

4D社では、次のようなプラグインを提供しています。

■ 4D Write：ワードプロセッサ

■ 4D Draw：グラフィカルな描画用プログラム

■ 4D View：スプレッドシートおよびリストエディタ

■ 4D Internet Commands：インターネットによる通信用ユーティリティ

■ 4D ODBC 2004（ビルトイン）：ODBC経由での接続

■ 4D for OCI：ORACLE Callインタフェースを用いた接続

■ 4D Open for Java：Javaアプリケーションを用いた接続

■ 4D Open for 4D：分散型4D情報システム構築用の接続（4D間接続）

詳細に関しては、4D社または4Dパートナーまでお問い合わせください。Webサイトでもこれらの情報を提供しています。

USAおよびインターナショナル：

http://www.4d.com

フランス：

http://www.4d.fr

日本：

http://www.4d-japan.com

4Dコミュニティおよびサードパーティツール

世界各国において、ユーザグループや電子フォーラム、4Dパートナーから構成されている4Dコミュニティが意欲的に活躍しています。4Dパートナーからはサードパーティツールが提供されています。4D CDには、4Dパートナー提供のデモや情報が収録されています。Web上で4Dパートナーに関する情報をご覧になってください。次のアドレスで、4th Dimensionのユーザフォーラムに登録することができます。

http://forum.4D.fr

4Dコミュニティからは、ヒントやTips、ソリューション、情報、その他のツールが提供され、開発にかかる時間やエネルギーを節約して生産性を高めることが可能です。

4th Dimensionのプログラミング言語

4th Dimensionのプログラミング言語は、以下のような要素から構成されています。これらの要素は、処理を実行、またはデータを管理する際の手助けをします。

■ **データタイプ：**データベース内のデータの分類。詳細はこの章の後述および「データタイプ」の章を参照してください。

■ **変数：**メモリ内でデータを一時的に記憶する領域。詳細は「変数」の章を参照してください。

■ **演算子：**2つの値を計算するために使われる記号。詳細はこの章の後述および「演算子」の章を参照してください。

■ **式：**値を求めるための構成要素の組み合わせ。詳細はこの章の後述を参照してください。

■ **コマンド：**処理を実行するために4th Dimensionに組み込まれている命令文。すべての4Dコマンド、例えば**ADD RECORD**コマンドなどはテーマ別このマニュアルに記載されています。また必要な場合は、序章に続きテーマの記載があります。4Dプラグインを使い、4Dの開発環境に新規のコマンドを追加することもできます。例えば、4Dシステムに4D Writeを追加すると、4D Writeのコマンドを利用してワードプロセスのドキュメントを作成および操作を行うことが可能になります。

■ **メソッド：**このマニュアルに記載されたプログラミング言語による命令文。詳細はメソッドの章および関連項目を参照してください。

この章では、データタイプ、演算子、式について説明します。その他の内容については、上記の章を参照してください。

追記：

・言語を構成する要素、例えば変数には識別子という名前がつけられています。識別子についての詳細およびオブジェクトに名前をつける場合の規則については識別子の章を参照してください。

・配列変数についての詳細は「配列コマンド」の章を参照してください。

・BLOB変数についての詳細は「BLOBコマンド」の章を参照してください。

・データベースをコンパイルする場合は、「コンパイラコマンド」の章および『4th Dimensionデザインリファレンス』マニュアルを参照してください。

データタイプ

4th Dimensionのデータベースで使用するデータには、いくつかの種別があります。これらのデータ種別を“データタイプ”と呼びます。

4th Dimensionには、以下のような8つのデータタイプがあります。

■ **文字列：**“こんにちは”等の一連の文字

文字フィールドとテキストフィールドは、ともに文字列タイプです。

■ **数値：**2や1,000.67等の数値

整数フィールド、倍長整数フィールド、実数フィールドはすべて数値タイプです。

■ **日付：**97.07.20等の日付

日付フィールドは、日付タイプです。

■ **時間：**1:00:00や4:35:30 P.M.等の時:分:秒

時間フィールドは、時間タイプです。

■ **ブール：**“True（真）”または“False（偽）”のどちらかの値を返すもの。

ブールフィールドは、ブールタイプです。

■ **ピクチャ：**PICTタイプで作成されたピクチャ

ピクチャフィールドは、ピクチャタイプです。

■ BLOB（Binary Large OBject）：サイズが2GB以内のバイト群

BLOBフィールドは、BLOBタイプです。

■ **ポインタ：**上級プログラミングで使用する特殊タイプのデータ

対応するフィールドはありません。

注：これらのデータタイプに格納されるデータは、『4th Dimensionデザインリファレンス』の中で説明されている4th Dimensionフィールドの中に格納されるデータと一致します。例外は、255バイトまで含むことができる文字列変数です。

データタイプの中で文字列タイプと数値タイプが、複数の類似するフィールドタイプに対応する点に注意してください。プログラミング言語でこれらのタイプのフィールドを操作する場合、プログラミング言語が自動的にサポートするデータタイプにデータを変換します。例えば、整数フィールドを使用すると、自動的に数値として扱います。つまり、プログラミング言語として使用する場合は、類似するフィールドタイプと厳密に区別する必要はありません。

しかし、プログラミング言語を使用する場合は、異なるデータタイプと混同しないようにすることは重要です。“ABC”を日付フィールドに格納しても意味がないように、日付として使用する変数に“ABC”を格納することも意味がありません。4th Dimensionは、実行したことをできるだけ有効にしようとします。例えば、日付に数値を加算した場合は、日付に日数を加算したいものと認識します。しかし、日付に文字列を加算した場合には、4th Dimensionはその操作に意味を持たないことをユーザに警告します。

あるタイプとしてデータを格納したものを、別のタイプとして使用する場合があります。4th Dimensionのプログラミング言語には、あるタイプから別のタイプへ変換するためのコマンドが用意されています。

例えば、数値で始まり、“abc”等の文字で終了する部品番号を作成するような場合、以下のように記述することができます。

```
部品番号:=String (数値)+"abc"
```

“数値”が17であれば、部品番号に“17abc”という文字列が代入されます。

演算子

プログラミング言語を使用する際に、データのみを必要とする場合は非常に稀です。データを加工、またはそれを何らかの目的のために使用することがほとんどです。例えば、演算子を使用して2つのデータから1つの新しいデータを生成する場合です。1+2は2つの数値を加算（演算子+）して、3という結果を返します。以下に、よく知られている4つの演算子を示します。

演算子	実行される演算	例
+	加算（足し算）	1 + 2の結果は3になります。
-	減算（引き算）	3 - 2の結果は1になります。
*	乗算（掛け算）	2 * 3の結果は6になります。
/	除算（割り算）	6 / 2の結果は3になります。

数値演算子は、使用可能な演算子のうちの1つのタイプにすぎません。

4th Dimensionは、数値、テキスト、日付、ピクチャ等、異なるタイプのデータを扱うために、各データタイプに対する演算を実行するための演算子を備えています。

同じ記号でも処理を実行するデータタイプによって、異なる意味に使用される場合があります。例えば、プラス記号（+）は下記のように、各データタイプによって異なる演算を実行します。

データタイプ	実行される演算	例
数値	加算	1+2は数値を加算し、結果は3になります。
文字列	連結（結合）	"みなさん　"+"こんにちは"は、文字列を連結し、結果は"みなさん　こんにちは"になります。
日付と数値	日付の加算	!97.07.01!+20は、1997年7月1日に20日を加算し、結果は!97.07.21!（1997年7月21日）になります。

式

式は、値を返します。実際に、プログラミング言語として使用している場合は、常に式を使用します。式は、“フォーミュラ”と呼ぶこともあります。

式は、コマンド、演算子、変数、フィールド等プログラミングのほとんどの部分で使用します。式により、ステートメント（メソッドの1文や1行）を構成します。必要に応じて、メソッドの任意の場所に式を使用することができます。

式が単独で使われることはほとんどありませんが、単独で使用できる場合がいくつかあります。「ユーザ」モードの「フォーミュラで検索」ダイアログボックスの場合や、デバッガで式の値をチェックする場合、「フォーミュラで更新」ダイアログボックスの場合、およびカラムをフォーミュラとして使用する「クイックレポート」エディタの場合です。

数値4または文字列“こんにちは”等の定数だけで構成された式の値は、常に一定で、決して変わることはありません。

演算子を使用すると、式でさまざまなことを実行することができます。前節で、演算子を使用する式を既に紹介しました。例えば、4+2は加算演算子を使用した式で2つの数値を加算し、結果に6を返します。

データタイプによって、以下の7つのタイプの式を使用することができます。

- 文字列式
- 数値式
- 日付式
- 時間式
- ブール式
- ピクチャ式
- ポインタ式

以下の表に、7つのタイプの式ごとに例を示します。演算子の詳細については、第33章を参照してください。

式	タイプ	説明
"みなさん　"	文字列	これは、文字列定数の"みなさん　"です。これが文字列定数であることを示すために2重引用符を使用している点に注意してください。
"みなさん　"+"こんにちは"	文字列	2つの文字列"みなさん　"と"こんにちは"を、文字列連結演算子（+）で連結して、文字列"みなさん　こんにちは"を返します。
[従業員]名前+"さん"	文字列	名前フィールドの現在の値と"さん"の2つの文字列を連結します。名前フィールドの値が"スミス"の場合は、"スミスさん"を返します。
Uppercase ("smith")	文字列	この式は、文字列"smith"を大文字に変換するために、**Uppercase**関数を使用して"SMITH"を返します。
4	数値	これは、数値定数4です。
4*2	数値	掛け算演算子を使用して、2つの数値4と2が乗算します。結果は数値8になります。
私のボタン	数値	これは、ボタンの名前でボタンの現在の値を返します。クリックされた場合は1を、クリックされない場合には0を返します。
!97.07.21! 日）	日付	これは、日付定数!97.07.21!（1997年7月21です。感嘆符（!）を使用して日付定数であることを示します。

式	タイプ	説明
Current date+30	日付	これは**Current date**関数を使用して、現在の日付を得るための日付式です。これは現在の日付に30日を加算した新しい日付を返します。
!!8:05:30!!	時間	これは、時間定数!!8:05:30!!（8時間5分30秒）です。2 つの感嘆符（!!）を使用して時間定数であることを示します。
!!2:03:04!!+!!1:02:03!!	時間	これは、2つの時間を加算し、!!3:05:07!!を返します。
True	ブール	これは、ブール値“True（真）”を返すコマンドです。
10#20	ブール	これは、2つの数値の間の論理比較です。この数値記号（#）は“等しくない”ことを意味します。10と20は、等しくないため“True（真）”を返します。
"ABC"="XYZ"	ブール	これは、2つの文字列の間の論理比較です。結果は等しくないため、式は“False（偽）”を返します。
私の絵+50	ピクチャ	これは、私の絵というピクチャを右へ50ピクセル移動します。
->[従業員]名前	ポインタ	この式は、[従業員]名前フィールドのポインタを返します。
Table (1)	ポインタ	この式は、テーブル番号1のテーブルのポインタを返します。

データタイプ

4th Dimensionのフィールド、変数、式は以下のデータタイプになります。

データタイプ	フィールド	変数	式
文字列（備考1参照）	○	○	○
数値（備考2参照）	○	○	○
日付	○	○	○
時間	○	○	○
ブール	○	○	○
ピクチャ	○	○	○
ポインタ	×	○	○
BLOB（備考3参照）	○	○	×
配列（備考4参照）	×	○	×
サブテーブル	○	×	×
未定義	×	○	○

備考

1. 文字列には、文字フィールド、固定長変数、テキストフィールド、テキスト変数があります。
2. 数値には、実数、整数、倍長整数のフィールドと変数があります。
3. BLOBはBinary Large OBjectの省略形です。BLOBの詳細については、「BLOBコマンド」の章を参照してください。
4. 配列にはすべてのタイプの配列が含まれます。詳細については、「配列コマンド」の章を参照してください。

文字列

文字列とは、以下を示す総称です。

■ 文字フィールド

■ 固定長変数

■ テキストフィールドまたはテキスト変数

■ 任意の文字列またはテキスト式

文字列は、文字で構成されます。ASCIIコードと、クロスプラットフォーム環境で4Dが ASCIIコードを処理する手順については、ASCIIコードの節を参照してください。

■ 文字フィールドは0から80バイトまでの文字を含むことができます（文字数の制限はフィールド定義で決まります）。

■ 固定長変数は0から255バイトまでの文字を含むことができます（文字数の制限は変数宣言によって決まります）。

■ テキストフィールド、テキスト変数、テキスト式は0個から32,000個までの文字を含むことができます。

テキストフィールドには文字列を代入することができ、また、反対に文字列にテキストフィールドの値を代入することができます。4Dが変換を行い、必要に応じて切り捨てを行います。文字列とテキストは1つの式の中で混在させて使用できます。

注：このマニュアルでは、コマンド説明における文字列とテキストの引数（パラメータ）は特に明記されていない限り、両方とも「文字列」と表記されています。

数値

数値とは、以下を示す総称です。

■ 実数のフィールド、変数、または式

■ 整数のフィールド、変数、または式

■ 倍長整数のフィールド、変数、または式

実数データタイプの範囲は、±1.7e±308（15バイト）です。

整数データタイプ（2バイト整数）の範囲は、-32,768から32,767まで（2^15から(2^15)-1まで）です。

倍長整数データタイプ（4バイト整数）の範囲は、-2^31から(2^31)-1までです。

数値データタイプは、異なる数値データタイプに代入することができます。このとき、4th Dimensionが必要に応じて変換、切り捨て、丸め処理を行います。ただし、値が範囲外の場合には、変換は正しい値を返しません。数値データタイプは式の中に混在させて使用することができます。

注：このマニュアルでは、実際のデータタイプに関わらず、コマンド説明における実数、整数、倍長整数の引数（パラメータ）は特に明記されていない限り、「数値」と表記されています。

日付

■ 日付フィールド、変数、式として認識できる範囲は、100/1/1から32,767/12/31までです。

■ 日付の順序は、「年．月．日」の順になります。

■ 年を2桁で指定すると2000年代であると解釈されます（このデフォルトが**SET DEFAULT CENTURY**コマンドで変更されていない場合に限る）。

注：このマニュアルでは、コマンド説明における日付引数（パラメータ）は特に明記されていない限り、「日付」と表記されています。

時間

■ 時間のフィールド、変数、式の範囲は00:00:00から596,000:00:00までです。

■ 時間の順序は、「時：分：秒」の順になります。

■ 時間は24時間制です。

■ 時間の値は数値として扱うことができます。時間から返される数値は時間が表す秒数です。詳細については後述の「時間演算子」の節を参照してください。

注：このマニュアルでは、コマンド説明における時間引数（パラメータ）は特に明記されていない限り、「時間」と表記されています。

ブール

ブールのフィールド、変数、式は、TRUE（真）またはFALSE（偽）のいずれかになります。

注：このマニュアルでは、コマンド説明におけるブール引数（パラメータ）は特に明記されていない限り、「ブール」と表記されています。

ピクチャ

ピクチャのフィールド、変数、式は、任意のWindowsまたはMacintoshのピクチャになります。一般的に、クリップボード上に置いたり、4Dコマンドやプラグインコマンドを使用してディスクから読み出すことのできる任意のピクチャがこれに当てはまります。

注：このマニュアルでは、コマンド説明におけるピクチャ引数（パラメータ）は特に明記されていない限り、「ピクチャ」と表記されています。

ポインタ

ポインタの変数または式は、別の変数（配列、配列要素を含む）、テーブルまたはフィールドへの参照です。ポインタタイプのフィールドは、存在しません。

ポインタの詳細については、第9章を参照してください。

注：このマニュアルでは、コマンド説明におけるポインタ引数（パラメータ）は特に明記されていない限り、「ポインタ」と表記されています。

BLOB

BLOBのフィールドまたは変数は、個別にまたはBLOBコマンドを使用して利用できる一連のバイト群（長さは0バイトから2ギガバイトまで）です。BLOBタイプの式は、存在しません。

注：このマニュアルでは、コマンド説明におけるBLOB引数（パラメータ）は特に明記されていない限り、「BLOB」と表記されています。

配列

配列は、実際にはデータタイプではありません。さまざまな種類の配列（整数配列、テキスト配列等）はこの配列という名前の下にグループ化されています。配列は、変数です。配列タイプのフィールドは存在せず、配列タイプの式も存在しません。配列の詳細については、第9章を参照してください。

注：このマニュアルでは、コマンド説明における配列引数（パラメータ）は特に明記されていない限り、「配列」と表記されています。

サブテーブル

サブテーブルは実際にはデータタイプではありません。サブテーブルのタイプになれるのはフィールドだけです。サブテーブルタイプの変数や式は存在しません。サブテーブルの詳細については、『4th Dimensionデザインリファレンス』マニュアルと後述の「サブレコードコマンド」の章を参照してください。

未定義

未定義は、実際にはデータタイプではありません。未定義は、まだ定義されていない変数を指しています。関数（結果を返すプロジェクトメソッド）は、メソッド内にある場合には未定義値を返すことができ、関数の結果（$0）は未定義式（少なくとも1つの未定義変数で計算された式）に代入されます。フィールドは、未定義にはできません。

データタイプの変換

4th Dimension言語には演算子とコマンドがあり、変換の実行が意味を持つ場合にデータタイプを変換できます。4th Dimension言語はデータタイプをチェックしています。例えば、"abc" + 0.5 + !97/08/21! - !!00:30:45!!のように表記することはできません。これは、シンタックス（構文）エラーになります。

次の表は、基本的なデータタイプ、変換できるデータタイプ、それを実行する際に使用するコマンドを示しています。

データタイプ	文字列変換	数値変換	日付変換	時間変換
文字列		**Num**	**Data**	**Time**
数値(*)	**String**			
日付	**String**			
時間	**String**			
ブール		**Num**		

(*) 時間値は数値として扱うことができます。

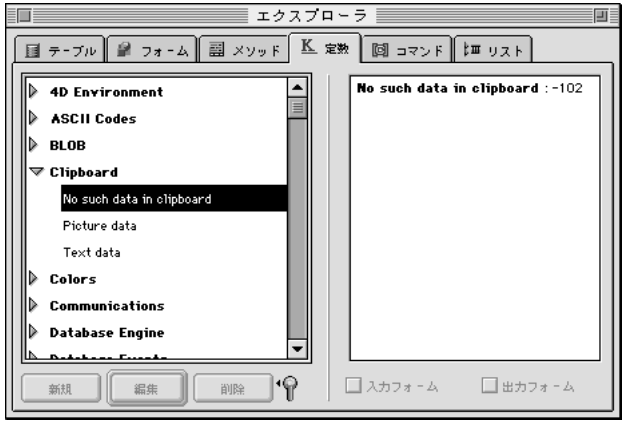
注：この表に示すデータ変換の他に、演算子と他のコマンドを組み合せることで、より洗練されたデータ変換を実行することができます。

定数

定数は、固定値を持つ式です。定数には、名前で選択できる「定義済定数」と実際の値を入力する「リテラル定数」の2種類があります。

定義済定数

4th Dimensionのバージョン6からあらかじめ定義されている「定義済定数」が導入されました。これらの定数は、「エクスプローラ」ウインドウに次のように一覧表示されています。



定義済定数は、テーマ別に一覧されています。「メソッドエディタ」ウインドウで定義済定数を使用するには、次のどちらかの手順で行います。

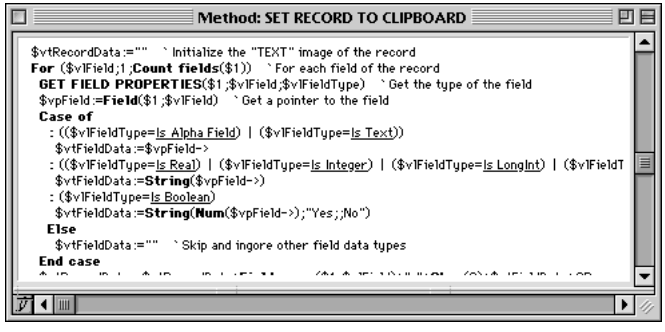
■「エクスプローラ」ウインドウから「メソッドエディタ」ウインドウに定数をドラッグ&ドロップします。

■「メソッドエディタ」ウインドウに定数の名前を直接入力します。

定義済定数名は、最大31文字まで指定できます。

Tips：定義済定数の名前を直接入力する場合に名前の先頭数文字をタイプし、タブキーを押下します。メソッドエディタの先行入力機能（タイプアヘッド）により、該当する定数がリストアップまたは確定され便利です。

「メソッドエディタ」ウインドウや「デバッグ」ウインドウにおいて、定義済定数はデフォルトとして下線付きで表示されます。



例えば、上記のウインドウでは「Is Alpha Field」が定義済定数です。

リテラル定数

リテラル定数は、次の4つのデータタイプにできます。

■ 文字列

■ 数値

■ 日付

■ 時間

文字列定数

文字列定数は、次のように2重引用符（"..."）で囲んで表します。文字列定数の例を次に示します。

```
"レコード追加"
"レコードが見つかりません"
"送り状"
```

空の文字列は、2つの引用符の間に何も入れない状態（""）で指定します。

数値定数

数値定数は、実数として記述します。下記に数値定数の例をいくつか次に示します。

```
27
123.76
0.0076
```

負の数値は、次のようにマイナス記号（-）を付けて指定します。

```
-27
-123.76
-0.0076
```

日付定数

日付定数は、エクスクラメーションマーク（! ... !）で囲んで表します。

日付は、“年.月.日”の順で表し、それぞれをWindows上ではスラッシュ（/）、Mac OS上ではピリオド（.）で区切ります。

下記に、日付定数の例を示します。

```
!76.1.1!
!04.4.4!
!97.07.25!
```

空の日付は、!00.00.00!のように指定します。

Tips：メソッドエディタでは空の日付を入力するためのショートカットが提供されています。空の日付を入力するには、エクスクラメーションマーク（!）の入力後にenterキーを押します。

注：2桁の年度は、2000年代であると認識します（このデフォルト設定がSET DEFAULT CENTURYコマンドで変更されていない場合に限る）。

時間定数

時間定数は、2つのエクスクラメーションマーク（!! ... !!）で囲んで表します。

時間は、“時:分:秒”の順で表し、それぞれをコロン（:）で区切ります。時間は、24時間制で指定します。

時間定数の例を次に示します。

!!00:00:00!!	` 午前0時
!!09:30:00!!	` 午前9時30分
!!13:01:59!!	` 午後1時1分59秒

空の時間は、!!00.00.00!!のように指定します。

Tips：メソッドエディタでは空の時間を入力するためのショートカットが提供されています。空の時間を入力するには、2つのエクスクラメーションマーク（!!）の入力後にenterキーを押します。

変数

4th Dimensionのデータは、フィールドまたは変数に格納されます。フィールドがディスクにデータを保存するのに対して、変数はメモリ上にだけデータを格納します。データベースを作成する際、フィールドに名前とデータタイプを指定するのと同様に、変数にも名前とデータタイプを指定します。

以下の変数タイプは、各データタイプと一致します。

■ **文字：**最大255バイトまでの固定文字列

■ **テキスト：**最大32000バイトまでの文字

■ **整数：**-32,768から32767までの整数

■ **倍長整数：**-2^31から(2^31)-1までの整数

■ **実数：**±1.7e±308の数字（15バイト）

■ **日付：**100/1/1から32767/12/31（Windows）、または100.1.1から32767.12.31（Macintosh）

■ **時間：**00:00:00から596000:00:00

■ **ブール：**TrueまたはFalse

■ **ピクチャ：**WindowsまたはMacintoshのピクチャ

■ **BLOB (Binary Large OBject)：**サイズが2GB以内のバイト群

■ **ポインタ：**ファイル、フィールド、変数、配列、配列要素のポインタ

変数は、画面上への表示やフォームからのデータ入力、レポートの印刷に使用することができます。また、フィールドと同様に、以下のような4th Dimension内部に用意された組み込み制御ツールを使用することができます。

■ 表示用フォーマット

■ データチェック（入力フィルタ、デフォルト値等）

■ 文字フィルタ

■ 選択項目リスト、指定項目リスト、除外項目リスト

■ 入力可属性または入力不可属性

また、変数は以下のような特殊なこともできます。

■ ボタン（ボタン、チェックボックス、ラジオボタン、3Dボタン等）の制御

■ スライダ（メータ、ルーラ、ダイアル）の制御

■ スクロール可能エリア、ポップアップメニュー、ドロップダウンリストボックスの制御

■ 階層リスト、階層ポップアップメニューの制御

■ ボタングリッド、タブコントロール、ピクチャボタン等の制御

■ 保存の必要がない計算結果の表示

変数の作成

変数は、作成するだけで使用することができます。フィールドのように正式に定義（宣言）する必要はありません。例えば、現在の日付に30日を加えたものを変数に保存したい場合は、以下のように定義します。

```
MyDate:=Current date +30
```

この式は、現在の日付に30日を加えたものを変数“MyDate”に代入することを意味します。また、以下のようにフィールド“MyField”に変数“MyDate”の値を代入することもできます。

```
MyField:=MyDate
```

また、変数はデータタイプを明確にするために使用される場合もあります。

注：このマニュアル上では、上記の例のCurrent date関数のようにコマンドや関数が太字（ボールド）で表記されています。これは、このマニュアルの表記規則に従っているためです。

変数にデータを代入する

変数にデータを格納、または変数からデータをコピーすることができます。変数に対してデータを代入（格納）するには、代入演算子（:=）を使用します。代入演算子はフィールドに対してデータを代入する場合にも使用します。

代入演算子は、変数を作成、または変数にデータを代入するために使用します。作成する変数名は代入演算子の左側に入力します。変数“MyNumber”を作成する例を次に示します。“MyNumber”に数値3を代入しています。

```
MyNumber:= 3
```

これで、数値3を値に持つ“MyNumber”という変数が作成されました。“MyNumber”が、既に存在している場合には、数値3は単純に“MyNumber”に代入されます。

もちろん、変数からデータを取り出すことができなければ、便利とは言えません。再度代入演算子を使用してみましょう。“Size”というフィールドに“MyNumber”の値を代入する場合に、代入演算子の右側に“MyNumber”を記述します。

```
Size:=MyNumber
```

この“Size”の値は3になります。この例はとても単純ですが、データをある場所から別の場所へ転送する基本的な方法です。

代入演算子（:=）を比較演算子（=）と混同しないように注意してください。代入演算子と比較演算子はまったく異なる演算子です。比較演算子に関する詳細は、第11章を参照してください。

ローカル変数、プロセス変数、インタープロセス変数

4th Dimensionでは、ユーザはローカル変数、プロセス変数、およびインタープロセス変数の3種類の変数を作成することができます。この3種類の変数の違いは、その使用できる「範囲」にあります。また、それらを使用することのできるオブジェクトによっても異なります。

ローカル変数

ローカル変数はその名のとおり、メソッド内でローカル（局所）に、つまり変数が作成されたメソッド内でのみアクセスすることができ、その他のメソッドからはアクセスできません。メソッド内でローカルであるというのは、「その範囲がローカルである」という意味です。ローカル変数は、その使用範囲をメソッド内に限定する場合に用います。

ローカル変数は、以下のような目的のために使用されます。

■ 他の変数名との重複を避ける。

■ データを一時的に使用する。

■ プロセス変数の数を減らす。

ローカル変数の名前は、必ずドル記号（$）で始め、31バイトまでの文字を指定できます。これより長い名前を指定すると、4th Dimensionにより余分の桁数は切り捨てられます。

多くのメソッドや変数を持つデータベースで作業する場合、現在作業しているメソッド内でのみ使用可能な変数を必要とする場合がよくあります。この場合、同じ変数名が他で使用していないかどうかを気にすることなくローカル変数を作成することができます。

また、ユーザからのデータを必要とする場合があります。**Request**関数は、ユーザから1つのデータを取得します。**Request**関数は、ユーザにデータの入力を求めるダイアログボックスを表示し、ユーザがデータを入力した時点で、その入力情報を返します。このデータはメソッドに保存する必要はありません。これは、ローカル変数を使用する良い例です。

例えば、以下のメソッドは、

```
$応答:=Request ("従業員番号を入力してください。")
If (OK=1)
QUERY ([従業員] ; [従業員]ID番号=$応答)
End if
```

ユーザに従業員のID番号を入力するように要求するものです。これはローカル変数“$応答”にID番号を代入し、それをもとにユーザが入力した従業員ID番号を検索します。このメソッドが終了した時点で、“$応答”というローカル変数は消去されます。この変数は1回しか必要としないためローカル変数で十分です。

ローカル変数は引数として使用した場合、メソッド間のデータの受け渡しにも使用されます。

プロセス変数

プロセス変数は、1つのプロセス内でだけ使用可能です。この変数は、そのプロセス内で呼び出された他のメソッドで使用することができます。

プロセス変数には、名前の前に付ける接頭辞はありません。プロセス変数名の長さは、最大31バイトまで指定できます。

インタープロセス変数

インタープロセス変数は、データベース全体で使用することができ、すべてのプロセスで共用します。

インタープロセス変数の名前は、必ずインタープロセス記号（<>）で始めます。記号の後に31バイトまでの名前を指定できます。例えば、変数「<>Color」はインタープロセス変数です。

クライアント／サーバでは、各マシン間（クライアントマシンとサーバマシン）で同じインタープロセス変数定義を共有しますが、マシンごとに各変数に対する異なる実体が存在します。

フォーム上のオブジェクトと変数

「フォーム」エディタを使って、ボタン、スクロールエリア、サーモメータ等のオブジェクトを作成すると、それに付けた名前と同じ名前を持つ変数を自動的に生成します。例えば、“MyButton”という名前のボタンを作成すると、“MyButton”という名前の変数が自動的に生成されます。しかし、この変数名がこのボタンのラベル名ではなく、ボタンの名前になっている点に注意してください。

この変数を使用してオブジェクトの制御、チェックを実行することができます。例えば、このボタンがクリックされた時点で、その変数に1が設定されます。それ以外の場合はすべて0になります。サーモメータやダイアルに付随する変数は、現在の設定値を読み込んで、その設定値を変更することができます。例えば、サーモメータをドラッグした場合に、変数の値は新しい設定値に変更されます。同様にメソッドで変数の値を変更すると、サーモメータは新しい値を再表示します。

システム変数

4th Dimensionには、システム変数という特別な変数があります。この種類の変数を使用するとさまざまな操作を監視することができます。すべてのシステム変数がプロセス変数です。

最も重要なシステム変数は、OKシステム変数です。名前が示すように、何かの事象がOKかどうかを判断します。例えば、レコードは保存されたか、データ読み込み処理は終了したか、ユーザがOKボタンをクリックしたか等を判断します。システム変数OKは、処理が正常に完了した場合には1が設定され、処理がうまく終了しない場合には0が設定されます。

システム変数に関する詳細は、このマニュアルの付録Aを参照してください。

システム変数

4th Dimensionのシステム変数を使用して、異なる操作の実行を制御することができます。システム変数はすべてプロセス変数であり、ひとつのプロセス内でのみアクセスできます。

OK

システム変数OKは、システム変数の中で最も頻繁に使用されます。一般的には、処理が成功すると1が代入され、成功しなかったときに0が代入されます。大半の4Dコマンドは、システム変数OKの値を変更します。各コマンドの説明を参照し、コマンドがこのシステム変数に影響を与えるかどうかを調べてください。

Document

システム変数Documentには、以下のコマンドで表示されたり、作成された最後のMacintoshのディスクファイルの名前が格納されます。

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

FldDelimit

システム変数FldDelimitは、テキストを読み込んだり書き出す際に、フィールドの区切り文字として使用する文字のASCIIコードが格納されています。デフォルトの値は、ASCIIコードでタブに相当する9です。区切り文字を変更する場合は、この値を変更してください。

RecDelimit

システム変数RecDelimitは、テキストを読み込んだり書き出す際に、レコードの区切り文字として使用する文字のASCIIコードが格納されています。デフォルトの値はASCIIコードでキャリッジリターン（CR）に相当する13です。区切り文字を変更する場合は、この値を変更してください。

Error

システム変数Errorは、**ON ERR CALL**コマンドによってインストールされたエラー処理メソッドの中でのみ有効です。この変数にはエラーコードが格納されています。付録Cに4th DimensionとWindows（Macintosh）のエラーコードのリストを示します。

MouseDown、MouseX、MouseY、KeyCode、Modifiers、MouseProc

これらのシステム変数は、**ON EVENT CALL**コマンドによってインストールされたイベントメソッドの中でのみ有効です。

- システム変数MouseDownは、マウスのボタンが押されたときに1が、それ以外の場合は0に代入されます。
- イベントがMouseDownの場合（MouseDown=1）は、システム変数MouseXとMouseYにはマウスがクリックされた場所の水平と垂直が代入されます。両方の値ともピクセル単位で表わされ、ウインドウのローカルな座標システムを使用します。
- システム変数KeyCodeは、押されたキーのASCIIコードが代入されます。押されたキーがファンクションキーの場合、には特殊コードがセットされます。ファンクションキーの一覧は、付録Cに記載されています。
- システム変数Modifiersは、キーボードのモディファイアキーの値を含んでいます（Ctrl/command、Alt/option、shift、CapsLock）。システム変数Modifiersは、**ON EVENT CALL**コマンドによってインストールされた“割り込み処理”においてのみ意味を持ちます。
- システム変数MouseProcは、イベントが発生したプロセス番号を含みます。

ポインタ

ポインタの使用は、データを参照するための上級技法です。第1章から第12章までで示した概念を完全に理解した上でポインタを使用してください。

プログラミング言語を使用する場合、名前を用いてテーブル、フィールド、変数、配列等をアクセスします。通常は、名前によるデータの参照が最も簡単な方法ですが、名前を使用しないでデータを参照する、またはアクセスした方が便利な場合もあります。この場合、ポインタを使用すると実現できます。

ポインタの背景にある概念は、日常生活でもよく用いられています。対象物を正確に知らないで何かを示すことがあります。例えば、友人に対して“登録番号123ABDの車に乗ろう”と言わないで“君の車に乗ろう”という場合があります。つまり、“登録番号123ABDの車”を“君の車”で示したわけです。この場合、“登録番号123ABDの車”はオブジェクトの名前で、“君の車”はオブジェクトを参照するためのポインタと考えることができます。

あるものの対象物を明示しないで参照することができると非常に便利です。例えば友人が新しい車に買い替えても、同じく“君の車”と言うことができます。ポインタも同じように機能します。例えば、ある場合は数値フィールド“Age”を参照したポインタで、別の場合には数値変数“Old Age”を参照することもできます。この場合のポインタは、計算に使用する数値データを参照しています。

テーブル、フィールド、変数、配列、配列要素を参照するためにポインタを使用することができます。以下の表に、各タイプの例を示します。

データタイプ	参照	引数として使用	代入
テーブル	**My Table**:=->[ﾃｰﾌﾞﾙ]	**DEFAULT TABLE (My Table->)**	不使用
フィールド	**My Field**:=->[ﾃｰﾌﾞﾙ]ﾌｨｰﾙﾄﾞ	**ALERT(My Field->)**	**My Field->**:= "ジョン"
変数	**My Var**:=->変数	**ALERT(My Var->)**	**My Var->**:= "ジョン"
配列	**My Arr**:=->配列	**COPY ARRAY(My Arr->** ; B)	**COPY ARRAY**(B;**My Arr->**)
配列要素	**My Erem**:=->配列{1}	**ALERT(My Erem->)**	**My Erem->**:= "ジョン"

ポインタの使用例

例題を用いてポインタの使用方法について説明します。以下の例は、ポインタを通して変数にアクセスする方法を示しています。まず、変数の作成から始めます。

```
MyVar:="こんにちは"
```

“MyVar”は、文字列“こんにちは”を含む変数です。“MyVar”に対するポインタを作成します。

```
MyPointer:=->MyVar
```

ポインタ記号（->）は、“･･･に対するポインタを求める”ことを意味します。ここでは、“MyVar”を参照するポインタを呼び出します。このポインタは、代入演算子（:=）で“MyPointer”に対して割り当てられます。

“MyPointer”は、“MyVar”に対するポインタを含む変数です。“MyPointer”は、“こんにちは”という“MyVar”の値を含みませんが、“MyVar”に含まれる値を指すことはできます。以下の式は“MyVar”の値を返します。

```
My Pointer->
```

前述の式は、“こんにちは”という文字列を返します。ポインタ記号（->）をポインタの後につけると、そのポインタの指すオブジェクトの値を示します。

オブジェクトの名前を直接使用して参照する代わりに、ポインタ記号（->）を後につけたポインタを使用してオブジェクトを参照することの意味を理解することが重要です。つまり、変数“MyVar”を使用することと、式“MyPointer->”を使用することは、全く同じ意味です。

例えば、以下のステートメントは警告ボックスに文字列“こんにちは”を表示します。

```
ALERT(My Pointer->)
```

“MyPointer”を使用して“MyVar”の値を変更することもできます。下記のステートメントは、変数“MyVar”に文字列"さようなら"を代入します。

```
MyPointer->:="さようなら"
```

この2つの“MyPointer->”を使用したステートメントのとおり、“MyVar”を使用した場合と全く同じ動作を実行します。以下の2つのステートメントも、全く同じ動作を実行します。両方とも、変数“MyVar”の現在の値を警告ボックスに表示します。

```
ALERT(MyPointer->)
ALERT(MyVar)
```

以下の2つのステートメントも、全く同じ動作を実行します。両方とも“MyVar”に、文字列“さようなら”を代入します。

```
MyPointer ->:="さようなら"
```

```
MyVar:="さようなら"
```

ボタンに対するポインタの使用

この節では、ボタンを参照するためのポインタの使用方法について説明します。ボタンは（プログラミング言語の観点から）、変数に属します。この節では、ボタンを参照するためのポインタの例を使用していますが、ここで示す概念は、すべての種類のポインタに適用されます。

フォーム上に、それぞれの条件によって、使用可または使用不可にする必要のあるボタンが数多くあるとします。ボタンを使用可にする、または使用不可にする必要があるたびに、以下のような判定を実行します。

```
If(条件)          ` 条件が“True”の場合...
      ENABLE BUTTON(My Button)     ` ボタンを使用可にする
Else  ` それ以外の場合...
      DISABLE BUTTON(My Button)    ` ボタンを使用不可にする
End if
```

フォーム中のその他のボタンに対しても、同様の判定を実行する必要があります。さらに効率良く判定処理を実行するためには、各ボタンの参照にポインタを用いて判定を実行するサブルーチンを採用します。

サブルーチンを使用する場合は、ポインタを使用しなければなりません。なぜなら、これ以外の方法では、ボタンの変数を参照することができないからです。例えば、以下のサブルーチン“Set Button”はポインタを使用してボタンを参照しています。

```
      `$1ーブール：“True”は、ボタンを使用可に、“False”は、ボタンを
`使用不可にする
      `$2ーボタンのポインタ
If($1)  ` 条件が“True”の場合...
      ENABLE BUTTON($2->)    ` ボタンを使用可にする
Else  ` それ以外の場合...
      DISABLE BUTTON($2->)   ` ボタンを使用不可にする
End if
```

サブルーチン“Set Button”は以下のように呼び出されます。

```
Set Button(判定1 ; ->ボタン1)
Set Button(判定2 ; ->ボタン2)
```

テーブルに対するポインタの使用

プログラミング言語でテーブルにアクセスする場合には、ポインタを使用してテーブルを参照することができます。テーブルへのポインタは、主にテーブルを操作するコマンドの最初の引数として指定されます。

以下のようなステートメントで、テーブルのポインタを作成します。

```
MyTable p:=->[MyTable 1]
```

あるいは、以下のように**Table**コマンドを使用してテーブルのポインタを呼び出すこともできます。

```
MyTable p:=Table(1)
```

以下のようにコマンドに対して、参照するテーブルを指定することもできます。

```
DEFAULT Table(MyTable p->)
```

フィールドに対するポインタの使用

プログラミング言語でフィールドをアクセスする場合には、ポインタを使用して、フィールドを参照することができます。

以下のようなステートメントで、フィールドのポインタを作成します。

```
MyField p:=->[MyTable1] My Field2
```

あるいは、以下のように**Field**コマンドを使用してフィールドのポインタを呼び出すこともできます。

```
MyField p:=Field(1 ; 2)
```

以下のようにコマンドに対して、参照するフィールドを指定することもできます。

```
FONT(MyField p-> ; "Osaka")
```

ポインタ変数を使用する

この例では、ポインタ変数の使い方について説明します。

```
MyVar:="Hello"
MyPointer:=->MyVar
```

ポインタは、インタープロセス変数、プロセス変数の他、バージョン2004.1からはローカル変数にも利用できるようになりました。

プロセス変数もしくはローカル変数にポインタを使う場合、参照される変数はポインタが使用される時点で既に定義されていなければなりません。ローカル変数は、それらを作成したメソッドの実行が終わると破棄され、プロセス変数もそれを作成したプロセスの終了時に削除される点に留意してください。存在しない変数をポインタが呼び出そうとすると、インタープリタモードでは（「変数が設定されていません」という内容の）シンタックスエラーが起きます。コンパイルモードでは、さらに重大なエラーが発生する可能性があります。

ローカル変数に関する注意：ローカル変数のポインタを使用すると、プロセス変数の使用を控えることができます。ポインタは、同じプロセス内のローカル変数にのみ使用することができます。ポインタの参照先が、別のメソッドで宣言されたローカル変数の場合、これをデバッガに表示すると、オリジナルのメソッド名が、ポインタの後の括弧内に表示されます。

```
$MyVar:="Hello world"
Method2(->$MyVar)
```

Method2では、デバッガは$1を次のように表示します。

```
$1      ->$MyVar（Method1）
```

$1の値は、次のようになります。

```
$MyVar（Method1）          "Hello world"
```

配列要素に対するポインタの使用

配列要素に対するポインタを作成することができます。下記のステートメントは配列を作成し、変数“要素p”に最初の配列要素を指し示すポインタを割り当てます。

```
ARRAY REAL(配列 ; 10)    ` 配列の作成
要素p:=->配列{1}    ` 配列要素に対するポインタの作成
```

以下のように、ポインタを使用して要素に値を代入することができます。

```
要素p->:=1
```

配列に対するポインタの使用

配列に対するポインタを作成することができます。以下のステートメントは配列を作成し、変数“配列p”に最初の配列を指し示すポインタを割り当てます。

```
ARRAY REAL(配列 ; 10)    ` 配列の作成
配列p:=->配列          ` 配列へのポインタの作成
```

これは、ポインタが配列の要素を指しているのではなく配列を指しています。ポインタを、以下のように使用することもできます。

```
SORT ARRAY(配列p-> ; >) ` 配列をソートする
```

例えば、ポインタを使用して配列の4番目の要素を参照する場合は以下のように記述します。

```
配列p->{4}:=84
```

この方法はポインタ配列を使用することとは異なります。ポインタ配列については次の節を参照してください。

ポインタ配列の使用

関連するオブジェクトのグループを参照する場合にポインタ配列を持つと便利な場合があります。

そのようなオブジェクトのグループの例として、フォーム上の変数のグリッドがあります。グリッドのそれぞれの変数には、“変1”、“変2”、 . . . 、“変10”という連番が付いています。間接的にこれらの変数を数字で参照することができます。ポインタの配列を作成し、各変数を指すためにポインタを初期化すれば、変数を簡単に参照することができます。例えば、配列を作成し各要素を初期化するために、以下のようなステートメントを使用します。

```
ARRAY POINTER(変 ; 10) `10個のポインタを保存するため配列を作成する
For($i ; 1 ; 10)          ` 各変数に対して1回ずつループする
    変{$i}:=Get pointer("変" +String ($i))          ` 配列要素を初期化する
End for
```

Get pointer関数は、指定されたオブジェクトへのポインタを返します。

ポインタに対応する変数を参照するためには、配列要素を使用します。例えば、変数に10個の日付を代入する場合に（変数が日付タイプであると想定して）、以下のようなステートメントを使用します。

```
For($i ; 1 ; 10)          ` 各変数に対して1回ずつ処理を行い10回ループする
    変{$i}->:=Current date + $i ` 日付を代入する
End for
```

ポインタを使用したボタンの設定

フォーム中に関連する一連のラジオボタンがある場合に、それらを素早く設定しなければならないことがあります。名前を使用して各ラジオボタンを直接参照することは、効率が良くありません。以下の図に、ボタン1、ボタン2、. . . 、ボタン5という名前の5つのラジオボタンを示します。

○ ボタン1
○ ボタン2
○ ボタン3
○ ボタン4
○ ボタン5

一連のラジオボタンでは、1つのラジオボタンのみがオンになります。オンになっているラジオボタンの番号は、数値フィールドに記憶されます。例えば、“設定”フィールドの値が3の場合は、ボタン3に1を設定します。フォームメソッドで、以下のようなステートメントを使用して、ボタンをセットします。

```
If(Before)
    Case of        `“設定”フィールドを判定する
    ¥([初期設定]設定=1) `“設定”フィールドが“True”の場合...
        ボタン1:=1    `ラジオボタンをオンにする
    ¥([初期設定]設定=2)
        ボタン2:=1
    ¥([初期設定]設定=3)
        ボタン3:=1
    ¥([初期設定]設定=4)
        ボタン4:=1
    ¥([初期設定]設定=5)
        ボタン5:=1
    End case
End if
```

各ラジオボタンごとに判定しなければなりません。フォーム内に多くのラジオボタンがあると、非常に長いメソッドになる場合もあります。この問題を解決するために、ポインタを使用します。

Get pointer関数を使用して、ラジオボタン（あるいは任意のボタン）のポインタを作成することができます。値を設定する必要のあるラジオボタンを参照するためにポインタを使用します。次に改善されたメソッドの例を示します。

```
Case of
        ¥ (Form event=On Load)
        ` ...
        $vpRadio:=Get pointer("ボタン"+String ([初期設定]設定))
        $vpRadio->:=1
        ` ...
End case
```

ラジオボタンの番号が、“設定”フィールドに記憶される必要があります。これは、フォームメソッドで**On Clicked**イベントに対し以下の行を加えると実現できます。

```
[初期設定]設定:=ボタン1+(ボタン2*2)+(ボタン3*3)+(ボタン4*4)+(ボタン5*5)
```

メソッドに対するポインタの受け渡し

ポインタを引数としてメソッドに渡すことができます。メソッド内で、ポインタによって参照されるオブジェクトを修正することができます。

例えば、以下のメソッド“Take Two”は、引数に2つのポインタ持ちます。最初の引数は、大文字に変換するオブジェクトを参照します。2番目の引数は小文字に変換するオブジェクトを参照します。次に、そのプメソッドを示します。

```
        ` $1ー文字列のポインタで、大文字に変更する
        ` $2ー文字列のポインタで、小文字に変更する
$1->:=Uppercase ($1->)
$2->:=Lowercase ($2->)
```

以下のステートメントは、フィールドを大文字に、変数を小文字に変更するためにメソッド“Take Two”を使用します。

```
Take Two(->[MyTable] MyField ; ->MyVar)
```

このフィールド“[MyTable] MyField”が、“jones”であれば、“JONES”に変更されます。一方、変数“MyVar”が“HELLO”であれば、“hello”に変更されます。

メソッド“Take Two”で使用するポインタと、参照されるオブジェクトのデータタイプが一致していることが重要です。この例では、ポインタに対して必ず文字列のオブジェクトを割り当てなければなりません。

ポインタに対するポインタ

ポインタを使用して他のポインタを参照することができます。以下の例を見てみましょう。

```
MyVar:="こんにちは"
MyPointer 1:=->MyVar
MyPointer 2:=->MyPointer 1
(MyPointer 2->)->:="さようなら"
ALERT((MyPointer 2->)->)
```

このステートメントは、警告ボックスに文字列“さようなら”を表示します。この例は、ポインタの使用に内在する複雑性を示しています。つまり、ポインタがどこの何を指すのかを知る必要があります。

各行について説明していきます。

■ MyVar:="こんにちは"

これは、単に文字列“こんにちは”を変数“MyVar”に代入します。

■ MyPointer 1:=->MyVar

“MyPointer 1”を“MyVar”のポインタにします。

■ MyPointer 2:=->ポイント1

“MyPointer 2”は“MyPointer 1”のポインタで、“MyVar”を指し示します。

■ (MyPointer 2->)->:="さようなら"

この場合の“MyPointer 2->”は、“MyPointer 1”の値を参照し、“MyVar”の位置を指します。“(MyPointer 2->)->”は、“MyPointer 1”の値の指す位置を参照します。つまり、“(MyPointer 2 ->)->”は、“MyVar”の値を参照することになります。したがって、“MyVar”に文字列“さようなら”を代入します。

■ **ALERT**((MyPointer 2->)->)

上記のステートメントは、“MyVar”の値をアクセスします。“(MyPointer 2->)->”は、“MyVar”の値を呼び出します。このステートメントでは、“MyVar”の位置を指すものとシンタックスが異なる点に注目してください。

以下のステートメントは、文字列“こんにちは”を“MyVar”に代入します。

```
(MyPointer 2->)->:="こんにちは"
```

以下のステートメントは、“MyVar”から文字列“こんにちは”を取り出し、“新変数”にそれを代入します。

```
新変数:=(MyPointer 2->)->
```

重要：複合参照を実行する場合は、カッコを使用してください。

識別子

本節は、4th Dimension言語のさまざまなオブジェクトを命名するための規則を説明します。すべてのオブジェクトの名前は、次の規則に従います。

■ 名前は、文字（全角文字または半角アルファベット）で始めます。

■ 名前には、文字、数字、スペース、アンダースコアが使用できます。

■ ピリオド（.）、スラッシュ（/）、コロン（:）は使用することはできません。

■ +、-、*、/、(、) 等の演算子に用いられる文字は使用できません。

■ 名前の最後につけたスペースは無視されます。

テーブル

角カッコ内に名前を入れると、テーブルを表します。テーブル名は、31バイト以内で指定します。

テーブル名	ステートメント内のテーブル名
[注文]	**DEFAULT TABLE** ([注文])
[顧客]	**INPUT FORM** ([顧客] ; "入力")
[レター]	**ADD RECORD** ([レター])

フィールド

フィールドが属するテーブルを最初に指定して、フィールドを表します。フィールドの名前はテーブル名のすぐ後に続けます。フィールド名は、31バイト以内で指定します。

フィールド名の先頭文字にアンダースコア（_）を使用してはいけません。このアンダースコア文字は4Dモジュール用の予約語として使用されます。4th Dimensionは「メソッド」エディタ内のフィールドの先頭でこの文字を見つけると、アンダースコアを取り除きます。

フィールド名	ステートメント内のフィールド名
[注文]総合計	[注文]総合計:=**Sum** ([明細]合計)
[顧客]名前	**QUERY** ([顧客]; [顧客]名前="山田")
[レター]テキスト	*Capitalize* ([レター]テキスト)

トリガ、フォームメソッド、オブジェクトメソッド内で指定する必要がない場合でも、フィールドの前にテーブル名を指定することは、良いプログラミング技法です。

サブテーブル

サブテーブルは、サブテーブルの属するテーブルを最初に指定します。このテーブルは、そのサブテーブルの親テーブルです。サブテーブルの名前は、テーブル名のすぐ後に続けます。サブテーブル名は、31バイト以内で指定します。

サブテーブル名	ステートメント内のサブテーブル名
[従業員]子供	**ALL SUBRECORDS** ([従業員]子供)
[顧客]電話番号	**ADD SUBRECORD** ([顧客]電話番号 ; "追加")
[レター]キーワード	**NEXT SUBRECORD** ([レター]キーワード)

サブテーブルは、フィールドタイプとして扱われます。したがって、フォームで使用する場合は、フィールドと同じ規則に従います。トリガ、フォームメソッド、親テーブルのオブジェクトメソッドでサブテーブルを指定する場合に、親テーブル名を指定する必要はありません。しかし、サブテーブル名の前にテーブル名を指定することは良いプログラミング技法です。

サブフィールド

フィールドと同じようにサブフィールドを表します。サブフィールドは、最初にサブフィールドが属するサブテーブルを指定して表します。サブフィールドの名前は、アポストロフィ（'）でサブテーブルの名前と区切ります。サブフィールド名は、31バイト以内で指定します。

サブフィールド名	ステートメント内のサブフィールド名
[従業員]子供'名前	[従業員]子供'名前:=**Uppercase** ([従業員]子供'名前)
[顧客]電話番号'番号	[顧客]電話番号'番号:="03-1234-5678"
[レター]キーワード'ワード	*Capitalize* ([レター]キーワード'ワード)

サブテーブルメソッド、フォームメソッド、サブテーブルのオブジェクトメソッドでサブフィールドを指定する場合に、サブテーブル名を指定する必要はありません。しかし、サブフィールド名の前にテーブル名やサブテーブル名を指定しておくことは、良いプログラミング技法です。

インタープロセス変数

名前の先頭にインタープロセス（<>）記号を付けることによって、インタープロセス変数を表します。

インタープロセス変数名は、インタープロセス（<>）記号を除いて31バイト以内で指定します。

インタープロセス変数名	ステートメント内のインタープロセス変数名
<>プロセスID	<>プロセスID:=**Current process**
<>キー	<>キー:=**Char**(Keycode)
<>カラー	**If** (<>カラー # 0)

プロセス変数

名前を使用して、プロセス変数を表します。プロセス変数名は、31バイト以内で指定します。

プロセス変数名	ステートメント内のプロセス変数名
総合計	総合計:=**Sum** ([会計]金額)
ボタン1	**If** (ボタン1 =1)
MyVar	MyVar:="定数文字列"

ローカル変数

ドル記号（$）を名前の先頭につけてローカル変数を表します。ローカル変数名は、ドル（$）記号を除いて31バイト以内で指定します。

ローカル変数名	ステートメント内のローカル変数名
$i	**For** ($i ; 1 ; 100)
$TempVar	**If** ($TempVar="No")
$My String	$My String:="こんにちは"

配列

名前を使用して、配列を表します。これは配列作成時に配列宣言コマンド（**ARRAY LONGINT**等）に渡す名前です。配列には次の3種類があります。

■ インタープロセス配列

■ プロセス配列

■ ローカル配列

インタープロセス配列

インタープロセス配列の名前は、名前の先頭にインタープロセス（<>）記号が付きます。インタープロセス配列名は、インタープロセス（<>）記号を除いて31バイト以内で指定します。

インタープロセス配列名	ステートメント内のインタープロセス配列名
<>ID配列	**ARRAY TEXT** (<>ID配列 ; **Records in table**([ID]))
<>キーワード	**SORT ARRAY** (<>キーワード ; >)
<>大きい配列	**ARRAY INTEGER** (<>大きい配列 ; 10000)

プロセス配列

名前を使用して、プロセス配列を表わします（<>記号も$記号も名前の先頭につきません）プロセス配列名は、31バイト以内で指定します。

ローカル配列名	ステートメント内のローカル配列名
$アイテム	**ARRAY TEXT** ($アイテム ; **Records in table**([ID]))
$キーワード	**SORT ARRAY** ($キーワード ; >)
$ファイル	**COPY ARRAY** ($ファイル ; スクロール)

ローカル配列

配列名がドル記号（$）で始まるものは、ローカル配列です。ローカル配列名は、ドル（$）記号を除いて31バイト以内で指定します。

ローカル配列名	ステートメント内のローカル配列名
$アイテム	**ARRAY TEXT** ($アイテム ; **Records in table**([ID]))
$キーワード	**SORT ARRAY** ($キーワード ; >)
$ファイル	**COPY ARRAY** ($ファイル ; スクロール)

配列の要素

中カッコ（{}）を使用して、インタープロセス配列、プロセス配列、ローカル配列の要素を参照します。参照される配列要素は数式で表されます。次に例を示します。

インタープロセス配列

インタープロセス配列要素	ステートメント内のインタープロセス配列要素
<>ID配列{$i}	<>ID配列{$i}{$j}:=[従業員]ID番号
<>キーワード{3}	**If** (<>キーワード{3}="終わり")
<>大きい配列{$ID}	私のファイル:=<>大きい配列{$ID}

プロセス配列

プロセス配列要素	ステートメント内のプロセス配列要素
アイテム{$i}	アイテム{$i}:=[従業員]名前
キーワード{3}	**If** (キーワード{3}="終わり")
ファイル{$ID}	私のフィールド:=ファイル{$ID}

ローカル配列

ローカル配列要素	ステートメント内のローカル配列要素
$アイテム{$i}	$アイテム{$i}:=[従業員]名前
$キーワード{3}	**If** ($キーワード{3}="終わり")
$ファイル{$名前}	私のフィールド:=$ファイル{$名前}

二次元配列の要素

中カッコ（{}）を2回使用して、2次元配列の要素を参照します。参照される要素は2組の中カッコ内の2つの数式で表されます。

インタープロセス配列

インタープロセス配列要素	ステートメント内のインタープロセス配列要素
<>ID配列{$i}{$j}	<>ID配列{$i}{$j}:=[従業員]名前
<>キーワード{3}{2}	**If** (<>キーワード{3}{2}="終わり")
<>ファイル{$名前1}{$名前2}	MyField:=<>ファイル{$名前1}{$名前2}

プロセス配列

プロセス配列要素	ステートメント内のプロセス配列要素
ID配列{$i}{$j}	ID配列{$i}{$j}:=[従業員]名前
キーワード{3}{2}	**If** (キーワード{3}{2}="終わり")
ファイル{$名前1}{$名前2}	MyField:=ファイル{$名前1}{$名前2}

ローカル配列

ローカル配列要素	ステートメント内のローカル配列要素
$アイテム{$i}{$j}	$アイテム{$i}{$j}:=[従業員]名前
$キーワード{3}{2}	**If** ($キーワード{3}{2}="終わり")
$ファイル{$名前1}{$名前2}	MyField:=$ファイル{$名前1}{$名前2}

フォーム

フォームの名前は、文字列式を使用して表します。フォーム名は、31バイト以内で指定します。

フォーム名	ステートメント内のフォーム名
"入力"	**INPUT FORM**([従業員];"入力")
"出力"	**OUTPUT FORM**([従業員];"出力")
"名前"+String($i)	**DIALOG** ([ステージ];"名前"+String($i))

メソッド

名前を使用して、メソッド（プロシージャおよび関数）を表します。メソッド名は31バイト以内で指定します。

注：結果を返さないメソッドはプロシージャとも呼ばれます。結果を返すメソッドを関数と呼びます。

メソッド名	ステートメント内のメソッド名
新規顧客	**If** (新規顧客)
重複データ削除	重複データ削除
大文字変換	**APPLY TO SELECTION** (大文字変換)

Tips：4Dの組み込みコマンドと同じ命名規約を利用することは良いプログラミングテクニックです。メソッド名には大文字を使用しますが、メソッドが関数の場合、メソッド名の最初の文字だけを大文字にします。このように命名することにより、数ヶ月後に保守のためデータベースを再度開いたときに、「エクスプローラ」ウインドウでその名前を見ただけで、メソッドが結果を返すかどうかがわかります。

注：メソッドの呼び出しを実行するには、メソッド名を入力するだけです。しかしON EVENT CALL等4Dの組み込みコマンドの一部やプラグインコマンドは、メソッドの引数が渡される際に文字列としてのメソッド名を必要とします。次に例を示します。

```
` このコマンドにはメソッド(関数)またはフォーミュラが必要です
QUERY BY FORMULA ([テーブル1];Special query)
` このコマンドにはメソッド（プロシージャ）または文が必要です
APPLY TO SELECTION ([従業員];INCREASE SALARIES)
` しかし、このコマンドにはメソッド名が必要です
ON EVENT CALL ("HANDLE EVENTS")
` このプラグインコマンドにはメソッド名が必要です
CT ON ERROR ("CT HANDLE ERRORS")
```

メソッドに引数を渡すことができます。引数は、メソッドまたは関数の後のカッコ内に記述します。各引数は、セミコロン（;）で次のものと区切ります。引数は、呼び出されたメソッド内で、連番付きのローカル変数$1、$2、. . .、$nとして使用できます。さらに、複数の連続する引数は、${n}というシンタックスを用いて使用できます。nは数値で引数の番号を示します。

関数内のローカル変数“$0”に、返される値が代入されます。

例題

```
` DROP SPACES内で $1 はフィールド [従業員]名前へのポインタです
DROP SPACES(->[People]Name)

` Calc creator内で
` - $1 は数値であり、その値は1です
` - $2 は数値であり、その値は5です
` - $3 はテキストまたは文字列であり、その値は“Nice”です
` - 結果の値は$0に代入されます
$v結果:=Calc creator(1; 5; "Nice")

` Dump内において
` - 3つの引数はテキストまたは文字列です
` - これらの引数は$1,$2または$3を使って使用できます
` - 例えば次のように指定することもできます。${$vlParam} $vlParamの値は
                                                    1,2または3です
` - 結果の値は$0に代入されます
vtClone:=Dump("is"; "the"; "it")
```

プラグインコマンド（プラグイン、関数、エリア）

プラグインで定義された名前を使用して、プラグインルーチンを表します。プラグインルーチン名は、31バイト以内で指定します。

プラグイン名	ステートメント内のプラグイン名
SP New offscreen area	新規エリア:=***SP New offscreen area***
DR PUBLISH	***DR PUBLISH***(エリア ; 0 ; $Temp)

セット

セットには、2種類あります。

■ インタープロセスセット

■ プロセスセット

4D Serverには次のセットもあります。

■ クライアントセット

インタープロセスセット

インタープロセスセットの名前は、名前の先頭にインタープロセス（<>）記号が付きます。インタプロセスセット名は、インタープロセス（<>）記号を除いて80バイト以内で指定します。

プロセスセット

セットの名前を表す文字列式を使用してプロセスセットを表します（<>記号も$記号も名前の先頭につきません）。インタプロセスセット名は、80バイト以内で指定します。

クライアントセット

クライアントセット名は、名前の先頭にドル（$）記号を指定します。クライアントセット名は、ドル記号を除いて31バイト以内で指定します。

注：バージョン6より前の4D Client／Serverでは、セットはセットが作成されたクライアントマシン上で保守されていました。バージョン6からは、セットはサーバマシン上で保守されるようになりました。効率や特殊目的のために、クライアントマシン内でローカルにセットで作業をする必要がある場合もあります。これを実行するときにはクライアントセットを使用します。

インタープロセスセット

インタープロセスセット名	ステートメント内のインタープロセスセット名
"<>レコード削除"	**USE SET** ("<>レコード削除")
"<>顧客注文"	**CREATE SET** ([顧客] ; "<>顧客注文")
"<>選択" + **String** ($i)	**Records in set** ("<>選択" + **String** ($i))

プロセスセット

プロセスセット名	ステートメント内のプロセスセット名
"レコード削除"	**USE SET** ("レコード削除")
"顧客注文"	**CREATE SET** ("顧客注文")
"選択" + **String** ($i)	**Records in set** ("選択" + **String** ($i))

クライアントセット

クライアントセット名	ステートメント内のクライアントセット名
"$レコード削除"	**USE SET** ("$レコード削除")
"$顧客注文"	**CREATE SET** ("$顧客注文")
"$選択"+**String**($i)	**Records in set** ("$選択" + **String** ($i))

命名セレクション

文字列式を使用して、命名セレクションを表します。命名セレクション名は、31バイト以内で指定します。

命名セレクションには、2種類あります。

■ インタープロセス命名セレクション

■ プロセス命名セレクション

インタープロセス命名セレクション

インタープロセス命名セレクションの名前は、名前の先頭にインタープロセス（<>）記号が付きます。インタプロセス命名セレクション名は、インタープロセス（<>）記号を除いて31バイト以内で指定します。

プロセス命名セレクション

プロセス命名セレクションの名前を表す文字列式を使用してプロセスセットを表します(<>記号も$記号も名前の先頭につきません)。インタプロセスセット名は、80バイト以内で指定します。

インタープロセス命名セレクション

インタープロセス命名 セレクション名	ステートメント内のインタープロセス命名 セレクション名
"<>郵便番号"	**USE NAMED SELECTION** ([顧客] ; "<>郵便番号")
"<>顧客注文"	"<>顧客注文"
"<>選択" + **String** ($i)	"<>選択" + **String** ($i))

プロセス命名セレクション

プロセス命名セレクション	ステートメント内の命名セレクション
郵便番号	**USE NAMED SELECTION** ([顧客] ; "郵便番号")
顧客注文	顧客注文
選択 + **String** ($i)	選択 + **String** ($i)

プロセス

シングルユーザ版およびクライアント／サーバ版のクライアント側において、2種類のプロセスがあります。

■ グローバルプロセス

■ ローカルプロセス

グローバルプロセス

プロセスの名前を表す文字列式を使用してグローバルプロセスを表します。グローバルプロセス名は、31バイト以内で指定します。

ローカルプロセス

ドル記号（$）を名前の前につけてローカルプロセスを表します。ローカルプロセス名は、ドル（$）記号を除いて31バイト以内で指定します。

グローバルプロセス

グローバルプロセス名	ステートメント内の外部グローバルプロセス名
"イベントプロセス"	MyProc:=**New Process** ("EventProc" ; 32*1024 ; "イベントプロセス")

ローカルプロセス

ローカルプロセス名	ステートメント内のローカルプロセス名
"$イベントマネージャ"	"$イベントマネージャ"
"$チェック"	MyLocal:=**New Process** ("My Proc" ; 16*1024 ;"$チェック")
"$シリアルマネージャ"	"$シリアルマネージャ"

名前規則のまとめ

次の表は、4th Dimensionの名前規則についてまとめています。

タイプ	長さ	例
テーブル	31	[テーブル1]
フィールド	31	[テーブル1] フィールド 1
サブテーブル	31	[テーブル1] サブファイル
サブフィールド	31	[テーブル1] サブファイル'サブフィールド
インタープロセス変数	<>+31	<>インタープロセス変数
プロセス変数	31	変数
ローカル変数	$+31	$ローカル変数
フォーム	31	"フォーム"
インタープロセス配列	<>+31	<>インタープロセス配列
プロセス配列	31	プロセス配列
ローカル配列	$+31	$ローカル配列
メソッド	31	メソッド
プラグインルーチン	31	***CT GET DOCUMENT SIZE***
インタープロセスセット	<>+80	"<>インタープロセスセット"
プロセスセット	80	"プロセスセット"
クライアントセット	$+80	"$クライアントセット"
ローカルプロセス	$+31	$ローカルプロセス
グローバルプロセス	31	グローバルプロセス
命名セレクション	80	"命名セレクション"
インタープロセス命名セレクション	<>+80	"<>インター命名セレクション"

名前が重複する場合

特定のオブジェクトが別タイプのオブジェクトと同じ名前を持つ場合（例えば、フィールドが“Person”という名前で、変数も“Person”という名前の場合）に、4th Dimensionは、オブジェクトを識別するために優先順位システムを使用します。したがって、重複しない名前の使用に関してはユーザ自身に委ねられます。

4th Dimensionは、メソッドで使用される名前を次の順位で識別します。

1. フィールド
2. コマンド
3. メソッド
4. プラグインルーチン
5. 定義済定数
6. 変数

例えば、4th Dimensionには、“**Date**”という組み込み関数があります。メソッドに“Date”という名前を付けると、4th Dimensionは組み込み関数の“**Date**”として認識し、メソッドとしては認識しません。つまり、メソッドの呼び出しができなくなります。しかし、もし、フィールドに“Date”と定義すると、4th Dimensionは**Date**関数の代わりにフィールドとして使用します。

制御フロー

ユーザは、メソッドの単純性や複雑性に関係なく、プログラミング構造のシーケンス、分岐、ループの3種類のうち1つまたは複数をいつでも使用し、メソッド内でステートメントの実行や順序を制御します。

これらの構造のそれぞれを制御するステートメントがあります。この節では、これらのステートメントについての紹介にとどめます。

シーケンス構造

シーケンス構造は、単純な線形構造です。シーケンスは、4th Dimensionが最初から最後まで次々に実行する一連のステートメントです。例えば、以下のようなものです。

```
OUTPUT FORM (([人事]) ; "リスト")
ALL RECORDS ([人事])
DISPLAY SELECTION ([人事])
```

1行から成るルーチンも、オブジェクトメソッドに対しては頻繁に使用されます。最も簡単なシーケンス構造の例を次に示します。

```
[人事]名字:= Uppercase ([人事]名字)
```

分岐構造

分岐構造は、条件を判定した結果に応じて、メソッドに、別の選択肢を与えることができます。この条件は“True（真）”または“False（偽）”に評価される式、つまりブール式です。分岐構造には、**If...Else...End if**構造があります。これは、2つの選択肢のうちのいずれかにプログラムの流れを導きます。他の分岐構造には、**Case of...Else...End case**構造があり、これは多くの選択肢の中の1つだけにプログラムの流れを導きます。

ループ構造

メソッドの作成にあたって、何度も同じ処理を繰り返すことがあります。これを実行するために、4th Dimensionは**For**、**While**、**Repeat**の3つのループ構造を備えています。ループを制御する方法には、条件が満たされるまでループする方法と指定した回数だけループする方法の2通りがあります。各ループ構造にはいずれの方法も用いることができますが、**While**ループと**Repeat**ループは条件が満たされるまで繰り返す場合に、**For**ループは指定した回数だけループする場合に有効です。

If...Else...End if構造

```
If(ブール式)
        ステートメント
Else
        ステートメント
End if
```

If...Else...End if構造は、プログラムが2つの動作のうちから、判定結果（ブール式）が“True（真）か“False（偽）”かに応じて選択するものです。ブール式が“True（真）”の場合は、テストのすぐ後のステートメントを実行し、ブール式が“False（偽）”の場合には、**Else**文のすぐ後のステートメントを実行します。**Else**文はオプションです。**Else**を省略した場合の、実行は、**End if**の後の最初のステートメント（存在する場合だけ）を実行します。

例えば、

```
$名字:=Request ("名字を入力してください。")  ` ユーザに名字の入力を求める
If($名字 # "")  ` 何か名字が入力されたら...
        QUERY([人事] ; [人事]名字 = $名字)
Else
        ALERT("名字が入力されていません。")  ` ブール式“False”を実行する
End if
```

Tip：一方の条件に実行ステートメントがなくても分岐処理は行えます。

```
If(ブール式)
Else
        ステートメント
End if
```

または

```
If(ブール式)
        ステートメント
Else
End if
```

Case of...Else...End case構造

```
Case of
        ¥(ブール式)
                ステートメント
        ¥(ブール式)
                ステートメント
.
.
.
Else
        ステートメント
End Case
```

Case of...Else...End case構造は、複数の選択肢にメソッドの動作を分岐させます。**If...Else...End if**構造と異なり、**Case of...Else...End case**構造は複数のブール式を評価し、“True（真）”であるものに対して処理を実行します。

それぞれのブール式の前には円記号（¥）を付けます。この組み合わせ（円記号とブール式）を“選択式”と呼びます。例えば、以下の行は選択式です。

```
¥(番号=1)
```

最初に“True（真）”になった選択式の後のステートメントだけを実行します。“True（真）”になる選択式がない場合には、ステートメントを何も実行しません（**Else**文が指定されていない場合）。また、最後の選択式の後に**Else**文を含むことができます。すべての選択式が“False（偽）”の場合に、**Else**文の後のステートメントを実行します。

下記の例は数値変数を判定し、警告ボックスに対応する数字を表示します。

```
Case of
        ¥(数値=1)      ` 数値が1の場合の判定
                ALERT("1")   ` 1の場合の警告表示
        ¥(数値=2)      ` 数値が2の場合の判定
                ALERT("2")   ` 2の場合の警告表示
        ¥(数値=3)      ` 数値が3の場合の判定
                ALERT("3")   ` 3の場合の警告表示
        Else   ` 数値が1、2、3のどれでもない場合
                ALERT("1、2、3のどれでもない。") ` どれでもない場合の警告表示
End case
```

比較するために、同じことを**If...Else...End if**構文で記述すると以下のようになります。

```
If(数値=1)      ` 数値が1の場合の判定
        ALERT("1")   ` 1の場合の警告表示
Else
        If(数値=2)      ` 数値が2の場合の判定
                ALERT("2")   ` 2の場合の警告表示
        Else
                If(数値=3)      ` 数値が3の場合の判定
                        ALERT("3")   ` 3の場合の警告表示
                Else   ` 数値が1、2、3のどれでもない場合
                        ALERT("1、2、3のどれでもない。") ` どれでもない
                                                        場合の警告表示
                End if
        End if
End if
```

Case of...Else...End case構造の場合には、最初に“True（真）”になったケースだけを実行します。2つ以上のケースが“True（真）”の場合は、最初に“True（真）”になったケースの後のステートメントだけを実行します。

したがって、階層的なテストを実行するときには、階層上で低い位置にある条件がテスト順序ではじめに記述されていることを確認する必要があります。例えば、条件1ののテスト表記は、条件1と条件2のテスト表記をカバーするので、テスト順序の最後に位置すべきです。次のコードでは最後の条件が検出されることはありません。

```
Case of
        ¥(vResult = 1)
                `実行文
        ¥((vResult = 1) & (vCondition #2)) `この条件は検出されない
                `実行文
End case
```

上記の例では、"vResult = 1"により他の条件を見る前に分岐するので、第2の条件の表記は検出されません。コードが正しく実行されるためには次のように書きます。

```
Case of
¥((vResult = 1) & (vCondition #2))
        `実行文
¥(vResult = 1)
        `実行文
End case
```

さらに階層的なテストを実行したい場合、階層コードを使用する必要があります。

While...End whileループ

```
While (ブール式)
        ステートメント
End while
```

Whileループは、ブール式が“True（真）”である限り、ループ内のステートメントを実行し続けます。ループのはじめにブール式を評価し、ブール式が“False（偽）”の場合にはループを行いません。

一般に、**While**ループに入る手前で、ブール式で判定する値を初期化しておきます。通常はブール式が“True（真）”になるように設定してからループに入ります。

代表的な**While**ループの例として、レコードの追加があります。

```
CONFIRM ("新しいレコードを追加したいですか？") ` レコードを追加したいか？
While (OK=1) ` ユーザが希望する限りループを続行する
        ADD RECORD ([従業員])    ` 新規レコードを追加する
End while     ` End whileで終了する
```

この例では、ループに入る前に**CONFIRM**コマンドによりシステム変数OKをセットします。ユーザがダイアログボックスで「OK」ボタンをクリックすると、システム変数OKに1がセットされ、ループを開始します。それ以外の場合はシステム変数OKに0が設定され、ループをスキップします。ループに入ると、**ADD RECORD**コマンドはループを続けます。これは、ユーザがレコードを保存した時点で、システム変数OKに1が設定されるからです。ユーザが最後のレコードを取り消した（保存しない）時点で、システム変数OKに0がセットされ、ループは終了します。

ループが永久に実行されないようにするために、ブール式に対してループ内で何らかの設定を行わなければなりません。以下のループは、変数“永久”が常に“True（真）”のため、ループを永久に続行します。

```
永久:= True
While(永久)
End while
```

注：メソッドが制御できない状態で実行されているメソッドの実行を停止するには、トレース機能を使用してください。

Repeat...Untilループ

```
Repeat
        ステートメント
Until(ブール式)
```

Repeatループは、ループの後でブール式を判定する以外は**While**ループとまったく同じです。つまり、**Repeat**ループはループを必ず1回は実行しますが、**While**ループはブール式が最初に“False（偽）”である場合には、ループを実行しません。

その他の**While**ループとの相違点は、ブール式が“True（真）”になるまでループを続行することです。以下の例を、**While**ループの例と比較してください。ブール式を、初期化する必要がない点に注目してください。システム変数OKを初期化する**CONFIRM**コマンドはありません。

```
Repeat
        ADD RECORD([従業員])
Until(OK=0)
```

For...End forループ

```
For(カウンタ ; 開始値 ; 終了値 ; {増分値})
        ステートメント
End for
```

Forループは、＜カウンタ＞によって制御されるループです。このカウンタは、数値変数（実数、整数、倍長整数）で、＜開始値＞に指定した値で初期化されます。ループを実行するたびに、任意の引数＜増分値＞の値が加算されます。＜増分値＞を指定しない場合、増分値は1になります。カウンタが＜終了値＞の値を超えた時点で、ループを停止します。

重要：＜開始値＞、＜終了値＞、＜増分値＞の値は、ループの初めで一度評価されます。これらの数値が変数で指定されている場合、ループ内でこの変数の値を変更してもループは影響を受けません。

Tips：特別な目的のために、カウンタ変数の値を変更することができます。ループ内でカウンタ変数を変更すると、ループはその影響を受けます。

■ 通常、＜開始値＞は＜終了値＞より小さい。

■ ＜開始値＞と＜終了値＞が等しい場合、1回だけループが行われる。

■ ＜開始値＞が＜終了値＞より大きい場合、＜増分値＞に負の値を指定しない限り、ループは行われない。

基本的な例

1. 以下のループは、1で開始し100回ループします。

```
For($i ; 1 ; 100)
    `処理を実行する
End for
```

2. 以下の例は配列anArrayの要素をすべてループします。

```
For($vlElem;1;Size of array(anArray))
    ` 配列要素を用いて処理を実行する
    anArray{$vlElem}:=...
End for
```

3. 以下の例はテキストvtSomeTextのすべての桁に対してループを行います。

```
For($vlChar;1;Length(vtSomeText))
` TABの場合その桁を用いて処理を実行する
    If (Ascii(vtSomeText[[$vlChar]])=Tab)
        ` ...
    End if
End for
```

4. 以下の例はテーブル[テーブル1]のレコードセレクションに対してループを行います。

```
FIRST RECORD([テーブル1])
For($vlRecord;1;Records in selection([テーブル1]))
    ` レコードを用いて処理を実行する
    SEND RECORD([テーブル1])
    ` ...
    ` 以下のレコードへ
    NEXT RECORD([テーブル１])
End for
```

データベースで作成する大部分のForループは、上記例題のいずれかの形式になります。

カウンタ変数の減少

ループに際してカウンタ変数を増加させるのではなく、減少させたい場合があります。その場合、＜開始値＞に＜終了値＞より大きい値を設定し、＜増分値＞に負の数を指定する必要があります。次に挙げる例題は、前述の例と同じ処理を逆の順序で行います。

1. 以下の例は、100回ループします。

```
For($i ; 100; 1; -1)
    `処理を実行する
End for
```

2. 以下の例は配列anArrayの要素をすべてループします。

```
For($vlElem;Size of array(anArray);1;-1)
    ` 配列要素を用いて処理を実行する
    anArray{$vlElem}:=...
End for
```

3. 以下の例はテキストvtSomeTextのすべての桁に対してループを行います。

```
For($vlChar;Length(vtSomeText);1;-1)
` TABの場合その桁を用いて処理を実行する
    If(Ascii(vtSomeText[[$vlChar]])=Char(Tab))
        ` ...
    End if
End for
```

4. 以下の例はテーブル[テーブル1]の選択したレコードに対してループを行います。

```
LAST RECORD([テーブル１])
For($vlRecord;Records in selection([テーブル１]);1;-1)
    ` レコードを用いて処理を実行する
    SEND RECORD([テーブル１])
    ` ...
    ` 前のレコードへ
    PREVIOUS RECORD([テーブル１])
End for
```

カウンタ変数を1より大きな値で増加する

必要に応じて、＜増分値＞（正または負の値）に、その絶対値が1より大きな値を指定できます。

9. 以下のループは配列anArrayの偶数要素に対してのみ処理を行います。

```
For($vlElem;2;((Size of array(anArray)+1)//2)*2;2)
        ` 偶数要素に対して処理を実行する
        anArray{$vlElem}:=...
End for
```

((**Size of array**(anArray)+1)//2)*2という式により、配列サイズが偶数、奇数いずれの場合でも対応できる点に注意してください。

カウンタ変数を変更してループから抜ける

一定回数のループを行いたいが、他の条件が真になった場合はループから抜けたい場合があります。この場合、ループ内で条件判定を行い、判定結果が真であれば、カウンタ変数に＜終了値＞を越える値を明示的に設定します。

10.以下の例は、実際に処理が終了するか、あるいは最初にFALSEに設定されているインタープロセス変数<>vbWeStopがTRUEになるまでレコードセレクションを参照します。この変数は**ON EVENT CALL**プロジェクトメソッドで処理されこのメソッドにより処理を中断します。

```
<>vbWeStop:=False
ON EVENT CALL("HANDLE STOP")
` HANDLE STOPは「Ctrl+.（ピリオド）」キー（Windows）または「command+.
    （ピリオド）」キー（Macintosh）が押されると<>vbWeStopをTrueに設定する
$vlNbRecords:=Records in selection([テーブル1])
FIRST RECORD([テーブル1])
For($vlRecord;1;$vlNbRecords)
        ` レコードを用いて処理を実行する
        SEND RECORD([テーブル1])
        ` ...
        ` 以下のレコードへ
        If(<>vbWeStop)
                $vlRecord:=$vlNbRecords+1 `カウンタ変数にループ終了条件となる
                                                        値を設定
        Else
                NEXT RECORD([テーブル1])
        End if
End for
ON EVENT CALL("")
If(<>vbWeStop)
        ALERT("処理が中断されました")
Else
        ALERT("処理が正常に終了しました")
End if
```

ループ構造を比較する

Forループの例をもう一度見てみましょう。

以下の例は、100回ループします。

```
For($i ; 1 ; 100)
    `処理を実行する
End for
```

Whileループと**Repeat**ループで、同じ処理を実行する方法を調べてみましょう。

以下の例は、同じ処理を実行する**While**ループです。

```
$i:=1                   ` カウンタを初期化する
While($i <=100)         ` 100回までループを実行する
    `処理を実行する
    $i:=$i+1            ` カウンタを1加算する
End while
```

以下の例は、**Repeat**ループで同じ処理を実行します。

```
$i:=1                   ` カウンタを初期化する
Repeat
    `処理を実行する
$i:=$i+1                ` カウンタを1加算する
Until($i >100)` 100回ループする
```

Tip：4th Dimensionが、ループの条件を各サイクルで内部的に判定し、カウンタを加算するため**For**ループは、**While**や**Repeat**ループよりも高速です。なるべく、**For**ループを使用してください。

Forループの実行を最適化する

Forループの一般的な用途は、セレクションの各レコード間の移動を繰り返すことです。インタープロセス、プロセス、ローカル変数のカウンタに対し実数、整数、倍長整数タイプを用いることができます。数多く繰り返されるループの場合、とくにコンパイルモードでは、倍長整数タイプのローカル変数を使用してください。

11.次に例を示します。

```
C_LONGINT($vlCounter) ` 倍長整数タイプのローカル変数を使用
For($vlCounter;1;10000)
    ` 処理を実行する
End for
```

ネスト（入れ子）された**For**ループ構造

必要に応じて、制御構造は好きなだけネストすることができます。**For**ループも同じです。誤りを避けるため、各ループ構造ごとに別のカウンタ変数を使用してください。

次に例を示します。

12.以下の例は、2次元配列のすべての要素に対して処理を行います。

```
For($vlElem;1;Size of array(anArray))
    ` ...
    ` この行を用いて処理を実行する
    ` ...
    For($vlSubElem;1;Size of array(anArray{$vlElem}))
        ` この要素を用いて処理を実行する
        anArray{$vlElem}{$vlSubElem}:=...
    End for
End for
```

13.以下の例は、データベースのすべての日付フィールドに対するポインタの配列を作成します。

```
ARRAY POINTER($apDateFields;0)
$vlElem:=0
For($vlTable;1;Count table)
    For($vlField;1;Count fields($vlTable))
        $vpField:=Field($vlTable;$vlField)
        If(Type($vpField->)=Is Date)
            $vlElem:=$vlElem+1
            INSERT ELEMENT($apDateFields;$vlElem)
            $apDateFields{$vlElem}:=$vpField
        End if
    End for
End for
```

メソッド

コマンドや演算子がプログラミング言語として機能するには、それらをメソッド内に配置する必要があります。この章では、すべての種類のメソッドの共通機能について説明します。メソッドには、トリガ（テーブルメソッド）、フォームメソッド、プロジェクトメソッド、データベースメソッドがあります。オブジェクトメソッドも特別なメソッドの1つです。

メソッドは、ステートメントで構成されます。ステートメントとは、メソッドの1行のことで1つの命令を実行します。ステートメントは単純な場合もあれば、複雑な場合もあります。ステートメントは常に1行ですが最大32,000バイトまで使用することができます。これは、ほとんどの処理で十分な長さです。例えば、以下の行は[従業員]テーブルに新しいレコードを追加するステートメントです。

```
ADD RECORD ([従業員])
```

メソッドは、テストとループも含んでいます。これらは、制御フローをコントロールしています。制御フローに関する詳細は、後述の「制御フロー」の節を参照してください。

メソッドの種類

4th Dimensionのメソッドには、以下の5種類があります。

■「オブジェクトメソッド」：オブジェクトメソッドとは、任意のオブジェクトプロパティです。これは、通常、有効なオブジェクトフォームに関連する短いメソッドのことです。一般に、オブジェクトメソッドはそのフォームが表示または印刷されているときにオブジェクトを「管理」します。ユーザがオブジェクトメソッドを呼び出すことはありません。オブジェクトメソッドが属しているオブジェクトをイベントが含んでいる場合に、4th Dimensionがオブジェクトメソッドを自動的に呼び出します。

■「フォームメソッド」：フォームメソッドとは、任意のフォームプロパティです。ユーザはフォームメソッドを使用してデータとオブジェクトの管理を実行することができます。ただし、上述の目的には、オブジェクトメソッドを使用する方が通常は簡単であり、より効果的です。ユーザがフォームメソッドを呼び出すことはありません。フォームメソッドが属しているフォームをイベントが含んでいる場合に、4th Dimensionがフォームメソッドを自動的に呼び出します。
オブジェクトメソッドとフォームメソッドの詳細に関しては、『4th Dimensionデザインリファレンス』および第4章を参照してください。

■「トリガ（テーブルメソッド）」：トリガとは、任意のテーブルプロパティです。ユーザがトリガを呼び出すことはありません。ユーザがテーブル（追加、削除、修正、読み込み）のレコードを操作する度にトリガは4Dデータベースエンジンによって自動的に呼び出されます。トリガはユーザのデータベースのレコードに対して「不正な」オペレーションが行われることを防ぎます。例えば、「送り状」システムでは、送り状を送付した顧客名を明記せずに第三者が送り状を追加することを防ぎます。トリガはとても強力なツールで、テーブル上のオペレーションの制限を実行することができます。同様に、思いがけないデータの損失や不正な変更も防ぐことができます。ユーザは簡単なトリガを作成し、それを徐々に精巧にしていくことができます。
トリガについての詳細は、第57章の「トリガコマンド」を参照してください。

■「プロジェクトメソッド」：オブジェクトメソッド、フォームメソッド、トリガはすべて特定のオブジェクト、フォーム、テーブルとそれぞれ関連があります。これとは異なり、プロジェクトメソッドはユーザのデータベースを通して使用可能となります。プロジェクトメソッドは再利用でき、他のメソッドによる使用が可能です。作業の反復が必要になった場合でも、ユーザは状況ごとに同じメソッドを書く必要はありません。ユーザは必要に応じていつでもプロジェクトメソッドを他のプロジェクトメソッド、フォームオブジェクト、フォームメソッドから呼び出すことができます。ユーザがプロジェクトメソッドを呼び出すと、プロジェクトメソッドはユーザが呼び出した場所でユーザが作成したようにスムーズに機能します。他のメソッドから呼び出されたプロジェクトメソッドは「サブルーチン」として参照されることがよくあります。結果を返すプロジェクトメソッドは、関数とも呼ばれます。

プロジェクトメソッドには、メニューと関連付けるというもう1つの使用方法があります。ユーザがプロジェクトメソッドをメニューに関連付けると、そのメニューが選択されたときにそのメソッドが実行されるようになります。これでメニューをプロジェクトメソッドの呼び出しと同じように考えることができます。

プロジェクトメソッドに関する詳細は、第6章を参照してください。

■「データベースメソッド」：フォーム内でイベントが生じるときにオブジェクトとフォームのメソッドが呼び出されるのと同様に、作業セッションのイベントが生じると呼び出されるデータベースと連結するメソッドがあります。これが、データベースメソッドです。例えば、ユーザがデータベースを開くたびに、作業セッション中に使用する変数をいくつか初期化したいとします。このために、ユーザは「On Startup」データベースメソッドを使用します。これはユーザがデータベースを開くときに、4Dによって自動的に実行されます。

データベースメソッド関する詳細は、第6章を参照してください。

■「プラグイン」：プラグインは、4th Dimension以外で作成したメソッドです。プラグインは一般にPascalやC言語で作成されます。このメソッドはコンパイルされた後、4th Dimensionのアプリケーションでの使用が可能になります。異なる機能を持ったプラグインを4th Dimensionに追加することができます。
プラグイン以外のメソッドは、4th Dimensionの「メソッド」エディタを使用して作成します。

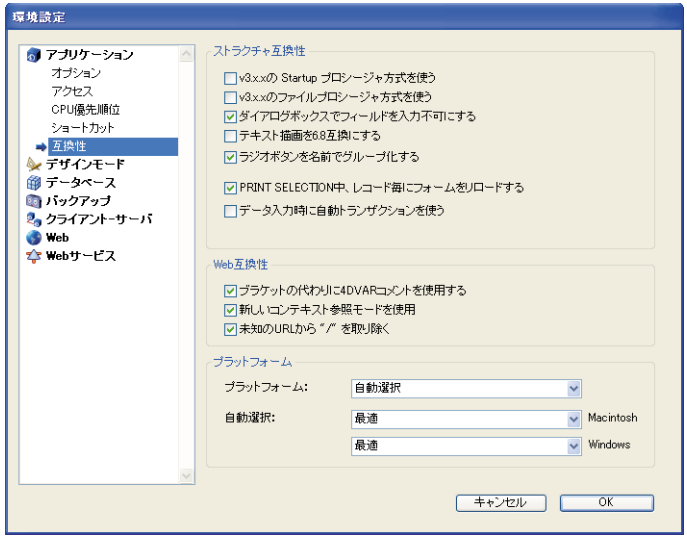
4th Dimensionバージョン3との互換性

4th Dimensionのバージョン6以降で新しく作成されたデータベースで作業している場合は、この互換性に関する注記は読み飛ばしてかまいません。

1. バージョン6以降では、新しいオブジェクトとフォームイベント（**On Double Clicked**、**On Getting Focus**等）が数多く提供されています。これらは、バージョン3の実行サイクルと置き替わります。バージョン3のデータベースをバージョン6以降に変換した場合、そのフォームも変換されます。これは、ユーザのフォームとオブジェクトが「期待通り」に動作するように管理するためです。バージョン3の4th Dimensionで作成したフォームとオブジェクト用に新しいイベントを利用したい場合、そのフォームとオブジェクト用に「フォームプロパティ」または「オブジェクトプロパティ」ウインドウ内で新しいイベントを有効にする必要があります。

2. テーブルメソッドとも呼ばれるトリガは、バージョン6以降で新しく追加されたメソッドです。バージョン3の4th Dimensionでは、テーブル用のフォームがデータ入力、表示、印刷に使用されるときだけ、4Dが（ファイルプロシージャと呼ばれる）テーブルメソッドを実行しました。このため、これらは滅多に使用されませんでした。トリガは、旧バージョンのファイルプロシージャのかなり低いレベルで実行されることに注意してください。（データ入力のような）ユーザアクションを通して、または（**SAVE RECORD**コマンドへの呼び出しのような）プログラムから、レコードに対して実行する処理が何であれ、4Dはテーブルのトリガを呼び出します。トリガは、旧バージョンのファイルプロシージャとはまったく異なります。バージョン3のデータベースをバージョン6以降用に変換し、新しいトリガの機能を利用したい場合、下図の「データベースプロパティ」ダイアログボックス内の「旧バージョンのファイルプロシージャ方式を使用する」チェックボックスを選択解除する必要があります。

3. データベースメソッドは、バージョン6以降で新しく追加されたメソッドです。バージョン3の4th Dimensionでは、ユーザがデータベースを開いたときに4Dが自動的に実行するメソッド（プロシージャ）は1つだけでした。このプロシージャを呼び出すには、「Startup」プロシージャを呼び出す必要がありました。バージョン3のデータベースをバージョン6以降に変換した場合、そして新しい「On Startup」データベースメソッドの機能を利用したい場合、下図の「データベースプロパティ」ダイアログボックス内の「旧バージョンのStartupプロシージャ方式を使用する」チェックボックスを選択解除する必要があります。このチェックボックスは、「Startup」プロシージャまたは「On Startup」データベースメソッドのどちらか一方にしか効果はありません。この属性の選択解除を行わず、そして例えば、「On Exit」データベースメソッドを作成した場合、4Dは後者の「On Exit」データベースメソッドを呼び出します。



メソッドの例と用語

この節では、メソッドの用語、概念に共通する部分について説明するためにメソッドを詳しく調べてみることにします。この節に記述した内容の詳細は、このマニュアルの他の部分で説明されています。

メソッドは先頭の行から始まり、最後の行に到達するまで、各ステートメント（命令文）を実行します。基本的には、すべてのメソッドは同じです。次にプロジェクトメソッドの例を示します。

```
QUERY([従業員])    `「クエリ」エディタを表示する
If(Records in selection([従業員]) =0)    `何も見つからない場合、
    ADD RECORD([従業員])    `レコードを追加する
End if `終了
```

まず、プログラミング言語の用語と機能を説明することにします。上記の各行を“ステートメント”または“コード行”と呼びます。プログラミング言語を使用して作成したものを、単にコードとも呼びます。4th Dimensionは、コードで指定した処理を実行します。

それでは、最初の行を詳しく見てみましょう。

```
QUERY([従業員])    `「クエリ」エディタを表示する
```

この行の最初の要素である**QUERY**は、コマンドです。コマンドは、4th Dimensionのプログラミング言語の一部で、処理を実行します。**QUERY**コマンドは「クエリ」エディタを表示します。「ユーザ」モードの「クエリ」メニューから「検索」を選択することと同じ機能です。

この行の2番目の要素である括弧は、**QUERY**コマンドに対する引数（パラメータ）を指定します。引数は、コマンドが処理を実行するために必要なものです。この例では、[従業員]はテーブル名です。テーブル名は常に角カッコ（[...]）の中で指定します。つまり、“従業員テーブルが**QUERY**コマンドの引数である”ということを意味します。コマンドの中には複数の引数を持つものもあります。

3番目の要素は、行の終りに指定されたコメントです。コメントは逆アポストロフィ（`）によって示します。ユーザ（コードを解析する人）にこのコードが何を行っているのかを説明します。またコメント記号に続く内容は、コードを実行する時点で無視されます。コメントはそれ自体で1つの行になりますが、通常は上記の例で示すようにコードの右側に記述します。コメントを使用することにより、メソッドの内容を読みやすく、理解しやすいものにします。コメントは、80バイトまで入力することができます。

以下の行は、レコードが見つかったかどうかを調べます。

```
If(Records in selection ([従業員])=0)      `何も見つからない場合...
```

Ifはフロー制御ステートメント（“フロー制御文”または単に“文”）です。これはメソッドの実行の流れを制御します。**If**文は、カッコ内の式を判定し、結果が“True（真）”の場合、実行が以下の行に継続されます。本マニュアルでは、2つのブール値“True（真）”と“False（偽）”を使用します。

Records in selectionは関数です。これは値を返すコマンドです。ここでは、**Records in selection**関数は引数として渡されたテーブルのカレントセレクションのレコード数を返します。

注：関数名の頭文字だけが大文字になっていることに注意してください。これは、4th Dimensionの関数に対する命名規則によるものです。

既に、カレントセレクションとは何かについて説明しました。これは、その時点で作業対象となっているレコードの集まりのことです。レコードの数が0件の場合（レコードが全く見つからない場合）に以下の行を実行します。

```
ADD RECORD([従業員])    `レコードを追加する。
```

ADD RECORDコマンドは入力フォームを表示し、新しいレコードを追加します。この行はインデントされています。4th Dimensionは、自動的にコードをフォーマットします。この行は、先ほどのフロー制御ステートメント（**If**）に従属することを示すためにインデントされています。

```
End if `終了
```

End if文は**If**文の制御セクションを終了します。フロー制御ステートメントを使用した場合は、常に制御が終了する場所を示す対応ステートメントを指定する必要があります。

この節の概念をしっかりと把握し、理解されるようお薦めします。

■ オブジェクトメソッドとフォームメソッドに関する詳細は、第4章を参照してください。

■ トリガに関する詳細は、第57章の「トリガコマンド」を参照してください。

■ プロジェクトメソッドに関する詳細は、第6章を参照してください。

■ データベースメソッドに関する詳細は、第6章を参照してください。

プロジェクトメソッド

プロジェクトメソッドには、適切な名前を付ける必要があります。フォームメソッドやオブジェクトメソッドはフォームやオブジェクトと密接に関連付けられていますが、プロジェクトメソッドはどこでも使用できます。特に、データベースの特定のオブジェクトに付属しているわけではありません。プロジェクトメソッドはその実行方法や使用方法に応じて、次のような役割を果たします。

■ メニューメソッド

■ サブルーチンや関数

■ プロセスメソッド

■ イベント管理メソッド

■ エラー管理メソッド

これらは、プロジェクトメソッドをその名称で識別するのでなく、その動作で識別します。

メニューメソッドは、カスタムメニューから呼び出されるプロジェクトメソッドです。これは、ユーザのアプリケーションの流れを管理します。メニューメソッドは、必要とされる場所での分岐、フォームの提供、レポートの生成、ユーザデータベースの一般的な管理といった制御を行います。

別のメソッドから呼び出されたプロジェクトメソッドは、しばしばサブルーチンとして参照されます。結果を返すプロジェクトメソッドは関数という言い方もします。

プロセスメソッドは、プロセスの開始時に呼び出されるプロジェクトメソッドです。このプロセスは、プロセスメソッドが実行されている間だけ存続します。プロセスに関する詳細は、第10章を参照してください。「新規プロセス開始」チェックボックスが選択されているメニューに属しているメニューメソッドは、新しく開始されるプロセス用のプロセスメソッドでもあることに注意してください。

イベント管理メソッドは、イベントを管理するプロセスメソッドとして個々のプロセス内で実行されます。通常、イベント管理の大部分を4Dに任せています。例えば、データ入力中、4Dがキーストロークやクリックを検出し、それから正しいオブジェクトとフォームメソッドを呼び出します。このためユーザは、これらのメソッド内のイベントに対し適切に応答できるのです。一方、ユーザがイベントを直接操作したい場合があります。例えば、ユーザが（**For...End For**ループ、レコードの操作等）処理時間の長い操作を実行する場合、ユーザは「Ctrl+.（ピリオド）」キー（Windows）や「command+.（ピリオド）」キー（Macintosh）を押して、その操作への割り込みを行いたいとします。この場合、ユーザはイベント管理メソッドを使用する必要があります。詳細は、**ON EVENT CALL**コマンドの説明を参照してください。

エラー管理メソッドは、割り込みを実行するプロジェクトメソッドです。エラーや例外が起こる度に、エラー管理メソッドがインストールされたプロセス内で実行します。詳細は、**ON ERR CALL**コマンドの説明を参照してください。

メニューメソッド

メニューメソッドは、それが付属するカスタムメニューを選択すると、カスタムメニューモードで起動されます。「メニュー」エディタを使用して、メソッドをメニューに割り当てます。メニューが選択されると、それに対応するメソッドが起動されます。このプロセスは、データベースをカスタマイズする上で大きな特長の1つです。特定の処理を実行するメニューメソッドを割り当てたカスタムメニューを作成することで、データベースをカスタマイズすることができます。詳細は、『4th Dimension デザインリファレンス』の「メニュー」エディタの説明を参照してください。

カスタムメニューは、単一または複数の処理を実行することができます。例えば、データの入力処理を実行するメニューは、以下の2つの処理を実行するメソッドを呼び出します。第1に、適切な入力フォームを表示します。第2に、ユーザがキャンセルするまでの**ADD RECORD**コマンドによるデータ入力を繰り返します。

連続処理の自動化は、プログラミング言語の強力な機能の1つです。カスタムメニューを使用すると、「ユーザ」モードにおいて手作業で行っていた処理を自動化することができます。また、カスタムメニューを使用することで、ユーザにより良い操作環境を提供することができます。

サブルーチン（メソッドから起動されるメソッド）

プロジェクトメソッドを作成すると、それは同じデータベースシステムの言語の一部になります。プロジェクトメソッドは、4th Dimensionに組み込まれたコマンドと同様に呼び出すことができます。このように使用されるプロジェクトメソッドを“サブルーチン”と呼びます。

サブルーチンには、以下の4つの利点があります。

■ コーディングの重複をなくすことができる。

■ メソッドの機能を明確にすることができる。

■ メソッドの編集を容易にすることができる。

■ メソッドをモジュール化することができる。

例えば、“顧客”というデータベースがあるとします。メソッドを作成していくうちに同じ処理を繰り返しコーディングしていることに気づきました。それは顧客テーブルを検索してレコードを修正するという一連の作業でした。そのコーディングは以下のようになっています。

```
      ` 顧客テーブルの検索
QUERY BY EXAMPLE ([顧客])
      ` 入力フォームの設定
INPUT FORM (([顧客]) ; "入力フォーム")
      ` 顧客レコードの修正
MODIFY RECORD ([顧客])
```

サブルーチンを使用しなければ、顧客レコードの修正を実行するたびにコードを作成しなければなりません。10箇所で同じ処理が必要になると、同じコーディングを10回も行わねばなりません。しかし、サブルーチンを使用すれば、1回コーディングするだけで済みます。これが“コーディングの重複をなくす”というサブルーチンの第1の利点です。

このメソッドに、例えば“M_顧客修正”という名前をつければ、その名前を他のメソッドで、記述するだけで呼び出すことができます。顧客テーブルの検索修正を行い、その後で印刷を実行するメソッドのコーディング例を次に示します。

```
M_顧客修正
PRINT SELECTION ([顧客])
```

この例のようにサブルーチンを使用することにより、メソッドを単純化することができます。サブルーチンを呼び出す側は、サブルーチンがどのように処理するのかを知る必要はありません。サブルーチンが何を実行するのかを理解していれば十分です（この例ではサブルーチンの名前どおりに、顧客テーブルを修正するということを理解していれば十分です）。これが“メソッドの機能を明確にする”というサブルーチンの第2の利点です。このように、メソッドは4th Dimension言語の機能を拡張します。

また、例題のデータベースで顧客テーブルの検索処理を変更する必要が発生した場合でも、“M_顧客修正”のメソッドの修正だけで済みます。これが“メソッドの編集を容易にする”というサブルーチンの第3の利点です。

サブルーチンを使用して、“コードのモジュール化”を実行することができます。つまり、コードを論理的な処理単位で、モジュール（サブルーチン）に分割します。以下の当座預金データベースの例で考察してみましょう。

M_決済済み検索 `決済済みの小切手を見つける
M_口座の照合 `口座の照合
M_出納印刷 `出納記録の印刷

データベースを知らない人でも、このプログラムが何をしているかはわかります。各サブルーチンの処理手順を知る必要はありません。各サブルーチンは長く、複雑な処理で構成されていることもありますが、そのサブルーチンが何を実行するのかだけを知っていれば十分です。

プログラムを論理的な処理単位やモジュールにできるだけ分割することをお勧めします。メソッドが20行を越える場合には、そのメソッドを分割することを検討してください。これが“メソッドのモジュール化”というサブルーチンの第4の利点です。

サブルーチンへ渡す引数（パラメータ）

サブルーチンへデータを渡すときに、引数（パラメータ）を使用すると簡単に実行することができます。

引数は、サブルーチン内で行う処理に必要な一連のデータです。引数やパラメータという用語はこのマニュアルの随所で使用されています。引数は、4th Dimensionのコマンドへデータを渡す場合にも使用します。以下の例は、文字列“こんにちは”という引数を**ALERT**コマンドへ渡します。

```
ALERT ("こんにちは")
```

引数は、サブルーチンに対しても同じように渡すことができます。例えば、サブルーチンが、3つの引数を受け取る“MyProc”というメソッドを呼び出すには以下のようにコーディングします。

```
MyProc (This ; That ; The Other)
```

引数の間は、セミコロン（;）で区切ります。

サブルーチン内で、それぞれの引数の値は自動的に、順に番号が付けられたローカル変数（$1、$2、$3...）に格納されます。ローカル変数の番号は、引数の順序を表わします。

このローカル変数は呼び出しメソッドから渡された実際のフィールドや変数、式ではなく、渡された値を保持しているだけです。

サブルーチン内で、他のローカル変数と同様にこれらの引数（$1、$2...）を使用できます。

ローカル変数は、サブルーチン内でのみ使用可能です。サブルーチンから戻ると、これらのローカル変数はクリアされます。このため、ローカル変数の値を変更しても、サブルーチンを呼び出したメソッドの引数の値は、変更されません。次に例を示します。

```
` メソッド“MYメソッド”のコードは以下の通りです
` ...
   処理 ([人事]名字)      ` [人事]名字の内容が“williams”であるものとする
   ALERT([人事]名字)

   ` メソッド“処理”のコードは以下の通りです
   $1:=Uppercase($1)
   ALERT($1)
```

メソッド“処理”の警告ボックスには“WILLIAMS”が表示され、メソッド“MYメソッド”の警告ボックスには“williams”が表示されます。このメソッドではローカルに引数$1の値が変更されますが、メソッド“MYメソッド”から渡された引数である[人事]名字フィールドの値は影響されません。

メソッド“処理”でフィールドの値を変更する方法には2通りあります。

1. メソッドにフィールドを渡すのではなく、フィールドへのポインタを渡します。

```
   ` メソッド“MYメソッド”のコードは以下の通りです
   ` ...
   ` [人事]名字の内容が“williams”であるものとする
   処理 (->[人事]名字)
   ALERT([人事]名字)

   ` メソッド“処理”のコードは以下の通りです
   $1->:=Uppercase($1->)
   ALERT($1->)
```

上の例では、引数はフィールドではなくフィールドへのポインタです。したがって、メソッド“処理”で$1はフィールドの値ではなく、フィールドへのポインタになります。$1により参照される（例では$1->）オブジェクトは実際のフィールドです。この結果、参照されたオブジェクトを変更することは、サブルーチンという枠を越えてしまい、実際のフィールドに影響を与えます。上の例の警告ボックスはともに“WILLIAMS”を表示します。

ポインタに関する詳細は、第9章を参照してください。

2. メソッド“M処理A”に処理を行わせるだけではなく、値を返すようにコードを書き換えることができます。

```
   ` メソッド“MYメソッド”のコードは以下の通りです
   ` ...
   ` [人事]名字の内容が“williams”であるものとする
   [人事]名字:=M処理A ([人事]名字)
   ALERT([人事]名字)

   ` メソッド“M処理A”のコードは以下の通りです
   $0:=$1
   ALERT($0)
```

サブルーチンから値が返される2番目の手法は、“関数”と呼ばれますが、以下の項で説明します。

注：“{ }”を使用してサブルーチンへ引数を渡すこともできます。例えば、${i}は、iの値が1の場合の$1と同じ意味です。詳細はCount Parameters関数の説明を参照してください。

関数としてのサブルーチン

サブルーチンから、データを返すこともできます。値を返すサブルーチンを“関数”と呼びます。値を返す4Dコマンドや4Dプラグインコマンドも“関数”と呼びます。

以下の行は、文字列のデータ長を返す**Length**関数を用いたステートメントです。このステートメントは、**Length**関数が“長さ”という変数に値を返します。

```
長さ:= Length ("How did I get here?")
```

どのようなサブルーチンでも値を返すことができます。返す値を、ローカル変数$0に格納します。例えば、“Uppercase4”という以下の関数は、始めの4文字を大文字に変換した文字列を返します。

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

次は、Uppercase4関数を使用した例です。

```
NewPhrase:=Uppercase4 ("This is good.")
```

この例では、変数「NewPhrase」に“THIS is good.”が格納されます。

関数の結果である$0は、サブルーチン内のローカル変数です。サブルーチン内では、ユーザは他のローカル変数を使用するときと同様に$0を使用できます。（サブルーチンが終了したときのままの）$0の値を呼ばれたメソッドに返すのは4Dです。

再帰プロジェクトメソッド

プロジェクトメソッドは、自分自身を呼び出すことができます。例えば、

■ メソッドAは、Aを呼び出すメソッドBを呼び出します。このためAが再びBを呼び出します。

■ メソッドAは自身を呼び出すことができます。

これは、再帰性と呼ばれています。4D言語は再帰性を完全にサポートしています。

ここに例題があります。2つのフィールドで構成された[友人関係]テーブルを持っているとします。

- [友人関係] 名前
- [友人関係] 子供' 名前

この例題では、フィールド内の値は重複しない（つまり、同じ名前の人がいない）ものとします。名前を与え、次に以下の文を組み立てます。「私の友人であり、正雄の子供である弘子の子供である光男の子供である祐也の子供である栄治は生活のためにこれを行っています。」

1. ユーザは、以下のように文を組み立てることができます。

```
$vsName:=Request("名前を入力してください";"正雄")
If(OK=1)
    QUERY([友人関係];[友人関係]名前=$vsName)
    If(Records in selection([友人関係])>0)
        $vtTheWholeStory:="私の友人であり、"+$vsName
        Repeat
            QUERY([友人関係];[友人関係]子供'名前=$vsName)
            $vlQueryResult:=Records in selection([友人関係])
            If($vlQueryResult > 0)
                $vtTheWholeStory:=[友人関係]名前+"の子供である "
                                                    +$vtTheWholeStory
                $vsName:=[友人関係]名前
            End if
        Until($vlQueryResult=0)
        $vtTheWholeStory:=$vtTheWholeStory+"は生活のためにこれを
                                                    行っています。"
        ALERT($vtTheWholeStory)
    End if
End if
```

2. また、ユーザは以下のように文を組み立てることもできます。

```
$vsName:=Request("名前を入力してください";"正雄")
If(OK=1)
    QUERY([友人関係];[友人関係]名前=$vsName)
    If(Records in selection([友人関係])>0)
        ALERT("私の友人であり、"+世代($vsName)+"は
                                    生活のためにこれを行っています。")
    End if
End if
```

次は、「世代」再帰性関数のコードです。

```
`「世代」プロジェクトメソッド
` 世代(文字列) → テキスト
` 世代(名前) → 文の一部
$0:=$1
```

```
QUERY([友人関係];[友人関係]子供'名前=$1)
If(Records in selection([友人関係])>0)
        $0:=$0+"の子供である "+世代([友人関係]名前)
End if
```

自分自身を呼び出す「世代」メソッドであることに注意してください。

最初に挙げた方法は反復性のアルゴリズムです。2番目に挙げた方法は再帰性のアルゴリズムです。

前述の例題のような場合にコードを履行する場合、反復性や再帰性を使用してメソッドを書くことが常にできるということに注意してください。

一般的に、再帰性はより明瞭で、読みやすく、維持しやすいコードを提供します。ただし、この使用は強制ではありません。

4D内での再帰性の代表的な使用方法は以下の通りです。

■ 例題と同じく、お互いに関連するテーブル内でのレコードの取り扱い。

■ **FOLDER LIST**コマンドと**DOCUMENT LIST**コマンドを使用してのユーザディスク上にあるドキュメントとフォルダのブラウズ処理。フォルダにはフォルダとドキュメントが含まれており、サブフォルダはフォルダとドキュメントとその他を含むことができます。

重要：再帰性の呼び出しは、常にある地点で終了する必要があります。例えばGenealogyメソッドが自身の呼び出しを止めるのは、クエリがレコードを返さないときです。この条件のテストをしないと、メソッドは際限なく自身を呼び出します。最終的に、4Dは“スタックがいっぱいです。”エラーを返します。その理由は4Dが既に呼び出しを「蓄積する」容量を持たないためです（メソッド内で使用される引数とローカル変数についても同様です）。

Application type

Application type→倍長整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	倍長整数	←	アプリケーションタイプを示す数値

説明

Application type関数は、現在実行している4D環境のアプリケーションタイプを示す数値を返します。4th Dimensionは、以下のようにあらかじめ定義された定数を持っています。

定数	タイプ	値
4th Dimension	倍長整数	0
4D Runtime Volume License	倍長整数	1
4D Runtime Interpreted	倍長整数	2
4D Runtime Single User	倍長整数	3
4D Client	倍長整数	4
4D Server	倍長整数	5
4D First	倍長整数	6

例題

▼ 以下の例は、4D Serverが実行されているかどうかを検査します。以下のコードを「On Server Startup」データベースメソッド以外の場所に記述します。

```
If(Application type=4D Server)
    `4D Server上におけるそれ相応の動作を実行する
End if
```

参照

Application version、Version type

Version type

Version type→倍長整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	倍長整数	←	0=フルバージョン（製品版） 1=デモ版（制限付き）

説明

Version type関数は、現在実行している4D環境のバージョンタイプを示す数値を返します。4th Dimensionは、次のようにあらかじめ定義された定数を持っています。

定数	タイプ	値
Full Version	倍長整数	0
Demo Version	倍長整数	1

例題

▼4Dアプリケーションが4D環境のデモ版では利用できないいくつかの機能を含んでいる場合に、判定式で**Version type**関数をコールして処理を分岐します。

```
If(Version type=Full Version)
    `何らかの処理を実行する
Else
    ALERT("デモ版では、いくつかの機能を使用することができません。")
End if
```

参照

Application type、Application version

Application version

Application version({*})→文字列

引数	タイプ		説明
*	*	→	指定した場合、ロングバージョン番号 指定しなかった場合、ショートバージョン番号
戻り値	文字列	←	バージョン番号のエンコードされた文字列

説明

Application version関数は、現在使用している4D環境のバージョン番号を表すエンコードされた文字列値を返します。

■引数オプション<*>を指定しない場合、以下のようにフォーマットされた4バイト文字列を返します。

桁	説明
1-2	バージョン番号
3	更新番号
4	訂正版番号

例えば、文字列“0600”は、バージョン6.0.0を表します。

■引数オプション<*>を指定した場合、以下のようにフォーマットされた8バイト文字列を返します。

桁	説明
1	"F"は最終版を意味する "B"はベータ版を意味する それ以外の文字は、4Dの内部バージョンを意味する
2-3-4	4Dの内部コンパイル番号
5-6	バージョン番号
7	更新番号
8	訂正版番号

例えば、文字列“B0120602”は、バージョン6.0.2のベータ12を表します。

例題

▼次の例は、4D環境のバージョン番号を表示します。

```
$vs4Dversion:=Application version
ALERT("現在使用している4Dのバージョン： "+String(Num(Substring
                ($vs4Dversion; 1 ; 2)))+"."+$vs4Dversion 3+"."+$vs4Dversion 4 )
```

▼以下の例は、最終版の4Dを使用しているかどうかを検査します。

```
If(Subtring(Application version(*) ; 1 ; 1) # "F")
    ALERT("最終版の4Dでこのデータベースを使用しているかどうか
                                        確認してください。")
    QUIT 4D
End if
```

参照

Application type、Version type

Compiled application

Compiled application→プール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	True=コンパイル、False=インタプリタ

説明

Compiled application関数は、データベースをコンパイルモード（True）、またはインタプリタモード（False）のどちらで実行しているかを検査します。

例題

▼ インタプリタモードで実行している場合にだけ使用したいデバッグコードをルーチン内に作成する場合、以下のようにIf文で囲まれた中にデバッグ用のコードを記述します。

```
If(Not(Compiled application))
   ` ここにデバッグ用のコードを記述する
End if
   ` ...
```

参照

IDLE、Undefined

Application file

Application file→文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	4D実行形式のファイルまたは アプリケーションのロング名

説明

Application file関数は、現在使用している4D実行形式のファイルまたはアプリケーションのロング名を返します。

Windows上

例えば、ボリュームE上の「4DWIN600¥ PROGRAM」の中に配置された4th Dimensionを使用している場合、この関数は、「E:¥ 4DWIN600¥ PROGRAM¥ 4D,EXE」を返します。

Macintosh上

例えば、ハードディスク「Macintosh HD」上の「4th Dimension® 6.0 f」フォルダの中にある4th Dimensionを使用している場合、この関数は、「Macintosh HD:4th Dimension® 6.0 f:4th Dimension® 6.0」を返します。

例題

▼以下の例は、Windowsの起動時に、任意のDLLライブラリが4D実行形式のファイルと同じ階層に配置されているかどうかをチェックします。「On Startup」データベースメソッドの中に次のコードを記述します。

```
If(On Windows &(Application type#4D Server))
   If(Test path name( Long name to path name(Application file)
                                  +"XRAYCAPT.DLL")#Is a document)
      `「XRAYCAPT.DLL」ライブラリが見つからないことを説明した
                                  ダイアログボックスを表示する。
                          そのため、x-rayキャプチャの機能を利用できない
   End if
End if
```

注：プロジェクトメソッド「On Windows」と「Long name to path name」は、第52章の「システムドキュメントコマンド」の節で説明されています。

参照

Data file、DATA SEGMENT LIST、Structure file

Structure file

Structure file→文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	データベースストラクチャファイルのロング名

説明

Structure file関数は、現在使用しているデータベースのストラクチャファイルのロング名を返します。

Windows上

例えば、ボリュームG上の「DOCS¥ MyCD」の中に配置されたデータベースを使って作業している場合、この関数は、「G:¥ DOCS¥ MyCD¥ myCD.4DB」を返します。

Macintosh上

例えば、ハードディスク「Macintosh HD」上の「MyCD f」フォルダの中に配置されたデータベースを使って作業している場合、この関数は、「Macintosh HD:MyCD f :My CD」を返します。

注：データベースがコンパイルされて4D Runtimeに統合されている場合、このコマンドはアプリケーションファイル（実行可能なアプリケーション）のパス名を返します。Macintosh上では、このファイルはソフトウェアパッケージの中の「Contents：MacOS」フォルダに置かれます。ソフトウェアパッケージ自体のフルアクセスパスを取得したい場合には、Application file関数を利用することをお勧めします。方法としては、Application type関数を使用してアプリケーションタイプを調べた後、その結果に応じてStructure file関数またはApplication type関数を実行します。

警告：4D Clientを実行している最中にこの関数を呼び出すと、ロング名ではなくストラクチャファイルの名前だけが返されます。

例題

▼ 以下の例は、現在使用中のストラクチャファイルの名前と配置場所を表示します。

```
If(Application type#4D Client)
    $vsStructureFilename:= Long name to file name(Structure file)
    $vsStructurePathname:= Long name to path name(Structure file)
    ALERT("現在下記のデータベースを使用しています"+Char(34)
        +$vsStructureFilename+Char(34)+" located at "+Char(34)
```

```
                                        +$vsStructurePathname+Char(34)+".")
    Else
        ALERT("次のデータベースに接続されます："+Char(34)
                                        +Structure file+Char(34))
    End if
```

注：プロジェクトメソッド「Long name to file name」および「Long name to path name」は、「System Documents」の節に記載します。

参照

Application file、Data file、DATA SEGMENT LIST

Data file

Data file({セグメント})→文字列

引数	タイプ		説明
セグメント	数値	→	セグメント番号
戻り値	文字列	←	データベースのデータファイルのロング名

説明

Data file関数は、現在使用しているデータベースのデータファイルのロング名、または現在使用しているデータベースの1つのデータセグメントを返します。

オプション引数＜セグメント＞を指定しない場合、**Data file**関数はそのデータファイルまたは1番目のセグメント（データベースが分割されている場合）のロング名を返します。

＜セグメント＞を指定した場合、**Data filee**関数は対応しているデータセグメントのロング名を返します。データセグメントの数よりも大きいセグメント番号を指定すると、**Data file**関数は空の文字列を返します。

Windows上

例えば、ボリュームG上の「¥DOCS¥MyCD」の中に配置されたデータベースMyCDを使って作業している場合、この関数は、「G:¥DOCS¥MyCD¥MyCD.4DD」を返します(データベース作成時にデフォルトの場所と名前を保存した場合)。

Macintosh上

例えば、ハードディスク「Macintosh HD」上の「MyCD」フォルダの中に配置されたデータベースを使って作業している場合、この関数は、「Macintosh HD:MyCD f :My CD.data」を返します（データベース作成時にデフォルトの場所と名前を保存した場合)。

警告：4D Clientを実行している最中にこの関数を呼び出すと、ロング名ではなくデータファイルの名前、あるいは1番目のセグメントの名前が返されます。また、たとえデータベースが分割されていようとも、Data file関数は他のデータセグメントに関しては空の文字列を返します。もし、4D Client上にデータセグメントの一覧を表示したい場合、ストアドプロシージャを使ってデータセグメントリストを作成し、サーバマシン上の任意の変数内にそのリストを格納します。そして、GET PROCESS VARIABLEコマンドを使って、この変数の内容を取得します。

例題

▼ 以下の例は、データベースのデータベースセグメントを調べます。

```
If(Application type#4D Client)
    $vlDataSegNum:=0
    Repeat
        $vlDataSegNum:=$vlDataSegNum+1
        $vsDataSegName:=Data file($vlDataSegNum)
        If($vsDataSegName#"")
            ALERT("データセグメント"+String($vlDataSegNum)
                +":"+Char(34)+$vsDataSegName+Char(34)+".")
        End if
    Until($vsDataSegName="")
    ALERT(String($vlDataSegNum-1)+"個のデータセグメントがあります。")
End if
```

参照

Application file、Data file、DATA SEGMENT LIST

Is data file locked

Is data file locked→ブール

引数	タイプ		説明
このコマンドに引数はありません。			
戻り値	ブール	←	True= ファイル／セグメントがロックされている False= ファイル／セグメントはロックされていない

説明

Is data file locked関数は、開かれているデータベースのデータファイル、または少なくとも1つのセグメントがロックされている（つまり、書き込み禁止）場合にTrue（真）を返します。

例えば、「On Startup」データベースメソッドにおいてこのコマンドを使用すると、ロックされたデータファイルを誤ってオープンする危険性を回避することができます。

例題

このメソッドは、データファイルがロックされている場合、データベースのオープンを回避します。

```
If(Is data file locked)
   ALERT("データファイルはロックされています。データベースを開くことができません。")
   QUIT 4D
End if
```

Get 4D folder

Get 4D folder{(folder)}→文字列

引数	タイプ		説明
folder	倍長整数	→	フォルダのタイプ （省略されている場合= Active 4D Folder）
Function result	文字列	←	4Dフォルダのパス名

説明

Get 4D folder関数は、カレントアプリケーションのアクティブな 4D フォルダ、もしくは指定された4D環境フォルダへのパス名を返します。この関数を使用すると、4Dアプリケーションで使用されたフォルダの実際のパス名を取得することができます。また、この関数を使用することで、記述したコードがローカルシステムのどのプラットフォームでも作動することを確認することができます。

フォルダには、"4D環境"テーマ内にある、次のいずれかの定数を渡します。

定数	タイプ	値
Active 4D Folder	倍長整数	0（デフォルト）
Licenses Folder	倍長整数	1
Extras Folder	倍長整数	2
4D Client Database Folder	倍長整数	3

Active 4D Folder

4D環境は、4Dフォルダを使用して以下の情報を保存します。

- ユーザレジストレーションファイル
- 4D環境アプリケーション、ツール、ユーティリティプログラムで使用される初期設定（Preference）ファイル
- TCP/IPネットワークプロトコルのオプションファイル
- 4D Serverからダウンロードされたリソース、プラグイン、Extra folderなどを格納するために4D Clientによって作成されたローカルデータベースフォルダ

4Dフォルダは、以下の場所に作成されます。

- Windows：{Disk}:¥Documents and Settings¥All Users¥Application Data¥4D

4D Clientでの注意：4D Clientを使用する場合、Active 4D Folderは次の場所に作成されます。Disk}:¥Documents and Settings¥Current user¥Application Data¥4D

Current userとは、カレントウインドウズセッションを開いたユーザ名を指します。

■ Macintosh: {Disk}:Library:Application Support:4D

Licenses Folder

マシンのライセンスファイルを含むフォルダです。

Licenses Folderは、次の場所に置かれます。

■ Windows: {Disk}:¥Documents and Settings¥All Users¥Application Data¥4D¥Licenses

■ Macintosh: {Disk}:Library:Application Support:4D:Licenses

Extras Folder

カスタマイズされた内容を含むフォルダで、各4D Clientマシンにダウンロードされます。このフォルダは、カスタマイズされたアイテム（リソースファイル、テキスト文書、XML preferences filesなど）をサーバからクライアントマシンに移動するために使用します。オリジナルのフォルダの階層は、それぞれのクライアントマシンで再構築されます。

4D Serverは、このフォルダへの変更を自動的に管理し、必要なものだけを移動します。また、ネットワークでの複製時間を最適化するため、フォルダの内容は圧縮されます。

サーバ側のオリジナルのExtrasフォルダは、データベースのストラクチャファイルと同じ階層に置いてください。それぞれの4D Clientには、他のストラクチャ要素と同じ場所にExtrasフォルダがダウンロードされます。例えば、

■ Windows：{Disk}:¥Documents and Settings¥Current user¥Application Data¥4D¥DatabaseName_Address¥Extras
Current userとは、カレントウインドウズセッションを開いたユーザ名を指します。

■ Macintosh：{Disk}:Library:Application Support:4D:DatabaseName_Address: Extras

4D Client Database Folder (クライアントマシン)

各4D Clientのマシンに作成された4Dのデータベースフォルダ。リーソス、プラグイン、Extrasフォルダその他データベースに関連したファイルやフォルダを格納するためのフォルダです。

4D Client Database Folderは各クライアントマシンの次の場所に置かれます。

■ Windows：{Disk}:¥Documents and Settings¥Current user¥Application Data¥4D¥DatabaseName_Address
Current userとは、カレントウインドウズセッションを開いたユーザ名を指します。

■ Macintosh：{Disk}:Library:Application Support:4D:DatabaseName_Address:

例題：

シングルユーザデータベースの起動中に、4Dフォルダの中に配置されたファイル内にユーザ自身が設定した内容をロード（もしくは作成）したいとします。これを実行するには、「On Startup」データベースメソッド内に以下のようなコードを記述します。

```
MAP FILE TYPES("PREF";"PRF";"Preferences file")
        ` Map PREF MacOS file type to .PRF Windows file extension
⇒       $vsPrefDocName:=Get 4D folder+"MyPrefs"
                                                ` Build pathname to the Preferences file
        ` Check if the file exists
If (Test path name($vsPrefDocName+(".PRF"*Num(On Windows)))#Is a
                                                                document)
$vtPrefDocRef:=Create document($vsPrefDocName;"PREF") ` If not, create it
Else
        $vtPrefDocRef:=Open document($vsPrefDocName;"PREF") ` If so, open it
End if
If (OK=1)
        ` Process document contents
        CLOSE DOCUMENT($vtPrefDocRef)
Else
        ` Handle error
End if
```

参照

System folder、Temporary folder、Test path name

DATA SEGMENT LIST

DATA SEGMENT LIST({セグメント})

引数	タイプ		説明
セグメント	文字列配列	←	データベースのデータセグメントのロング名

説明

DATA SEGMENT LISTコマンドは、引数<セグメント>配列に現在使用しているデータベースのデータセグメントのロング名を代入します。

警告：このコマンドは、4D Client上では何も行いません。4D Client上にデータセグメントの一覧を表示したい場合、ストアードプロシージャを使ってデータセグメントリストを作成し、サーバマシン上の任意の変数内にそのリストを格納します。そして、GET PROCESS VARIABLEコマンドを使って、この変数の内容を取得します。

例題

1. 以下の例は、[ダイアログ]テーブルの“データセグメント情報”フォームにおいて、データセグメントの名前を持ったドロップダウンリストを表示したいとします。これを実行するには、“データセグメント情報”フォームのフォームメソッドに以下のコードを記述します。

```
` [ダイアログ];"データセグメント情報" フォームのフォームメソッド
Case of
	¥(Form event=On Load )
		` ...
		ARRAY STRING(255 ; asDataSegName ; 0)
		DATA SEGMENT LIST(asDataSegName)
		` ...
End case
```

2. 次のメソッドは、データベースが分割されているかどうかをユーザに知らせます。

```
` データベースが分割されているかどうか → ブール
C_BOOLEAN($0)
DATA SEGMENT LIST($asDataSegName)
$0:=(Size of array($asDataSegName)>1)
```

3. 次の例は、**DATA SEGMENT LIST**コマンドを呼び出した後、新しくセグメントを追加したいかどうかを検査します。

```
DATA SEGMENT LIST(($asBefore)
```

```
ADD DATA SEGMENT
DATA SEGMENT LIST($asAfter)
If(Size of array($asBefore) # Size of array($asAfter))
        `データセグメントが多くある場合
Else
        ` セグメントの数が同じ場合
End if
```

参照

Application type、Application version

ADD DATA SEGMENT

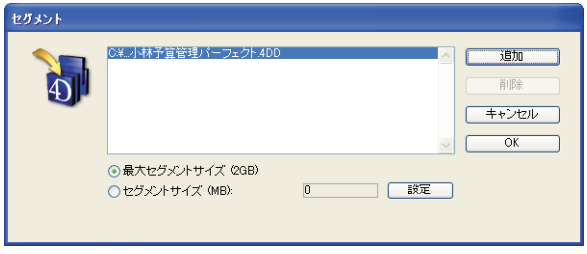
ADD DATA SEGMENT

引数	タイプ	説明

このコマンドには、引数はありません。

説明

このコマンドは、「データセグメント管理」ダイアログボックスを表示します。



「終了」ボタンをクリックしてダイアログボックスを有効にすると、システム変数OKに1が設定されます。「キャンセル」ボタンをクリックすると、システム変数OKに0が設定されます。

注：4D Serverでは、このコマンドは動作しません。

アプリケーションの環境設定（“データベース→データ管理”ページ）で「必要に応じて新しいデータセグメントを作成する」のオプションを選択した場合、データセグメントがフルになると、4th Dimentisonまたは4D Serverは自動的に新しいセグメントを作成します。

このオプションが選択されていない場合は、エラー-9999が発生します。エラーメッセージが表示され、ディスクに空きがないことが伝えられます。

4th Dimensionを使っている場合は、エラーをプロシージャで処理できるように**ON ERR CALL**コマンドを使用して、このエラーメッセージをトラップすることができます。その後で、**ADD DATA SEGMENT**コマンドを使うようにすれば、使用可能な別の空きボリューム上に新しいデータセグメントを追加することができます。

4D Serverを使っている場合、データベース管理者がサーバマシンから新しいデータセグメントを追加する必要があることを示す警告を表示できます。

参照

ON ERR CALL

システム変数またはシステムセット

「データセグメント管理」ダイアログボックスを有効にすると、システム変数OKには1が代入されます。

FLUSH BUFFERS

FLUSH BUFFERS

引数	タイプ	説明

このコマンドには、引数はありません。

説明

FLUSH BUFFERSコマンドを実行すると、即座にデータバッファの内容をディスクに保存します。データベースへのすべての変更をディスクに保存します。

通常、4th Dimensionがデータの変更内容を定期的に保存するため、このコマンドを呼び出す必要はありません。環境設定（「データベース→データ管理」ページ）の「データベースキャッシュ設定」プロパティでは、保存頻度の指定ができます。また、通常このプロパティを使用してバッファフラッシュをコントロールします。

注：4Dには、I/O操作を高速化するためのビルトインデータキャッシュスキームが統合されています。データキャッシュにはデータの変更が存在するのに、ディスク上にはない場合、プログラムは見えない状態になっています。例えば、QUERYを呼び出した場合、4Dのデータベースエンジンは、データキャッシュをクエリ操作に統合します。

OPEN DATA FILE

OPEN DATA FILE(アクセスパス)

引数	タイプ		説明
アクセスパス	文字列	←	開こうとするデータベースの名前、または完全なアクセスパス

説明

OPEN DATA FILEコマンドは、4Dアプリケーションによって開かれたデータファイルをオンザフライで変更します。

引数<アクセスパス>には、開こうとするデータファイルの名前、または完全なアクセスパスを渡します。ファイル名だけを渡す場合、データファイルはデータベースのストラクチャファイルと同じ階層に配置されていなければなりません。

アクセスパスが有効なデータファイルを指している場合、4Dは現在使用しているデータベースを終了した後、指定されたデータファイルを使って再度オープンします。データベースメソッドの「On Exit」と「On Startup」が続けて呼び出されます。

警告：このコマンドを使用すると、指定されたデータファイルで再びデータベースが開かれる前に、アプリケーションは一度終了します。このため、「On Startup」データベースメソッドやこのデータベースメソッドによって呼び出されたメソッド内で、このコマンドを使用することはできません。

このコマンドは非同期的に実行されます。つまり、コマンド呼び出しの後、4Dはメソッドの残りの部分を続けて実行します。この後、アプリケーションは「ファイル」メニューの「終了」コマンドが選択された場合と同様の処理を行います。表示されているダイアログボックスはキャンセルされ、実行中のプロセスは10秒間の猶予の後に打ち切られます。

処理を開始する前に、コマンドは指定されたデータファイルが有効かどうかを検証します。つまり、ファイルの拡張子がWindowsでは“.4dd”、Mac OSではファイルタイプが“dat5”でなくてはなりません。また、そのファイルが既にオープンされている場合、コマンドはファイルが現在使用中のストラクチャファイルに対応するかどうかを確認します。

<アクセスパス>に空の文字列を渡した場合、このコマンドはデータファイルを変更せずにデータベースを再度開きます。

4D Server：4D Clientや4D Serverではこのコマンドは使用できません。

参照

CREATE DATA FILE

CREATE DATA FILE

CREATE DATA FILE(アクセスパス)

引数	タイプ		説明
アクセスパス	文字列	←	作成するデータベースの名前、 または完全なアクセスパス

説明

CREATE DATA FILEコマンドは、オンザフライで新しいデータファイルをディスク上に作成し、4Dアプリケーションで開かれているデータファイルと置き換えます。

このコマンドの全般的な動作は、**OPEN DATA FILE**コマンドと同じです。唯一の相違点は、ストラクチャファイルを再オープンした後に、引数<アクセスパス>で設定された新しいデータファイルを作成するところです。

処理を開始する前に、コマンドは指定されたアクセスパスが既存のファイルに該当していないかどうかを調べます。

4D Server：4D Clientや4D Serverではこのコマンドは使用できません。

参照

OPEN DATA FILE

QUIT 4D

QUIT 4D{(time)}

引数	タイプ	説明
time	数値	4D Server終了までの時間

説明

QUIT 4Dコマンドは、4th Dimensionもしくは4D Client、4D Serverを終了してデスクトップに戻ります。

コマンドが実行されたアプリケーションに応じ、このコマンドは異なる処理を行います。

4th Dimensionおよび4D Clientで実行された場合

QUIT 4Dコマンドを呼び出すと、カレントプロセスはそれ自身の実行を中止して、4th Dimensionは次の動作を行います。

■ 「**On Exit**」データベースメソッドがある場合には、4Dは新しく作成されたローカルプロセス内でこのメソッドの実行を開始します。例えば、このデータベースメソッドを使用して、インタープロセスコミュニケーションを介して他のプロセスに（データ入力の）終了や「On Startup」データベースメソッドによって開始した操作（4D から他のデータベースサーバへの接続）の実行を中止するように通知することができます。4Dは、いずれ終了してしまうことを覚えておいてください。「On Exit」データベースメソッドは、必要なすべてのクリーンアップ操作や終了を実行しますが、quitを拒否することはできないため、ある時点でデータベースは終了します。

■ 「**On Exit**」データベースメソッドがない場合には、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。

ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。

現在開いているウインドウ上で行われたデータ入力の変更内容をユーザに保存させたい場合は、プロセス間通信を使って、データベースが終了されようとしていることを他のすべてのユーザプロセスに合図することができます。これを実行するには、以下の2つの方法があります。

■ **QUIT 4D**コマンドを呼び出す前に、カレントプロセスの中から上記の操作を実行する

■ 「On Exit」データベースメソッドの中から上記の操作を取り扱う。

3番目の方法もあります。**QUIT 4D**コマンドを呼び出す前に、ウインドウが妥当性検査を必要とするかどうかをチェックします。もし、このケースの場合は、ユーザにこのウインドウを有効にするか、または取り消すかを尋ね、それから再度、「終了」を選択するか尋ねます。しかし、ユーザインタフェースの観点から見ると、最初の2つの方法を使用するのをお勧めします。

注：4th Dimensionや4D Clientでは、引数＜時間＞は使用することができません。

4D Server（ストアドプロシージャ）で実行された場合

QUIT 4Dコマンドは、サーバマシン上のストアドプロシージャで実行できるようになりました。その場合、オプションのパラメータ＜時間＞を受け付けます。

引数＜時間＞により、アプリケーションが実際に終了するまでの4D Serverのタイムアウトを設定し、クライアントマシンが接続を切るための時間を与えることができます。＜時間＞には分単位の値を渡します。この引数は、サーバマシン上で実行された際にのみ考慮されます。4D Clientや4th Dimensionでは、この引数は無視されます。

引数＜時間＞を渡さない場合、終了する前に4D Serverはすべてのクライアントマシンが接続を切るまで待機します。

4D Clientや4th Dimensionとは異なり、4D Serverによる**QUIT 4D**の処理は非同期的に行われます。つまり、このコマンドの実行後、コマンドの呼び出しを行ったメソッドが中断されることはありません。

「On server stop」データベースメソッドが存在する場合、指定した引数に応じて、引数＜時間＞で設定した遅延時間後、またはすべてのクライアント接続が切断された後にメソッドが実行されます。

ストアドプロシージャ内で使用された場合の**QUIT 4D**コマンドの働きは、4D Serverの「ファイル」メニューから「終了」コマンドを選択した場合と同じです。すなわち、各クライアントマシンにダイアログボックスを表示して、サーバが終了することを通知します。

例題

▼以下の例は、「ファイル」メニューに「終了」メニューを持ったプロジェクトメソッドを示しています。

```
`「M_終了」プロジェクトメソッド
CONFIRM("終了してもよろしいですか？")
If(OK=1)
      QUIT 4D
End if
```

参照

On Exit Database Method、On Server Shutdown Database Method

SELECT LOG FILE

SELECT LOG FILE(ログファイル | *)

引数	タイプ	説明
ログファイル名	文字列	ログファイルの名前
*	*	カレントログファイルを閉じる場合は“*”

説明

このコマンドは、引数<ログファイル名>で指定されたログファイルのオープン、作成、クローズを行います。

注：このコマンドは、「ユーザ」モードの「ファイル」メニューから「ログファイル...」アイテムを選択するのと同じです。

<ログファイル>には、開こうとしているログファイルもしくは作成するログファイルのフルパス名を渡します。ファイル名のみが渡された場合、そのファイルが検索されるか、データベースのストラクチャファイルと同じ階層にファイルが作成されます。

<ログファイル名>が空の文字列の場合、**SELECT LOG FILE**コマンドは「ファイルオープン」ダイアログボックスを表示し、ログファイルを開くか、新しく作成することができます。ユーザが「開く」ボタンをクリックし、ログファイルが正常に開かれると、システム変数OKに1が設定されます。それ以外の場合、つまりユーザが「キャンセル」ボタンをクリック、またはログファイルのオープンやクローズが行えない場合は、システム変数OKに0が設定されます。

<ログファイル名>が“*”の場合、**SELECT LOG FILE**コマンドはカレントログファイルを閉じます。ログファイルが閉じられると、システム変数OKに0が設定されます。

フルバックアップがまだ行われていない場合やデータファイルに既にレコードが含まれている場合に**SELECT LOG FILE**コマンドを使って、ログファイルを作成、または開くと警告メッセージが表示され、エラーコード - 4447が返されます。

参照

ON ERR CALL

システム変数またはシステムセット

ログファイルが正しくオープン、作成、クローズされた場合、システム変数OKには1が代入されます。

エラーの処理

操作が実行されない場合、バックアップが必要になるため、エラー-4447が発生します。エラーはON ERR CALLメソッドを使用して独自のエラー処理を記述することができます。

GET SERIAL INFORMATION

GET SERIAL INFORMATION(キー; ユーザ名; 会社名; 接続ユーザ; 最大ユーザ)

引数	タイプ		説明
キー	倍長整数	←	（暗号化された）製品独自のキー
ユーザ名	文字列	←	登録名
会社名	文字列	←	登録組織名
接続ユーザ	倍長整数	←	接続ユーザ数
最大ユーザ	倍長整数	←	最大接続ユーザ数

説明

GET SERIAL INFORMATIONコマンドは、4Dのカレントバージョンのシリアル番号に関する各種情報を返します。

■ **キー：**インストールされた製品のユニークなID。固有の番号がマシンにインストールされた4Dアプリケーション（4D Server、4th Dimension、4D Runtime等）に関連付けられます。もちろん、この番号は暗号化されています。

■ **ユーザ名：**インストール時に定義された、アプリケーションのユーザ名

■ **会社名：**インストール時に定義された、ユーザの会社名または組織名

■ **接続ユーザ：**コマンド実行時の接続ユーザ数

■ **最大ユーザ：**同時接続した最大ユーザ数

注：最後の2つの引数は、シングルユーザの4Dでは常に1を返し、例外としてデモバージョンでは0が返されます。

GET SERIAL INFORMATIONコマンドは、バージョン6.7以降の4Dで採用された包括的なコンポーネント保護機構の一部です（コンポーネントに関する詳細は、『4D Insiderリファレンス』を参照してください）。コンポーネントの開発者は、違法コピーを防ぐため、自分の製品のコピーとインストールした所定の4Dアプリケーションとを関連付けることができます。

コンポーネントを必要とするユーザは、**GET SERIAL INFORMATION**コマンドが返す"key"値をコンポーネントのデベロッパーへ送ります（そのためのオーダーフォームがコンポーネントのデモ版パッケージに含まれているかもしれません）。前述の"key"値はユニークで特定の4D製品とだけ関連づけられています。

コンポーネントデベロッパーは、ユーザから送られてきた"key"値と何らかの暗号値を組み合わせて独自のシリアル番号を生成し、その番号をコンポーネントを購入したいユーザへ送ります。配付してあるコンポーネント（デモ版）にはデベロッパーより発行されたシリアル番号と**GET SERIAL INFORMATION**コマンドが返す"key"値の適合性を検査する機能を盛り込んでおきます。この機能を実行し入力したシリアル番号が正しいものであればコンポーネントの機能を利用可能にします。

注：プラグインの開発者も、この保護機構を使用することができます。詳細は、『4D Plugin APIリファレンス』を参照してください。

参照

Get component resource ID

Is license available

Is license available{(license)}→ブール値

引数	タイプ		説明
license	数値	→	ライセンスの有効性テストを行う製品／プラグイン
戻り値	ブール値	←	製品／プラグインが、使用可能な場合はTrue、そうでない場合はFalse

説明

このコマンドを使って、製品やプラグインの利用可能性を確認できます。例として、プラグインが必要な機能を表示または非表示にする際に有効です。

Is license availableコマンドは次のような3通りの使用が可能です。

■ license引数を省略する場合：4Dアプリケーションがデモモードの場合に、コマンドはFalseを返します。

■ 以下の“Is license available”テーマの定数のうちの1つをlicense引数にパスする場合。

定数	タイプ	値
4D Draw License	倍長整数	808464694
4D for OCI License	倍長整数	808465208
4D View License	倍長整数	808465207
4D Web License	倍長整数	808464945
4D Write License	倍長整数	808464697
4D Client Web License	倍長整数	808465209
4D Client SOAP License	倍長整数	808465465
4D SOAP License	倍長整数	808465464
4D ODBC Pro License	倍長整数	808464946
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715

このとき、対応する製品がロードされライセンスが使用可能な場合（4D Serverで）に、コマンドはTrueを返します。

例として、4D Drawのシリアルナンバーがありexpansionナンバーがない場合は、コマンドは4th Dimensionのシングルユーザシステムではtrueを、4D ServerではFalseを返します。反対に、4D Drawのexpansionナンバーがありシリアルナンバーがない場合は、コマンドは4D ServerではTrueを、4th DimensionではFalseを返します。シリアルナンバーおよびexpansionナンバーのどちらもある場合は、コマンドはいかなる場合もTrueを返します。

■ プラグイン“4BNX”リソースのID番号をlicense引数に直接パスする場合、コマンドは、前述の通りに動作します。

OPEN 4D PREFERENCES

OPEN 4D PREFERENCES(セレクタ)

引数	タイプ		説明
セレクタ	文字列	→	「環境設定」ダイアログボックスのテーマ、ページ、パラメータグループを指定するキー

説明

OPEN 4D PREFERENCESコマンドは、現行の4th Dimensionアプリケーションの「環境設定」ダイアログボックスを表示し、＜セレクタ＞に渡されたキーに対応するテーマやページを表示します。

引数＜セレクタ＞には、「環境設定」ダイアログボックスのテーマやページ、パラメータグループを示す“キー”を1つ以上納めなくてはなりません。これらキーの一覧は、この節の最後に記載されています。

固定アクセスパスまたは単独項目の名前のいずれかを渡すことができます。

■ **固定アクセスパス：**引数＜セレクタ＞は次の方法で組み立てることができます。

/テーマ{/ページ{/パラメータグループ}}

文字列は必ず“/”記号で開始し、各レベルを“/”で区切らなくてはなりません。

例えば、「デザインモード」ページの「コンパイラ」ページを設定する場合、引数＜セレクタ＞には“/Design Mode/Compiler”を代入します。

■ **名前（相対パス）：**この場合、引数＜セレクタ＞は“/”記号で開始しません。目的の項目の名前を渡すだけで、4th Dimensionは「パラメータグループ→ページ→テーマ」という検索順に従い、最初に一致する項目を開きます。

例えば、＜セレクタ＞に“Progress indicator”を渡した場合、4th Dimensionは「アプリケーション」テーマの「オプション」ページを開きます。

ダイアログボックスの最初のページを直接開くには、＜セレクタ＞に“/”を渡します。

コマンドは、＜セレクタ＞に指定された項目に関する「環境設定」ページを開きますが、その他すべてのテーマやページにもアクセスすることができます。ユーザが「環境設定」にアクセスすることにより、アプリケーションの妨げとならないよう確認することは開発者の責任です。ユーザ動作を制御するために、ユーザアクセス管理システムを有効にすることをお勧めします。

パスのキー

引数<セレクタ>として使用できるキーの一覧を次に示します。

/Application（アプリケーション）
/Application/Options（アプリケーション/オプション）
/Application/Options/Options（アプリケーション/オプション/オプション）
/Application/Options/Temporary Folder Location（アプリケーション/オプション/テンポラリフォルダの場所）
/Application/Access（アプリケーション/アクセス管理）
/Application/Access/Data Access（アプリケーション/アクセス管理/データアクセス権）
/Application/Access/User Access（アプリケーション/アクセス管理/ユーザアクセス）
/Application/CPU Priorities（アプリケーション/CPUの優先度）
/Application/CPU Priorities/Set CPU Priority to:（アプリケーション/CPUの優先度/他のアプリケーション）
/Application/Shortcuts（アプリケーション/ショートカット）
/Application/Shortcuts/Keys（アプリケーション/ショートカット/編集）
/Application/Compatibility（アプリケーション/互換）
/Application/Compatibility/Structure Compatibility（アプリケーション/互換/ストラクチャの互換性）
/Application/Compatibility/Web Compatibility（アプリケーション/互換/Web互換性）
/Application/Compatibility/Platform（アプリケーション/互換/プラットフォーム）
/Design Mode（デザインモード）
/Design Mode/Structure（デザインモード/ストラクチャ）
/Design Mode/Structure/General Font（デザインモード/ストラクチャ/通常フォント）
/Design Mode/Structure/Forms and Methods Automatic Comments（デザインモード/ストラクチャ/フォーム／メソッド用自動コメント）
/Design Mode/Form Editor（デザインモード/フォームエディタ）
/Design Mode/Form Editor/Object Templates（デザインモード/フォームエディタ/オブジェクトテンプレート）
/Design Mode/Form Editor/Move（デザインモード/フォームエディタ/移動）
/Design Mode/Form Editor/Auto Alignment（デザインモード/フォームエディタ/自動整列）
/Design Mode/Form Editor/Default Display（デザインモード/フォームエディタ/デフォルト表示）
/Design Mode/Method Editor（デザインモード/メソッドエディタ）
/Design Mode/Method Editor/Font（デザインモード/メソッドエディタ/フォント）
/Design Mode/Method Editor/Default Display（デザインモード/メソッドエディタ/デフォルト表示）
/Design Mode/Method Editor/Options（デザインモード/メソッドエディタ/オプション）
/Design Mode/Method Editor/Structure Style-Sheets（デザインモード/メソッドエディタ/ストラクチャスタイルシート）
/Design Mode/Compiler（デザインモード/コンパイラ）

/Design Mode/Compiler/Compilation Options（デザインモード/コンパイラ/コンパイルオプション）
/Design Mode/Compiler/Compiler Methods for...（デザインモード/コンパイラ/コンパイラメソッド...）
/Design Mode/Documentation（デザインモード/ドキュメント）
/Design Mode/Documentation/Documentation Access from the Explorer（デザインモード/ドキュメント/エクスプローラからのドキュメントアクセス）
/Database（データベース）
/Database/Data Management（データベース/データ管理）
/Database/Data Management/General（データベース/データ管理/一般設定）
/Database/Data Management/Database Cache Settings（データベース/データ管理/データベースキャッシュ設定）
/Database/Data Management/WEDD（データベース/データ管理/WEDD）
/Database/Script Manager（データベース/スクリプトマネージャ）
/Database/Script Manager/Script Manager（データベース/スクリプトマネージャ/スクリプトマネージャ）
/Backup（バックアップ）
/Backup/Configuration（バックアップ/設定）
/Backup/Configuration/Backup Contents（バックアップ/設定/バックアップの内容）
/Backup/Configuration/Backup File Destination Folder（バックアップ/設定/バックアップファイル保存先フォルダ）
/Backup/Configuration/Last Backup Information（バックアップ/設定/最後のバックアップ情報）
/Backup/Configuration/Log Management（バックアップ/設定/ログ管理）
/Backup/Scheduler（バックアップ/スケジューラ）
/Backup/Scheduler/Backup Frequency（バックアップ/スケジューラ/バックアップスケジュール）
/Backup/Backup（バックアップ/バックアップ）
/Backup/Backup/General（バックアップ/バックアップ/一般設定）
/Backup/Backup/Archive（バックアップ/バックアップ/アーカイブ）
/Backup/Restore（バックアップ/復元）
/Client-Server（クライアント-サーバ）
/Client-Server/Configuration（クライアント-サーバ/設定）
/Client-Server/Configuration/Network（クライアント-サーバ/設定/ネットワーク）
/Client-Server/Configuration/Client-Server Connections Timeout（クライアント-サーバ/設定/クライアント-サーバ接続タイムアウト）
/Client-Server/Configuration/Client-Server Communication（クライアント-サーバ/設定/クライアント-サーバ通信）
/Client-Server/Configuration/4D Open（クライアント-サーバ/設定/4D Open）
/Client-Server/Publishing（クライアント-サーバ/公開）
/Client-Server/Publishing/Publishing Information（クライアント-サーバ/公開/公開情報）

/Client-Server/Publishing/Allow-Deny Table Configuration（クライアント-サーバ/公開/許可-拒否テーブル設定）
/Client-Server/Publishing/Encryption（クライアント-サーバ/公開/暗号化）
/Web（Web）
/Web/Configuration（Web/設定）
/Web/Configuration/Web Server Publishing（Web/設定/Webサーバ公開時オプション）
/Web/Configuration/Default HTML Path（Web/設定/デフォルトHTMLパス）
/Web/Configuration/Starting Mode（Web/設定/開始時のモード）
/Web/Advanced（Web/詳細）
/Web/Advanced/Cache（Web/詳細/キャッシュ）
/Web/Advanced/Web Process（Web/詳細/Webプロセス）
/Web/Advanced/Options（Web/詳細/オプション）
/Web/Advanced/Web Passwords（Web/詳細/Webパスワード）
/Web/Options（Web/オプション）
/Web/Options/Text Conversion（Web/オプション/テキスト変換）
/Web/Options/4D WebSTAR（Web/オプション/4D WebSTAR）
/Web/Options/Persisent Connections（Web/オプション/keep Alive接続）
/Web Services（Webサービス）
/Web Services/SOAP（Webサービス/SOAP）
/Web Services/SOAP/Server Side（Webサービス/SOAP/サーバ側設定）
/Web Services/SOAP/Client Side（Webサービス/SOAP/クライアント側設定）

例題

1. 「環境設定」の最初のページを開く

```
OPEN 4D PREFERENCES("/")
```

2. 「アプリケーション」テーマの「ショートカット」ページを開く

```
OPEN 4D PREFERENCES("/Application/Shortcuts")
```

3. 「デザインモード」テーマの「メソッドエディタ」ページを開く

```
OPEN 4D PREFERENCES("/Design Mode/Method Editor")
```

システム変数またはセット

要求した項目が見つからない場合や、正常に開けなかった場合は、システム変数OKは1を返し、それ以外の場合は0を返します。

配列

配列とは、同じタイプの変数を番号付きで並べたものです。各変数は、配列の要素といいます。配列のサイズとは、それが持つ要素の数を指します。配列は作成時にサイズが与えられ、要素の追加、挿入、削除によって、またはそれを作成するのに使用したのと同じコマンドを使用して配列のサイズを変更すれば、何度でもサイズを変更することができます。

配列は、配列宣言コマンドのいずれかを使用して作成します。詳細については、次節の「配列を作成する」を参照してください。

要素には、1からNの番号が付けられ、Nは配列のサイズとなります。配列には、常に要素ゼロがあります。この要素は他の要素と同様にアクセスできますが、配列をフォームに表示しても表示されません。要素ゼロは、配列がフォームオブジェクトをサポートする場合には表示されませんが、プログラミング言語で使用する際には制限がありません。要素ゼロについての詳細は、後述の「配列の要素ゼロを使用する」の節を参照してください。

配列は、4Dの変数です。他の変数と同様、配列にもスコープがあり、4D言語の規則に従いますが、他と異なるところがいくつかあります。詳細については、後述の「配列と4D言語」および「配列とポインタ」の節を参照してください。配列は言語のオブジェクトですので、画面に絶対表示されない配列を作成、使用することができます。

配列はまた、ユーザインタフェースオブジェクトでもあります。配列とフォームオブジェクトの間の相互作用についての詳細は、「配列とフォームオブジェクト」および「グループ化されたスクロールエリア」の節を参照してください。配列は短時間に妥当な量のデータを保持するように設計されています。配列はメモリ内に保持されるため、取り扱いやすく、高速に操作できます。詳細については、「配列とメモリ」の節を参照してください。

配列を作成する

配列は、この章で説明する配列宣言コマンドのいずれかを使用して作成します。次の表に、配列宣言コマンドを記載します。

コマンド	配列の作成またはサイズ変更
ARRAY INTEGER	2バイト整数値
ARRAY LONGINT	4バイト整数値
ARRAY REAL	実数値
ARRAY TEXT	テキスト値（1要素当たり0から32000バイト）（参照：備考）
ARRAY STRING	文字列値（1要素当たり0から32000バイト）（参照：備考）
ARRAY DATE	日付値
ARRAY BOOLEAN	ブール値
ARRAY PICTURE	ピクチャ値
ARRAY POINTER	ポインタ値

配列宣言コマンドは、1次元または2次元の配列の作成やサイズ変更が可能です。

2次元配列についての詳細は、「2次元配列」の節を参照してください。

logint配列により時間タイプのデータを操作することができます。フォーム中で時間配列を表示するためには、関係付けられたフォームオブジェクトに表示フォーマット&/xを適用します。このxは時間フォーマットリスト中のフォーマットの番号を表しています（表示順）。例えば、&/4は時分フォーマットで表示します。

備考：テキスト配列と文字列配列の違いは、要素の性質にあります。どちらのタイプの配列も、要素がテキスト値（文字）を持つことができます。しかし、以下の点が異なります。

■ テキスト配列では、各要素は可変長で、その文字はメモリの異なる部分に格納されます。

■ 文字列配列では、すべての要素が固定長です（配列の作成時に長さが渡されます）。要素はすべて、その内容に関わらずメモリの同じ部分に順番に格納されます。

この構造上の相違のため、文字列配列はテキスト配列より高速に動作します。ただし、文字列配列は最高255バイトしか保持できないため注意してください。

次のコードは、10個の要素からなる整数配列を作成（宣言）します。

```
ARRAY INTEGER(aiAnArray ; 10)
```

次のコードは、同じ配列の要素を20個にします。

```
ARRAY INTEGER(aiAnArray ; 20)
```

次のコードは、同じ配列の要素をゼロにします。

```
ARRAY INTEGER(aiAnArray ; 0)
```

配列中の要素は中カッコ（{}）を使用して参照します。中カッコの中には数字を入れて特定の要素を指定します。この数字を要素番号といいます。次の行は、5つの名前をatNamesという配列に入れ、それらを警告ウインドウに表示します。

```
ARRAY TEXT(atNames ; 5)
atNames{1} := "吉川"
atNames{2} := "伊藤"
atNames{3} := "安藤"
atNames{4} := "田中"
atNames{5} := "中村"
For($vlElem ; 1 ; 5)
    ALERT("配列要素 #"+String($vlElem)+"は"+atNames{$vlElem} + "です。")
End for
```

atNames{$vlElem}というシンタックスに注意してください。atNames{3}のような数値そのものを指定するのではなく、数値変数を使用して配列のどの要素を指定するかを示すことができます。ループ構造による反復を使用すると（**For**...**End for**、**Repeat**...**Until**または**While**...**End while**）、短いコードで配列のすべて、または一部のコードを指定することができます。

配列および4D言語のその他のエリア

この他にも配列の作成や処理を行う4Dコマンドがあります。特に次のようなコマンドです。

- 配列とレコードセレクションを操作するには、**SELECTION RANGE TO ARRAY**、**SELECTION TO ARRAY**、**ARRAY TO SELECTION**、**DISTINCT VALUES**コマンドを使用します。
- リストボックスタイプのオブジェクトは、配列に基づいています。いくつかの「リストボックス」テーマのコマンド（例えば**INSERT LISTBOX ROW**）は配列を処理します。
- テーブル、サブテーブルおよび配列に格納された値を元にグラフや図を作成することができます。詳細については、**GRAPH**コマンドを参照してください。
- バージョン6には階層リストを扱う新規コマンドが数多く用意されていますが、旧バージョンの**LIST TO ARRAY**コマンドと**ARRAY TO LIST**コマンドが互換性を保つために残されています。

■ コマンドの多くは、一度の呼び出しで配列を作成することができます。例えば、**FONT LIST**、**WINDOW LIST**、**VOLUME LIST**、**FOLDER LIST**、**DOCUMENT LIST**、**GET SERIAL PORT MAPPING**、**SAX GET XML ELEMENT**などのコマンドです。

参照

ARRAY BOOLEAN、ARRAY DATE、ARRAY INTEGER、ARRAY LONGINT、ARRAY PICTURE、ARRAY POINTER、ARRAY REAL、ARRAY STRING、ARRAY TEXT、Arrays、2次元配列

配列とフォームオブジェクト

配列は、プログラム言語のオブジェクトです。決して画面に表示されない配列を作成、使用することができます。しかし、配列は同時にユーザインタフェースでもあります。配列では、以下のタイプのフォームオブジェクトがサポートされます。

■ ポップアップメニュー／ドロップダウンリスト

■ コンボボックス

■ スクロールエリア

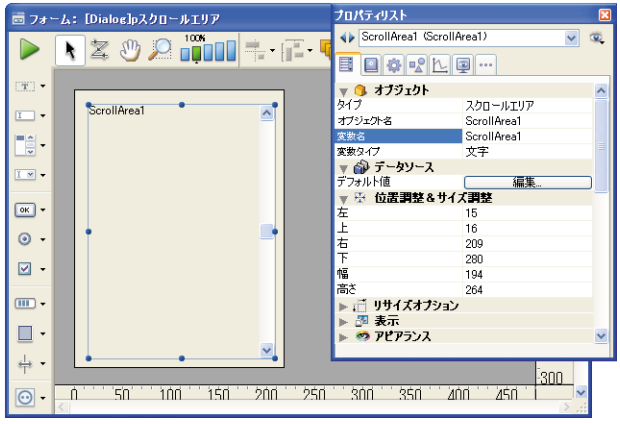
■ タブコントロール

■ リストボックス

これらのオブジェクトは、「デザイン」モードのフォームエディタで「プロパティリスト」ウインドウの「デフォルト値」ボタンを用いて、事前に定義することができますが（リストボックスを除く）、配列コマンドを使用してプログラムから定義することもできます。いずれの場合も、フォームオブジェクトは手動または4Dが作成した配列によってサポートされます。

これらのオブジェクトを使用するとき、その配列で選択された要素を調べることにより、オブジェクト内のどの項目が選択されているかを検出することができます。逆に、選択した要素を配列に設定して、オブジェクトの中の特定の項目を選択することができます。

配列がフォームオブジェクトのサポートに使用される場合、この配列は、ランゲージオブジェクトとユーザインタフェースの2つの性質を併せ持ちます。例えば、フォームを設計する場合、スクロールエリアを作成します。



関連付けられた変数の名前（この例ではatName）が、スクロールエリアの作成と処理に使用する配列名となります。

注：

・2次元配列やポインタ配列は表示できません。

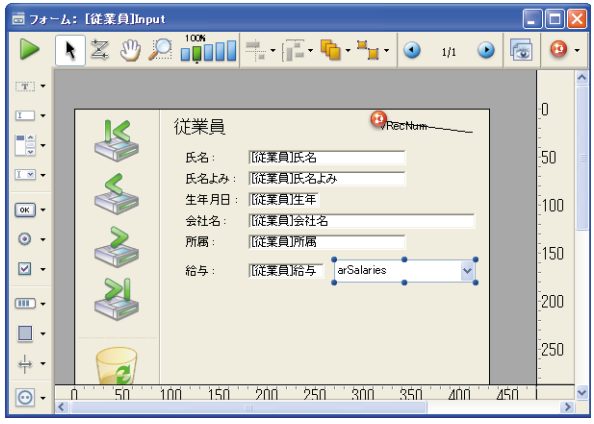
・リストボックスタイプのオブジェクト（複数の配列を含む場合があります）を管理する上で、特定の状況が数々存在します。詳細については、「リストボックスオブジェクトの管理」の節で説明しています。

例：ドロップダウンリストを作成する

次の例では、配列に値を格納し、それをドロップダウンリストに表示する方法を示します。配列arSalariesは、**ARRAY REAL**コマンドを使用して作成します。これには、社内で社員に支払われる基準給与がすべて入っています。ユーザがドロップダウンリストから要素を選択すると、[従業員]給与フィールドに「ユーザ」モードまたは「カスタムメニュー」モードで選択された値が割り当てられます。

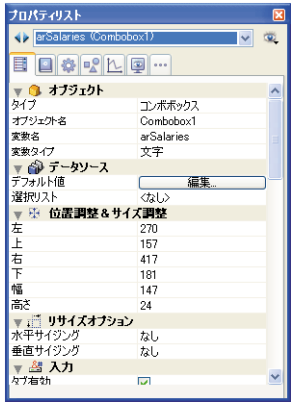
フォーム上で「arSalaries」ドロップダウンリストを作成する

ドロップダウンリストを作成し、それを“arSalaries”と名付けます。ドロップダウンリストの名前は、配列の名前と同じでなければなりません。

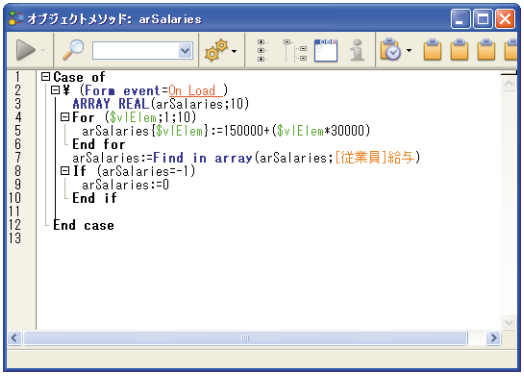


配列を初期化する

「On Load」イベントをオブジェクトに対して使用し、配列arSalariesを初期化します。これを行うには、次に示すように、必ず「プロパティリスト」ウインドウでこのイベントを有効にしてください。



「オブジェクトメソッド」ボタンをクリックして、次のようにメソッドを作成します。



次の行は、

```
ARRAY REAL(arSalaries;10)
For($vlElem;1;10)
        arSalaries{$vlElem}:=150000+($vlElem*30000)
End for
```

税引き前の給与に応じて、2500、3000 ...7000の数値配列を作成します。

次の行は、

```
arSalaries:=Find in array(arSalaries ; [従業員]給与)
If(arSalaries=-1)
        arSalaries:=0
End if
```

新規レコードの作成と既存レコードの修正の両方を処理します。

■ 新規レコードを作成する場合、最初は[従業員]給与フィールドはゼロです。この場合、**Find in array**関数は配列の中で値を検出できず、-1を返します。**If**(arSalaries=-1)という判定式により、arSalariesをゼロにリセットして、ドロップダウンリストで要素がまったく選択されていないことを示します。

■ 既存レコードを修正する場合、**Find in array**関数により配列内の値が取得されてドロップダウンリストの選択した要素をフィールドの現在の値に設定します。特定の従業員の値がリストにない場合、**If**(arSalaries=-1)という判定式により、リストの任意の要素の選択が解除されます。

注：配列選択要素についての詳細は、次の節を参照してください。

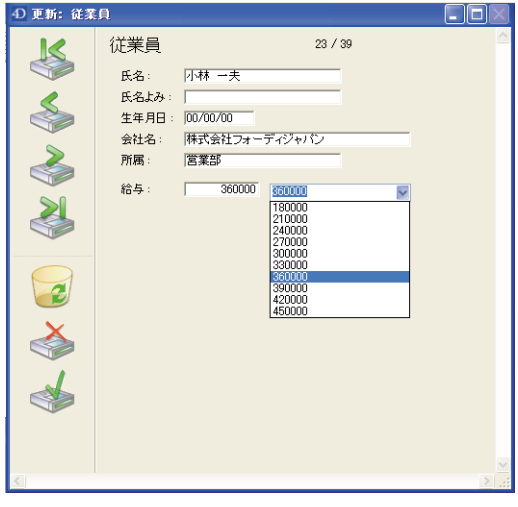
選択した値の[従業員]給与フィールドへのレポート

「arSalaries」ドロップダウンリストから選択した値をレポートするには、オブジェクトに対する「On Clicked」イベントを処理する必要があります。選択した要素の要素番号は、配列arSalaries自体の値です。したがって、式arSalaries{arSalaries}はドロップダウンリストで選択した値を返します。

「arSalaries」ドロップダウンリストのオブジェクトメソッドは、以下のようになります。

```
Case of
	¥(Form event=On Load )
		ARRAY REAL(arSalaries;10)
		For($vlElem;1;10)
			arSalaries{$vlElem}:=150000+($vlElem*30000)
		End for
		arSalaries:=Find in array(arSalaries;[従業員]給与)
		If(arSalaries=-1)
			arSalaries:=0
		End if

	¥(Form event=On Clicked)
		[従業員]給与:=arSalaries{arSalaries}

End case
```

「ユーザ」モードまたは「カスタムメニュー」モードでは、このドロップダウンリストは次のように表示されます。



次の節では、配列をフォームオブジェクトとして使用する際に配列に対して実行する一般的な基本操作を説明します。

配列のサイズを取得する

配列の現在のサイズは、**Size of array**コマンドを使用して取得することができます。前述の例を使用すると、次のコードは、5を表示します。

```
ALERT("atNames配列のサイズは"+String(Size of array(atNames)) +"です。")
```

配列の要素を並べ替える

単一レベルの並び替えには**SORT ARRAY**コマンドを、マルチレベルの並び替えには**MULTI SORT ARRAY**コマンドを使用することができます。前述の例を使用し、配列がスクロールエリアに表示されているとします。

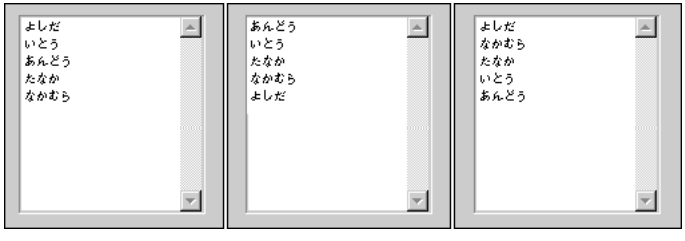
a. まず、エリアは左のリストのように表示されています。

b. 以下のコードを実行すると、エリアは中央のリストのように表示されます。

```
SORT ARRAY(atNames ; >)
```

c. コードの以下の行を実行すると、エリアが右のリストのように表示されます。

```
SORT ARRAY(atNames ; <)
```



要素の追加と削除

APPEND TO ARRAYコマンド、**INSERT ELEMENT**コマンドおよび**DELETE ELEMENT**コマンドを使用して、要素の追加、挿入、削除を実行できます。

配列内のクリックの処理：選択した要素のテスト

前述の例を使用し、配列がスクロールエリアに表示されているとすると、このエリアでのクリックは、以下のように処理できます。

```
`「atNames」スクロールエリアのオブジェクトメソッド
Case of
	¥(Form event=On Load)
		`配列を初期化する(前述の例を参照)
		ARRAY TEXT(atNames;5)
		`...
	¥(Form event=On Unload)
		`配列が必要なくなった場合、
		ARRAY TEXT(atNames;0)

	¥(Form event=On Clicked)
		If(atNames#0)
			ALERT(atNames{atNames} + "をクリックしました。")
		End if

	¥(Form event=On Double Clicked)
		If(atNames#0)
			ALERT(atNames{atNames}+"をダブルクリックしました。")
		End if
End case
```

注：各イベントは、そのオブジェクトのプロパティでアクティブにしなければなりません。

シンタックス「atNames{$vlElem}」は、特定の配列要素を使用して作業を行い、シンタックス「atNames」は、配列内で選択した要素の要素番号を返します。したがって、シンタックス「atNames{atNames}」は配列atNames内で選択した要素の値を意味します。

要素が選択されていない場合、atNamesはゼロになるため、「If(atNames#0)」という判定式は、要素が実際に選択されているかどうかを調べます。

選択した要素を設定する

同様に、配列に値を割り当てることにより、プログラムから選択した要素を変更することができます。

```
`最初の要素を選択する(配列要素が1つ以上)
atNames:=1

`最後の要素を選択する(配列要素が1つ以上)
atNames:=Size of array(atNames)
```

```
` 選択した要素を選択解除する
atNames:=0

If((0<atNames)&(atNames<Size of array(atNames))
    ` 可能なら、選択した要素の次の要素を選択する
    atNames:=atNames+1
End if

If(1<atNames)
    ` 可能なら、選択した要素の1つ前の要素を選択する
    atNames:=atNames-1
End if
```

配列内の値を検索する

Find in array関数は、配列内の特定の値を検索します。前述の例を使用すると、次のコードは、「リクエスト」ダイアログボックスに“田中”と入力した場合に、その値が“田中”である要素を選択します。

```
$vsName:=Request("名字を入力してください：")
    If(OK=1)
        $vlElem:=Find in array(atNames ; $vsName)
        If($vlElem>0)
            atNames:=$vlElem
        Else
            ALERT($vsName+"という名字の人はいません。")
        End if
    End if
```

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは、通常同じ方法で扱われます。もちろん、要素の値を変更、要素を追加、あるいは削除するたびに別のコードを作成して、画面のオブジェクトを再描画する必要はありません。

注：アイコンと有効、無効なタブによってタブコントロールを作成、使用するには、タブコントロールの支援オブジェクトとして階層リストを使用する必要があります。詳細については、New list関数の例を参照してください。

コンボボックスの扱い

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは前節で説明したアルゴリズムによって制御できますが、コンボボックスの扱いは、これらとは異なります。

コンボボックスは、実際は、値のリスト（配列の要素）を添付されたテキスト入力可能なエリアです。ユーザはこのリストから値を取り出し、テキストを編集することができます。したがって、コンボボックスでは、選択した要素という概念は適用されません。

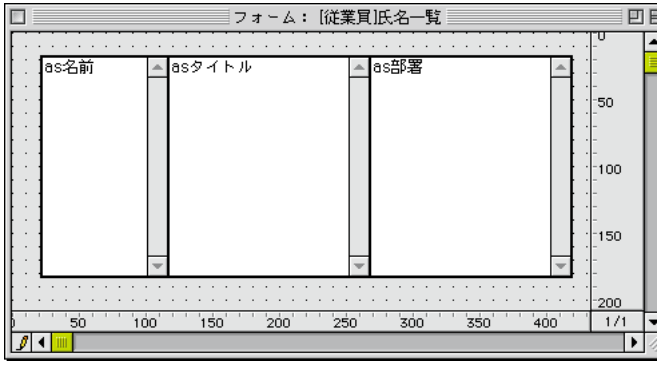
コンボボックスには、選択された要素というものはありません。ユーザがエリアに添付された値のいずれかを選択するたびに、その値が配列の要素ゼロに入ります。次にユーザがテキストを編集すると、ユーザが変更した値が要素ゼロに入ります。

```
`「asColors」コンボボックスのオブジェクトメソッド
Case of
	¥(Form event=On Load)
		ARRAY STRING(31 ; asColors ; 3)
		asColors{1}:="青"
		asColors{2}:="白"
		asColors{3}:="赤"
	¥(Form event=On Clicked)
		If(asColors{0}#"")
			`オブジェクトは自動的にそれの値を変更します
			`コンボボックスでの「On Clicked」イベントの使用は、
			`さらに処理を行わなければならない時にのみ必要となります
		End if
	¥(Form event=On Data Change)
		`Find in array関数は要素ゼロを無視し-1または0以上の値を返します
		If(Find in array(asColors ; asColors{0})<0)
			` 入力した値がオブジェクトに付属した値の1つではない場合
			` 次回のために、そのリストに値を追加する
			APPEND TO ARRAY(asColors;asColors{0})
		Else
			` 入力した値がオブジェクトに付属の値の中にある場合
		End if
End case
```

グループ化されたスクロールエリア

互換性に関する注意：4th Dimensionでは、依然としてグループ化されたスクロールエリアを利用することができます。しかし、バージョン2004以降、これらはリストボックスタイプのオブジェクトに置き換えることができます。この件に関する詳細は、「リストボックス概要」の節を参照してください。

スクロールエリアは、グループ化してフォームに表示することができます。複数のスクロールエリアをグループ化すると、それらは1つのスクロールエリアとして動作します。それぞれのスクロールエリアには独自のフォントとフォントサイズが指定できますが、各カラムのフォントの高さ（フォントとフォントサイズに依存します）は同一にすることをお勧めします。データ入力中に表示する場合、一番前にあるスクロールエリアがスクロールバーを表示します。次の図は、「デザイン」モードにおいて1つにグループ化されたスクロールエリアです。



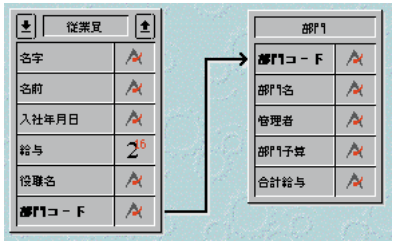
以下にグループ化されたスクロールエリアを作成する際のヒントを示します。

- すべての配列が同じサイズ（同じ要素数）になっているかを確認する。
- 各スクロールエリアに対して、すべて同じ文字サイズ（ポイント数）を使用する。
- 各スクロールエリアを同じ高さにする。
- 各スクロールエリアの一番上と一番下を揃える。
- 隣接するスクロールエリアが互いに重ならないようにする。
- 一番右側のスクロールエリアを前面に移動する。これは、最前面のスクロールエリアにスクロールバーが表示されるためである。
- 各スクロールエリアに対する単独の操作を終了してから、すべてのエリアを選択し、「設定」メニューから「グループ」を選択する。

次のメソッドは、3つの配列にデータを格納し、画面上に表示します。

```
ALL RECORDS([従業員])
SELECTION TO ARRAY([従業員]名字 ; asName ; [従業員]役職名 ; asTitle
                                                    ;[部門]部門名 ; asDepartment)
DIALOG([部門];"グループ化")
```

このメソッドは、[従業員]テーブルと[部門]テーブルのフィールドにあるデータを使用します。これらのテーブルを次に示します。



注：[部門]テーブルは、[従業員]から[部門]への自動リレーションが存在する場合に使用できます。

結果は、次の通りです。



スクロールバーが1つしか表示されていないことに注意してください。スクロールバーは、常に一番前のスクロールエリアに表示されます。このスクロールバーは、3つの配列をあたかも1つの配列のように制御します。ユーザが行をクリックすると、3つのエリア全部が同時に反転表示されます。それぞれのスクロールエリアに関連付けられた変数は、ユーザがクリックした行の番号に設定され、クリックされたエリアのオブジェクトメソッドだけが実行されます。例えば、ユーザが「菊地芳枝」をクリックすると、asName、asTitle、asDepartmentがすべて2に設定されますが、実行されるのはasNameのオブジェクトメソッドだけです。グループ化されたスクロールエリアのいずれかの配列で選択した要素を設定すると、次のイベントのためにグループの他の配列が選択した同じ要素に設定され、スクロールエリアのそれぞれの行が反転表示されます。

配列は、**SORT ARRAY**コマンドを使用してソートすることができます。次に例を示します。

SORT ARRAY(asTitle ; asName ;asDepartment ; >)

次にソート結果を示します。



配列が**SORT ARRAY**コマンドの最初の引数に基づいてソートされたことに注意してください。他の2つの配列は、行が同期するように指定されています。**SORT ARRAY**コマンドは、常に最初の配列の値に基づいて配列をソートし、他の配列をそれに同期させます。

注：SORT ARRAYコマンドは、配列に対するマルチレベルのソートはできません。上記のようなテーブルを表示して、マルチレベルのソートを実行するには（すなわち部門でソートしてから、役職名でソートし、次に部門名でソートする）、テーブルを表示するサブフォームを使用してから、ORDERED BYコマンドを使用します。

配列と4D言語

配列は、4D変数です。他の変数と同様、配列にも適用範囲があり、4D言語の規則に従いますが、いくつか他の変数とは異なる点があります。

ローカル配列、プロセス配列、インタープロセス配列

ローカル配列、プロセス配列、インタープロセス配列を作成して使用することができます。次に例を示します。

ARRAY INTEGER($aiCodes;100) `2バイト整数値のローカル配列を100個作成
ARRAY INTEGER(aiCodes;100) `2バイト整数値のプロセス配列を100個作成
ARRAY INTEGER(<>aiCodes;100) `2バイト整数値のインタープロセス配列を100個作成

これらの配列の適用範囲は、他のローカル変数、プロセス変数、インタープロセス変数と同一です。

ローカル配列

ローカル配列は、配列の名前をドル記号（$）で始めることによって宣言されます。
ローカル配列の適用範囲は、それが作成されたメソッド内です。メソッドが終了すると配列も消去されます。2つの異なるメソッドにある同じ名前のローカル配列は、実際には適用範囲の異なる2つの別個の変数なため、それぞれに異なるタイプを指定することができます。

フォームメソッド、オブジェクトメソッド、またはこれらのメソッドによってサブルーチンとして呼び出されるプロジェクトメソッドの中でローカル配列を作成する場合、配列はフォームメソッドまたはオブジェクトメソッドが起動されるたびに作成され消去されます。つまり、配列はフォームイベントごとに作成され消去されます。したがって、目的が表示であれ印刷であれ、ローカル配列をフォームに使用することはできません。

ローカル変数については、可能であればローカル配列を使用することをお勧めします。そうすることにより、多くの場合、アプリケーションを実行するために必要なメモリの量を小さくできます。

プロセス配列

プロセス配列は、配列名を文字で始めることによって宣言されます。

プロセス配列の適用範囲は、それが作成されたプロセス内です。配列は、プロセスの終了時またはアボート時に消去されます。プロセス配列には、プロセスごとに1つのインスタンスが自動的に作成されます。したがって、配列はプロセスのどこでも同じタイプです。ただし、その内容はプロセスによって異なります。

インタープロセス配列

インタープロセス配列は、配列名を<>で始めることによって宣言されます。

インタープロセス配列の適用範囲は、作業セッションのすべてのプロセスで構成されます。これらの使用は、プロセス間でのデータの共有や情報の転送のみに限る必要があります。

Tips：インタープロセス配列が複数のプロセスによってアクセスされ、それが衝突するおそれがあると予測できる場合、その配列へのアクセスをセマフォによって保護します。詳細については、Semaphore関数を参照してください。

注：プロセス配列とインタープロセス配列をフォームの中で使用して、スクロールエリアやドロップダウンリスト等のフォームオブジェクトを作成することができます。

配列の引数（パラメータ）としての受け渡し

配列を引数（パラメータ）として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列をパラメータとしてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。詳細については、後述の「配列とポインタ」を参照してください。

配列の他の配列への割り当て

テキストや文字列変数とは異なり、配列を他の配列に割り当てることはできません。配列を他の配列にコピーする（割り当てる）には、**COPY ARRAY**コマンドを使用します。

配列とポインタ

配列を引数（パラメータ）として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列を引数としてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。

次に例を示します。

▼次のような例を考えます。

```
If((0<atNames)&(atNames<Size of array(atNames))
    `可能であれば、選択した要素の次の要素を選択する
    atNames:=atNames+1
End if
```

異なるフォームの50の異なる配列に同じ処理を実行する場合、50回同じことを書かずに次のプロジェクトメソッドを使用することができます。

```
`「SELECT NEXT ELEMENT」プロジェクトメソッド
` SELECT NEXT ELEMENT(ポインタ)
` SELECT NEXT ELEMENT(-> 配列)
C_POINTER($1)
If((0<$1->)&($1-><Size of array($1->))
    $1->:=$1->+1 `可能であれば、選択した要素の次の要素を選択します。
End if
```

次に、以下のように記述します。

```
SELECT NEXT ELEMENT(->atNames)
` ...
SELECT NEXT ELEMENT(->asZipCodes)
```

```
` ...
SELECT NEXT ELEMENT(->alRecordIDs)
` ...
```

▼次のプロジェクトメソッドは、数値配列（整数、倍長整数または実数）のすべての要素の合計を返します。

```
` 「Array sum」プロジェクトメソッド
` Array sum(ポインタ)
` Array sum(->配列)
C_REAL($0)
$0:=0
For($vlElem ; 1 ; Size of array($1->))
        $0:=$0+$1->{$vlElem}
End for
```

次に、以下のように記述します。

```
$vlSum:=Array sum(->arSalaries)
` ...
$vlSum:=Array sum(->aiDefectCounts)
` ..
$vlSum:=Array sum(->alPopulations)
```

▼次のプロジェクトメソッドは、文字列またはテキスト配列のすべての要素を大文字に変換します。

```
` 「CAPITALIZE ARRAY」プロジェクトメソッド
` CAPITALIZE ARRAY(ポインタ)
` CAPITALIZE ARRAY(->配列)
For($vlElem ; 1 ; Size of array($1->))
        If($1->{$vlElem}#"")
                $1->{$vlElem}:=Uppercase($1->{$vlElem}[[1]])+Lowercase
                                                        (Substring( $1->{$vlElem} ; 2))
        End if
End if
```

次に、以下のように記述します。

```
CAPITALIZE ARRAY(->atSubjects )
` ...
CAPITALIZE ARRAY(->asLastNames )
```

配列、ポインタ、およびループ構造（**For**... **End for**等）を組み合わせることにより、配列を扱うための短くて便利なプロジェクトメソッドを作成することができます。

配列の要素ゼロを使用する

配列には、常に要素ゼロがあります。配列がフォームオブジェクトをサポートする場合は要素ゼロは表示されませんが、ランゲージでの使用については制限はありません。

要素ゼロの用途の一例として、「配列とフォームオブジェクト」の節で説明するコンボボックスの例があります。

その他の例を2つ挙げます。

1. 前に選択した要素以外の要素をクリックした場合のみ動作を行わせる場合、選択した要素のそれぞれを追跡する必要があります。これを実行する方法の1つは、選択する要素の要素番号を保持するプロセス変数を使用することです。もう1つの方法は、次のように配列の要素ゼロを使用する方法です。

```
`「 atNames」スクロールエリアのオブジェクトメソッド
Case of
	¥(Form event=On Load)
		`配列を初期化する
		ARRAY TEXT(atNames ; 5)
		` ...
		`要素ゼロをその文字列フォームの中で選択した
                                        現在の要素の番号で初期化する
		`これで選択した要素なしで開始します。
		atNames{0}:="0"
	¥(Form event=On Unload)
		` この配列はもう必要ありません
		CLEAR VARIABLE(atNames)
	¥(Form event=On Clicked)
		If(atNames#0)
			If(atNames#Num(atNames{0}))
				vtInfo:=atNames{atNames}+"上がクリックされました。
                                        これは、以前選択されていませんでした。"
				atNames{0}:=String(atNames)
			End if
		End if
	¥(Form event=On Double Clicked)
		If(atNames#0)
			ALERT(atNames{atNames}
                                        +"上がダブルクリックされました。")
		End if
End case
```

2. ドキュメントまたはシリアルポートから一連の文字群を送受信する場合、4Dは異なるASCIIマップを使用するプラットフォームおよびシステム間でASCIIコードをフィルタリングする方法として、**USE ASCII MAP**、**Mac to ISO**、**ISO to Mac**、**Mac to Win**、**Win to Mac**コマンドを提供します。

場合によっては、ASCIIコードを変換する方法を完全に制御したいことがあります。これには、255要素からなる整数配列を使用して、送信元のASCIIコードがNである文字の変換後のASCIIコードとしてN番目の要素を設定します。例えば、ASCIIコード#187を#156に変換する場合、データベース中で使用されるインタープロセス配列を初期化するメソッドの中に、「<>aiCustomOutMap{187}:=156」と「<>aiCustomInMap{156}:=187」と記述します。すると、次のカスタムプロジェクトメソッドを使用して、一連の文字群を送信することができます。

```
` X SEND PACKET(テキスト { ; 時間} )
For($vlCha r ; 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 Receive packet(テキスト { ; 時間} )→テキスト
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
```

この応用例では、NULL文字（ASCIIコードのゼロ）が入っている一連の文字群を送受信する場合、配列<>aiCustomOutMapおよび<>aiCustomInMapのゼロ要素は255要素の配列の他の要素としての役割を果たします。

2次元配列

配列宣言コマンドは、すべて1次元配列または2次元配列の作成またはサイズ変更を実行することができます。次に例を示します。

ARRAY TEXT(atTopics ; 100 ; 50) `100行50列からなるテキスト配列を作成します。

2次元配列は、本質的にはランゲージオブジェクトなため、表示も印刷もできません。

前述の例で、

■ atTopicsは、2次元配列です。

■ atTopics{8}{5}は、8行目の5番目(5列目)の要素です。

■ atTopics{20}は、20行目で配列自体は1次元配列です。

■ **Size of array**(atTopics)は、行数の100を返します。

■ **Size of array**(atTopics{17})は、17行目の列数である50を返します。

以下の例では、データベースの各テーブルの各フィールドへのポインタが2次元配列に格納されます。

```
` まず、空の行をテーブルと同じ数だけ作成する
ARRAY POINTER(<>apFields ; Count tables ; 0)
`各テーブルに対して、
For($vlTable ; 1 ; Size of array(<>apFields))
    `行の列数がテーブルのフィールドと同じ数になるようにサイズ変更する
    INSERT ELEMENT(<>apFields{$vlTable} ; 1 ; Count tables($vlTable))
        `要素の値を設定する
    For($vlField ; 1 ; Size of array(<>apFields{$vlTable}))
        <>apFields{$vlTable}{$vlField}:=Field($vlTable;$vlField)
    End for
End for
```

この2次元配列が初期化されている場合、以下の方法で特定のテーブルのフィールドへのポインタを取得することができます。

```
`現在画面表示されているテーブルのフィールドへのポインタを取得する
COPY ARRAY(<>apFields{Table(Current form table)};
                                        $apTheFieldsIamWorkingOn)
`ブールと日付フィールドを初期化する
For($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
```

注：この例でわかるように、2次元配列の行は同じサイズでも異なるサイズでも構いません。

配列とメモリ

テーブルやレコードを使用してディスク上に格納したデータと異なり、配列は常に全部がメモリに保存されます。

例えば、米国内の郵便番号がすべて[郵便番号]テーブルに入力されている場合、約100,000件のレコードになります。加えて、そのテーブルは郵便番号自体、対応する市、郡、州という複数のフィールドを持つとします。カリフォルニアからのみ郵便番号を選択する場合、4Dデータベースエンジンが[郵便番号]テーブルの対応するレコードセレクションを作成して、必要な場合にのみそのレコードをロードします（例えば表示や印刷時）。すなわち、4Dのデータベースエンジンによってディスクからメモリに部分的にロードされた（フィールドごとに同じタイプの）順序づけられた一連の値で作業するということです。

同じことを配列で実行するのは、次の理由で禁止されています。

- 4つの情報タイプ（郵便番号、市、郡、州）を維持するためには、4つの大きな配列をメモリ内で維持する必要があります。
- 配列は、常に全体がメモリ内に維持されるため、あまり使用しない場合でも、作業セッションの間すべての郵便番号をメモリに置いておく必要があります。
- 同じく配列全体が常にメモリ内に維持されることから、データが作業セッション中に使用も変更もされていない場合でも、データベースが開始されて終了するたびに、4つの配列をロードしてからディスクに保存する必要があります。

結論：配列は、ほどよい量のデータを短時間維持するためのものです。他方、配列はメモリ内に置かれるため、扱いやすく高速操作が可能です。

しかし、状況によっては何百、何千という要素を持った配列で作業する必要があります。次の表に、各配列がメモリ上に占めるバイト数を求めるための計算式を示します。

配列タイプ	メモリ上に占めるバイト数を求めるための計算式
ブール	(31 + 配列の要素の数) // 8
日付	(1 + 配列の要素の数) *6
文字列	(1 + 配列の要素の数) * 指定した文字列の長さ (+1 奇数の場合,+2 偶数の場合) * 2
整数	(1 + 配列の要素の数) * 2
倍長整数	(1 + 配列の要素の数) * 4
ピクチャ	(1 + 配列の要素の数) * 4 + 各ピクチャサイズの合計
実数	(1 + 配列の要素の数) * 16
テキスト	((1 + 配列の要素の数) * 6) + 各テキストサイズの合計
2次元配列	(1 + 配列の要素の数) * 12 + 各配列サイズの合計

注：選択した要素、要素番号、配列自体の情報を保存するために、別に数バイトを要します。非常に大きな配列で作業する場合、メモリが一杯という状況を扱う最善の方法は、配列の作成をON ERR CALLプロジェクトメソッドで囲むことです。以下に例を示します。

```
`大きな配列を作成する必要がある晩に一晩かけてバッチ操作を実行します。
`夜間にエラーが発生する危険がないように、操作の始めに配列を作成して、
`この時点でエラーをテストします。
gError:=0       `エラーがないと仮定する
ON ERR CALL("エラー処理") `エラーを検出するためのメソッドをインストールする
ARRAY STRING(63 ; asThisArray ; 50000) ` 約3125K
ARRAY REAL(arThisAnotherArray ; 50000) ` 488K
ON ERR CALL("") `これ以上エラーを検出する必要はない
If(gError=0)
        `配列を作成することができました。操作を続行します。
Else
        ALERT("この処理は多くのメモリを必要とします。")
End if
`どのような場合でも、これ以上配列は必要ありません。
CLEAR VARIABLE(asThisArray)
CLEAR VARIABLE(arThisAnotherArray)
```

「エラー処理」プロジェクトメソッドを次に示します。

```
`「エラー処理」プロジェクトメソッド
gError:=Error `エラーコードだけが返される
```

ARRAY INTEGER

ARRAY INTEGER(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY INTEGERコマンドは、メモリ上に2バイトの整数の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

■ サイズを増やした場合には、既存の値は保持され、あたらしい要素は0で初期化されます。

■ サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、2バイト整数の配列を100個作ります。

⇒ **ARRAY INTEGER**(aiValues;100)

2. この例は、50個の2バイト整数のローカル配列を100行作ります。

⇒ **ARRAY INTEGER**($aiValues;100;50)

3. この例は、2バイト整数のインタープロセス配列を50個作り、その要素番号を格納します。

```
⇒ ARRAY INTEGER(<>aiValues;50)
   For($vlElem;1;50)
      <>aiValues{$vlElem}:=$vlElem
   End for
```

ARRAY LONGINT

ARRAY LONGINT(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY LONGINTコマンドは、メモリ上に4バイトの倍長整数の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

■ サイズを増やした場合には、既存の値は保持され、あたらしい要素は0で初期化されます。

■ サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、倍長整数の配列を100個作ります。

⇒ **ARRAY LONGINT**(alValues;100)

2. この例は、倍長整数のローカル配列50個を100行作ります。

⇒ **ARRAY LONGINT**($alValues;100;50)

3. この例は、50個の倍長整数のインタープロセス配列を作り、その要素番号を格納します。

```
⇒ ARRAY LONGINT(<>alValues;50)
  For($vlElem;1;50)
      <>alValues{$vlElem}:=$vlElem
  End for
```

ARRAY REAL

ARRAY REAL(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY REALコマンドは、メモリ上に実数の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

■ サイズを増やした場合には、既存の値は保持され、あたらしい要素は0で初期化されます。

■ サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、実数の配列を100個作ります。

⇒ **ARRAY REAL**(arValues;100)

2. この例は、50個の実数のローカル配列を100行作ります。

⇒ **ARRAY REAL**($arValues;100;50)

3. この例は、実数のインタープロセス配列を50個作り、その要素番号を格納します。

```
⇒ ARRAY REAL(<>arValues;50)
   For($vlElem;1;50)
      <>arValues{$vlElem}:=$vlElem
   End for
```

ARRAY STRING

ARRAY STRING(長さ ; 配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
長さ	数値	→	文字列の長さ（1～255）
配列名	配列	→	配列の名前
サイズ1	数値	→	配列の要素の数 サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	2次元配列の要素の数

説明

ARRAY STRINGコマンドは、メモリ上に文字列の配列を作成またはリサイズします。

引数＜長さ＞は、配列中の各要素に含むことのできる最大文字数を指定します。＜長さ＞は1から255まで指定できます。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

- サイズを増やした場合には、既存の値は保持され、あたらしい要素は""（空文字列）で初期化されます。
- サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、31文字からなる文字列のプロセス配列を100個作ります。

⇒ **ARRAY STRING**(31;asValues;100)

2. この例は、63文字からなる文字列のローカル配列50個を100行作ります。

⇒ **ARRAY STRING**(63;$asValues;100;50)

3. この例は、255文字からなる文字列のインタープロセス配列を50個作り、文字列”要素 #”と、それに続けて要素番号を格納します。

⇒
```
ARRAY STRING(255;<>asValues;50)
For($vlElem;1;50)
        <>asValues{$vlElem}:="要素 #"+String($vlElem)
End for
```

ARRAY TEXT

ARRAY TEXT(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY TEXTコマンドは、メモリ上にテキスト要素の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

■ サイズを増やした場合には、既存の値は保持され、あたらしい要素は0で初期化されます。

■ サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、テキスト要素の配列を100個作ります。

⇒ **ARRAY TEXT**(atValues;100)

2. この例は、50個のテキスト要素のローカル配列を100行作ります。

⇒ **ARRAY TEXT**($arValues;100;50)

3. この例は、テキスト要素のインタープロセス配列を50個作り、文字列”要素 #”と、それに続けて要素番号を格納します。

```
⇒ ARRAY TEXT(<>atValues;50)
   For($vlElem;1;50)
      <>atValues{$vlElem}:="要素 #"+String($vlElem)
   End for
```

ARRAY DATE

ARRAY DATE(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY DATEコマンドは、メモリ上に日付要素の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

■ サイズを増やした場合には、既存の値は保持され、あたらしい要素はヌル値（!00/00/00!）で初期化されます。

■ サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、日付要素の配列を100個作ります。

⇒ **ARRAY DATE**(adValues;100)

2. この例は、50個の日付要素のローカル配列を100行作ります。

⇒ **ARRAY DATE**($adValues;100;50)

3. この例は、日付要素のインタープロセス配列を50個作り、現在の日付に要素番号を足した値を格納します。

```
⇒ ARRAY DATE(<>adValues;50)
   For($vlElem;1;50)
      <>adValues{$vlElem}:=Current date+$vlElem
   End for
```

ARRAY BOOLEAN

ARRAY BOOLEAN(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY BOOLEANコマンドは、メモリ上にブール要素の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

- サイズを増やした場合には、既存の値は保持され、あたらしい要素は“False（偽）”で初期化されます。
- サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、ブール要素の配列を100個作ります。

⇒ **ARRAY BOOLEAN**(abValues;100)

2. この例は、50個のブール要素のローカル配列を100行作ります。

⇒ **ARRAY BOOLEAN**($abValues;100;50)

3. この例は、ブール要素のインタープロセス配列を50個作り、偶数の要素には“True（真）を格納します。

⇒
```
ARRAY BOOLEAN(<>abValues;50)
For($vlElem;1;50)
        <>abValues{$vlElem}:=(($vlElem%2)=0)
End for
```

ARRAY PICTURE

ARRAY PICTURE(配列名 ; サイズ1 ; {サイズ2})

説明

ARRAY PICTUREコマンドは、メモリ上にピクチャ要素の配列を作成またはリサイズします。

引数<配列名>は、作成する配列の名前です。

引数<サイズ1>は、配列の要素の数です。

引数<サイズ2>はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、<サイズ1>に配列の行数を、<サイズ2>には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

- サイズを増やした場合には、既存の値は保持され、あたらしい要素は空のピクチャで初期化されます。これらのピクチャのサイズは0になります。
- サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、ピクチャ要素の配列を100個作ります。

⇒ **ARRAY PICTURE**(agValues;100)

2. この例は、50個のピクチャ要素のローカル配列を100行作ります。

⇒ **ARRAY PICTURE**($agValues;100;50)

3. この例は、日付要素のインタープロセス配列を50個作り、それぞれのピクチャを配列の要素に読み込みます。配列の大きさはデータベース上で有用な"PICT"リソースの数と等しくなります。配列のリソース名は"User Intf/"で始められます。

```
   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)
```

ARRAY POINTER

ARRAY POINTER(配列名 ; サイズ1 ; {サイズ2})

引数	タイプ		説明
配列名	配列	→	新しい配列の名前
サイズ1	数値	→	配列の要素の数
		→	サイズ2を指定した場合は、配列の列数
サイズ2	数値	→	２次元配列の要素の数

説明

ARRAY POINTERコマンドは、メモリ上にブール要素の配列を作成またはリサイズします。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、2次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の行数を、＜サイズ2＞には列数を指定します。2次元配列の各配列は要素と同じように扱います。したがって、このコマンドを使用して、2次元配列中の配列に要素を挿入、または削除することができます。

このコマンドを使用して既存の配列の大きさを変更する場合、

- サイズを増やした場合には、既存の値は保持され、あたらしい要素はヌルポインタで初期化されます。**Nil**を要素に使用した場合、“True”をかえします。
- サイズを減らした場合には、後ろの要素は配列から削除され、失われます。

例題

1. この例は、ポインタ要素の配列を100個作ります。

⇒ **ARRAY POINTER**(apValues;100)

2. この例は、50個のポインタ要素のローカル配列を100行作ります。

⇒ **ARRAY POINTER**($apValues;100;50)

3. この例は、ポインタ要素のインタープロセス配列を作り、要素と同じ番号のテーブルを示すポインタを格納します。配列の大きさはデータベース上のテーブルの数と等しくなります。

```
ARRAY POINTER(<>apValues;Count tables)
For($vlElem;1;Size of array(<>apValues))
        <>apValues{$vlElem}:=Table($vlElem)
End for
```

Size of array

Size of array(配列) → 数値

引数	タイプ		説明
配列	配列	→	大きさを求める配列

説明

Size of array関数は、＜配列＞の大きさ（要素の数）を返します。＜配列＞が2次元配列の場合には、配列の行数を返します。

▼以下の例は、配列“MyArr”の大きさを変数“vサイズ”に代入します。

```
vサイズ:=Size of array(MyArr)      ` MyArrの大きさを求める
```

参照

DELETE ELEMENT、INSERT ELEMENT

SORT ARRAY

SORT ARRAY(配列1{ ;...; 配列N} ; {ソート種別})

引数	タイプ		説明
配列	配列	→	ソートする配列
ソート種別	>または<	→	>昇順、<降順

説明

SORT ARRAYコマンドは、1つまたは複数の配列を昇順や降順にソートします。

注：ポインタ配列やピクチャ配列のソートは行えません。2次元配列の要素（a2DArray{$vlThisElem}）をソートすることはできますが、2次元配列そのもの（a2DArray）をソートすることはできません。

ピクチャ以外の、すべての配列タイプに使用することができます。また、配列を2次元配列の最初の次元にすることはできません。

引数＜ソート種別＞は、配列を昇順にソートするのか、降順にソートするのかを指定します。＜ソート種別＞に（>）を指定すると昇順にソートします。＜ソート種別＞に（<）を指定すると降順にソートします。＜ソート種別＞を省略した場合は昇順にソートします。

複数の配列を指定した場合は、最初の配列の順番でソートします。各配列ごとに個々にソートするわけではありません。この機能は、フォーム上のグループ化したスクロールエリアを扱う場合に特に有用です。**SORT ARRAY**コマンドはスクロールエリアを構成する配列の同期を維持します。

例題

▼次の例は、配列を2つ作成し、元のレコードに影響を与えないで会社を昇順でソートします。

```
ALL RECORDS([従業員])
SELECTION TO ARRAY([従業員]名前 ; arNames ; [従業員]会社 ; arComp)
SORT ARRAY(arComp ; arNames  ; >)
```

しかし、コマンドはマルチレベルソートを行わないため、会社ごとの従業員名の順序はばらばらのままです。各会社の従業員名をソートするには、次のようにします。

```
ALL RECORDS([従業員])
ORDER BY([従業員];[従業員]会社;>;[従業員]名前;>)
SELECTION TO ARRAY([従業員]名前;arNames;[従業員]会社;arCompanies)
```

▼次の例では、[従業員]テーブルの名前をフローティングウインドウに表示します。ウインドウ上のボタンをクリックすると、名前のリストをAからZへ、またはZからAへソートします。同じ名前の人が複数存在するため、インデックス付きで重複不可である[従業員]ID番号フィールドを使用できます。名前のリストをクリックすると、クリックした名前を持つレコードを取得します。同期がとられ、表には表示されないID番号フィールドを利用することにより、必ずクリックした名前に対応するレコードにアクセスされます。

```
` asNames 配列オブジェクトメソッド
Case of
        ¥(Form event=On Load)
                ALL RECORDS([従業員])
                SELECTION TO ARRAY([従業員]名前;asNames;[従業員]
                                                        ID番号;alIDs)
                SORT ARRAY(asNames;alIDs;>)
        ¥(Form event=On Unload)
                CLEAR VARIABLE(asNames)
                CLEAR VARIABLE(alIDs)
        ¥(Form event=On Clicked)
                If(asNames#0)
                ` 正しいレコードを取得するために配列alIDsを使用する
                        QUERY([従業員];[従業員]ID番号=alIDs{asNames})
                        ` レコードを用いて処理を実行する
                End if
End case

` bA2Zボタンのオブジェクトメソッド
` 配列を昇順にソートし、同期を維持する
SORT ARRAY(asNames;alIDs;>)
` bZ2Aボタンのオブジェクトメソッド
` 配列を昇順にソートし、同期を維持する
SORT ARRAY(asNames;alIDs;<)
```

参照

ORDER BY、SELECTION TO ARRAY

MULTI SORT ARRAY

MULTI SORT ARRAY(配列{; 順序}{; 配列2; 順序2; ...; 配列N; 順序N})

引数	タイプ		説明
配列	配列	→	並び替える配列
順序	> または <	→	>：昇順並び替え、または <：降順並び替え 省略した場合：並び替えは行わない

MULTI SORT ARRAY(ポインタ配列名; ソート配列名)

引数	タイプ		説明
ポインタ配列名	ポインタ配列	→	配列ポインタの配列
ソート配列名	倍長整数配列	→	並び替え順の配列（1= 昇順並び替え、-1= 降順並び替え、0= 前の並び替えとの同期化）

説明

MULTI SORT ARRAYコマンドにより、一連の配列に対してマルチレベルソートを実行することができます。この機能は、フォーム内のグループ化したスクロールエリアのコンテキストにおいて特に役立ちます。

このコマンドは2種類の構文を受け入れます。

■ 第1の構文：**MULTI SORT ARRAY**(配列{; 順序}{; 配列2; 順序2; ...; 配列N; 順序N})

この構文が最もシンプルです。複合条件による並び替えを適用しようとする同期化配列の名前を直接渡すことができます。

配列だけを渡すか、あるいは「配列；“>”または“<”」の組み合わせを好きな数だけ渡すことができます。引数として渡された配列はすべて、同期化されて並び替えられます。

また、ポインタタイプやピクチャタイプの配列を除き、任意のタイプの配列を渡すことができます。二次元配列の要素（例：a2DArray{$vlThisElement}）を並び替えられますが、二次元配列自体を並び替えることはできません（つまり、a2DArray）。

配列の内容を並び替え条件として使用するには、引数＜順序＞を渡します。この引数の値（“>”または“<”）により、その配列の並び替え順序（昇順または降順）が決まります。引数＜順序＞を省略した場合、その配列の内容は並び替え条件として使用されません。

注：コマンドが正常に機能するには、少なくとも1つの並び替え条件を渡さなければならないということを覚えておいてください。並び替え条件が設定されていない場合はエラーが生成されます。

コマンドに渡した配列の順番により、並び替えレベルが決定します。つまり、この構文において、並び替え条件が指定された配列の位置により、その並び替えレベルが決まります。

■ 第2の構文：**MULTI SORT ARRAY**(ポインタ配列名; ソート配列名)

この構文は、より複雑であり、汎用的な開発では非常に有効です（例えば、あらゆるタイプの配列を並び替える汎用メソッドを作成したり、汎用的な**SORT ARRAY**コマンドに相当するものを作成する場合）。

引数＜ポインタ配列名＞には、配列ポインタの配列名を指定します。この配列の各要素は、並び替える配列を示すポインタです。＜ポインタ配列名＞に指定した配列ポインタの順に、並び替えが実行されます。

注：＜ポインタ配列名＞には、ローカル（$ptrArrayName）、プロセス（ptrArrayName）、インタープロセス（<>ptrArrayName）タイプのポインタの配列を指定することができます。これとは逆に、この配列の要素が指す対象は、プロセス配列またはインタープロセス配列でなくてはなりません。

引数＜ソート配列名＞には配列名を渡し、この配列の各要素は対応するポインタ配列要素の並び替え順を示します。

■ 1=降順並び替え

■ 0=配列は並び替え条件として使用されませんが、他の並び替えに応じて並び替えられます。

■ 1=昇順並び替え

注：ポインタタイプやピクチャタイプの配列を並び替えることはできません。二次元配列（つまり、a2DArray{$vlThisElement}）の要素を並び替えることができますが、二次元配列そのもの（つまり、a2DArray）を並び替えることはできません。

配列＜ポインタ配列名＞の各要素に対して、対応する配列＜ソート配列名＞の要素が存在していなければなりません。したがって、必ずこの2つの配列の要素数は全く同じになります。

例題

1. 次の例題では1番目の構文を使用します。4つの配列を作成して都市順（昇順）の給与順（降順）で並び替えます。最後の2つの配列、namesとtelNumsは、前の並び替え条件に応じて同期化されます。

```
ALL RECORDS([従業員])
SELECTION TO ARRAY([従業員]都市;cities;[従業員]給与;salaries;[従業員]名前;
                                                    names;[従業員]電話番号;telNums)
MULTI SORT ARRAY(cities;>;salaries;<;names;telNums)
```

配列namesを3番目の並び替え条件として使用したい場合は、引数であるnames配列の後に“>”または“<”を付加します。

次の構文は、

```
MULTI SORT ARRAY(cities;>;salaries;names;telNums)
```

次の命令と同じであることに注意してください。

```
SORT ARRAY(cities;salaries;names;telNums;>)
```

2. 次の例題では2番目の構文を使用して、4つの配列を作成し、都市（昇順）と会社（降順）で並び替えます。最後の2つの配列、names_ArrayとtelNum_Arrayは、前の並び替え条件に応じて同期化されます。

```
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]City;cities;[Employees]Company;companies;
                                    [Employees]Name;names;[Employees]TelNum;telNums)
ARRAY POINTER(pointers_Array;4)
ARRAY LONGINT(sorts_Array;4)
pointers_Array{1}:=->cities
sorts_Array{1}:=1
pointers_Array{2}:=->companies
sorts_Array{2}:=-1
pointers_Array{3}:=->names
sorts_Array{3}:=0
pointers_Array{4}:=->telNums
sorts_Array{4}:=0
MULTI SORT ARRAY(pointers_Array;sorts_Array)
```

3番目の並び替え条件としてnames配列を使用したい場合には、sorts_Array{3}要素に値“1”を割り当てる必要があります。または、都市だけを条件として配列を並び替えたい場合は、sorts_Array{2}、sorts_Array{3}、sorts_Array{4}の要素に値“0”を割り当てます。この方法で、**SORT ARRAY**(cities; companies;names;telNums;>)と同じ結果を得ることができます。

参照：

SELECTION TO ARRAY、ORDER BY、SORT ARRAY

Find in array

Find in array(配列 ; 値 ; {開始}) → 数値

引数	タイプ		説明
配列	配列	→	検索する配列
値	文字列または数値、日付またはブール	→	検索する値
開始	数値	→	検索を開始する要素の番号

説明

Find in array関数は、＜配列＞から引数＜値＞と同じものを検索し、最初に発見された要素の番号を返します。

Find in array関数は、タイプがテキスト、文字列、数値、日付、ブールの配列に使用します。配列タイプと値のタイプは、必ず同じにしてください。

引数＜値＞と同じものを発見できない場合には、**Find in array**関数は-1を返します。

引数＜開始＞を指定するとその番号の要素から検索を始めます。引数＜開始＞を指定していない場合には、コマンドは第1要素から検索します。

例題

1. 以下のプロジェクトメソッドは、ポインタを引数として渡した文字列配列またはテキスト配列から空の要素をすべて削除します。

```
` CLEAN UP ARRAYプロジェクトメソッド
` CLEAN UP ARRAY( ポインタ)
` CLEAN UP ARRAY( -> テキスト配列または文字列配列 )
C_POINTER($1)
Repeat
    $vlElem:=Find in array($1->;"")
    If($vlElem>0)
        DELETE ELEMENT($1->;$vlElem)
    End if
Until($vlElem<0)
```

このプロジェクトメソッドの作成後、以下のように記述します。

```
ARRAY TEXT(atSomeValues;...)
` ...
` 配列を使用したさまざまな処理をここで実行する
` ...
```

```
` 空の文字列を含む要素を取り除く
CLEAN UP ARRAY(->atSomeValues)
```

2. 次のプロジェクトメソッドは、ポインタを最初の引数で渡した配列の中から、ポインタを引数として渡した変数やフィールドの値に一致する最初の要素を選択します。

```
` SELECT ELEMENT プロジェクトメソッド
` SELECT ELEMENT(ポインタ ; ポインタ)
` SELECT ELEMENT(->テキスト配列または文字列配列 ; ->
                                        テキストか文字列の変数またはフィールド )

$1->:=Find in array($1->;$2->)
If($1->=-1)
    $1->:=0 ` 該当する要素が見つからない場合、選択された要素なしに設定する
End if
```

このプロジェクトメソッドの作成後、以下のように記述します。

```
` asGender ポップアップメニューのオブジェクトメソッド
Case of
    ¥(Form Event=On Load)
        SELECT ELEMENT(->asGender;->[People]Gender)
End case
```

参照

DELETE ELEMENT、INSERT ELEMENT、Size of array

Count in array

Count in array(配列; 値) → 倍長整数

引数	タイプ		説明
配列	配列	→	集計を行う配列
値	4D表記	→	集計する値
戻り値	倍長整数	←	出現回数

説明

Count in array関数は、＜配列＞内における＜値＞の出現回数を返します。

このコマンドは、テキスト、文字、数値、日付、ポインタ、ブールタイプの配列に対して使用できます。引数＜配列＞と＜値＞は同じタイプか、または互換性があるタイプでなくてはなりません。

＜値＞と一致する項目が＜配列＞内に存在しない場合、コマンドは0を返します。

例題

▼次の例題を使用し、リストボックス内で選択された行の数を表示することができます。

```
`tBListはリストボックス列配列の名前です
ALERT(String(Count in array(tBList;True))+
                    "行がリストボックス内で選択されました。")
```

参照

Find in array

INSERT ELEMENT

INSERT ELEMENT(配列 ; 挿入位置 ; {要素数})

引数	タイプ		説明
配列	配列	→	配列の名前
挿入位置	数値	→	要素を挿入する位置
要素数	数値	→	挿入する要素の数

説明

INSERT ELEMENTコマンドは、＜配列＞に1つまたは複数の要素を挿入します。引数＜挿入位置＞の示す要素の前に新しい要素を挿入します。新しく挿入された要素には、配列タイプに応じた空の値が代入されます。＜挿入位置＞より後ろの要素は、＜要素数＞で指定した数だけ後ろに移動します。

＜挿入位置＞が配列より大きい（配列の要素の数より多い）場合は、配列の最後に挿入します。

引数＜要素数＞は、挿入する要素の数を指定します。省略した場合は、1と解釈します。配列の大きさは、＜要素数＞によって動的に増加します。

例題

▼ 以下の例は、配列“リスト”の10番目の要素の位置に新しい5つの要素を挿入します。

```
INSERT ELEMENT(リスト ; 10 ; 5)
```

▼ 次の例は、配列に要素を追加します。

```
$vlElem:=Size of array(anArray)+1
INSERT ELEMENT(anArray;$vlElem)
anArray{$vlElem}:=...
```

参照

DELETE ELEMENT、Size of array

DELETE ELEMENT

DELETE ELEMENT(配列 ; 削除位置 ; {要素数})

引数	タイプ		説明
配列	配列	→	要素を削除する配列
削除位置	数値	→	要素の削除を開始する位置
要素数	数値	→	削除する要素の数

説明

DELETE ELEMENTコマンドは、1つまたは複数の要素を＜配列＞から削除します。引数＜削除位置＞の示す位置から要素の削除を開始します。

＜削除位置＞が配列より大きい(配列の要素の数より多い)場合は、配列の最後を削除します。引数＜要素数＞は、削除する要素の数を指定します。省略した場合は、1と解釈します。配列の大きさは、＜要素数＞によって動的に減少します。

例題

▼ 以下の例は、配列“リスト”の5番目の要素から3つの要素を削除します。

```
DELETE ELEMENT(リスト ; 5 ; 3)
```

▼ 次の例は、配列の最後の要素を削除します。

```
$vlElem:=Size of array(anArray)
If($vlElem>0)
    DELETE ELEMENT(anArray;$vlElem)
End if
```

参照

INSERT ELEMENT、Size of array

APPEND TO ARRAY

APPEND TO ARRAY(配列; 値)

引数	タイプ		説明
配列	配列	→	項目を追加する配列
値	4D表記	→	追加する値

説明

APPEND TO ARRAYコマンドは、＜配列＞の最後に新規項目を追加し、その項目に＜値＞を割り当てます。インタプリタモードにおいて、＜配列＞が存在しない場合、コマンドは＜値＞のタイプに従って配列を作成します。

このコマンドはあらゆるタイプの配列（文字列、数値、ブール、日付、ポインタ、ピクチャ）に対応します。

＜値＞のタイプは配列のタイプと一致しなくてはなりません。一致しない場合は、シンタックスエラー54“引数のタイプが違います”が生成されます。ただし、次の値は受け入れられます。

■ 文字列タイプの＜配列＞（テキストまたは文字列）はテキストや文字列タイプの＜値＞を受け入れます。

■ 数値タイプの＜配列＞（整数、倍長整数、実数）は、整数、倍長整数、実数、または時間タイプの＜値＞を受け入れます。

例題

▼次のコードの場合

```
INSERT ELEMENT($myarray;Size of array($myarray)+1)
$myarray{Size of array($myarray)}:=$myvalue
```

次の命令に書き換えることができます。

```
APPEND TO ARRAY($myarray;$myvalue)
```

参照

DELETE ELEMENT、INSERT ELEMENT

COPY ARRAY

COPY ARRAY(コピー元 ; コピー先)

引数	タイプ		説明
コピー元	配列	→	コピー元の配列
コピー先	配列	←	コピー先の配列

説明

COPY ARRAYコマンドは、既存の配列を複製します。＜コピー先＞が存在しない場合は、＜コピー元＞の配列と全く同じ大きさとタイプを持つ配列を作成します。＜コピー先＞が存在する場合には、新しく作成した配列に置き換えます。

＜コピー元＞と＜コピー先＞の配列はローカル配列、プロセス配列、インタープロセス配列にすることができます。配列の適用範囲は、配列を複製する際には問題ではありません。

例題

▼以下の例は、配列“arCompany”と同じ大きさと同じ内容を持つ配列“arData”を作成します。

```
` 従業員テーブルのすべてのレコードを選択
ALL RECORDS([従業員])
      ` 会社フィールドのデータを配列「arCompany」に格納
SELECTION TO ARRAY([従業員]会社 ; arCompany)
      ` 配列“arCompany”を配列“arData”にコピー
COPY ARRAY(arCompany ; arData)
```

LIST TO ARRAY

互換性に関する注意

選択リストに関する新しい機能により、このコマンドの互換性が完全には保たれなくなっています。また、バージョン6からは、「デザイン」モードの「リストエディタ」で定義された階層リストを使って作業する場合には、Load listコマンドの使用をお勧めします。

LIST TO ARRAY(リスト ; 配列 ; {項目参照})

引数	タイプ		説明
リスト	文字列	→	コピー元のリスト
配列	配列	←	コピー先の配列
項目参照	配列	←	リスト項目の参照番号

説明

LIST TO ARRAYコマンドは、＜リスト＞から＜配列＞を作成します。このコマンドは、データを＜リスト＞から＜配列＞にコピーします。配列が既存の場合は、配列を上書きします。配列のタイプを前もって文字型として定義しない限り、テキストの配列を作成します。

任意の引数＜項目参照＞（数値配列）は、リスト項目の参照番号を返します。

互換性に関する注意：前バージョンの4Dでは、この配列にはリンクされたリストの名前＜リンク配列＞を代入していました。リストの項目がリンクしたリストを持っている場合、リンク先のリストの名前をリンク配列の同じ番号の要素に代入します。リンク先のリストが存在しない場合は、空の文字列を代入します。＜リンク配列＞は、リストと同じ大きさです。リンク配列に含まれる名前は、リンク先のリストを参照するために使用します。

今バージョンでも**LIST TO ARRAY**コマンドを使い、階層リストの第1階層の項目をもとにして配列を作成できますが、このコマンドでは子アイテムを使っての作業は行えません。階層リストを使用する場合、バージョン6から導入された新しい階層リストコマンドをお使いください。

例題

▼ 以下の例は、リスト“地方”の項目を配列“a地方”にコピーします。

```
LIST TO ARRAY("地方" ; a地方)
```

参照

ARRAY TO LIST、Load list、SAVE LIST.

ARRAY TO LIST

互換性に関する注意

選択リストに関する新しい機能により、このコマンドの互換性が完全には保たれなくなっています。また、バージョン6からは、「デザイン」モードの「リストエディタ」で定義された階層リストを使って作業する場合には、SAVE LISTコマンドの使用をお勧めします。

ARRAY TO LIST(配列 ; リスト ; {項目参照})

引数	タイプ		説明
配列	配列	→	コピー元の配列
リスト	文字列	→	コピー先のリスト
項目参照	配列	→	項目参照番号を格納した数値配列

説明

ARRAY TO LISTコマンドは、＜配列＞を＜リスト＞にコピーします。＜リスト＞が存在しない場合は、リストを作成します。

このコマンドでは、リストの第1階層の項目だけを定義できます。

任意の引数＜項目参照＞（指定されている場合）は、数値配列で、＜配列＞と同期がとられている必要があります。各要素は、対応する＜配列＞の要素に対するリスト項目の参照番号を表わします。この引数を省略した場合、4Dにより自動的に1、2...Nという項目参照番号が設定されます。

互換性に関する注意：前バージョンの4Dでは、この引数は他のリストを＜配列＞の各要素にリンクするために使用していました（＜リンク配列＞）。＜リンク配列＞の要素が既存のリストの名前である場合に、対応するアイテムにそのリストがリンクされます。

今バージョンでも**ARRAY TO LIST**コマンドを使い、配列の要素ををもとにしてリストを作成できますが、このコマンドでは子アイテムを使っての作業は行えません。階層リストを使用する場合、バージョン6から導入された新しい階層リストコマンドをお使いください。

例題

▼ 以下の例は、配列“a地方”をリスト“地方”にコピーします。

```
ARRAY TO LIST(a地方 ; "地方")
```

エラー処理

「デザイン」モードのリストエディタで編集中のリストに対して、**ARRAY TO LIST**コマンドが使用されると、エラー-9957が発生します。

参照

LIST TO ARRAY、Load list、ON ERR CALL、SAVE LIST

SELECTION TO ARRAY

SELECTION TO ARRAY(フィールド| テーブル; 配列{; フィールド2| テーブル2; 配列2; ...; フィールドN| テーブルN; 配列N})

引数	タイプ		説明
フィールド¦テーブル	フィールドまたはテーブル	→	データの取得に使用するフィールド、またはレコード番号の取得に使用するテーブル
配列	配列	←	フィールドデータまたはレコード番号を受け取る配列

説明

SELECTION TO ARRAYコマンドは、1つ以上の配列を作成し、カレントセレクションのフィールドデータやレコード番号を配列にコピーします。

SELECTION TO ARRAYコマンドは、1番目の引数として指定したテーブルのセレクションに対して適用され、次の事柄を実行します。

■ 1つ、または複数のフィールドから値をロードします。

■ 構文“...;[テーブル];配列;...”を用いてレコード番号をロードします。

■ テーブル間にn対1の自動リレートが設定されている場合や、**SET AUTOMATIC RELATIONS**を事前に呼び出してn対1のマニュアルリレートを自動に変更した場合は、リレートフィールドの値をロードします。いずれの場合も、複数レベルのn対1リレートを経てテーブルから値がロードされます。

各配列は、そのフィールドタイプに応じてタイプ定義されます。ただし、次の2つの例外があります。

■ テキストフィールドを文字列タイプの配列にコピーした場合、その配列は文字列配列のままになります。

■ 時間フィールドは倍長整数タイプの配列にコピーされます。

注：サブテーブルのフィールドやサブフィールドを指定することはできません。

第2のシンタックスでは、**SELECTION TO ARRAY**コマンドは、＜配列＞に＜テーブル＞のレコード番号の配列を作成します。デフォルトのデータタイプは倍長整数型の配列です。

4D Server：コマンドは4D Server用に最適化されています。各配列はサーバ上で作成され、配列全体がクライアントマシンに送信されます。

警告：SELECTION TO ARRAYコマンドは、セレクションの大きさによって非常に大きな配列を作成する場合があります。配列はメモリに常駐しますので、十分なメモリがあるかどうか必ず確認してください。

注：SELECTION TO ARRAYコマンドを呼び出した後、カレントセレクションとカレントレコードは同じままですが、カレントレコードはロードされません。そのため、もし、カレントレコードのフィールド値を使いたい場合は、SELECTION TO ARRAYコマンドの後にLOAD RECORDコマンドを使用してください。

例題

▼ 以下の例は、自動リレートで関連している[従業員]テーブルと[会社]テーブルのデータから“arFirst”と“arAddress”の2つの配列を作成します。この場合に、配列の大きさは[従業員]テーブルのセレクションのレコード数と一致します。

SELECTION TO ARRAY([従業員]名前 ; arFirst ; [会社]住所 ; arAddress)

▼ 以下の例は、配列“AFileNum”にレコード番号を返し、配列“ANames”に[顧客]名前フィールドを返します。

SELECTION TO ARRAY([顧客] ; AFileNum ; [顧客]名前 ; ANames)

参照

ARRAY TO SELECTION、AUTOMATIC RELATIONS、ON ERR CALL、SELECTION RANGE TO ARRAY

SELECTION RANGE TO ARRAY

SELECTION RANGE TO ARRAY(開始 ; 終了 ; フィールドまたはテーブル ;
配列 ; {フィールド2またはテーブル2 ; 配列2...;フィールドNまたは
テーブルN ; 配列N})

引数	タイプ		説明
開始	数値	→	データ抽出を開始するレコード位置番号
終了	数値	→	データ抽出を終了するレコード位置番号
フィールド またはテーブル	フィールド またはテーブル	→	データ抽出用に使用するフィールドまたは レコード番号抽出用に使用するテーブル
配列	配列	←	フィールドデータまたはレコード番号を 抽出するための配列

説明

SELECTION RANGE TO ARRAYコマンドは1つまたは複数の配列を作成し、その配列にカレントセレクションのフィールドデータまたはレコード番号をコピーします。

配列にカレントセレクション全体を用いる**SELECTION TO ARRAY**コマンドと違って、**SELECTION RANGE TO ARRAY**コマンドは、引数＜開始＞と＜終了＞によって指定されたレコード位置番号の範囲に適用されます。

次の式（1 <= ＜開始＞ <= ＜終了＞ <= **Records in selection**([...])）を満たしている＜開始＞と＜終了＞で指定されるレコード位置番号の範囲のフィールド値またはレコード番号を受け取ります。

もし、以下の式（1 <= ＜開始＞ = ＜終了＞<**Records in selection**([...])）を満たす値を受け渡すと、レコード位置番号が(＜開始＞ = ＜終了＞)であるレコードからフィールドをロードしてくるか、またはレコード番号を取得します。

もし、間違ったレコード位置番号を受け渡すと、このコマンドは以下のようなことを実行します。

■ （＜終了＞ > **Records in selection**([...])）の場合は、＜開始＞によって指定されたレコード位置番号から最後までのレコードを対象とします。

■ （＜開始＞ > ＜終了＞）の場合は、レコード位置番号が＜開始＞であるレコードから値を返します。

■ 両方の引数がセレクションのサイズと一致しない場合は、空の配列が返されます。

SELECTION TO ARRAYコマンド同様、**SELECTION RANGE TO ARRAY**コマンドも第1引数で指定したテーブルのセレクションを用います。

SELECTION RANGE TO ARRAYコマンドは**SELECTION TO ARRAY**コマンド同様、以下のようなことを実行することができます。

■ 1つまたは複数のフィールドから値をロードする

■ シンタックス（[テーブル] ; 配列 ; ...）を使って、レコード番号をロードする

■ テーブル間にn対1の自動リレートが設定されている場合や、**SET AUTOMATIC RELATIONS**を事前に呼び出してn対1のマニュアルリレートを自動に変更した場合は、リレートフィールドの値をロードします。いずれの場合も、複数レベルのn対1リレートを経てテーブルから値がロードされます。

各配列のタイプは、下記の2つの例外を除いてフィールドタイプに依存します。

■ テキストフィールドが文字列配列にコピーされた場合、その配列は文字列配列のままです。

■ 時間フィールドは、倍長整数配列にコピーされます。

注：サブテーブルおよびサブフィールドを指定することはできません。

レコード番号をロードすると、それらは倍長整数配列の中にコピーされます。

4D Server：コマンドは4D Server用に最適化されています。各配列はサーバ上で作成され、配列全体がクライアントマシンに送信されます。

警告：SELECTION RANGE TO ARRAYコマンドは、ロードしてくるデータのタイプやサイズや＜開始＞と＜終了＞で指定した範囲次第では、大きな配列を作成する場合があります。配列はメモリ上に常駐するため、返された配列のサイズを調べることによってそのコマンド実行後の結果を調べたり、またはON ERR CALLコマンドを使用したプロジェクトメソッドを使い、そのコマンドの呼びだしをカバーすることは良いアイデアです。

コマンドが正常に実行されると、返される配列のサイズは（＜終了＞-＜開始＞+1）になります。ただし、引数＜終了＞がセレクションのレコード数より大きい場合は、結果の配列は（Records in selection([...])-＜開始＞+1）個の要素を含みます。

例題

▼ 以下の例は、[送り状]テーブルのカレントセレクションの先頭から50レコードを選択します。そして、[送り状]送り状番号フィールドおよびリレートフィールドの[顧客]顧客番号から値をロードしてきます。

```
SELECTION RANGE TO ARRAY(1 ; 50 ; [送り状]送り状番号 ; arInvoID ;
                         [顧客]顧客番号 ; arCustID)
```

▼ 以下の例は、[送り状]テーブルのカレントセレクションの最終から50レコードを選択します。そして、[送り状]レコードおよび[顧客]リレートレコードのレコード番号をロードしてきます。

```
vSelSize := Records in selection([送り状])
SELECTION RANGE TO ARRAY(vSelSize - 49 ; vSelSize ; [送り状] ; arInvoID ;
                                                    arInvRecNo ; [顧客] ; arCustRecNo)
```

▼ 以下の例は、配列の中にダウンロードできない大きいセレクション内の1000レコードのかたまりを使って処理を行い続けます。

```
vMaxPage := 1000
vSelSize := Records in selection([電話帳])
For($vPage ; 1 ; 1 +(vSelSize - 1) // vMaxPage))
    `値またはレコード番号をロードする
    SELECTION RANGE TO ARRAY(1+(vMaxPage *($vPage - 1)) ;
    vMaxPage * $vPage ; ... ; ... ; ... ; ... ; ... )
    `何らかの配列処理を実行する
End for
```

参照

AUTOMATIC RELATION、ON ERR CALL、SELECTION TO ARRAY、Size of array

ARRAY TO SELECTION

ARRAY TO SELECTION(配列1 ; フィールド1{ ;...; 配列N ; フィールドN})

引数	タイプ		説明
配列	配列	→	セレクションをコピーした配列
フィールド	フィールド	←	配列のデータを格納するフィールド

説明

ARRAY TO SELECTIONコマンドは、1つまたは複数の配列をセレクションのレコードにコピーします。すべてのフィールドは、同一テーブルのものでなければなりません。

セレクションが既存の場合は、配列要素を要素の順番とレコードの順番に従ってレコードに書き込みます。要素の方がレコードよりも多い場合は、新しいレコードを作成します。レコードは、既存でも新規でも、自動的に保存されます。

大きさが異なる配列を複数指定した場合に、最初に指定した配列の大きさが採用されます。もし、他の配列が大きいと、追加された要素は無視され、小さい場合は、追加された要素はヌル値になります。

このコマンドは、**SELECTION TO ARRAY**コマンドとは逆の作業を行います。**ARRAY TO SELECTION**コマンドは、自動リレートが設定されていても、リレートテーブルのフィールドを使用することはできません。

警告：ARRAY TO SELECTIONコマンドは、既存のレコードの情報を上書きします。十分に注意して使用してください。

ARRAY TO SELECTIONコマンド実行中にレコードがロックされると、そのレコードは修正されません。ロックされたレコードは、「LockedSet」というシステムセットの中に置かれます。**ARRAY TO SELECTION**コマンドを実行した後に、任意のレコードがロックされていないかを確認するために「LockedSet」を調べることができます。

4D Server：このコマンドは4D Server用に最適化されています。配列はクライアントマシンからサーバへ渡され、レコードの修正や追加はサーバ上で実行されます。この処理は同時に行われるため、クライアントマシンは処理が正常に終了するまで待機しなくてはなりません。マルチユーザ・マルチプロセス環境では、ロックされたレコードは上書きされません。

例題

▼ 以下の例は、“arFirst”、“arComp”の2つの配列のデータを[従業員]テーブルにコピーします。配列“arFirst”は“従業員]名前”フィールドに、配列“arComp”は“[従業員]会社”フィールドに、それぞれ書き込みます。

ARRAY TO SELECTION(arFirst ; [従業員]名前 ; arComp ; [従業員]会社)

参照

SELECTION TO ARRAY

DISTINCT VALUES

DISTINCT VALUES(フィールド ; 配列)

引数	タイプ		説明
フィールド	フィールド またはサブ フィールド	→	使用するインデックスが使えるフィールド またはサブフィールド
配列	配列	←	フィールドから値を受け取るための配列

説明

DISTINCT VALUESコマンドは、フィールド、またはサブフィールドが属するテーブルのカレントセレクションに対する＜フィールド＞フィールドから得られる繰り返しのない（ユニークな）値で作成された＜配列＞配列を作成します。

DISTINCT VALUESコマンドには、インデックス付けされていなくてもインデックスをサポートしているすべてのインデックス可能なフィールドを渡すことができます。しかし、インデックス付けされていないフィールドでこのコマンドを実行すると遅くなります。また、この場合、コマンドはカレントレコードをアンロードすることに注意してください。

注：このコマンドはインデックス付けされたフィールドおよびインデックス付けされていないフィールドの両方で機能します。SET DATABASE PARAMETERコマンドを用いて実行モードを設定することができます。

テーブルのフィールドを渡たした場合、**DISTINCT VALUES**コマンドは現在選択されたレコード中に存在する繰り返しのない値だけをブラウズし、保持します。しかし、サブフィールドを渡した場合は、**DISTINCT VALUES**は現在選択された各レコードに存在するすべてのサブレコードをブラウズします。

注：4Dのバージョン6.5から、DISTINCT VALUESコマンドはトランザクション中（終了しない間）に呼び出されると、トランザクション中に作成された評価用のレコードを得るようになりました。

呼び出す前に配列を作成した場合、**DISTINCT VALUES**コマンドはユーザが渡すフィールド、またはサブフィールドと互換性を持つ配列のタイプを想定します。一方、インタプリタモードでは、**DISTINCT VALUES**は、適当なタイプの配列を作成します。しかし、フィールド、またはサブフィールドが時間タイプならば、このコマンドは倍長整数の配列で処理または作成します。

呼び出し後、配列のサイズはセレクションの中から見つけられる異なった値の番号と等しいです。このコマンドは、カレントセレクション、またはカレントレコードを変更しません。**DISTINCT VALUES**コマンドはフィールドのインデックスを使用するため、＜配列＞での要素は昇順でソートされて返されます。これが目的の並べ替え順であれば、**DISTINCT VALUES**を使用した後に**SORT ARRAY**コマンドを呼び出す必要はありません。

警告：DISTINCT VALUESコマンドは、セレクションのサイズやレコード中の異なった値によって、大きな配列を作成することがあります。メモリ上での配列のリサイズなので、このコマンドの終了後に結果を確認することはよいことです。そのためには、結果の配列のサイズをテストするか、ON ERR CALLプロジェクトメソッドを使用し、コマンドの呼び出しをカバーします。

4D Server：このコマンドは、4D Server用に最適化されています。配列をサーバ側で作成し、値が計算され、その全体をクライアントに送ります。

例題

▼ 以下の例は、カレントセレクションから都市のリストを作成します。そして、会社の店舗がある都市の数字を求めます。

```
ALL RECORDS([Retail Outlets])  ` レコードのセレクションの作成
   DISTINCT VALUES([Retail Outlets]City ; asCities)
ALERT("この会社が店を持っている都市は " +String(Size of array(asCities))
                                          +" 都市です。")
```

▼ 以下の例は、asKeywordに、テーブル[Documentation]に保存された4D Writeドキュメントに（サブテーブルを用いて）付けられたもので、テーマが「経済」であるすべてのキーワードを返します。

```
QUERY([Documentation] ; [Documentation]Theme="経済")
   DISTINCT VALUES([Documentation]Keywords'Keyword ; asKeywords)
```

この配列が構築された後は、選択されたキーワードが割り当てられたすべてのドキュメントの位置を速やかに見つけるために再利用できます。

```
QUERY([Documentation] ; [Documentation]Keywords'Keyword =
                                          asKeywords{asKeywords})
   SELECTION TO ARRAY([Documentation]Subject ; asSubjects)
      ` ...
```

参照

ON ERR CALL、SELECTION RANGE TO ARRAY、SELECTION TO ARRAY、SET DATABASE PARAMETER

LONGINT ARRAY FROM SELECTION

LONGINT ARRAY FROM SELECTION(テーブル;レコード番号配列{;セレクション})

引数	タイプ		説明
テーブル	テーブル	→	カレントセレクションのテーブル
レコード番号配列	倍長整数配列	←	レコード番号の配列
セレクション名	文字列	→	命名セレクション この引数を省略した場合は現在の カレントセレクションの名前

説明

このコマンドは、セレクション名に属している各レコードに対する（絶対）レコード番号の配列を、レコード番号配列として作成します。

セレクション名引数を省略した場合、コマンドはテーブルのカレントセレクションを使用します。

注：配列要素0は-1に初期化されます。

参照

CREATE SELECTION FROM ARRAY

BOOLEAN ARRAY FROM SET

BOOLEAN ARRAY FROM SET(ブール配列{; セット })

引数	タイプ		説明
ブール配列	ブール配列	←	レコードがセット内にあるかどうかを示す配列
セット	文字列	→	セットの名前 この引数を省略した場合UserSetを使用

説明

このコマンドは、テーブル内の各レコードが指定されたセットに含まれているかそうでないかの、ブール配列を作成します。ブール配列には、セットに含まれているレコード数より多くの要素数があります。配列内の要素は、レコードがテーブル内で作成された順番で並べられています。

配列の各要素は、テーブルに作成されたレコードと同じ順序で整列されます（レコード番号順）。Nがテーブルに含まれるレコード数とすると、配列の 0 番目の要素は、レコード番号0のレコードにあたり、配列のエレメント1はレコード番号1のレコードにあたります。

配列の各要素は以下の通りです。

■ 対応するレコードがセットに属している場合は、True

■ 対応するレコードがセットに属さない場合は、False

配列要素0はレコード番号0に対応し、配列要素1はレコード番号1に対応します。引数セットを指定しない場合、コマンドは現在のプロセスのUserSetを使用します。

注意：配列“booleanArr”の総要素数は重要ではありません。構造上の理由により、この数はテーブル上の実存のレコード数と異なっています。起こりうる余分の要素はFalseにセットされます。

セット引数を渡さない場合は、コマンドはカレントプロセスのUserSetを使います。

参照

CREATE SELECTION FROM ARRAY

BACKUP

BACKUP

引数	タイプ	説明

このコマンドに引数は必要ありません。

説明

BACKUPコマンドは、現在のバックアップ設定を用いて、データベースのバックアップを開始します。確認ダイアログボックスは表示されませんが、進捗バーが画面上に現れます。

バックアップ設定は、アプリケーションの「環境設定」で設定します。これらの設定は、データベースの「Preferences/backup」サブフォルダ内にある「Backup.XML」ファイルにも保存されます。

BACKUPコマンドは、実行前に「**On backup startup**」データベースメソッドを呼び出し、実行終了時に「**On backup shutdown**」データベースメソッドを呼び出します。

このようなメカニズムなので、このコマンドを「**On backup startup**」と「**On backup shutdown**」データベースメソッドからコールすべきではありません。

4D Server：クライアントマシンから呼び出された場合、BACKUPコマンドはストアドプロシージャとみなされ、サーバー上で実行されます。

参照

GET BACKUP INFORMATION、「On Backup Startup」データベースメソッド、RESTORE

システム変数またはセット

バックアップが正常に終了すると、システム変数OKには値1が代入され、それ以外の場合は値0が代入されます。

エラー処理

障害が発生した場合はエラーが生成されます。このエラーは**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて遮ることができます。

GET BACKUP INFORMATION

GET BACKUP INFORMATION(セレクタ; 情報1; 情報2)

引数	タイプ		説明
セレクタ	倍長整数	→	取得する情報のタイプ
情報1	日付¦整数	←	セレクタの値1
情報2	日付¦整数	←	セレクタの値2

説明

GET BACKUP INFORMATIONコマンドを使用し、データベースのデータに関して行われた前回のバックアップに関連する情報を取得することができます。

＜セレクタ＞には取得する情報タイプを渡します。この引数の値として、「Backup」テーマ内にある次の定数のいずれかを使用することができます。

定数	タイプ	値
Last Backup Date	倍長整数	0
Last Backup Status	倍長整数	2
Next Backup Date	倍長整数	4

引数＜情報1＞と＜情報2＞のタイプおよび内容は、＜セレクタ＞の値によって決まります。

■ ＜セレクタ＞=0：＜情報1＞には前回のバックアップ日付、＜情報2＞には前回のバックアップ時間が返されます。

■ ＜セレクタ＞=2：＜情報1＞には前回のバックアップステータスの数値、＜情報2＞にはそのテキストが返されます。

■ ＜セレクタ＞=4：＜情報1＞には次回の定期バックアップ日付、＜情報2＞には次回の定期バックアップ時間が返されます。

参照

RESTORE

RESTORE

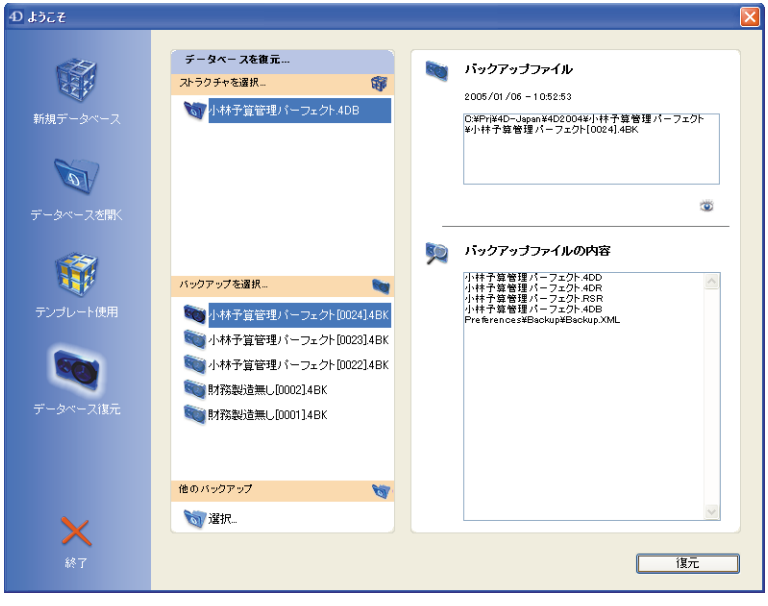
RESTORE

引数　　タイプ　　説明

このコマンドに引数は必要ありません。

説明

RESTOREコマンドは4th Dimensionの「ようこそ」ダイアログボックスの「復元」ページを表示します。



ユーザはここで復元するアーカイブを選択することができます。

このコマンドは、カスタマイズされたインタフェースを使用してバックアップを管理する際に役立ちます。

参照

BACKUP、GET RESTORE INFORMATION

GET RESTORE INFORMATION

GET RESTORE INFORMATION(セレクタ; 情報1; 情報2)

引数	タイプ		説明
セレクタ	倍長整数	→	取得する情報のタイプ
情報1	日付 ¦ 整数	←	セレクタの値1
情報2	日付 ¦ 整数	←	セレクタの値2

説明

GET RESTORE INFORMATIONコマンドを使用し、前回の復元に関連する情報を取得することができます。

＜セレクタ＞には取得する情報タイプを渡します。この引数の値として、「Backup」テーマ内にある次の定数のいずれかを使用することができます。

定数	タイプ	値
Last Restore Date	倍長整数	0
Last Restore Status	倍長整数	2

引数＜情報1＞と＜情報2＞のタイプおよび内容は、＜セレクタ＞の値によって決まります。

■ ＜セレクタ＞=0：＜情報1＞には前回のデータベース復元日付、＜情報2＞には前回のデータベース復元時間が返されます。

■ ＜セレクタ＞=2：＜情報1＞には前回のデータベース復元のステータスを表わす数値が、＜情報2＞にはそのテキストが返されます。

参照

RESTORE

「On Backup Startup」データベースメソッド

このデータベースメソッドは、データベースのバックアップを開始しようとするたびに呼び出されます（手動でのバックアップ、定期的自動バックアップ、または**BACKUP**コマンドによるバックアップ）。

これは、すべての4D環境、つまり4th Dimension、4D Server、4D Client、4D Runtime、ならびに4D Engineが組み込まれたデータベースに関係します。

「**On Backup Startup**」データベースメソッドを使用すると、開始されたバックアップを検証することができます。このメソッドにおいて、バックアップの許可または拒否を示す値を引数$0に返してください。

■ $0= 0：バックアップを開始することができます。

■ $0# 0：バックアップは許可されません。処理はキャンセルされ、エラーが返されます。このエラーは、**GET BACKUP INFORMATION**コマンドを使用して取得することができます。

このデータベースメソッドを使用すると、バックアップの実行状況を確認することができます（ユーザ、前回のバックアップ日付など）。

引数$0（倍長整数タイプ）は、データベースメソッド内で宣言しなくてはなりません。

```
C_LONGINT($0)
```

参照

BACKUP、「On Backup Shutdown」データベースメソッド

「On Backup Shutdown」データベースメソッド

このデータベースメソッドは、データベースのバックアップが終了するたびに呼び出されます。バックアップが中止される原因として、コピーの終了やユーザによる中断、エラーがあります。

これは、すべての4D環境、つまり4th Dimension、4D Server、4D Client、4D Runtime、ならびに4D Engineが組み込まれたデータベースに関係します。

「**On backup shutdown**」データベースメソッドを使用すると、バックアップが正常に実行されたかどうかを確認できます。バックアップが完了すると、このメソッド内の引数$1にはバックアップのステータスを示す値が返されます。

■ バックアップが正常に終了すると、引数$1には0が代入されます。

■ バックアップが中断された場合、引数$1には0以外が代入されます。

他にも、「**On Backup Startup**」データベースメソッド（$0# 0）によりバックアップが拒否された場合は、引数$0へ実際に返された値が引数$1に代入されます。これにより、独自のエラー管理システムを実装することができます。

引数$1（倍長整数タイプ）は、データベースメソッド内で宣言しなくてはなりません。

```
C_LONGINT($1)
```

参照

BACKUP、「On Backup Startup」データベースメソッド

BLOBの定義

4th Dimensionバージョン6では、BLOB（Binary Large OBjects）データタイプが新しく追加されました。

BLOBフィールドおよびBLOB変数は、以下のように定義します。

- BLOBフィールドを作成するには、「フィールドプロパティ」ウインドウ内の「フィールドタイプ」ドロップダウンリストでBLOBを選択します。
- BLOB変数を作成するには、コンパイラ宣言コマンド**C_BLOB**コマンドを使用します。タイプがBLOBのローカル変数、プロセス変数、インタープロセス変数を作成することができます。

注：BLOBの配列はありません。

4th Dimensionの中では、1つのBLOBは連続した可変長バイトであり、1つのまとまったオブジェクトまたは各バイトが個々にアドレス指定できるオブジェクトとして取り扱うことができます。1つのBLOBは空（長さがNUL）でもよく、また最大2,147,483,647バイト（2GB）まで含むことができます。

BLOBとメモリ

BLOBは、そっくりそのままメモリの中にロードされます。BLOB変数は、メモリ内にだけ保持され、存在します。BLOBフィールドは、そのフィールドが属すレコードの他の部分と同様に、ディスクからメモリにロードされます。

大量のデータを保持できる他のフィールドタイプ（ピクチャフィールドやサブテーブルフィールド）と同様に、レコードを更新してもBLOBフィールドはメモリに複製されません。その結果、**Old**関数および**Modified**関数をBLOBフィールドに適用しても、返される結果は意味を持ちません。

BLOBの表示

1つのBLOBには、どのような種類のデータでも保持できるため、画面上でのデフォルトの表現はありません。フォーム内でBLOBフィールドまたは変数を表示すると、どのような内容であっても常に空白になります。

BLOBフィールド

BLOBフィールドを使用すると、最大で2GB（ギガバイト）までのあらゆる種類のデータでも保存することができます。BLOBフィールドにインデックス付けすることはできないため、BLOBフィールドに保存された値のレコードを検索するには、式を使用しなければなりません。検索処理を使用して保存するデータを素早く取得したい場合は、BLOBを使用してはいけません。例えば、BLOBフィールドには、キーワードを格納してはいけません。その代わりに、キーワードサブフィールドをインデックス付けできるサブテーブルを使用します。

引数（パラメータ）の受け渡し、ポインタおよび関数の結果

4th DimensionのBLOBは、BLOB引数を想定している4Dコマンドまたは4Dプラグインの引数として受け渡すことができます。一方では、BLOBをユーザメソッドのパラメータとして受け渡すことはできません。BLOBは関数の結果として戻すことができません。

BLOBを自分の独自のメソッドに渡すには、そのBLOBへのポインタを定義し、そのポインタをパラメータとして受け渡します。

例題

```
`タイプがBLOBの変数を宣言する
C_BLOB(anyBlobVar)
` BLOBは、パラメータとして任意の4Dコマンドに受け渡される
SET BLOB SIZE(anyBlobVar ; 1024*1024)
` BLOBは、パラメータとしてプラグインに受け渡される
$errCode:= Do Something With This BLOB(anyBlobVar)
` BLOBへのポインタは、ユーザメソッドへパラメータとして渡される
COMPUTE BLOB(->anyBlobVar )
` タイプがポインタの変数を宣言する
C_POINTER(aPointer)
` BLOBへのポインタを定義する
aPointer :=->anyBlobVar
` BLOBへのポインタは、ユーザメソッドへパラメータとして渡される
COMPUTE BLOB(aPointer)
```

4Dプラグイン開発者のための注意点：BLOB引数は“&O”（数字の"0"ではなく、アルファベットの"O"）として宣言されます。

BLOBの割り当て

BLOBを相互に割り当てることができます。

例題

```
`タイプがBLOBの2つの変数を宣言する
C_BLOB(vBlobA ; vBlobB)
`最初のBLOBのサイズを10Kに設定する
SET BLOB SIZE(vBlobA ; 10*1024)
` 最初のBLOBを2番目のBLOBに割り当てる
vBlobB:=vBlobA
```

ただし、BLOBに演算子を適用することはできません。BLOBタイプの式はありません。

BLOBの内容のアドレス指定

中カッコ{...}を使用すれば、BLOBの各バイトは、個別にアドレス指定できます。1つのBLOB内では、各バイトには0～N-1の番号が割り当てられています。NはBLOBのサイズです。例は、以下のとおりです。

```
` BLOBのタイプの変数を宣言する
C_BLOB(vBlob)
` BLOBのサイズを256バイトに設定する
SET BLOB SIZE(vBlob ; 256)
` このループは、BLOBの256バイトをゼロに初期化する
For( vByte ; 0 ; BLOB size(vBlob)-1)
      vBlob{vByte}:=0
End for
```

BLOBの各バイトはすべて、個別にアドレス指定できるため、BLOBフィールドまたはBLOB変数に格納したいものは実際には何でも格納できます。

BLOBの4th Dimensionコマンド

4th Dimensionには、作業しているBLOBに対し、以下のようなコマンドがあります。

■ **SET BLOB SIZE**コマンドは、BLOBフィールドやBLOB変数のサイズを変更します。

■ **BLOB size**関数は、BLOBのサイズを戻します。

■ **DOCUMENT TO BLOB**コマンドおよび**BLOB TO DOCUMENT**コマンドを使用すると、ドキュメント全体をBLOBからロード、またはBLOBに書き込むことができます（オプションで、Macintosh上のデータフォークおよびリソースフォークとの間でもこの操作が可能です）。

■ **VARIABLE TO BLOB**コマンドおよび**BLOB TO VARIABLE**コマンドを使用すると、**LIST TO BLOB**コマンドおよび**BLOB to list**関数と同様に、4D変数をBLOBに格納、または取り出すことができます。

■ **COMPRESS BLOB**コマンド、**EXPAND BLOB**コマンド、および**BLOB PROPERTIES**コマンドを使用すると、圧縮されたBLOBを操作することができます。

■ **BLOB to integer**関数、**BLOB to longint**関数、**BLOB to real**関数、**BLOB to text**関数、**INTEGER TO BLOB**コマンド、**LONGINT TO BLOB**コマンド、**REAL TO BLOB**コマンド、および**TEXT TO BLOB**コマンドを使用すると、ディスク、リソース、OS等から入力される構造化されたデータを操作することができます。

■ **DELETE FROM BLOB**コマンド、**INSERT IN BLOB**コマンド、および**COPY BLOB**コマンドを使用すると、BLOB内にある大きいサイズのデータのまとまりをすばやく処理することができます。

■ **ENCRYPT BLOB**と**DECRYPT BLOB**により4Dデータベース上のデータの暗号化と解読ができます。

これらのコマンドについては、この章で説明しています。

追記：

■ **C_BLOB**コマンドは、タイプがBLOBの変数を宣言します。詳細は、「コンパイラコマンド」の章を参照してください。

■ **GET CLIPBOARD**コマンドおよび**APPEND CLIPBOARD**コマンドを使用すると、クリップボードに格納されているどのデータタイプでも操作できます。詳細は、「クリップボードコマンド」の章を参照してください。

■ **GET RESOURCE**コマンドおよび**SET RESOURCE**コマンドを使用すると、ディスク上に格納されているリソースでも操作できます。詳細は、「リソースコマンド」の章を参照してください。

■ **SEND HTML BLOB**はウェブブラウザ上にどのようなタイプのデータでも送ることができます。詳細は「Webサーバコマンド」の章を参照してください。

■ **PICTURE TO BLOB**、**BLOB TO PICTURE**と**PICTURE TO GIF**により画像を開いたり、変換することができます。詳細は「ピクチャコマンド」の章を参照してください。

■ **GENERATE ENCRYPTION KEYPAIR**と**GENERATE CERTIFICATE REQUEST**は接続プロトコルをセキュリティ化したSSL(Secured Socket Layer)により使われるコマンドです。詳細は「セキュアドプロトコル」の章を参照してください。

SET BLOB SIZE

SET BLOB SIZE(blob ; サイズ {;フィラー})

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
サイズ	数値	→	BLOBの新しいサイズ
フィラー	数値	→	フィラー文字のASCIIコード

説明

SET BLOB SIZEコマンドは、引数＜サイズ＞に渡された値に従って、BLOB＜blob＞のサイズを変更します。

BLOBに新しいバイトを割り当て、それらのバイトを特定の値で初期化したい場合には、その値（0～255）をオプション引数の＜フィラー＞に渡します。

例題

1. 大きなBLOBプロセス変数またはBLOBインタープロセス変数を終了した後、占有していたメモリを解放することをお勧めします。そのためには、以下のように記述します。

```
SET BLOB SIZE(aProcessBLOB ; 0)
SET BLOB SIZE(<>anInterprocessBLOB ; 0)
```

2　以下の例では、0xFFが埋め込まれた16KBのBLOBが1つ作成されます。

```
C_BLOB(vxData)
SET BLOB SIZE(vxData ; 16*1024 ; 0xFF)
```

参照

BLOB size

エラー処理

メモリ不足のためにBLOBのサイズを変更できない場合には、エラーコード-108が発生します。**ON ERR CALL**割り込みメソッドを使用すれば、このエラーを取り出すことができます。

BLOB size

BLOB size(blob) → 数値

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
戻り値	数値	←	BLOBのサイズ（バイト単位）

説明

BLOB size関数は、バイトで表現された＜blob＞のサイズを返します。

例題

▼ 以下の例は、“myBlob”BLOBに100バイトが追加されます。

```
SET BLOB SIZE(BLOB size(myBlob)+100)
```

参照

SET BLOB SIZE

COMPRESS BLOB

COMPRESS BLOB(blob {;圧縮})

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
圧縮	数値	←	省略しなかった場合、 1=できるだけ小さく圧縮する 2=できるだけ速く圧縮する

説明

COMPRESS BLOBコマンドは、4th Dimension内部の圧縮アルゴリズムを使用して、引数＜blob＞に渡されたBLOBを圧縮します。

注：このコマンドは、サイズが254バイトを超えるBLOBだけを圧縮します。

オプション引数の＜圧縮＞を使用すると、BLOBを圧縮する方法を設定できます。

- 1を渡すと、圧縮および解凍の操作の速度と引き換えに、BLOBができるだけ小さく圧縮されます。
- 2を渡すと、圧縮率と引き換えに（圧縮されたBLOBのサイズは大きくなります）、BLOBができるだけ速く圧縮されます（展開の速度もできるだけ速くなります）。
- 他の値を渡す、または引数を省略すると、圧縮モード1を使用してBLOBができるだけ小さく圧縮します。

4th Dimensionには、以下のようにあらかじめ定義された定数があります。

定数	タイプ	値
Compact compression mode	倍長整数	1
Fast compression mode	倍長整数	2

呼び出し後、BLOBが正常に圧縮された場合には、システム変数OKは1に設定されます。BLOBを圧縮するために必要なメモリがない、またはサイズが255バイト未満である等の理由で圧縮が実行できなかった場合には、システム変数OKは0に設定されます。その他の場合はエラー10600になります。このエラーは、**ON ERROR CALL**コマンドを用いて取得することができます。

圧縮されたBLOBは、**EXPAND BLOB**コマンドを使用して解凍することができます。

BLOBが正常に圧縮されたかどうかを確認するには、**BLOB PROPERTIES**コマンドを使用します。

警告：圧縮されてもBLOBは依然BLOBであるため、その内容を更新できなくなるわけではありません。ただし、圧縮されたBLOBの内容を更新すると、EXPAND BLOBコマンドでBLOBを正しく解凍することができません。

例題

▼ 以下の例では、“vxMyBlob”BLOBが圧縮されたかどうかテストし、圧縮されていない場合には、これを圧縮します。

```
BLOB PROPERTIES(vxMyBlob ; $vlCompressed ; $vlExpandedSize ;
                                                $vlCurrentSize)
If($vlCompressed=Is not compressed)
    COMPRESS BLOB(vxMyBlob)
End if
```

ただし、既に圧縮されているBLOBに対して**COMPRESS BLOB**を実行すると、このコマンドはこれを認識し、何も実行しません。

▼ 以下の例では、ドキュメントを選択して、これを圧縮します。

```
$vhDocRef := Open document("")
If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    DOCUMENT TO BLOB(Document ; vxBlob)
    If(OK=1)
        COMPRESS BLOB(vxBlob)
        If(OK=1)
            BLOB TO DOCUMENT(vxBlob ; Document)
        End if
    End if
End if
```

参照

BLOB PROPERTIES、EXPAND BLOB

システム変数またはシステムセット

BLOBが正常に圧縮された場合には、システム変数OKは1に設定されます。そうでない場合には、0に設定されます。

EXPAND BLOB

EXPAND BLOB(blob)

引数	タイプ		説明
blob	BLOB	→	解凍するBLOB

説明

EXPAND BLOBコマンドは、**COMPRESS BLOB**コマンドを使用して既に圧縮されている＜blob＞を解凍します。

呼び出し後、BLOBが解凍された場合は、システム変数OKは1に設定されます。BLOBが解凍できなかった場合は、システム変数OKは0に設定されます。

メモリ不足で解凍できない場合は、エラーが表示されず、メソッドはその解凍を再度行います。その他の場合（例えば、BLOBが圧縮されていない、またはダメージを受けている場合）は、10600のエラーを返します。このエラーは、**ON ERR CALL**コマンドで取得することができます。

BLOBが圧縮されているかどうかを確認するには、**BLOB PROPERTIES**コマンドを使用します。

例題

▼以下の例では、“vxMyBlob”BLOBが圧縮されたかどうかテストし、圧縮されている場合には、これを展開します。

```
BLOB PROPERTIES(vxMyBlob ; $vlCompressed ; $vlExpandedSize ;
                                                $vlCurrentSize)
If($vlCompressed # Is not compressed)
    EXPAND BLOB(vxMyBlob)
End if
```

▼以下の例では、ドキュメントを選択し、そのドキュメントが圧縮されている場合には解凍します。

```
$vhDocRef := Open document("")
If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    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(vxBlob ; Document)
                    End if
                End if
            End if
        End if
```

参照

BLOB PROPERTIES、COMPRESS BLOB

システム変数またはシステムセット

BLOBが正常に解凍された場合には、システム変数OKは1に設定されます。そうでない場合には、0に設定されます。

BLOB PROPERTIES

BLOB PROPERTIES(blob ; 圧縮 {;サイズ {現在サイズ}})

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
圧縮	数値	←	0=BLOBは圧縮されていない 1=できるだけ小さく圧縮されたBLOB 2=できるだけ速く圧縮されたBLOB
サイズ	数値	←	圧縮されてない場合のBLOBサイズ
現在サイズ	数値	←	BLOBの現在サイズ（バイト単位）

説明

BLOB PROPERTIESコマンドは、引数＜blob＞に関する情報を返します。

■ 引数＜圧縮＞はBLOBが圧縮されたかどうかを示し、以下のようなあらかじめ定義された定数の1つの値を返します。

定数	タイプ	値
Is not compressed	倍長整数	0
Compact compression mode	倍長整数	1
Fast compression mode	倍長整数	2

■ BLOBの圧縮状態がどのようなものであっても、引数＜サイズ＞は、圧縮されていない時のBLOBのサイズを戻します。

■ 引数＜現在サイズ＞は、BLOBの現在のサイズを戻します。BLOBが圧縮されている場合には、通常、＜サイズ＞より小さい＜現在サイズ＞を取得します。BLOBが圧縮されていない場合には、常に、＜サイズ＞に等しい＜現在サイズ＞を取得します。

例題

▼ **COMPRESS BLOB**コマンドおよび**EXPAND BLOB**コマンドの例を参照してください。

▼ BLOBが圧縮された後、以下のプロジェクトメソッドはこの圧縮でできた空間の割合を取得します。

```
`「Space saved by compression」プロジェクトメソッド
` Space saved by compression(ポインタ {; ポインタ } ) → 倍長整数
` Space saved by compression(-> BLOB {; ->savedBytes } ) → 割合
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
```

このメソッドがアプリケーションに追加された後は、これを以下のように使用することができます。

```
COMPRESS BLOB(vxBlob)
$vlPercent:= Space saved by compression(->vxBlob ; ->vlBlobSize)
ALERT("圧縮サイズと割合："+String(vlBlobSize)+"バイト"+String
                                    ($vlPercent;"#0%")+"%")
```

参照

COMPRESS BLOB、EXPAND BLOB

DOCUMENT TO BLOB

DOCUMENT TO BLOB(ドキュメント ; blob {; *})

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントファイルの名前
blob	BLOB	→	ドキュメントファイルを受信するBLOBフィールドまたはBLOB変数
		←	ドキュメントファイルの内容
*	*	→	Macintosh上でのみ： *が指定された場合は、リソースフォーク、それ以外は、データフォークがロードされる

説明

DOCUMENT TO BLOBコマンドは、ドキュメントの内容全体を引数<blob>にロードします。まだ開かれていない既存のドキュメントの名前を渡す必要があり、それ以外の場合には、エラーが発生します。BLOBへロードするドキュメントをユーザが選択できるようにするには、**Open document**関数およびプロセス変数Documentを使用します（下の例を参照してください)。

Macintoshでの注意点：

Macintoshのドキュメントファイルは、データフォークおよびリソースフォークの2つのフォークから構成されています。デフォルトでは、**DOCUMENT TO BLOB**コマンドはドキュメントファイルのデータフォークをロードします。ドキュメントファイルのリソースフォークをロードするには、オプション引数<*>を渡します。Windowsでは、オプション引数<*>は無視されます。4D環境では、Macintoshのリソースフォークと同等のものをWindowsで実現することに注意してください。例えば、4Dデータベースのデータフォークは、「.4DB」ファイル拡張子のあるファイルに格納され、リソースフォークは、同じ名前でファイル拡張子が「.RSR」のファイルに格納されます。Windowsでは、データフォークおよびリソースフォークをBLOBに格納して4Dアプリケーションを記述する場合には、操作したい方のフォークに対応するファイルにアクセスするだけですみます。

例題

▼以下の例は、ドキュメントファイルをすばやく格納、または取得できるような情報システムを記述する場合を想定します。データ入力フォームで、ドキュメントファイルをBLOBフィールドにロードできるようなボタンを作成します。このボタンに以下のようなメソッドを作成します。

```
$vhDocRef:=Open document("") `目的のドキュメントを選択する
If(OK=1) `ドキュメントが選択されている場合、
    CLOSE DOCUMENT($vhDocRef) `開いたままにしておく必要はない
    DOCUMENT TO BLOB(Document; [テーブル1]BLOBフィールド)
                                        `ドキュメントをロードする
    If(OK=0)
        `エラー処理を実行する
    End if
End if
```

参照

BLOB TO DOCUMENT、Open document

システム変数

ドキュメントファイルが正常にロードされた場合には、OK変数は1に設定されます。そうでない場合には、0に設定され、エラーが発生します。

エラー処理

■存在しないドキュメントファイルや、既に他のプロセスやアプリケーションで開かれているドキュメントファイルを（BLOBに）ロードしようとすると、それぞれに対応するファイルマネージャエラーが発生します。

■ドキュメントファイルがロックされていたり、ロックされているボリュームにあったり、ドキュメントファイルを読み込む際に問題が発生すると、I／Oエラーが発生する場合があります。

■メモリ不足のためにドキュメントファイルをロードできない場合には、エラーコード-108が発生します。

いずれの場合でも、**ON ERR CALL**割り込みメソッドを使用すれば、このエラーをトラップできます。

BLOB TO DOCUMENT

BLOB TO DOCUMENT(ドキュメント ; blob {; *})

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントファイルの名前
blob	BLOB	→	ドキュメントファイルの新しい内容
*	*	←	Macintosh上でのみ： *が指定された場合は、リソースフォーク、それ以外は、データフォークがロードされる

説明

BLOB TO DOCUMENTコマンドは、＜blob＞に格納されているデータを使用してドキュメントファイルの内容全体を上書きします。ドキュメントファイルをユーザが選択できるようにするには、**Open document**関数または**Create document**関数、およびプロセス変数documentを使用します（例を参照してください）。

Macintoshでの注意点：

Macintoshのドキュメントファイルは、データフォークおよびリソースフォークの2つのフォークから構成されています。デフォルトでは、**DOCUMENT TO BLOB**コマンドはドキュメントファイルのデータフォークをロードします。ドキュメントファイルのリソースフォークをロードするには、オプション引数＜*＞を渡します。Windowsでは、オプション引数＜*＞は無視されます。4D環境では、Macintoshのリソースフォークと同等のものをWindowsで実現することに注意が必要です。例えば、4Dデータベースのデータフォークは、「.4DB」ファイル拡張子のあるファイルに格納され、リソースフォークは、同じ名前でファイル拡張子が「.RSR」のファイルに格納されます。Windowsでは、データフォークおよびリソースフォークをBLOBに格納して4Dアプリケーションを記述する場合には、操作したい方のフォークに対応するファイルにアクセスするだけですみます。

例題

▼以下の例は、ドキュメントファイルをすばやく格納、または取得できるような情報システムを記述する場合を想定します。データ入力フォームで、BLOBフィールドにロードされているデータが含まれているドキュメントファイルを保存できるようなボタンを作成します。このボタンのメソッドは、以下のように作成します。

```
$vhDocRef:=Create document("") `目的のドキュメントを保存する
If(OK=1) `ドキュメントが作成された場合、
    CLOSE DOCUMENT($vhDocRef) `開いたままにしておく必要はない
    DOCUMENT TO BLOB(Document; [テーブル1]BLOBフィールド)
                                    `ドキュメントの内容を記述する
    If(OK=0)
        `エラー処理を実行する
    End if
End if
```

参照

Create document、DOCUMENT TO BLOB、Open document

システム変数

ドキュメントが正常にロードされた場合には、システム変数OKは1に設定されます。そうでない場合には、0に設定され、エラーが発生します。

エラー処理

■存在しないドキュメントや、既に他のプロセスやアプリケーションで開かれているドキュメントを（BLOBに）ロードしようとすると、それぞれに対応するファイルマネージャエラーが発生します。

■ドキュメントがロックされていたり、ロックされているボリュームにあったり、ドキュメントを書き込む際に問題が発生すると、I／Oエラーが発生する場合があります。

■メモリ不足のためにドキュメントをロードできない場合には、エラーコード-108が発生します。

いずれの場合でも、**ON ERR CALL**割り込みメソッドを使用すれば、このエラーをトラップできます。

VARIABLE TO BLOB

VARIABLE TO BLOB(変数 ; blob {; *})

引数	タイプ		説明
変数	変数	→	BLOBの中に格納する変数
blob	BLOB	→	変数を受けとるBLOB
*	文字	→	値を追加する場合には*

説明

VARIABLE TO BLOBコマンドは、＜変数＞を＜blob＞に格納します。

オプション引数＜*＞を指定した場合には、変数はBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜*＞を使用すれば、BLOBがメモリ容量内であれば変数やリスト（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜*＞を指定しない場合には、変数はBLOBの先頭に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

VARIABLE TO BLOBコマンドは、以下のものを除いて、どのようなタイプの変数でも（他のBLOBも）受け付けます。

■ ポインタ

■ ポインタ配列

■ 2次元配列

ただし、階層リスト（ListRef）への参照である倍長整数の変数を格納した場合には、**VARIABLE TO BLOB**コマンドは階層リストではなく倍長整数の変数を格納します。BLOB内に階層リストを格納、またはBLOBから階層リストを取り出すには、**LIST TO BLOB**コマンドおよび**BLOB to list**関数を使用します。

警告：BLOBを使用して変数を格納すると、変数は4D内部形式を使用してBLOBに格納されるため、格納されたBLOBの内容を読み出すにはBLOB TO VARIABLEコマンドを使用しなければなりません。

呼び出し後、変数が正常に格納された場合には、システム変数OKは1に設定されます。変数を格納するために必要なメモリがない、等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォームからの独立性に関する注意点：

VARIABLE TO BLOBコマンドおよび**BLOB TO VARIABLE**コマンドは、4D内部形式を使用してBLOB内に格納された変数を処理します。この利点は、これら2つのコマンドを使用する際にプラットフォーム間でのバイトのスワップを配慮する必要がないところです。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

例題

1. 以下の2つのプロジェクトメソッドを使用すると、ディスク上のドキュメントへすばやく配列を格納、またはドキュメントからすばやく配列を取得できます。

```
`「配列保存」プロジェクトメソッド
` 配列保存(文字列 ; ポインタ)
` 配列保存(ドキュメント ; ->配列)
C_STRING(255 ; $1)
C_POINTER($2)
C_BLOB($vxArrayData)
VARIABLE TO BLOB($2-> ; $vxArrayData) ` BLOBの中に配列を格納する
COMPRESS BLOB($vxArrayData) ` BLOBを圧縮する
BLOB TO DOCUMENT($1 ; $vxArrayData) ` ディスク上にBLOBを保存する

`「配列読み込み」プロジェクトメソッド
` LOAD  ARRAY( 文字列 ; ポインタ )
` LOAD  ARRAY( ドキュメント ; ->配列 )
C_STRING(255;$1)
C_POINTER($2)
C_BLOB($vxArrayData)
DOCUMENT TO BLOB($1 ; $vxArrayData) ` ディスクからBLOBを読み込む
EXPAND BLOB($vxArrayData) `BLOBを解凍する
BLOB TO VARIABLE($vxArrayData ; $2->) `BLOBから配列を取り出す
```

上記のメソッドをアプリケーションに追加すれば、以下のように記述することができます。

```
ARRAY STRING(...;asAnyArray;...)
      ` ...
配列保存( $vsDocName ; ->asAnyArray)
      ` ...
配列読み込み( $vsDocName ; ->asAnyArray)
```

2. 以下の2つのプロジェクトメソッドを使用すると、BLOBへすばやく変数のセットを格納、またはBLOBからすばやく変数のセットを取得することができます。

```
`「STORE VARIABLES INTO BLOB」プロジェクトメソッド
` STORE VARIABLES INTO BLOB( ポインタ { ; ポインタ ... { ; ポインタ } } )
` STORE VARIABLES INTO BLOB( BLOB { ; 変数1 ... { ; 変数2 } } )
C_POINTER(${1})
C_LONGINT($vlParam)
SET BLOB SIZE($1-> ; 0)
For($vlParam ; 2 ; Count parameters)
    VARIABLE TO BLOB(${$vlParam}-> ; $1-> ; *)
End for

`「RETRIEVE VARIABLES FROM BLOB」プロジェクトメソッド
` RETRIEVE VARIABLES FROM BLOB(ポインタ { ; ポインタ ... { ; ポインタ}})
` RETRIEVE VARIABLES FROM BLOB(BLOB { ; 変数1 ... { ; 変数2}})
C_POINTER(${1})
C_LONGINT($vlParam ; $vlOffset)
$vlOffset:=0
For($vlParam ; 2 ; Count parameters)
    BLOB TO VARIABLE($1-> ; ${$vlParam}-> ; $vlOffset)
End for
```

これらのメソッドをアプリケーションに追加すれば、以下のように記述することができます。

```
STORE VARIABLES INTO BLOB(->vxBLOB ; ->vgPicture ; ->asAnArray ;
                                                    ->alAnotherArray)
` ...
RETRIEVE VARIABLES FROM BLOB(->vxBLOB ; ->vgPicture ; ->asAnArray ;
                                                    ->alAnotherArray)
```

参照

BLOB to list、BLOB TO VARIABLE、LIST TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定されます。

BLOB TO VARIABLE

BLOB TO VARIABLE(blob ; 変数 {;オフセット})

引数	タイプ		説明
blob	BLOB	→	4D変数を含んだBLOB
変数	変数	←	BLOBの内容を上書きする変数
オフセット	数値	→	BLOB内の変数の位置
		←	BLOB内の以下の変数の位置

説明

BLOB TO VARIABLEコマンドは、引数＜オフセット＞で指定されたバイトオフセット（ゼロから開始）にある＜blob＞に格納されているデータを使用して＜変数＞を上書きします。

BLOBデータは宛先変数と整合性を保っていなければなりません。通常、**VARIABLE TO BLOB**コマンドを使用して格納されたBLOBを使用します。

オプション引数の＜オフセット＞を指定しない場合には、変数データはBLOBの最初から読み込まれます。複数の変数が格納されているBLOBを操作する場合には、＜オフセット＞の他に、数値変数も渡さなければなりません。呼び出しの前に、この数値変数を適切なオフセットに設定します。呼び出しの後で、この同じ数値変数はBLOB内に格納されている以下の変数のオフセットを返します。

呼び出し後、変数が正常に再記述された場合には、システム変数OKは1に設定されます。変数を再記述するために必要なメモリがない、等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォームからの独立性に関する注意点：

BLOB TO VARIABLEコマンドおよび**VARIABLE TO BLOB**コマンドは、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

▼ **VARIABLE TO BLOB**コマンドの例を参照してください。

参照

VARIABLE TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定されます。

LIST TO BLOB

LIST TO BLOB(変数 ; blob {; *})

引数	タイプ		説明
変数	変数	→	BLOBの中に格納する階層リスト
blob	BLOB	→	階層リストを受信するBLOB
*	*	→	値を追加する場合は*

説明

LIST TO BLOBコマンドは、BLOB内に階層リストを格納します。

オプション引数<*>を指定した場合には、階層リストはBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数<*>を使用すれば、BLOBがメモリ容量内であれば変数やリスト（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数<*>を指定しない場合には、階層リストはBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

階層リストの格納場所に関わらず、指定された位置に従ってBLOBのサイズは必要に応じて拡張されます（必要な場合にはリストサイズまで加算）。修正後のバイト数は（設定されたばかりのもの以外）0（ゼロ）にリセットされます。

警告：BLOBを使用して階層リストを格納すると、階層リストは4D内部形式を使用してBLOBに格納されるため、格納されたBLOBの内容を読み出すにはBLOB to list関数を使用しなければなりません。

呼び出し後、階層リストが正常に格納された場合には、システム変数OKは1に設定されます。階層リストを格納するために必要なメモリがない、等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォームからの独立性に関する注意点：

LIST TO BLOBコマンドおよび**BLOB to list**関数は、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

例題

▼ **BLOB to list**関数の例を参照してください。

参照

BLOB to list、BLOB TO VARIABLE、VARIABLE TO BLOB

BLOB to list

BLOB to list(blob {;オフセット}) → 数値

引数	タイプ		説明
blob	BLOB	→	階層リストを含んだBLOB
オフセット	数値	→	BLOB内でのオフセット（バイト単位）
		←	読み込み後の新規オフセット
戻り値	数値	←	新しく作成されたリストの参照番号

説明

BLOB to list関数は、オプション引数＜オフセット＞で指定されたバイトオフセット（ゼロから開始）にある＜blob＞に格納されているデータを使用して新しい階層リストを作成し、この新しいリストの数値を返します。

BLOBデータはコマンドと整合性を保っていなければなりません。通常、**LIST TO BLOB**コマンドを使用して格納されたBLOBを使用します。

オプション引数＜オフセット＞を指定しない場合には、変数データはBLOBの最初から読み込まれます。複数の変数やリストが格納されているBLOBを操作する場合には、＜オフセット＞の他に、数値変数も渡さなければなりません。

呼び出しの前に、この数値変数を適切なオフセットに設定します。呼び出しの後で、この同じ数値変数はBLOB内に格納されている以下の変数のオフセットを戻します。

呼び出し後、階層リストが正常に作成された場合には、システム変数OKは1に設定されます。階層リストを作成するために必要なメモリがない、等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォームからの独立性に関する注意点：

BLOB to list関数および**LIST TO BLOB**関数は、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

例題

この例では、データ入力フォームが画面に表示される前に、このフォームのフォームメソッドがBLOBフィールドからリストを抽出し、データ入力が検証されるとこのリストをBLOBフィールドに再び格納します。

```
` [テーブル1];"入力"フォームのフォームメソッド
Case of
	¥(Form event=On Load)
		hList:=BLOB to list([テーブル1]アイデア)
		If(OK=1)
			hList:=New list
		End if
	¥(Form event=On Unload)
		CLEAR LIST(hList;*)
	¥(bValidate=1)
		LIST TO BLOB(hList ; [テーブル1]アイデア)
End case
```

参照

LIST TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定されます。

INTEGER TO BLOB

INTEGER TO BLOB(整数 ; blob ; バイト順序 {;オフセット | *})

引数	タイプ		説明
整数	数値	→	BLOBの中に書き込む整数の値
blob	BLOB	→	整数の値を受信するBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット¦*	変数¦*	←	*でなければ、書き込み後の新しいオフセット

説明

INTEGER TO BLOBコマンドは、＜blob＞に2バイトの＜整数＞の値を書き込みます。

引数＜バイト順序＞は、2バイトの＜整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜*＞を指定した場合には、2バイトの整数の値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜*＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜*＞や＜オフセット＞変数を指定しない場合には、2バイトの整数の値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット（ゼロから開始）に書き込まれます。2バイトの整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って（必要に応じてさらに最大2バイトまで）増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例題

1　以下のコードを実行すると、

```
INTEGER TO BLOB(0x0206 ; vxBlob; Native byte ordering)
```

■ vxBlobのサイズは、2バイトです。

■ Macintoshでは、vxBLOB{0} = $02およびvxBLOB{1} = $06

■ PCでは、vxBLOB{0} = $06およびvxBLOB{1} = $02

2. 以下のコードを実行すると、

```
INTEGER TO BLOB(0x0206 ; vxBlob ; Macintosh byte ordering)
```

■ vxBlobのサイズは、2バイトです。

■ すべてのプラットフォームで、vxBLOB{0} = $02 およびvxBLOB{1} = $06

3. 以下のコードを実行すると、

```
INTEGER TO BLOB(0x0206 ; vxBlob ; PC byte ordering)
```

■ vxBlobのサイズは、2バイトです。

■ すべてのプラットフォームで、 vxBLOB{0} = $06およびvxBLOB{1} = $02

4. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 100)
INTEGER TO BLOB(0x0206 ; vxBlob ; PC byte ordering;*)
```

■ vxBlobのサイズは、102バイトです。

■ すべてのプラットフォームで、vxBLOB{100} = $06およびvxBLOB{101} = $02

■ BLOBの他のバイトは変更されません。

5. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 100)
vlOffset:=50
INTEGER TO BLOB(518 ; vxBlob ; Macintosh byte ordering ; vlOffset)
```

■ vxBlobのサイズは100バイトです。

■ すべてのプラットフォームでvxBLOB{50} = $02 およびvxBLOB{51} = $06

■ BLOBの他のバイトは変更されません。

■ 変数vlOffsetは2だけ増分されました。

参照

BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

LONGINT TO BLOB

LONGINT TO BLOB(倍長整数 ; blob ; バイト順序 {;オフセット | *}**)**

引数	タイプ		説明
倍長整数	数値	→	BLOBの中に書き込む倍長整数の値
blob	BLOB	→	倍長整数の値を受けとるBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット¦*	変数¦*	→	BLOB内でのオフセット（バイト単位）または値を追加するための*
		←	*でなければ、書き込み後の新しいオフセット

説明

LONGINT TO BLOBコマンドは、＜blob＞に4バイトの＜倍長整数＞の値を書き込みます。

引数＜バイト順序＞は、4バイトの＜倍長整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜*＞を指定した場合には、4バイトの倍長整数の値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜*＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜*＞や＜オフセット＞変数を指定しない場合には、4バイトの倍長整数の値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、4バイトの整数の値はBLOB内のオフセット（ゼロから開始）に書き込まれます。4バイトの整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って（必要に応じてさらに最大4バイトまで）増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例題

1. 以下のコードを実行すると、

```
LONGINT TO BLOB(0x01020304 ; vxBlob; Native byte ordering)
```

■ vxBlobのサイズは、4バイトです。

■ Macintoshでは、vxBLOB{0}=$01、vxBLOB{1}=$02、vxBLOB{2}=$03、vxBLOB{3}=$04。

■ PCでは、vxBLOB{0}=$04、vxBLOB{1}=$03、vxBLOB{2}=$02、vxBLOB{3}=$01。

2. 以下のコードを実行すると、

```
LONGINT TO BLOB(0x01020304 ; vxBlob ; Macintosh byte ordering)
```

■ vxBlobのサイズは、4バイトです。

■ すべてのプラットフォームで、vxBLOB{0}=$01、vxBLOB{1}=$02、vxBLOB{2}=$03、vxBLOB{3}=$04。

3. 以下のコードを実行すると、

```
LONGINT TO BLOB(0x01020304 ; vxBlob ; PC byte ordering)
```

■ vxBlobのサイズは、4バイトです。

■ すべてのプラットフォームで、vxBLOB{0}=$04、vxBLOB{1}=$03、vxBLOB{2}=$02、vxBLOB{3}=$01。

4. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 100)
LONGINT TO BLOB(0x01020304 ; vxBlob ; PC byte ordering ; *)
```

■ vxBlobのサイズは、104バイトです。

■ すべてのプラットフォームで、vxBLOB{100}=$04、vxBLOB{101}=$03、vxBLOB{102}=$02、vxBLOB{103}=$01。

■ BLOBの他のバイトは、変更されません。

5. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
LONGINT TO BLOB(0x01020304 ; vxBlob ; Macintosh byte ordering ; vlOffset)
```

■ vxBlobのサイズは、100Kです。

■ すべてのプラットフォームで、vxBLOB{50}=$01、vxBLOB{51}=$02、vxBLOB{52}=$03、vxBLOB{53}=$04。

■ BLOBの他のバイトは、変更されません。

■ 変数vlOffsetは、4だけ増分されました(現在の値は54です)。

参照

BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、REAL TO BLOB、TEXT TO BLOB

REAL TO BLOB

REAL TO BLOB(実数 ; blob ; 実数形式 {;オフセット | *})

引数	タイプ		説明
実数	数値	→	BLOBの中に書き込む実数の値
blob	BLOB	→	実数の値を受信するBLOB
実数形式	数値	→	0=ネイティブ実数形式 1=拡張実数形式 2=Macintosh倍精度実数形式 3=Windows倍精度実数形式
オフセット¦*	変数¦*	→	BLOB内でのオフセット（バイト単位）または値を追加するための*
		←	*でなければ、書き込み後の新しいオフセット

説明

REAL TO BLOBコマンドは、＜blob＞に＜実数＞の値を書き込みます。

引数＜実数フォーマット＞は、内部形式とバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native real format	倍長整数	0
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
PC double real format	倍長整数	3

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜*＞を指定した場合には、実数値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜*＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜*＞や＜オフセット＞変数を指定しない場合には、実数値はBLOBの最初に格納され、それ以前にその場所にあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、実数値はBLOB内のオフセット（ゼロから開始）に書き込まれます。実数値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って（必要に応じてさらに最大8または10バイトまで）増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例題

1. 以下のコードを実行すると、

```
C_REAL(vrValue)
vrValue := ...
REAL TO BLOB(vrValue ; vxBlob; Native real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは8バイトです。

2. 以下のコードを実行すると、

```
C_REAL(vrValue)
vrValue := ...
REAL TO BLOB(vrValue ; vxBlob ; Extended real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは10バイトです。

3. 以下のコードを実行すると、

```
C_REAL(vrValue)
vrValue := ...
REAL TO BLOB(vrValue ; vxBlob ; Macintosh Double real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは8バイトです。

4. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue := ...
REAL TO BLOB(vrValue ; vxBlob ; Windows Double real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは8バイトです。

5. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 100)
```

5. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 100)
REAL TO BLOB(vrValue ; vxBlob ; Extended real format ; *)
```

■ すべてのプラットフォームで、vxBlobのサイズは110バイトです。

■ すべてのプラットフォームで、実数値はバイト番号100～109にあるバイトに格納されます。

■ BLOBの他のバイトは、変更されません。

6. 以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue := ...
vlOffset:=50
REAL TO BLOB(518 ; vxBlob ; Windows Double real format ; vlOffset)
```

■ すべてのプラットフォームで、vxBlobのサイズは100バイトです。

■ すべてのプラットフォームで、実数値はバイト番号50～57にあるバイトに格納されます。

■ BLOBの他のバイトは、変更されません。

■ 変数vlOffsetは、8だけ増分されました（現在の値は58です）。

参照

BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、TEXT TO BLOB

TEXT TO BLOB

TEXT TO BLOB(テキスト ; blob ; テキスト形式 {;オフセット | *})

引数	タイプ		説明
テキスト	文字列	→	BLOBの中に書き込むテキストの値
blob	BLOB	→	テキストの値を受けとるBLOB
テキスト形式	数値	→	0=C言語文字列 1=Pascal言語文字列 2=長さ属性付きテキスト 3=長さ属性なしテキスト
オフセット¦*	変数¦*	→	BLOB内でのオフセット（バイト単位）または値を追加するための*
		←	*でなければ、書き込み後の新しいオフセット

説明

TEXT TO BLOBコマンドは、＜テキスト＞の値を＜blob＞の中に書き込みます。

引数＜テキスト形式＞は、＜テキスト＞の値が書き込まれる際の内部形式を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
C string	倍長整数	0
Pascal string	倍長整数	1
Text with length	倍長整数	2
Text without length	倍長整数	3

以下の表は、これらの内部形式それぞれについて説明しています。

テキスト形式	説明と例
C言語文字列	テキストはNULL文字（ASCII コード $00）で終了します。 "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal 言語文字列	テキストの前には 1バイト長が追加されます。 "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性付きテキスト	テキストの前には2バイトの長さ属性が追加されます。 "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性なしテキスト	テキストは文字列だけで構成されます。 "" → データなし "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21

注：このコマンドは、テキスト（C_TEXTコマンドで宣言されます）および文字列（C_STRINGコマンドで宣言されます）の両方の式に対応しています。テキスト変数には最大で32,000文字、文字列変数にはその宣言にある文字の数だけ、最大で255文字まで含むことができることを覚えておいてください。

オプション引数＜*＞を指定した場合には、テキストの値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜*＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜*＞や＜オフセット＞変数を指定しない場合には、テキストの値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、テキストの値はBLOB内のオフセット（ゼロから開始）に書き込まれます。テキストの値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って（必要に応じてさらに最大8または10バイトまで）増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例題

以下のコードを実行すると、

```
SET BLOB SIZE(vxBlob ; 0)
C_TEXT(vtValue)
vtValue := "Hello World!" `「vtValue」変数の長さは12バイト
TEXT TO BLOB(vtValue ; vxBlob;C string) ``BLOBのサイズは13バイトになる
TEXT TO BLOB(vtValue ; vxBlob;Pascal string) `BLOBのサイズは13バイトになる
TEXT TO BLOB(vtValue ; vxBlob;Text with length) ``BLOBのサイズは14バイトになる
TEXT TO BLOB(vtValue ; vxBlob;Text without length) ``BLOBのサイズは12バイトになる
```

参照

BLOB to integer、BLOB to longint、BLOB to real,、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB

BLOB to integer

BLOB to integer(blob ; バイト順序 {;オフセット}) → 数値

引数	タイプ		説明
blob	BLOB	→	整数の値を受けとるBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	変数	→	BLOB内でのオフセット（バイト単位で表す）
		←	読み込み後の新しいオフセット
戻り値	数値	←	2バイトの整数値

説明

BLOB to integerコマンドは、引数＜blob＞から読み込まれた2バイトの整数の値を返します。

引数＜バイト順序＞は、2バイトの＜整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット（ゼロから開始）に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の2バイトが読み込まれます。

注：ゼロから「BLOBのサイズ-2」の範囲のオフセット（バイト単位）の値を渡す必要があります。この範囲の値を渡さないと、エラー-111が発生します。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

例題

▼以下の例では、あるBLOBから、オフセット0x200から開始して整数値を20個読み込んでいます。

```
$vlOffset:=0x200
For($viLoop ; 0 ;19)
        $viValue:=BLOB to integer(vxSomeBlob ; PC byte ordering ; $vlOffset)
        `$viValueを使った何らかの処理を実行する
End for
```

参照

BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to longint

BLOB to longint(blob ; バイト順序 {;オフセット}) → 数値

引数	タイプ		説明
blob	BLOB	→	倍長整数の値を受けとるBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	変数	→	BLOB内でのオフセット（バイト単位で表す）
		←	読み込み後の新しいオフセット
戻り値	数値	←	4バイトの倍長整数値

説明

BLOB to longintコマンドは、＜blob＞から読み込まれた4バイトの倍長整数の値を返します。

引数＜バイト順序＞は、4バイトの＜倍長整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット（ゼロから開始）に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の4バイトが読み込まれます。

注：ゼロから「BLOBのサイズ-4」の範囲のオフセット（バイト単位）の値を渡す必要があります。この範囲の値を渡さないと、エラー-111が発生します。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

例題

▼以下の例では、あるBLOBから、オフセット0x200から開始して倍長整数値を20個読み込んでいます。

```
$vlOffset:=0x200
For($viLoop ; 0 ;19)
    $viValue:=BLOB to longint(vxSomeBlob ; PC byte ordering ; $vlOffset)
    `$viValueを使った何らかの処理を実行する
End for
```

参照

BLOB to integer、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to real

BLOB to real(blob ; 実数形式 {;オフセット})→実数

引数	タイプ		説明
blob	BLOB	→	実数の値を受けとるBLOB
実数形式	数値	→	0=ネイティブ実数形式 1=拡張実数形式 2=Macintosh倍精度実数形式 3=Windows倍精度実数形式
オフセット	変数	→	BLOB内でのオフセット（バイト単位で表す）
		←	読み込み後の新しいオフセット

説明

BLOB to realコマンドは、引数＜blob＞から読み込まれた実数の値を返します。

引数＜実数形式＞は、内部形式とバイト順序を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
Native real format	倍長整数	0
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
PC double real format	倍長整数	3

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを受け渡す場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット（ゼロから開始）に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の8または10バイトが読み込まれます。

注：ゼロから「BLOBのサイズ-8」または「BLOBのサイズ-10」の範囲のオフセットの値を渡す必要があります。この範囲の値を渡さないと、エラー-111が発生します。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

例題

▼以下の例では、あるBLOBから、オフセット0x200から開始して実数値を20個読み込んでいます。

```
$vlOffset:=0x200
For($viLoop ; 0 ;19)
        $viValue:=BLOB to real(vxSomeBlob ; PC byte ordering ; $vlOffset)
        `$viValueを使った何らかの処理を実行する
End for
```

参照

BLOB to integer、BLOB to longint、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to text

BLOB to text(blob ; テキスト形式 {;オフセット {;テキストの長さ}}) → テキスト

引数	タイプ		説明
blob	BLOB	→	テキストの値を受けとるBLOB
テキスト形式	数値	→	0=C言語文字列 1=Pascal言語文字列 2=長さ属性付きテキスト 3=長さ属性なしテキスト
オフセット	変数	→	BLOB内でのオフセット（バイト単位で表す）
		←	読み込み後の新しいオフセット
テキストの長さ	数値	→	読み込まれるバイト数
戻り値	数値	←	テキストの値

説明

BLOB to textコマンドは、＜blob＞から読み込まれたテキストの値を返します。

引数＜テキスト形式＞は、＜テキスト＞の値が書き込まれる際の内部形式を決定します。4th Dimensionにある以下の定義済み定数のうち、いずれか1つを渡します。

定数	タイプ	値
C string	倍長整数	0
Pascal string	倍長整数	1
Text with length	倍長整数	2
Text without length	倍長整数	3

以下の表は、これらの内部形式それぞれについて説明しています。

テキスト形式	説明と例
C言語文字列	テキストはNULL文字（ASCIIコード $00）で終了します。 "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal 言語文字列	テキストの前には 1バイト長が追加されます。 "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性付きテキスト	テキストの前には2バイトの長さ属性が追加されます。 "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性なしテキスト	テキストは文字列だけで構成されます。 "" → データなし "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21

警告：読み込まれる文字の数は、引数＜テキスト形式＞によって決まります。ただし、形式が長さ属性なしテキストの場合には、オプション引数＜テキストの長さ＞に読み込まれる文字の数を必ず指定しなければなりません。それ以外の形式では、＜テキストの長さ＞は無視されるため、省略しても構いません。

テキスト変数には最高で32,000文字、文字列変数にはその宣言にある文字の数だけ、最大255文字まで含むことができることを忘れないようにします。変数に含むことのできる文字数以上のデータを読み込もうとすると、4Dは結果を変数に割り当てる時に途中で切り捨ててしまいます。

オプション引数＜オフセット＞を指定した場合には、テキストの値はBLOB内のオフセット（ゼロから開始）で読み込まれます。オプション引数＜オフセット＞を指定しなかった場合には、＜テキスト形式＞に渡した値に従ってBLOBの最初から読み込まれます。

長さ属性なしのテキストを読み込む場合には、オプション引数＜オフセット＞を必ず渡さなければならないことに注意してください。

注：ゼロから「BLOBのサイズ-読み込まれるテキストのサイズ」の範囲のオフセットの値を渡す必要があります。このようにしない場合には、この関数の結果は予期しない値になります。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を書き込むことができます。

例題

▼ 以下の例では、内部形式が'STR#'リソースの内部形式とまったく同一である、架空のMacintoshベースのリソースを読み込みます。

```
GET RESOURCE('ABCD' ; viResID ; vxResData ; viMyResFile)
vlSize:=BLOB Size(vxResData)
If(vlSize>0)
    `リソースは、文字列の数を指定している2バイトの整数から開始します
    vlOffset:=0
    viNbEntries:=BLOB to integer(vxResData ; Macintosh Byte Ordering ;
                                                        vlOffset)
    `リソースには、埋め込みではなく連結されたPascal言語文字列が含まれている
    For(viEntry ; 1 ; viNbEntries)
        If(vlOffset<vlSize)
            vsEntry:=BLOB to text(vxResData ; Pascal string ; vlOffset)
            ``vsEntryを使った何らかの処理を実行する
        Else
            `リソースデータは無効であり、ループを終了する
            viEntry:=viNbEntries+1
        End if
    End for
End if
```

参照

BLOB to integer、BLOB to longint、BLOB to real、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

INSERT IN BLOB

INSERT IN BLOB(blob ; オフセット ; 挿入数 {;フィル})

引数	タイプ		説明
blob	BLOB	→	バイトが挿入されるBLOB
オフセット	変数	→	挿入される開始位置
挿入数	数値	→	挿入されるバイト数
フィル	数値	→	デフォルトのバイト値（0x00..0xFF） 省略した場合は、0x00

説明

INSERT IN BLOBコマンドは、引数＜挿入数＞で指定されたバイト数を＜blob＞の＜オフセット＞で指定された位置に挿入します。BLOBは、＜挿入数＞で指定されたバイト数分だけ拡大されます。

オプション引数の＜フィル＞を指定しない場合には、BLOBに挿入されるバイトは 0x00に設定されます。それ以外の場合には、＜フィル＞に渡した値に設定されます（モジューロ256-0..255）。

呼び出しの前に、オプション引数の＜フィル＞変数内に、挿入位置をBLOBの最初から相対的な位置として渡します。

参照

DELETE FROM BLOB

DELETE FROM BLOB

DELETE FROM BLOB(blob ; オフセット ; 削除数)

引数	タイプ		説明
blob	BLOB	→	バイトが削除されるBLOB
オフセット	数値	→	削除される開始位置
削除数	数値	→	削除されるバイト数

説明

DELETE FROM BLOBコマンドは、＜削除数＞で指定されたバイト数を＜blob＞の＜オフセット＞で指定された位置（BLOBの最初から相対的な位置として表される）から削除します。BLOBは、＜削除数＞で指定されたバイト数分だけ縮小されます。

参照

INSERT IN BLOB

COPY BLOB

COPY BLOB(コピー元BLOB ; コピー先BLOB ; コピー元オフセット ; コピー先オフセット ; コピー数)

引数	タイプ		説明
コピー元BLOB	BLOB	→	コピー元BLOB
コピー先BLOB	BLOB	→	コピー先BLOB
コピー元オフセット	変数	→	コピー元の位置
コピー先オフセット	変数	→	コピー先の位置
コピー数	数値	→	コピーされるバイト数

説明

COPY BLOBコマンドは、引数＜コピー数＞で指定されたバイト数を＜コピー元BLOB＞から＜コピー先BLOB＞にコピーします。

コピーは＜コピー元オフセット＞で指定された位置（コピー元のBLOBの最初から相対的な位置として表されます）から開始され、＜コピー先オフセット＞で指定された位置（コピー先のBLOBの最初から相対的な位置として表されます）にコピーされます。

注：コピー先のBLOBのサイズは、必要に応じて変更できます。

参照

DELETE FROM BLOB、INSERT IN BLOB

ENCRYPT BLOB

ENCRYPT BLOB(データ; 送信秘密鍵{; 受信公開鍵})

引数	タイプ		説明
データ	BLOB	→	暗号化するデータ
		←	暗号化されたデータ
送信秘密鍵	BLOB	→	送信者の秘密鍵
受信公開鍵	BLOB	→	受信者の公開鍵

説明

ENCRYPT BLOBコマンドは、BLOB＜データ＞の内容を送信者の秘密鍵＜送信秘密鍵＞を使用して暗号化します。オプションとして、同時に受信者の公開鍵＜受信公開鍵＞も使用できます。これらの暗号化鍵は、**GENERATE ENCRYPTION KEYPAIR**コマンド(「セキュアプロトコル」テーマ）を使用して生成してください。

注：このコマンドは、SSLプロトコルアルゴリズムおよび暗号化機能を利用します。したがって、このコマンドを使用可能にするため、4D Webサーバ通信にSSLを使用したくない場合でも、SSLプロトコルに必要となる各コンポーネントがマシン上に正しくインストールされているか確認してください。このプロトコルについての詳細は、「Webサービス：SSLプロトコルの使用」の節を参照してください。

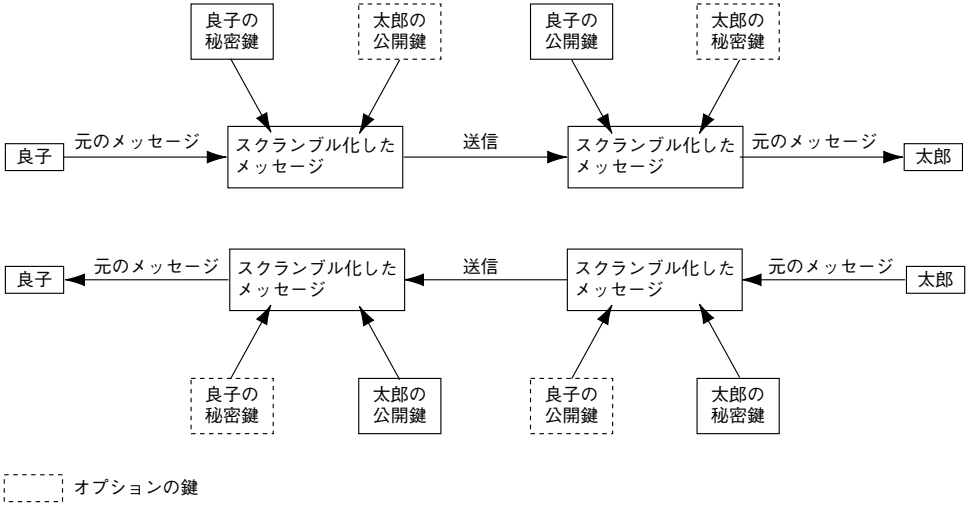
■ ある鍵が暗号化に使用されると（送信者の秘密鍵)、公開鍵の所有者だけがこの情報を読み取ることができます。このシステムにより、送信者自身が情報の暗号化を行ったということが保証されます。

■ 送信者の秘密鍵と受信者の公開鍵を同時に使用することにより、情報の読み取りを行えるのは1人の受信者だけであることが保証されます。

鍵を納めるBLOBは、PEM（Privacy Enhanced Mail）内部フォーマットです。クロスプラットフォームであるこの形式では、電子メールやテキストファイルにコピー＆ペーストすることにより簡単に鍵のやり取りや処理を行うことができます。

コマンドを実行すると、BLOB＜データ＞には暗号化されたデータが納められますが、このデータの解読は、引数として渡された送信者の公開鍵を使用した上で**DECRYPT BLOB**コマンドによってのみ行えます。さらに、情報の暗号化の際にオプションである受信者の公開鍵を使用すると、解読には受信者の秘密鍵も必要になります。

“良子”と“太郎”という名の2人の間でメッセージのやり取りを行う場合の
公開鍵と秘密鍵を用いた暗号化の原理



注：BLOB内容の変更（意図的かどうかに関わらず）を防ぐため、暗号にはチェックサム機能が含まれています。したがって、暗号化されたBLOBは変更しないでください。変更を行うと、解読できなくなるおそれがあります。

暗号化コマンドの最適化

データの暗号化を行うと、アプリケーションの実行速度が低下し、2 つの鍵を使用した場合は特に遅くなります。しかし、以下の最適化に関するヒントを考慮してみることをお勧めします。

■ 現時点で使用可能なメモリに応じて、コマンドは“同期”モードまたは“非同期”モードで実行されます。

非同期モードでは他のプロセスを中断しないので、より高速になります。使用可能なメモリが、暗号化するデータの少なくとも2倍ある場合、このモードが自動的に使用されます。

メモリがそれ以下の場合、セキュリティ上の理由から、非同期モードが使用されます。このモードでは他のプロセスを中断するため、速度はより低下します。

■ サイズの大きなBLOBに関しては、ヒントはBLOBの“戦略的”である小さな部分だけを暗号化するところにあります。これは、処理するデータサイズを減らして、処理時間を短くするためです。

例題

■ 単一の鍵を使用

ある会社が、4Dデータベースに保存されたデータを機密にしておきたいとします。これらの情報は定期的に、ファイルに納めてインターネット経由で子会社へ送信する必要があります。

1. 会社では、**GENERATE ENCRYPTION KEYPAIR**コマンドを使用して一組の鍵を生成します。

```
`GENERATE_KEYS_TXTメソッド
C_BLOB($BPublicKey; $BPrivateKey)
GENERATE ENCRYPTION KEYPAIR($BPrivateKey; $BPublicKey)
BLOB TO DOCUMENT("PublicKey.txt"; $BPublicKey)
BLOB TO DOCUMENT("PrivateKey.txt"; $BPrivateKey)
```

2. 会社側は秘密鍵を保存し、各子会社へは公開鍵を含むドキュメントのコピーを送信します。最高のセキュリティを維持するため、鍵は子会社に手渡すディスクにコピーしてください。

3. 次に、機密情報（例えば、テキストフィールドに保存したもの）をBLOBにコピーします。この情報は、秘密鍵を使用して暗号化されます。

```
`ENCRYPT_INFOメソッド
C_BLOB($vbEncrypted;$vbPrivateKey)
C_TEXT($vtEncrypted)

$tvEncrypted:=[Private]Info
VARIABLE TO BLOB($vtEncrypted;$vbEncrypted)
DOCUMENT TO BLOB("PrivateKey.txt"; $vbPrivateKey)

If(OK=1)
        ENCRYPT BLOB($vbEncrypted; $vbPrivateKey)
        BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
End if
```

4. 更新用ファイルは、インターネットのような非暗号化チャネルで子会社に送ることができます。万が一第三者がこの暗号化されたファイルを入手した場合でも、公開鍵なしではファイルを解読できません。

5. 各子会社では、公開鍵を使用してドキュメントの解読が可能です。

```
`DECRYPT_INFOメソッド
C_BLOB(($vbEncrypted;$vbPublicKey)
```

```
C_TEXT(($vtDecrytped)
C_TIME($vtDocRef)

ALERT("暗号化したドキュメントを選択してください。")
$vtDocRef:=Open document("") `Update.txtを選択する
If(OK=1)
    CLOSE DOCUMENT($vhRefDoc)
    DOCUMENT TO BLOB(Document;$vbEncrypted)
    DOCUMENT TO BLOB("PublicKey.txt"; $vbPublicKey)
    If(OK=1)
        DECRYPT BLOB($vbEncrypted; $vbPublicKey)
        BLOB TO VARIABLE($vbEncrypted; $vtDecrypted)
        CREATE RECORD([Private])
        [Private]Info:=$vtDecrypted
        SAVE RECORD([Private])
    End if
End if
```

■ 2つの鍵を使用

ある会社が情報のやり取りにインターネットを利用したいものとします。各子会社では、機密情報を受信し、また本社へ情報の送信も行います。したがって、要件は次の 2 つです。

■ 受信側は、メッセージの読み取りだけが可能です。

■ 受信側は、メッセージの送信が送信者自身によって行われたという証拠を取得しなければなりません。

1. 本社および各子会社では、それぞれ独自の鍵のペアを生成します（GENERATE_KEYS_TXTメソッドを使用）。

2. 秘密鍵は双方で内密にしておきます。各子会社は、自分の公開鍵を本社へ送り、本社もまた独自の公開鍵を送信します。公開鍵ではメッセージを解読するのに十分ではないため、この鍵の転送に暗号化のチャネルを使用する必要はありません。

3. 送信する情報を暗号化するため、子会社や本社ではENCRYPT_INFO_2メソッドを実行します。このメソッドは、送信側の秘密鍵と受信側の公開鍵を使用して情報の解読を行います。

```
`ENCRYPT_INFO_2メソッド
C_BLOB($vbEncrypted;$vbPrivateKey;$vbPublicKey)
C_TEXT($vtEncrypt)
C_TIME($vtDocRef)
```

```
$vtEncrypt:= [Private]Info
VARIABLE TO BLOB($vtEncrypt;$vbEncrypted)
` 独自の秘密鍵がロードされる...
DOCUMENT TO BLOB("PrivateKey.txt"; $vbPrivateKey)
If(OK=1)
	` ...受信側の公開鍵がロードされる
	ALERT("受信者の公開鍵を選択してください。")
	$vhDocRef:=Open document("") `ロードする公開鍵
	If(OK=1)
		CLOSE DOCUMENT($vhDocRef)
		DOCUMENT TO BLOB(Document;$vbPublicKey)
		`引数として2つの鍵を用いたBLOBの暗号化
		ENCRYPT BLOB($vbEncrypted; $vbPrivateKey; $vbPublicKey)
		BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
	End if
End if
```

4. 次に、暗号化したファイルが、インターネット経由で受信側に送信されます。万が一第三者がこのファイルを入手した場合、たとえ公開鍵を持っていたとしても、受信側の秘密鍵も必要となるため、メッセージを解読できません。

5. 受信側はそれぞれ、独自の秘密鍵と送信側の公開鍵を使用してドキュメントの解読が可能です。

```
`DECRYPT_INFO_2メソッド
C_BLOB($vbEncrypted;$vbPublicKey;$vbPrivateKey)
C_TEXT($vtDecrypted)
C_TIME($vhDocRef)

ALERT("暗号化したドキュメントを選択してください。")
$vhRefDoc:=Open document("") `Update.txtファイルを選択する
If(OK=1)
	CLOSE DOCUMENT($vhDocRef)
	DOCUMENT TO BLOB(Document;$vbEncrypted)
	`独自の秘密鍵がロードされる
	DOCUMENT TO BLOB("PrivateKey.txt"; $vbPrivateKey)
	If(OK=1)
		` ...送信側の公開鍵がロードされる
		ALERT("送信者の公開鍵を選択してください。")
		$vhDocRef:=Open document("") `ロードする公開鍵
		If(OK=1)
```

```
                    CLOSE DOCUMENT($vhDocRef)
                    DOCUMENT TO BLOB(Document;$vbPublicKey)
                    `引数として2つの鍵を用いたBLOBの解読
                    DECRYPT BLOB($vbEncrypted; $vbPublicKey;
                                                    $vbPrivateKey)
                    BLOB TO VARIABLE($vbEncrypted; $vtDecrypted)
                    CREATE RECORD([Private])
                    [Private]Info:=$vtDecrypted
                    SAVE RECORD([Private])
                End if
            End if
        End if
```

参照

DECRYPT BLOB、GENERATE ENCRYPTION KEYPAIR

DECRYPT BLOB

DECRYPT BLOB(データ; 送信公開鍵{; 受信秘密鍵})

引数	タイプ		説明
データ	BLOB	→	解読するデータ
		←	解読後のデータ
送信公開鍵	BLOB	→	送信者の公開鍵
受信秘密鍵	BLOB	→	受信者の秘密鍵

説明

DECRYPT BLOBコマンドは、BLOB＜データ＞の内容を送信者の公開鍵＜送信公開鍵＞を使用して解読します。オプションとして、受信者の秘密鍵＜受信秘密鍵＞も使用します。

送信者の公開鍵を納めたBLOBは、引数＜送信公開鍵＞で渡されます。この鍵は、送信者が**GENERATE ENCRYPTION KEYPAIR**コマンドを実行して生成し、受信者に送信しておく必要があります。

受信者の秘密鍵を納めたBLOBは、オプションの引数＜受信秘密鍵＞で受け渡すことができます。この引数を渡した場合、受信者は**GENERATE ENCRYPTION KEYPAIR**コマンドを実行して一組の暗号化鍵を生成し、独自の公開鍵を送信者に送る必要があります。この一組の鍵をもとにした暗号化システムでは、送信者のみがメッセージの暗号化を行い、かつ受信者だけがその解読を行える、ということが保証されます。一組の鍵をもとにした暗号化システムに関する詳細は、**ENCRYPT BLOB**コマンドを参照してください。

DECRYPT BLOBコマンドでは、BLOB内容の変更（意図的かどうかに関わらず）を防ぐため、チェックサム機能が提供されています。暗号化したBLOBが損傷したり、変更されると、このコマンドは何も行わず、エラーを返します。

例題

ENCRYPT BLOBコマンドの例題を参照してください。

参照

ENCRYPT BLOB、GENERATE ENCRYPTION KEYPAIR

4Dにはブール演算に用いるブール関数が用意されています。

True　　**False**　　**Not**

例題

▼以下は、Boolean変数を使用したボタン値の例です。myBoolean変数は、myButtonボタンがクリックされるとTrueを、クリックされなかった場合Falseとなります。ボタンは、クリックされると値が1になります。

```
If(MyButton=1) ` MyButtonがクリックされたら...
    myBoolean:=True ` myBoolean変数にTrueがセットする
Else ` MyButtonがクリックされなかったら...
    myBoolean:=False ` myBoolean変数にFalseがセットする
End if
```

上の例をもっと簡単に1行で記述することができます。

```
myBoolean:=(MyButton=1)
```

参照

False、論理演算子、Not, True

さらに、以下の4Dコマンドもブール値を返します。

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**

True

True → ブール値(True)

説明

True関数は、ブール値のTrueを返します。

例題

▼ 以下の例は、v変数にTrueを代入します。

```
v変数:=True
```

参照

False、Not

False

False → ブール値(False)

説明

False関数は、ブール値のFalseを返します。

例題

▼ 以下の例は、v変数にFalseを代入します。

```
v変数:=False
```

参照

True、Not

Not

Not(ブール値) → ブール値

引数	タイプ	説明
ブール値	ブール	否定を求めるブール値

説明

Not関数は、＜ブール値＞の否定を返します。TrueをFalseに、FalseをTrueにして返します。

例題

▼以下の例は、変数にTrueを代入し変数の値をFalseにした後、再びTrueにします。

```
v結果:=True  ` v結果にTrueを代入
v結果:=Not(v結果)  ` v結果にFalseを代入
v結果:=Not(v結果)  ` v結果にTrueを代入
```

APPEND TO CLIPBOARD

APPEND TO CLIPBOARD (データタイプ ; データ)

引数	タイプ		説明
データタイプ	文字列	→	4バイトの文字列データタイプ
データ	BLOB	→	クリップボードに追加するデータ

説明

APPEND TO CLIPBOARDコマンドは、引数＜データタイプ＞で指定されたデータタイプでBLOB＜データ＞内にあるデータをクリップボードに追加します。

警告：＜データタイプ＞に渡す値は、大文字と小文字が区別されます。例えば、“abcd”は“ABCD”と同じではありません。

BLOBデータがクリップボードに正しく追加された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定され、エラーが生成される可能性があります。

通常、同一データの複数のインスタンスをクリップボードに追加、またはテキストやPICT以外のタイプのデータを追加するときには、**APPEND TO CLIPBOARD**コマンドを使用します。クリップボードに新しいデータを追加するには、まず最初に**CLEAR CLIPBOARD**コマンドを使用してクリップボードを消去する必要があります。

消去と追加を実行する場合には、以下に従います。

- クリップボードに対してテキストを消去、または追加する場合には、**SET TEXT TO CLIPBOARD**コマンドを使用します。
- クリップボードに対してピクチャを消去、または追加する場合には、**SET PICTURE TO CLIPBOARD**コマンドを使用します。

ただし、BLOBに実際にテキストまたはピクチャが含まれている場合には、**APPEND TO CLIPBOARD**コマンドを使用して、クリップボードにテキストまたはピクチャを追加できることに注意してください。

例題

クリップボードコマンドとBLOBを使用すると、洗練された切り取り／コピー／貼り付けの仕組みを構築でき、たった1つのデータではなく構造化されたデータを扱うことができます。以下の例では、2つのプロジェクトメソッド「SET RECORD TO CLIPBOARD」と「GET RECORD FROM CLIPBOARD」は、クリップボードとの間でコピーするためにレコード全体を1つのデータとして扱うことができます。

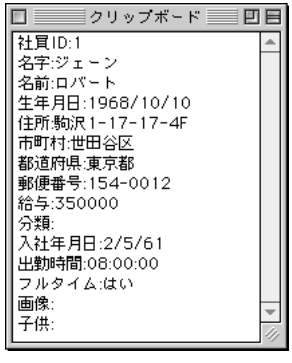
```
`「SET RECORD TO CLIPBOARD」プロジェクトメソッド
` SET RECORD TO CLIPBOARD(数値)
` SET RECORD TO CLIPBOARD(テーブル番号)
C_LONGINT($1 ; $vlField ; $vlFieldType)
C_POINTER($vpTable ; $vpField)
C_STRING(255 ; $vsDocName)
C_TEXT($vtRecordData ; $vtFieldData)
C_BLOB($vxRecordData)
` クリップボードの消去(カレントレコードがない場合は空のまま)
CLEAR CLIPBOARD
`パラメータとして渡される数値を持つテーブルへのポインタを取得する
$vpTable:=Table($1)
`そのテーブルのカレントレコードがある場合
If((Record number($vpTable->)>=0) ¦(Is new record($vpTable->)))
`レコードのテキストイメージを保持するテキスト変数を初期化する
    $vtRecordData:=""
    `レコードの各フィールドに対してループする
    For($vlField ; 1 ; Count fields($1))
        `フィールドタイプを取得する
        GET FIELD PROPERTIES($1 ; $vlField ; $vlFieldType)
        `フィールドへのポインタを取得する
        $vpField:=Field($1 ; $vlField)
        `フィールドのタイプに従って、適切な方法でそのデータをコピーする
                                                    (またはしない)
        Case of
            ¥(($vlFieldType=Is Alpha field ) |($vlFieldType=Is Text ))
                $vtFieldData:=$vpField->
            ¥(($vlFieldType=Is Real ) |($vlFieldType=Is Integer )
                                        |($vlFieldType=Is LongInt )
                |($vlFieldType=Is Date ) |($vlFieldType=Is Time ))
                $vtFieldData:=String($vpField->)
            ¥($vlFieldType=Is Boolean )
                $vtFieldData:=String(Num($vpField->) ;
                                                "Yes ; ; No")
```

```
        Else
            `他のフィールドデータタイプは読み飛ばすか、無視する
            $vtFieldData:=""
        End case
        `レコードのテキストイメージを保持しているテキスト変数に
                                        フィールドデータを積算する
        $vtRecordData:=$vtRecordData + Field name
                                ($1 ; $vlField)+":"+Char(9)+$vtFieldData+ CR
        `注：このメソッドでは、CRはMacintoshではChar(13)を返し、
        `Windowsでは、Char(13)+Char(10)を返す
    End for
    `クリップボードにレコードのテキストイメージを入れる
    SET TEXT TO CLIPBOARD($vtRecordData)
    ` テンポラリ（中間）フォルダにあるスクラップファイルの名前
    $vsDocName:=Temporary folder+"スクラップ"+String(1+(Random%99))
    `スクラップファイルが存在する場合には削除する
                                        （エラーはここでテストする）
    DELETE DOCUMENT($vsDocName)
    `スクラップファイルを作成する
    SET CHANNEL(10 ; $vsDocName)
    `レコード全体をスクラップファイルに送信する
    SEND RECORD($vpTable->)
    `スクラップファイルを閉じる
    SET CHANNEL(11)
    `BLOBのなかにスクラップファイルをロードする
    DOCUMENT TO BLOB($vsDocName ; $vxRecordData)
    `このスクラップファイルはもう必要ない
    DELETE DOCUMENT($vsDocName)
    `レコード全体のイメージをクリップボードに追加する
    `注：任意に"4Drc"をデータタイプとして使用する
    APPEND TO CLIPBOARD("4Drc" ; $vxRecordData)
    `この時点で、クリップボードには以下が含まれる
    `(1) レコードのテキストイメージ（以下の画面のとおり）
    `(2) レコードの全体イメージ
                        （ピクチャ、サブテーブル、BLOBフィールドが含まれる）
End if
```

以下のレコードを入力します。



「SET RECORD TO CLIPBOARD」プロジェクトメソッドに[Employees]テーブルを適用する場合には、クリップボードには以下に示すようにレコードのテキストイメージとレコード全体のイメージが含まれます。



以下のように、「*GET RECORD FROM CLIPBOARD*」プロジェクトメソッドを使用して、このレコードのイメージを別のレコードに貼り付けることができます。

```
`「GET RECORD FROM CLIPBOARD」プロジェクトメソッド
    ` GET RECORD FROM CLIPBOARD(数値)
    ` GET RECORD FROM CLIPBOARD(テーブル番号)
    C_LONGINT($1 ; $vlField ; $vlFieldType ; $vlPosCR ; $vlPosColon)
    C_POINTER($vpTable ; $vpField)
    C_STRING(255 ; $vsDocName)
    C_BLOB($vxClipboardData)
    C_TEXT($vtClipboardData ; $vtFieldData)
    `パラメータとして渡される数値を持つテーブルへのポインタを取得する
    $vpTable:=Table($1)
    `カレントレコードがある場合、
    If(Record number($vpTable->)>=0) | (Is new record($vpTable->))
        Case of
        `クリップボードにフルイメージレコードが含まれているか？
        ¥(Test clipboard("4Drc")>0)
            `含んでいる場合は、クリップボードの内容を取り出す
            GET CLIPBOARD("4Drc" ; $vxClipboardData)
            `テンポラリ（中間）フォルダにあるスクラップファイルの名前
            $vsDocName:=Temporary folder+"scrap"+
                                    String(1+(Random%99))
            `スクラップファイルが存在する場合には削除する
                                    (エラーはここでテストする)
            DELETE DOCUMENT($vsDocName)
            `スクラップファイルの中にBLOBを保存する
            BLOB TO DOCUMENT($vsDocName ; $vxClipboardData)
            `スクラップファイルを開く
            SET CHANNEL(10 ; $vsDocName)
            `スクラップファイルからレコード全体を受信する
            RECEIVE RECORD($vpTable->)
            `スクラップファイルを閉じる
            SET CHANNEL(11)
            `このスクラップファイルはもう必要ない
            DELETE DOCUMENT($vsDocName)
            `テキストをクリップボードが含んでいるか？
        ¥(Test clipboard("TEXT")>0)
            `クリップボードからテキストを取り出す
            $vtClipboardData:=Get text from clipboard
        `増分するフィールド番号を初期化する
            $vlField:=0
```

```
Repeat
   `テキスト内の以下のフィールド行を探す
   $vlPosCR:=Position(CR ; $vtClipboardData)
   If($vlPosCR>0)
   `フィールド行を取り出す
      $vtFieldData:=Substring($vtClipboardData ; 1 ; $vlPosCR-1)
      `コロン":"があるか？
      $vlPosColon:=Position(":" ; $vtFieldData)
         If($vlPosColon>0)
            `データタイプのみ取得する（フィールド名は削除）
            $vtFieldData:=Substring($vtFieldData ; $vlPosColon+2)
         End if
         `フィールド番号をインクリメントする
         $vlField:=$vlField+1
         `クリップボードは我々が望むより多い
                                          データを含んでいるかもしれない...
         If($vlField<=Count fields($vpTable))
            `フィールドタイプを取得する
            GET FIELD PROPERTIES($1 ; $vlField ; $vlFieldType)
            `フィールドポインタを取得する
            $vpField:=Field($1 ; $vlField)
            `フィールドのタイプに従って、適切な方法でそのデータを
                                                コピーする(またはしない)
            Case of
               ¥(($vlFieldType=ls Alpha field )  |
                                                ($vlFieldType=ls Text))
                  $vpField->:=$vtFieldData
               ¥(($vlFieldType=ls Real )  |  ($vlFieldType=ls Integer)  |
                     ($vlFieldType=ls LongInt))
                  $vpField->:=Num($vtFieldData)
               ¥($vlFieldType=ls Date )
                  $vpField->:=Date($vtFieldData)
               ¥($vlFieldType=ls Time )
                  $vpField->:=Time($vtFieldData)
               ¥($vlFieldType=ls Boolean )
                  $vpField->:=($vtFieldData="Yes")
            Else
               `他のフィールドデータタイプは読み飛ばすか、
                                                      無視する
            End case
         Else
```

```
            ` 全フィールドが割り当てられた場合、ループを抜ける
            $vtClipboardData:=""
        End if
        ` 取り出されたテキストを削除する
        $vtClipboardData:=Substring($vtClipboardData ;
                                    $vlPosCR+Length(CR))
    Else
        ` フィールド区切りが見つからない場合、ループを抜ける
        $vtClipboardData:=""
    End if
    ` データがあるまでループ
Until(Length($vtClipboardData)=0)
Else
    ALERT("レコードとして貼り付けられるデータがクリップボードにありません。")
End case
End if
```

参照

CLEAR CLIPBOARD、SET PICTURE TO CLIPBOARD、SET TEXT TO CLIPBOARD

システム変数

BLOBデータがクリップボードに正しく追加された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定され、エラーが生成される可能性があります。

エラー処理

BLOBデータをクリップボードにアペンドするための十分なメモリがない場合には、エラーコード-108が生成されます。

CLEAR CLIPBOARD

CLEAR CLIPBOARD

引数	タイプ	説明

このコマンドには、引数はありません。

説明

CLEAR CLIPBOARDコマンドは、クリップボードの内容を消去します。クリップボードに同じデータの複数のインスタンスが含まれる場合には、すべてのインスタンスが消去されます。**CLEAR CLIPBOARD**コマンドを呼び出した後、クリップボードは空になります。

APPEND TO CLIPBOARDコマンドを使用して新しいデータをクリップボードに追加する前に、**CLEAR CLIPBOARD**コマンドを1回呼び出す必要があります。これは、**APPEND TO CLIPBOARD**コマンドが新しいデータを追加する前にクリップボードを消去しないためです。

CLEAR CLIPBOARDコマンドを1回呼び出してから、**APPEND TO CLIPBOARD**コマンドを数回呼び出すと、異なるフォーマットで同じデータを切り取りまたはコピーすることができます。

一方、**SET TEXT TO CLIPBOARD**コマンドと**SET PICTURE TO CLIPBOARD**コマンドは、クリップボードにテキストデータやPICTデータを追加する前に自動的にクリップボードを消去します。

例題

▼ 以下の例は、クリップボードを消去し、次にデータをクリップボードに追加します。

```
CLEAR CLIPBOARD `必ずクリップボードを空にする
APPEND TO CLIPBOARD('XWKZ' ; $vxSomeData) `'XWKZ'タイプのデータを
                                                追加する
APPEND TO CLIPBOARD('SYLK' ; $vxSylkData) `Sylkデータだけデータを
                                                追加する
```

▼ **APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照

APPEND TO CLIPBOARD

GET CLIPBOARD

GET CLIPBOARD(データタイプ ; データ)

引数	タイプ		説明
データタイプ	文字列	→	4バイトの文字列データタイプ
データ	BLOB	←	クリップボードから取り出されたデータ

説明

GET CLIPBOARDコマンドは、クリップボードに存在するデータでユーザが<データタイプ>で指定したタイプのものをBLOBフィールドまたは変数データに返します。

警告：<データタイプ>に渡す値は、大文字と小文字が区別されます。例えば、“abcd”は“ABCD”と同じではありません。

データがクリップボードから正しく取り出された場合には、コマンドはシステム変数OKを1に設定します。クリップボードが空である場合や、指定したタイプのデータが存在しない場合には、コマンドは空のBLOBを返し、システム変数OKを0に設定してからエラー-102を生成します。クリップボードからデータを取り出すための十分なメモリがない場合には、コマンドはシステム変数OKを0に設定し、エラーコード-108を生成します。

例題

▼2つのボタンの以下のオブジェクトメソッドは、フォームに存在する配列asOptions（ポップアップメニュー、ドロップダウンリスト等）との間でデータをコピーまたは貼り付けます。

```
`「bCopyasOptions」ボタンのオブジェクトメソッド
    If(Size of array(asOptions)>0) コピーする物がある？
        ARRAY TO BLOB(asOptions ; $vxClipData) `BLOB内の配列要素を積算する
        CLEAR CLIPBOARD `クリップボードを空にする
        APPEND TO CLIPBOARD("artx" ; asOptions)
                                        `注：任意のデータタイプが選択される
    End if

    `「bPasteasOption」ボタンのオブジェクトメソッド
    If(Test clipboard("artx")>0) ` クリップボードに"artx"データが存在するか？
        GET CLIPBOARD("artx" ; $vxClipData)
                                        `クリップボードからデータを取り出す
        BLOB TO ARRAY($vxClipData ; asOptions)
                                        `BLOBデータを配列に登録する
        asOptions:=0 `配列用に選択した要素をリセットする
    End if
```

参照

GET PICTURE FROM CLIPBOARD、Get text from clipboard、Test clipboard

システム変数

データが正しく取り出された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。

■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

GET PICTURE FROM CLIPBOARD

GET PICTURE FROM CLIPBOARD(ピクチャ)

引数	タイプ		説明
ピクチャ	ピクチャ	←	クリップボードから取り出したピクチャ

説明

GET PICTURE FROM CLIPBOARDコマンドは、クリップボード内に存在するピクチャをピクチャフィールドやピクチャ変数に返します。

ピクチャがクリップボードから正しく取り出された場合には、コマンドはシステム変数OKを1に設定します。クリップボードが空である場合や、ピクチャが存在しない場合には、コマンドは空のピクチャを返し、システム変数OKを0に設定して、エラーコード-102を生成します。クリップボードからピクチャを取り出すための十分なメモリがない場合には、コマンドはシステム変数OKを0に設定し、エラーコード-108を生成します。

例題

▼以下のボタンのオブジェクトメソッドは、クリップボードに存在するピクチャをフィールド[従業員]写真フィールドに割り当てます。

```
If(Test clipboard("PICT")>0)
    GET PICTURE FROM CLIPBOARD([従業員]写真)
Else
    ALERT("クリップボードにピクチャはありません。")
End if
```

参照

GET CLIPBOARD、Get text from clipboard、Test clipboard

システム変数

データが正しく取り出された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。

■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

Get text from clipboard

Get text from clipboard → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	クリップボードにテキストがあればそれを返す

説明

Get text from clipboard関数は、クリップボードに存在するテキストを返します。

テキストがクリップボードから正しく取り出された場合には、コマンドはシステム変数OKを1に設定します。クリップボードが空である場合や、テキストが存在しない場合には、コマンドは空の文字列を返し、システム変数OKを0に設定して、エラーコード-102を生成します。クリップボードからテキストを取り出すための十分なメモリがない場合には、コマンドはシステム変数OKを0に設定し、エラーコード-108を生成します。

4th Dimensionのテキストフィールドとテキスト変数は、最大32,000文字まで含むことができます。クリップボード内にある文字が32,000文字を超える場合には、**Get text from clipboard**関数から返される結果は値を受け取るフィールドや変数に納められるときに切り捨てられます。非常に大量のクリップボードテキスト内容を処理するときには、まず最初に**Test clipboard**関数でデータのサイズを確認します。次に、そのテキストが32,000文字を超えている場合には、**Get text from clipboard**関数ではなく**GET CLIPBOARD**コマンドを使用します。

例題

▼以下の例は、クリップボード内のテキストがどの程度のサイズかをテストしてから、データのサイズに合わせて、テキストまたはBLOBとしてクリップボードからテキストを取り出します。

```
$vlSize:=Test clipboard("TEXT")
Case of
¥($vlSize<=0)
        ALERT("クリップボードにテキストはありません。")
    ¥($vlSize<=32000)
        $vtClipData:=Get text from clipboard
        If(OK=1)
            ` テキストを使って何らかの処理を実行する
        End if
    ¥($vlSize>32000)
```

```
            GET CLIPBOARD("TEXT" ; $vxClipData)
            If(OK=1)
                  ` BLOBを使った何らかの処理を実行する
            End if
      End case
```

参照

GET CLIPBOARD、GET PICTURE FROM CLIPBOARD、Test clipboard

システム変数

データが正しく取り出された場合には、システム変数OKは1に設定されます。それ以外の場合には、システム変数OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。

■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

SET PICTURE TO CLIPBOARD

SET PICTURE TO CLIPBOARD(ピクチャ)

引数	タイプ		説明
ピクチャ	ピクチャ	→	クリップボードに入れるコピーピクチャ

説明

SET PICTURE TO CLIPBOARDコマンドは、クリップボードを消去し、引数＜ピクチャ＞で渡したピクチャのコピーをクリップボードに入れます。

ピクチャをクリップボードに入れた後に、**GET PICTURE FROM CLIPBOARD**コマンドを使用するか、**GET CLIPBOARD**（"PICT" ; ...）を呼び出してそのピクチャを取り出すことができます。

ピクチャが正しくクリップボードに入れられた場合には、システム変数OKは1に設定されます。クリップボードにピクチャのコピーを入れるための十分なメモリがない場合には、システム変数OKは0に設定されますが、エラーは生成されません。

例題

▼ 以下の例はフローティングウインドウを使用して、配列asEmployeeNameを含むフォームを表示します。この配列は[従業員]テーブルの従業員名を一覧表示したものです。従業員の名字をクリックするたびに、その従業員の顔写真をクリップボードにコピーします。この配列用のオブジェクトメソッドを以下に示します。

```
If(asEmployeeName#0)
    QUERY([従業員];[従業員]名字=asEmployeeName{asEmployeeName})
    If(Picture size([従業員]写真)>0)
        SET PICTURE TO CLIPBOARD([従業員]写真)
                                        ` 従業員の顔写真をコピーする
    Else
        CLEAR CLIPBOARD ` 顔写真がないか、
                                    またはレコードが見つからない場合
    End if
End if
```

参照

APPEND TO CLIPBOARD、GET PICTURE FROM CLIPBOARD

システム変数またはシステムセット

ピクチャのコピーが正しくクリップボードに入れられた場合には、システム変数OKは1に設定されます。

SET TEXT TO CLIPBOARD

SET TEXT TO CLIPBOARD(テキスト)

引数	タイプ		説明
テキスト	テキスト	→	クリップボードに入れるテキスト

説明

SET TEXT TO CLIPBOARDコマンドは、クリップボードを消去し、クリップボードの中に<テキスト>で渡したテキストのコピーをクリップボードに入れます。

テキストをクリップボードに入れた後に、**Get text from clipboard**関数を使用するか、**GET CLIPBOARD**（"テキスト"；...）を呼び出すことでそのテキストを取り出すことができます。

テキストが正しくクリップボードに入れられた場合には、システム変数OKは1に設定されます。クリップボードにテキストのコピーを入れるための十分なメモリがない場合には、システム変数OKは0に設定されますが、エラーは生成されません。

4th Dimensionのテキスト式は最大32,000文字まで含めることができます。それよりも大きなテキスト値をコピーするには、テキストをBLOB内に積算して、**CLEAR CLIPBOARD**コマンドを呼び出してから**APPEND TO CLIPBOARD**（"テキスト"；...）を呼び出してください。

例題

▼ **APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照

APPEND TO CLIPBOARD、Get text from clipboard

システム変数またはシステムセット

テキストが正しくクリップボードに入れられた場合には、システム変数OKは1に設定されます。

Test clipboard

Test clipboard(データタイプ) → 数値

引数	タイプ		説明
データタイプ	文字列	→	4バイトの文字列データタイプ
戻り値	数値	←	クリップボードに格納されたデータサイズ（バイト単位）、またはエラーコード

説明

Test clipboard関数は、引数<データタイプ>にユーザが渡したデータがクリップボード内に存在するかどうかを調べます。

警告：<データタイプ>に渡す値は、大文字と小文字が区別されます。例えば、“abcd”は“ABCD”と同じではありません。

クリップボードが空である場合や、指定したタイプのデータが存在しない場合には、コマンドはエラーコード-102を生成します（下記の定義済定数の表を参照）。クリップボードに指定したタイプのデータが存在する場合には、コマンドはこのデータのサイズをバイト単位で返します。

ユーザが対象としたタイプのデータがクリップボードに存在することを確認した後は、以下のいずれか1つのコマンドを使用して、そのデータをクリップボードから取り出すことができます。

- クリップボードにあるデータがTEXTタイプの場合には、テキスト値を返す**Get text from clipboard**関数か、BLOBにテキストを返す**GET CLIPBOARD**コマンドを使用してそのデータを取得できます。
- クリップボードにあるデータがPICTタイプの場合には、ピクチャをピクチャフィールドまたは変数に返す**GET PICTURE FROM CLIPBOARD**コマンドか、ピクチャをBLOBに返す**GET CLIPBOARD**コマンドを使用してそのデータを取得できます。
- 上記以外の任意のデータタイプに対しては、データをBLOBに返す**GET CLIPBOARD**コマンドを使用します。

4th Dimensionには、以下のようなあらかじめ定義された定数があります。

定数	タイプ	値
No such data in clipboard	倍長整数	-102
Text data	文字列	テキスト
Picture data	文字列	PICT

例題

(1)以下のコードは、クリップボードにピクチャが存在するかどうかをテストし、存在する場合にはそのピクチャを4D変数にコピーします。

```
If(Test clipboard(Picture data) > 0) ` クリップボードにピクチャがあるか？
    `ピクチャがある場合、クリップボードからピクチャを取り出す
    GET PICTURE FROM CLIPBOARD($vPicVariable)
Else
    ALERT("クリップボードにピクチャはありません。")
End if
```

(2)通常、アプリケーションはテキストタイプまたはPICTタイプのデータをクリップボードに切り取ってコピーします。これは、ほとんどのアプリケーションでこの2つの標準データタイプが認識されているためです。ただし、アプリケーションは1つのデータの複数のインスタンスを異なるフォーマットでクリップボードに追加することができます。例えば、スプレッドシートの一部を切り取りまたはコピーするたびに、スプレッドシートアプリケーションはそのデータをSYLKフォーマットやTEXTフォーマットの他に、仮定的な「SPSH」フォーマットでも追加することができます。「SPSH」インスタンスにはアプリケーションのデータ構造を使用してフォーマットされたデータが含まれているはずです。SYLK形式には同じデータが含まれていますが、SYLKフォーマットを用いると、他の多くのスプレッドシートプログラムから認識されます。最後に、TEXTフォーマットには同じデータが含まれますが、SYLKや「SPSH」フォーマットに含まれる余分な情報は入っていません。この点で、4th Dimensionとその仮定的なスプレッドシートアプリケーション間での切り取り／コピー／貼り付けルーチンを記述している間に、「SPSH」フォーマットの内容を知り、SYLKデータの解析準備ができた場合には、以下のようなコードを作成することができます。

```
Case of
        `まず最初に、クリップボードに仮定的なスプレッドシート
                アプリケーションからのデータがあるのかをチェックする
    ¥(Test clipboard('SPSH') > 0)
        ` ...
        `次に、クリップボードにSylkデータが存在するかどうかをチェックする
    ¥(Test clipboard('SYLK') > 0)
        ` ...
        `最後に、クリップボードにTextデータが存在するかどうかをチェックする
    ¥(Test clipboard('TEXT') > 0)
` ...
End case
```

つまり、オリジナルの情報の大部分を含むデータのインスタンスをクリップボードから取り出そうとしています。

(3)**APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照

GET CLIPBOARD、GET PICTURE FROM CLIPBOARD、Get text from clipboard

SET CHANNEL

SET CHANNEL(ポート | 処理 {;セットアップ | ドキュメント})

引数	タイプ		説明
ポート¦処理	数値	→	使用するシリアルポート番号または ドキュメント処理番号
セットアップ¦ ドキュメント	数値	→	シリアルポートの設定または 処理対象のドキュメントファイル名

説明

SET CHANNELコマンドはシリアルポートまたはドキュメントファイルを開きます。このコマンドは、同時に1つのポートまたは1つのドキュメントファイルしか開くことができません。開いたシリアルポートを閉じるには**SET CHANNEL**(11)を用います。

注意：このコマンドは、本来最初の4Dコマンドで、シリアルポートやディスク上のドキュメントファイルを用いて作業を実行するために使用されました。それ以来、新しいコマンドが追加されています。現在では、ディスク上のドキュメントファイルを使った作業を実行する際には、Open document、Create document、Append documentといったコマンドを使用します。これらのコマンドを利用し、SEND PACKETやRECEIVE PACKETコマンドを使用してドキュメントファイルに文字を書き込んだり、読み込むことができます（この2つのコマンドはSET CHANNELとともに使用）。しかし、SEND VARIABLE、RECEIVE VARIABLE、SEND RECORD、RECEIVE RECORDコマンドを使用したい場合には、ディスクのドキュメントファイルにアクセスする際にSET CHANNELを使用しなくてはなりません。

SET CHANNELコマンドの説明は、２つの部分からなります。

■ シリアルポートに使用

■ ドキュメントに使用

シリアルポートに使用 - SET CHANNEL(ポート ;セットアップ)

SET CHANNELコマンドの第一の形式は、シリアルポートを開き、プロトコルやその他の情報をセットします。データの送信は、**SEND PACKET**コマンド、**SEND RECORD**コマンド、**SEND VARIABLE**コマンドで行います。データの受信は、**RECEIVE BUFFER**コマンド、**RECEIVE PACKET**コマンド、**RECEIVE RECORD**コマンド、**RECEIVE VARIABLE**コマンドで行います。

引数<ポート>

最初の引数<ポート>で、ポートとプロトコルを指定します。

シリアルポートは99まで使用できます（一度に1つ）。以下の表はポートの値を示します。

範囲	説明
0	無手順のプリンタポート（Macintosh）、またはCOM2（PC）
1	無手順のモデムポート（Macintosh）、またはCOM1（PC）
20	ソフトウェアフロー制御（XON/XOFF）のプリンタポート（Macintosh）またはCOM2（PC）
21	ソフトウェアフロー制御（XON/XOFF）のモデムポート（Macintosh）またはCOM1（PC）
30	ハードウェアフロー制御（RTS/CTS等）のプリンタポート（Macintosh）またはCOM2（PC）
31	ハードウェアフロー制御（RTS/CTS等）のモデムポート（Macintosh）またはCOM1（PC）
100から199	無手順のシリアル通信
201から299	ソフトウェアフロー制御（XON/XOFF）のシリアル通信
301から399	ハードウェアフロー制御（RTS/CTS等）のシリアル通信

重要：<ポート>に渡す値は、オペレーションシステムで認識される既存のシリアルCOMポートを示すものでなくてはなりません。例えば、101、103、125という値を使用できるようにするには、シリアルポートCOM1、COM3、COM25が必ず設定されている必要があります。

シリアルポートの注意点

MacintoshとWindowsでは2つのシリアルポートが提供されています。Macintoshではモデムポートとプリンタポート、WindowsではCOM1とCOM2ポートです。しかしながら、シリアルポートはエクステンションボードを使用して追加することができます。元は、4thDimensionは2つの標準のシリアルポートのみサポートし、追加されたポートのサポートが遅れていました。互換性のため、両方をサポートするシステムを残しました。

■ 標準のシリアルポート（プリンター／COM2またはモデム／COM1）にアクセスするには、ポートのパラメータの値を0、1、20、21、30、31のいずれかを指定します。(以前の使用法に相当)。または、100以上の値にします（以下の説明を参照)。

■ 追加したシリアルポートにアクセスするには、値をN+100にする必要があります（Nはポート番号)。ソフトウェア、ハードウェアそれぞれを選択するために（N+100）以上の値にするのに100か200を加えるのを考慮することもあります。

例題

1. 無手順のプリンタ／COM2ポートを使用する場合は、以下のシンタックスの1つを使用します。

    SET CHANNEL(0;param)

または、

    SET CHANNEL(102;param)

2. ソフトウェアフロー制御（XON/XOFF）のモデム／COM1ポートを使用する場合は、以下のシンタックスを使う必要があります。

    SET CHANNEL(21;param)

または

    SET CHANNEL(201;param)

3. ハードウェアフロー制御（RTS/CTS等）のCOM25を使用する場合は、以下のシンタックスを使用する必要があります。

    SET CHANNEL(325;param)

引数の設定

引数<セットアップ>で、転送速度、データビット、ストップビット、パリティを指定します。以下の表に示した転送速度、データビット、ストップビット、パリティの値を加算して<セットアップ>の値を指定します。例えば、転送速度を1200ボー、データビットを8ビット、ストップビットを1、パリティをなし、と設定する場合には、94+3072+16384+0と計算します。引数<セットアップ>の値として19550を指定します。

制御対象	セットアップ	設定
転送速度	380	300
	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	2	28800
	1	38400
	0	57600
	1022	115200
	1021	230400
データビット	0	5
	2048	6
	1024	7
	3072	8
ストップビット	16384	1
	-32768	1.5
	-16384	2
パリティ	0	なし
	4096	奇数
	12288	偶数

ディスク上のドキュメントとともに使用 - SET CHANNEL(処理; ドキュメント)

SET CHANNELコマンドの第2の形式は、ドキュメントファイルの作成やオープンおよび、クローズを行います。ドキュメントファイルコマンドと異なり、同時に1つのドキュメントファイルしか開くことができません。ドキュメントファイルは、読み込みと書き込みの両方が可能です。

引数＜処理＞で、オプション引数＜ドキュメント＞で指定したドキュメントファイルに対する処理を指定します。以下の表に、＜処理＞の値と、その結果としてドキュメントファイルに対して行われる処理を示します。“処理”の欄に引数＜処理＞の値を示します。“ドキュメント”の欄に＜ドキュメント＞の値を示します。“結果”の欄は指定した＜処理＞の結果を示します。

例えば、「ファイルを開く」ダイアログボックスを表示して、テキストファイルを開く場合には、以下のように記述します。

SET CHANNEL(13 ; "")

処理	ドキュメント	結果
10	文字列	文字列で指定したドキュメントファイルを開きます。指定したドキュメントファイルが存在しない場合には、そのドキュメントファイルを作成します。
10	""(空の文字列)	「ファイルを開く」ダイアログボックスを表示してファイルを開きます。この場合すべてのタイプのドキュメントファイルが表示されます。
11	なし	開いていたファイルを閉じます。
12	""(空の文字列)	「ファイル作成」ダイアログボックスを表示して新しいファイルを作成します。
13	""(空の文字列)	「ファイルを開く」ダイアログボックスを表示してファイルを開きます。TEXTタイプのドキュメントファイルだけが表示されます。

この表に示した処理はすべて、システム変数Documentに値を代入します。また、処理が正常に行われると、システム変数OKに1が代入されます。それ以外の場合には0が代入されます。

例題

RECEIVE BUFFER、**SET TIMEOUT**、**RECEIVE RECORD**の例題を参照して下さい。

参照

Append document、Create document、GET SERIAL PORT MAPPING、Open document、RECEIVE BUFFER、RECEIVE PACKET、RECEIVE RECORD、RECEIVE VARIABLE、SEND PACKET、SEND RECORD、SEND VARIABLE、SET TIMEOUT

SET TIMEOUT

SET TIMEOUT(秒)

引数	タイプ		説明
秒	数値	→	タイムアウトになるまでの秒数

説明

SET TIMEOUTコマンドは、シリアルポートコマンドの許容する待ち時間を設定します。シリアルポートコマンドが指定した時間＜秒＞以内に終了しないと、そのシリアルポートコマンドは、取り消されてエラー-9990が発生し、システム変数OKに0が代入されます。待ち時間は、コマンドが実行されるまでの時間です。データの送受信に要する時間ではない点に注意してください。

以前の設定値を取り消し、シリアルポートに対するタイムアウトの管理を中止する場合は、＜秒＞に0を指定します。

以下に、タイムアウトを適用することができるコマンドを示します。

コマンド

RECEIVE PACKET

RECEIVE RECORD

RECEIVE VARIABLE

例題

▼以下の例は、シリアルポートからデータを受信します。タイムアウトを設定し、**RECEIVE PACKET**でデータを受け取ります。データを設定した時間内に受け取れない場合は、エラーが発生します。

```
` 設定：モデムポート；9600ボー；データビット8；ストップビット1
SET CHANNEL(Macintosh Serial Port; Speed 9600 +
        Data Bits 8 + Stop Bits One + Parity None)
SET TIMEOUT(10)  ` タイムアウトを10秒に設定
RECEIVE PACKET(v ; Char(13))  ` 改行まで受信
If(OK=0)      ` エラー発生...
        ALERT("受信データエラー")      ` ユーザに通知
Else  ` その他の場合 ...
        [従業員]名前:=v      ` データ格納
End if
ON ERR CALL("")
```

参照

ON ERR CALL、RECEIVE BUFFER、RECEIVE PACKET、RECEIVE RECORD、RECEIVE VARIABLE

USE ASCII MAP

USE ASCII MAP(テーブル名 | * ; {I/O})

引数	タイプ	説明
テーブル名	文字列	使用するASCIIテーブルのドキュメント名
または *	文字列	使用するデフォルトのASCIIテーブル
I/O	数値	出力：0、入力：1 省略した場合、出力（0）

説明

USE ASCII MAPコマンドには2つの形式があります。第1の形式は、ディスクから＜テーブル名＞で指定したASCIIテーブルをロードし、それを使用します。＜I／O＞が0の場合は、ASCIIテーブルを出力用としてロードします。＜I／O＞が1の場合、ASCIIテーブルを入力用としてロードします。

ASCIIテーブルは、「ユーザ」モードの“ASCII出力テーブル編集”ダイアログボックスまたは“ASCII入力テーブル編集”ダイアログボックスで作成します。4th Dimensionは、ロードしたASCIIテーブルをデータベースとドキュメントファイルまたはシリアルポート間のデータの転送に使用します。転送処理には、TEXT（ASCII）、DIF、SYLKの各形式のファイルの“読み込み”と“書き出し”が含まれます。また、ASCIIテーブルは、**SEND PACKET**コマンド、**RECEIVE PACKET**コマンド、**RECEIVE BUFFER**コマンドで転送するデータにも適用されます。ただし、**SEND RECORD**コマンド、**SEND VARIABLE**コマンド、**RECEIVE RECORD**コマンド、**RECEIVE VARIABLE**コマンドには適用されません。

＜テーブル名＞に空の文字列を指定した場合は、**USE ASCII MAP**コマンドは、標準の「ファイルを開く」ダイアログボックスを表示します。ユーザは、ここでASCIIテーブルのドキュメントファイルを選択することができます。ASCIIテーブルを正常にロードすると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに0が代入されます。

USE ASCII MAPコマンドの第2の形式は、＜テーブル名＞の代わりにアスタリスク（*）を指定します。この場合、デフォルトのASCIIテーブルを復元します。＜I／O＞が0の場合は、出力用のASCIIテーブルを復元します。＜I／O＞が1の場合は、入力用のASCIIテーブルを復元します。デフォルトのASCIIテーブルは文字の変換を行いません。

注：アスタリスク（*）を指定すると、システム変数OKに0が代入されます。

例題

▼以下の例は、ディスクからASCIIテーブルをロードし、データを書き出します。最後にデフォルトのASCIIテーブルを復元します。

```
USE ASCII MAP("MyTable" ; 0)    ` ロードしたASCIIテーブルを使用
EXPORT TEXT([従業員] ; "MyText")    ` ASCIIﾃｰﾌﾞﾙを使ってデータ書き出し
USE ASCII MAP(* ; 0)    ` デフォルトのASCIIテーブルを復元
```

参照

EXPORT DIF、EXPORT SYLK、EXPORT TEXT、IMPORT DIF、IMPORT SYLK、IMPORT TEXT、Mac to Win、RECEIVE BUFFER、RECEIVE PACKET、SEND PACKET、Win to Mac

GET SERIAL PORT MAPPING

GET SERIAL PORT MAPPING(番号配列;名前配列)

引数	タイプ		説明
番号配列	数値配列	←	ポート番号の配列
名前配列	文字列配列	←	ポート名の配列

説明

GET SERIAL PORT MAPPINGコマンドは2つの配列を返し、現行のマシンのシリアルポート番号とシリアルポート名をそれぞれに代入します。

Mac OS Xでは、USBシリアルアダプターの使用時に、OSがポート番号を動的に割り当てるため、このコマンドが役立ちます。シリアルポートの実際のIDとは関係なく、その名前（固定）を用いてシリアルポートを取り扱うことができます。

参照

SET CHANNEL.

SEND PACKET

SEND PACKET({ドキュメントファイル参照番号 ;} パケット)

引数	タイプ		説明
ドキュメント	時間 ファイルまたは 参照番号	→	ドキュメントファイルの参照番号 カレントチャネル（シリアルポート またはドキュメントファイル）
パケット	文字列	→	送信するパケットまたはテキスト

説明

SEND PACKETコマンドは、シリアルポートやドキュメントファイルに＜パケット＞を送ります。＜ドキュメントファイル参照番号＞を指定した場合は、ドキュメントファイル参照番号の示すドキュメントファイルにパケットを書き込みます。＜ドキュメントファイル参照番号＞を指定しない場合は、あらかじめ**SET CHANNEL**コマンドで開かれたシリアルポートまたはドキュメントファイルにパケットを書き込みます。＜パケット＞はデータの一部であり、一般的には文字列です。

SEND PACKETコマンドを使用する前に**SET CHANNEL**コマンドでシリアルポートを開くか、またはドキュメントファイルコマンドでドキュメントファイルを開く必要があります。

既存のドキュメントファイルに書き込む場合は、**Append document**関数で開かれていない限り、最初の**SEND PACKET**コマンドはドキュメントファイルの先頭から書き始めます。それに続く**SEND PACKET**コマンドは、ドキュメントファイルが閉じられるまでパケットを書き加えます。

バージョン6における注意：このコマンドは、SET CHANNELを用いて開かれたドキュメントファイルに対して有効です。しかし一方で、Open document、Create document、Append documentで開かれたドキュメントファイルに関しては、ドキュメントファイル内での以下の書き込み位置（SEND PACKET）や読み込み位置（RECEIVE PACKET）を取得、または変更するためには、新しいコマンドであるGet document positionおよびSET DOCUMENT POSITIONを使用することができます。

SEND PACKETコマンドは、WindowsおよびMacintosh上でMacintoshのASCIIデータを書き込みます。MacintoshのASCIIは8ビットを使用しています。標準ASCIIでは下位の7ビットしか使用しないため、装置によっては、WindowsまたはMacintoshのようには8番目のビットを使用していないものもあります。このような装置として、PC-DOSやMS-DOSを使用したパーソナルコンピュータや、ImageWriterプリンタがあります。このような装置に、8ビットデータを送信する場合には、ASCIIコードを変換するために、ASCIIテーブルを定義し、USE ASCII MAPコマンドを実行してからSEND PACKETコマンドでデータ送信を行ってください。また、Mac to Win関数を使用することも可能です（この関数の例題を参照）。ただし、日本語DOSまたは日本語仕様のMS-DOSを使用したパーソナルコンピュータや、日本仕様のImageWriterプリンタに対しては、この処理は必要ありません。XON/XOFFのようなプロトコルでは、マシン間の接続を確立するのに下位のASCIIコードを使うことがあります。プロトコルへの干渉を行ったり、通信を絶ってしまうことがあるため、このようなASCIIコードは送信しないように注意してください。

例題

▼以下の例は、フィールドのデータをドキュメントファイルに書き込みます。この例では、固定長データとして書き込みます。固定長フィールドは常に長さが決まっています。フィールドのデータがフィールド長（15バイト）よりも短い場合には、その分のスペースを埋め込みます（つまり、指定された長さになるまでスペースを付け加えます）。固定長データの使用は、合理的な方法とはいえませんが、一部のコンピュータシステムやアプリケーションでは、広く使用されています。

```
ドキュメント:=Create document("")        ` ドキュメントファイルを作成
If(OK=1)        ` ユーザが新しいドキュメントファイルを開いた？
        For($i ; 1 ; Records in selection([従業員])) ` レコード数だけ繰り返す
        ` スペース15個と名字フィールドからパケットを作成し
                                                ドキュメントファイルに書き込む
                SEND PACKET(ドキュメント ; Change string(15*Char(Space);
                                                [従業員]名字 ; 1))
        ` スペース15個と名前フィールドからパケットを作成し、
                                                ドキュメントファイルに書き込む
                SEND PACKET(ドキュメント ; Change string(15*Char(Space);
                                                [従業員]名前 ; 1))
                NEXT RECORD([従業員])
        End for
` control-zをドキュメントファイルに書き込むこのコードはEOF(End-of-file)の
                                                意味を持つ
        SEND PACKET(ドキュメント ; Char(SUB ASCII Code))
        CLOSE DOCUMENT(ドキュメント)        ` ドキュメントファイルを閉じる
End if
```

参照

Get document position、RECEIVE PACKET、SET DOCUMENT POSITION

RECEIVE PACKET

RECEIVE PACKET({ドキュメントファイル参照番号 ;} 受信変数 ; 文字数 | 終了文字)

引数	タイプ		説明
ドキュメントファイル	時間 参照番号	→	ドキュメントファイルの参照番号またはカレントチャンネル（シリアルポートまたはドキュメントファイル）
受信変数	変数	→	受信データを格納する変数
文字数または	数値	→	受信する文字数（バイト）
終了文字	文字列		受信を終了する文字または記号

説明

RECEIVE PACKETコマンドは、シリアルポートまたはドキュメントファイルからデータを読み込みます。

＜ドキュメントファイル参照番号＞を指定した場合には、**Open document**、**Create document**、**Append document**で開かれたドキュメントファイルからデータを読み込みます。＜ドキュメントファイル参照番号＞を指定しない場合は、**SET CHANNEL**コマンドで開かれたシリアルポートかドキュメントファイルからデータを読み込みます。

ソースに関わらず、テキストか文字列である読み込まれた文字は、受信変数として返されます。特定の文字数まで読み込むためには、＜文字数＞にその数を渡します。特定の文字列（1桁以上の文字で構成される）が現われるまで文字を読み込むには、＜終了文字＞にその文字列を渡します（この文字列は＜受信変数＞に含まれません）。

ドキュメントを読み込む時に、＜文字数＞または＜終了文字＞が指定されていない場合、**RECEIVE PACKET**コマンドはドキュメントの最後まで読み込みます。しかし、文字列変数は固定長ですが、テキスト変数は32,000バイトまでのデータを格納できる点に注意してください。シリアルポートから読み込む際、**RECEIVE PACKET**コマンドはタイムアウトになるまで（指定されている場合）、あるいはユーザによって受信が中断されるまで待ち続けます。

RECEIVE PACKETコマンドを実行中に、Ctrl-Alt-Shift（Windows）またはCommand-Option-Shift（Macintosh）を押すことで、中断することができます。この中断はエラーコード-9994を発生させます。これは**ON ERR CALL**メソッドで検出できます。通常は、シリアルポート上で通信するときのみ、反応の中断を扱うことになります。

ドキュメントファイルを読み込む場合、**RECEIVE PACKET**コマンドは、ドキュメントファイルの先頭から読み込みを開始します。その後のデータ読み込みは、最後に読み込まれた文字の次から開始します。

バージョン6における注意：このコマンドは、SET CHANNELを用いて開かれたドキュメントファイルに対して有効です。しかし一方で、Open document、Create document、Append documentで開かれたドキュメントファイルに関しては、ドキュメントファイル内での以下の書き込み位置（SEND PACKET）や読み込み位置（RECEIVE PACKET）を取得、または変更するためには、新しいコマンドであるGet document positionおよびSET DOCUMENT POSITIONを使用することができます。

ファイルの最後を越えて読み込もうとした場合は、**RECEIVE PACKET**コマンドは、そのポイントまでに読み込んだデータを返し、システム変数OKに1を代入します。その次の**RECEIVE PACKET**コマンドは空の文字列を返し、システム変数OKに0を代入します。

RECEIVE PACKETの実行中に、「Ctrl＋Alt＋Shift」キー（Windows）または「command＋option＋shift」キー（Macintosh）を押して、受信を中断することができます。中断することにより、エラー-9994が発生します。**ON ERR CALL**を使用してインストールしたエラー処理メソッドにより、このエラーを検出することができます。通常、シリアルポート経由での通信の場合にのみ、受信の中断処理を実行する必要があります。

注：Windows上で、RECEIVE PACKETコマンドを使いドキュメントファイルの文字を読込む際に、Windows用の文字をMacintosh用の文字へ変換するためにASCII入力テーブルを使用しない場合（USE ASCII MAPコマンドを参照）には、Win to Mac関数を使用することができます（この関数の例題を参照）。

例題

▼以下の例は、20バイトのデータをシリアルポートから読み込み、変数“読込20”に格納します。

```
RECEIVE PACKET(読込20 ; 20)
```

▼以下の例は、変数“myDoc”に格納されたドキュメントファイル参照番号の示すドキュメントファイルからデータを読み込み、変数“vData”に格納します。ここでは改行（Char(13)）を発見するまで読み込みます。

```
RECEIVE PACKET(myDoc; vData; Char(13))
```

▼以下の例は、変数“myDoc”により参照されたドキュメントからデータを読み込み、変数“vData”に格納します。HTMLタグ“</TD>”（テーブルセルの終わり）が現われるまでデータを読み込みます。

```
RECEIVE PACKET(myDoc;vData;"</TD>")
```

▼以下の例は、ドキュメントファイルから読み込んだデータをフィールドに格納します。データは、固定長形式で格納されています。このメソッドは、サブルーチンを呼び出してデータの後ろに付随する不要なスペースを取り除きます。

```
$vhDocRef:=Open document("" ; "TEXT")` “テキスト”ファイルを開く
If(OK = 1)     ` ユーザがドキュメントファイルを開いたら？
    Repeat         ` データがなくなるまで繰り返す
        RECEIVE PACKET($vhDocRef ; $Var1 ; 15)
                                        `名字のデータを15バイト読み込む
        RECEIVE PACKET($vhDocRef ; $Var2 ; 15)
                                        ` 名前のデータを15バイト読み込む
        If(OK=1) ` まだドキュメントファイルを最後まで読み込んでない場合
            CREATE RECORD([従業員]) ` 新しいレコードを作成
            [従業員]名字:=Strip($Var1) ` 名字をフィールドに格納
            [従業員]名前:=Strip($Var2) ` 名前をフィールドに格納
            SAVE RECORD([従業員]) ` レコードを保存
        End if
    Until(OK=0)
    CLOSE DOCUMENT($vhDocRef) ` ドキュメントファイルを閉じる
End if
```

データに付随する不要なスペースを取り除くためのサブルーチン“Strip”を次に示します。

```
For($i ; Length($1) ; 1 ; -1) ` 文字列の最後からループを開始
    If($1[[$i]]#" ") ` スペース以外の場合...
        $i:=-$i ` ループを強制的に終了
    End if
End for
$0:=Delete string($1 ; -$i ; Length($1)) ` スペースを削除
```

参照

Get document position、RECEIVE PACKET、SEND PACKET、SET DOCUMENT POSITION、SET TIMEOUT

システム変数とセット

RECEIVE PACKETコマンドへの呼び出しの後に、パケットをエラーなく読み込むと、システム変数OKに1が代入されます。それ以外の場合、システム変数OKに0が代入されます。

RECEIVE BUFFER

RECEIVE BUFFER(受信変数)

引数	タイプ		説明
受信変数	変数	→	受信データを格納する変数

説明

RECEIVE BUFFERコマンドは、**SET CHANNEL**コマンドで前もって開いたシリアルポートからデータを読み込みます。シリアルポートは、コマンドで読み込まれるまでバッファの内容を保持しています。**RECEIVE BUFFER**コマンドは、バッファ中の文字を＜受信変数＞に格納して、バッファを消去します。バッファ中に文字が存在しなければ、＜受信変数＞は何も含みません。

Windows：

Windowsのシリアルポートバッファのサイズは限られています。つまり、バッファがオーバーフローする可能性があるということです。バッファがいっぱいになった後、新しい文字を受信すると、最も古いものと置き換えられます。古くなった文字は失われるため、新しい文字を受信する際は、すみやかにバッファを読み込むことが重要です。

Macintosh：

シリアルポートのバッファサイズは64バイトです。これは、バッファがオーバーフローするまで64バイトのデータを維持できることを意味します。オーバーフローした文字は失われます。したがって、バッファに文字が入力された時点で迅速に読み込む必要があります。

RECEIVE BUFFERコマンドは、バッファ中のデータが何であれ、それを即座に返す点が**RECEIVE PACKET**コマンドと異なります。**RECEIVE PACKET**コマンドはバッファ中に指定した文字を発見するまで、または指定した数の文字が入力されるまで待ちます。

RECEIVE BUFFERコマンドの実行中に、「Ctrl＋Alt＋Shift」キー（Windows）または「command＋option＋shift」キー（Macintosh）を押して、受信を中断することができます。中断することにより、エラー-9994が発生します。**ON ERR CALL**を使用してインストールしたエラー処理メソッドにより、このエラーを検出することができます。

例題

▼ 以下のプロジェクトメソッドLISTEN TO SERIAL PORTは、**RECEIVE BUFFER**コマンドを使用してシリアルポートからテキストを取得し、それをインタープロセス変数に追加します。

```
` シリアルポートの受信
` シリアルポート受信開始
SET CHANNEL(201; Speed 9600 + Data Bits 8 + Stop Bits One
                                        + Parity None) ` シリアルポートのオープン
<>IP_Listen_Serial_Port:=True
While(<>IP_Listen_Serial_Port)
        RECEIVE BUFFER($vtBuffer)
        If((Length($vtBuffer)+ Length(<>vtBuffer))>MAXTEXTLEN)
                <>vtBuffer:=""
        End if
        <>vtBuffer:=<>vtBuffer+$Buffer
End while
<>Buffer:=""
<>Len=0
```

この時点で、他のプロセスからインタープロセス変数<>vtBufferを読み込み、シリアルポートから受信したデータの処理を行うことができます。

シリアルポートの監視を中断するには、以下のコードを実行します。

```
` シリアルポート受信の終了
<>IP_Listen_Serial_Port:=False
```

プロセス間でのコンフリクトを避けるために、セマフォを利用してインタープロセス変数<>vtBufferへのアクセスを保護する必要がある点に注意してください。詳細は**Semaphore**コマンドを参照してください。

参照

ON ERR CALL、RECEIVE PACKET、Semaphore、SET CHANNEL、変数

SEND VARIABLE

SEND VARIABLE(変数)

引数	タイプ		説明
変数	変数	→	送信する変数

説明

SEND VARIABLEコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたはドキュメントファイルに＜変数＞を送信します。＜変数＞は**RECEIVE VARIABLE**コマンドでなければ読み込むことのできない特別な内部フォーマットで送信します。**SEND VARIABLE**コマンドは、変数のすべての情報（タイプと値）を送信します。

注：

1. このコマンドを使用してドキュメントファイルに＜変数＞を送信する場合、SET CHANNELコマンドを使ってあらかじめドキュメントファイルを開いておかなければなりません。Open document、Create document、Append documentで開かれたドキュメントファイルに対して、SEND VARIABLEを使用することはできません。
2. このコマンドでは配列変数はサポートされません。配列をドキュメントファイルまたはシリアルポート経由で送受信を行いたい場合、新しいBLOBコマンドを使用します。

例題

▼以下の例は、シリアルポートに変数を送信します。

```
SET CHANNEL(1 ; 10+3072+16384+0)
SEND VARIABLE(my var)
```

参照

RECEIVE RECORD、RECEIVE VARIABLE、SEND RECORD、SET CHANNEL

RECEIVE VARIABLE

RECEIVE VARIABLE(変数)

引数	タイプ		説明
変数	変数	→	受信する変数

説明

RECEIVE VARIABLEコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたはドキュメントファイルから**SEND VARIABLE**コマンドで送信した＜変数＞を受信します。

インタプリタモードでは、このコマンドのコール前に変数が存在しない場合、変数が作成され、受信内容に応じたタイプ付けが行われます。コンパイル後モードでは、変数のタイプは受信するものと同じでなくてはなりません。両方の場合とも、変数の内容は受信した値で置き換えられます。

注：

1. このコマンドを使用してドキュメントファイルに＜変数＞を送信する場合、SET CHANNELコマンドを使ってあらかじめドキュメントファイルを開いておかなければなりません。Open document、Create document、Append documentで開かれたドキュメントファイルに対して、SEND VARIABLEを使用することはできません。

2. このコマンドでは配列変数はサポートされません。配列をドキュメントファイルまたはシリアルポート経由で送受信を行いたい場合、新しいBLOBコマンドを使用します。

3. RECEIVE VARIABLEコマンドの実行中に、「Ctrl＋Alt＋Shift」キー（Windows）または「command＋option＋shift」キー（Macintosh）を押して、受信を中断することができます。中断することにより、エラー-9994が発生します。ON ERR CALLを使用してインストールしたエラー処理メソッドにより、このエラーを検出することができます。通常、シリアルポート経由での通信の場合にのみ、受信の中断処理を実行する必要があります。

レコードを正常に受信した場合、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに0が代入されます。

例題

▼以下の例は、シリアルポートから変数を受信します。

```
SET CHANNEL(1 ; 10+3072+16384+0)
SET TIMEOUT(20)  `エラーによる割り込みの備え
RECEIVE VARIABLE(MyVar)
```

SEND RECORD

SEND RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードを送信するテーブル 省略した場合、デフォルトテーブル

説明

SEND RECORDコマンドは、＜テーブル＞のカレントレコードを**SET CHANNEL**コマンドで開いたシリアルポートまたはドキュメントファイルに送信します。レコードは**RECEIVE RECORD**コマンドでなければ読み込むことのできない特別な内部フォーマットで送信します。カレントレコードが存在しなければ、**SEND RECORD**コマンドは何も行いません。

SEND RECORDコマンドは、サブレコードやBLOB、ピクチャも含めてレコードのすべての内容を送信します。

重要：SEND RECORD および RECEIVE RECORDコマンドを使用して、レコードの送受信を実行する場合、送信元と送信先のテーブル構造は互換性のあるものでなくてはなりません。互換性がない場合、RECEIVE RECORDコマンドの実行時に4Dがテーブル定義に応じて値を変換します。

注：このコマンドを使用してドキュメントファイルにレコードを送信する場合、SET CHANNELコマンドを使ってあらかじめドキュメントファイルを開いておかなければなりません。Open document、Create document、Append documentで開かれたドキュメントファイルに対して、SEND RECORDを使用することはできません。

例題

▼以下の例は、テーブルのすべてのレコードを送信します。

```
SET CHANNEL(10 ; "保管")｀テーブルを開く
ALL RECORDS([MyTable])｀すべてのレコードを選択
For($i ; 1 ; Records in table([MyTable]))  ｀レコードの数だけ繰り返す
    SEND RECORD([MyTable])｀レコード送出
    NEXT RECORD([MyTable])｀以下のレコードへ移動
End for
SET CHANNEL(11)｀テーブルを閉じる
```

参照

RECEIVE RECORD、RECEIVE VARIABLE、SEND VARIABLE

RECEIVE RECORD

RECEIVE RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	レコードを受信するテーブル 省略した場合、デフォルトテーブル

説明

RECEIVE RECORDコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたはドキュメントファイルからレコードを<テーブル>に受信します。受信するレコードは、**SEND RECORD**コマンドで送信したものでなければなりません。**RECEIVE RECORD**コマンドを実行すると、テーブルに新しいレコードが自動的に作成されます。レコードを正しく受信した時点で、**SAVE RECORD**コマンドを使用して新しいレコードをディスクに保存します。

RECEIVE RECORDコマンドは、サブレコードやBLOB、ピクチャも含めてレコードのすべての内容を受信します。

重要：SEND RECORD および RECEIVE RECORDコマンドを使用して、レコードの送受信を実行する場合、送信元と送信先のテーブル構造は互換性のあるものでなくてはなりません。互換性がない場合、RECEIVE RECORDコマンドの実行時に4Dがテーブル定義に応じて値を変換します。

注：

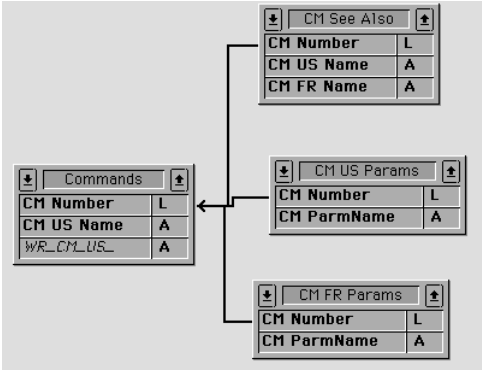
1. このコマンドを使用してドキュメントファイルにレコードを送信する場合、SET CHANNELコマンドを使ってあらかじめドキュメントファイルを開いておかなければなりません。Open document、Create document、Append documentで開かれたドキュメントファイルに対して、RECEIVE RECORDを使用することはできません。
2. RECEIVE RECORDコマンドの実行中に、「Ctrl＋Alt＋Shift」キー（Windows）または「command＋option＋shift」キー（Macintosh）を押して、受信を中断することができます。中断することにより、エラー-9994が発生します。ON ERR CALLを使用してインストールしたエラー処理メソッドにより、このエラーを検出することができます。通常、シリアルポート経由での通信の場合にのみ、受信の中断処理を実行する必要があります。

レコードを正常に受信すると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに0が代入されます。

例題

データのアーカイブ作成や、異なる場所で使用されている同じシングルユーザデータベース間でデータをやり取りする際には、**SEND VARIABLE**、**SEND RECORD**、**RECEIVE VARIABLE**、**RECEIVE RECORD**コマンドを組み合わせて使用するとよいでしょう。**EXPORT TEXT**および**IMPORT TEXT**等のデータ読み込み／書き出しコマンドを利用して、4Dデータベース間でデータのやり取りを実行することが可能です。しかし、データの中にグラフィックやサブテーブル、リレートテーブルが含まれる場合には、**SEND RECORD**と**RECEIVE RECORD**コマンドを使う方がはるかに便利です。

例えば、今お読みのドキュメントは4Dと4D Writeを使用して作成されました（英語版ドキュメント）。別々の場所に居る複数の製作者が作業を実行するため、異なるデータベース間でデータをやり取りする簡単な方法が必要となりました。以下の図はこのデータベースストラクチャを簡単に表わしたものです。



テーブル[Commands]には、各コマンドやトピックに関する説明が納められます。テーブル[CM US Params]および[CM FR Params]にはそれぞれ、英語版またはフランス語版の各コマンドに対する引数のリストが納められます。また、テーブル[CM See Also]には、各コマンドに対する参照としてリストされるコマンドが納められています。したがって、データベース間でドキュメントのやり取りを実行するということは、[Commands]レコードとそれにリレーとするレコードを送信するという意味になります。これを実行するには、**SEND RECORD**と**RECEIVE RECORD**コマンドを使用します。さらに、**SEND VARIABLE**と**RECEIVE VARIABLE**コマンドを使い、読み込み／書き出しドキュメントにタグを付けます。

ドキュメントの書き出しを実行するプロジェクトメソッド（簡約した）を以下に示します。

```
` CM_EXPORT_SELプロジェクトメソッド
` このメソッドでは [Commands] テーブルのカレントセレクションを使って
                                        作業を実行する。
```

```
SET CHANNEL(12;"") ` チャネルを開き、ユーザにドキュメントを作成させる
If(OK=1)
	` 内容を表わす変数でドキュメントにタグを付加する
	` 注：プロセス変数BUILD_LANGは
	` US(English) データが送信されたのか、FR(French) データが送信された
	                                                    のかを示す
	$vsTag:="4DV6COMMAND"+BUILD_LANG
	SEND VARIABLE($vsTag)
		` [Commands] が何件送られたかを示す変数を送信
	$vlNbCmd:=Records in selection([Commands])
	SEND VARIABLE($vlNbCmd)
	FIRST RECORD([Commands])
		` 各コマンドに対し
	For($vlCmd;1;$vlNbCmd)
		` [Commands] レコードを送信
		SEND RECORD([Commands])
		` リレートしたレコードをすべて選択
		RELATE MANY([Commands])
		` 言語に応じて、次に続く引数の数を示す変数を送信
		Case of
			¥(BUILD_LANG="US")
				$vlNbParm:=Records in selection([CM US Params])
			¥(BUILD_LANG="FR")
				$vlNbParm:=Records in selection([CM FR Params])
		End case
		SEND VARIABLE($vlNbParm)
			` 引数レコードを送信（存在する場合）
		For($vlParm;1;$vlNbParm)
			Case of
				¥(BUILD_LANG="US")
					SEND RECORD([CM US Params])
					NEXT RECORD([CM US Params])
				¥(BUILD_LANG="FR")
					SEND RECORD([CM FR Params])
					NEXT RECORD([CM FR Params])
			End case
		End for
		` 何件の“参照”が後に続くかを示す変数を送信
		$vlNbSee:=Records in selection([CM See Also])
		SEND VARIABLE($vlNbSee)
		` [See Also] レコードを送信（存在する場合）
```

```
                For($vlSee;1;$vlNbSee)
                    SEND RECORD([CM See Also])
                    NEXT RECORD([CM See Also])
                End for
                ` 以下の [Commands] レコードに進み、書き出しを続ける
                NEXT RECORD([Commands])
            End for
            SET CHANNEL(11) ` ドキュメントのクローズ
        End if
```

ドキュメントの書き出しを実行するプロジェクトメソッド（簡約した）を次に示します。

```
    ` CM_IMPORT_SEL プロジェクトメソッド
    SET CHANNEL(10;"") ` ユーザに既存のドキュメントを開いてもらう
    If(OK=1) ` ドキュメントが開かれている場合
        RECEIVE VARIABLE($vsTag) ` 求めるタグ変数の受信を試みる
        If($vsTag="4DV6COMMAND@") ` 正しいタグを取得したか？
            $CurLang:=Substring($vsTag;Length($vsTag)-1)` タグを元に言語
                                                        を取り出す
            If(($CurLang="US") |($CurLang="FR")) `有効な言語を取得したか？
                RECEIVE VARIABLE($vlNbCmd)
                ` このドキュメントにはいくつのコマンドが存在するか？
                If($vlNbCmd>0) ` 少なくとも1つ存在するなら
                    For($vlCmd;1;$vlNbCmd) ` アーカイブされた
                        [Commands] 各レコードに対してレコードを受信
                        RECEIVE RECORD([Commands])
                        ` 新規レコードの保存、または既存のレコード
                        にその値をコピーするサブルーチンを呼び出す
                        CM_IMP_CMD($CurLang)
                        ` 引数の数を受信（ある場合）
                        RECEIVE VARIABLE($vlNbParm)
                        If($vlNbParm>=0)
                        ` RECEIVE RECORDをコールした後、
                          新規レコードを保存、または既存レコードに
                                コピーするサブルーチンを呼び出す
                            CM_IMP_PARM($vlNbParm;$CurLang)
                        End if
                        ` “参照”の数を受信（ある場合）
                        RECEIVE VARIABLE($vlNbSee)
                        If($vlNbSee>0)
                        ` RECEIVE RECORDをコールした後、
                          新規レコードを保存、または既存レコードに
                                コピーするサブルーチンを呼び出す
```

```
                                    CM_IMP_SEEA($vlNbSee;$CurLang)
                                End if
                            End for
                        Else
                            ALERT("この書き出しドキュメントのコマンド数が
                                                        正しくない")
                        End if
                    Else
                        ALERT("この書き出しドキュメントの言語が不明")
                    End if
                Else
                    ALERT("このドキュメントはコマンド書き出しドキュメントでは
                                                        ありません")
                End if
                SET CHANNEL(11)｀ドキュメントのクローズ
            End if
```

データ受信中にシステム変数OKの評価も行わず、またエラーの検出も行っていない点に注意してください。しかし、ドキュメントそのものを表わすドキュメントに変数を保存するため、これらの変数がいったん受信されると意味を持つ場合には、エラーの可能性は低くなります。例えば、ユーザが誤ったドキュメントをオープンした場合、この処理は最初の判定式で即座に中断されます。

参照

RECEIVE VARIABLE、SEND RECORD、SEND VARIABLE

統合された4th Dimensionのコンパイラは、データベースアプリケーションをアセンブラ命令に翻訳します。コンパイラの利点は次の通りです。

■「スピード」：データベースの実行速度を3倍から1000倍速くします。

■「コードチェック」：データベースアプリケーションのコードを系統的にチェックし、論理的矛盾や構文的矛盾を検出します。

■「データベースの保護」：データベースをコンパイルすると、インタプリタコードが削除されます。コンパイルされたデータベースは、ストラクチャやメソッドを表示、または修正することができないこと以外は、オリジナルのデータベースと機能上は同一のものです。

■「スタンドアロンかつダブルクリックで起動するアプリケーション」：コンパイル後のデータベースは、独自のアイコンを持つスタンドアロンアプリケーション（.EXEファイル）に作り変えることもできます。

4D Compilerはカスタムアイコンを持ったスタンドアロンアプリケーション（.EXEファイル）を作成することができます。

この節のコマンドは、コンパイラの使用に関連があります。これらのコマンドは、データベース中のデータタイプを定義します。**IDLE**コマンドは、コンパイルされたデータベースで特別に使用されるコマンドです。

C_BLOB	**C_LONGINT**	**C_STRING**
C_BOOLEAN	**C_PICTURE**	**C_TEXT**
C_DATE	**C_POINTER**	**C_TIME**
C_GRAPH	**C_REAL**	**IDLE**
C_INTEGER		

IDLEコマンド以外のこれらのコマンドは、変数を宣言し、それらを指定したデータタイプとしてキャストします。変数を宣言することによって、変数のデータタイプに関連する曖昧さが解決されます。変数がこれらのコマンドのいずれかで宣言されていない場合には、コンパイラが変数のデータタイプを判断しようとします。フォームで使用される変数のデータタイプは、多くの場合、コンパイラで判断するのは困難です。このため、開発者がこれらのコマンドによってフォームで使用される変数を宣言することが特に重要です。

注：時間を節約するために、コンパイラウインドウにあるオプションを使用し、変数定義メソッドの生成や更新を行うことができます。このオプションは、データベース内で使用されているすべての変数を調査してタイプ指定を行う変数定義メソッドを自動作成します。

配列は変数で、コンパイルについては標準変数と同じ規則に従わなければなりません。配列の宣言命令は「配列」テーマ内でまとめられます。

コンパイルするコードの記述に関する一般規則

■ 2つ以上のメソッドまたは変数に同じ名前を付けてはなりません。また、メソッドと変数を同じ名前にすることはできません。

■ バージョン1の4th Dimensionで使用されていたような変数の間接参照はできません。パーセント記号（%）を使って間接的に変数を参照する文字型間接参照も、中カッコ（{...}）を用いる数値型間接参照も実行することはできません。中カッコ（{...}）は定義された配列の要素を指定する場合にのみ使用されます。しかし、引数による間接参照は使用できます。

■ あらゆる変数や配列のデータタイプは、変更することはできません。

■ 1次元配列を2次元配列に、また2次元配列を1次元配列に変更することはできません。

■ 文字(列)変数の長さや、文字列配列の要素の長さは変えられません。

■ コンパイラにより変数のタイプ定義は行われますが、フォーム上の変数のように、データタイプが明確でない場合は、コンパイラ命令を使用して変数のデータタイプを指定するべきです。

■ 変数を明示的にタイプ定義するもう1つの理由は、コードの最適化です。このことは、特にカウンタとして使用される変数に対して当てはまります。最大限の能力を得るために、可能な限り倍長整数型の変数を使用してください。

■ 変数をクリアする（ヌルに初期化する）には、変数の名前を用いて**CLEAR VARIABLE**コマンドを使用します。**CLEAR VARIABLE**コマンド内で変数の名前を表わす文字列を使用してはいけません。

■ **Undefined**関数は、常にFalseを返します。変数は常に定義されています。

■ 通常、倍長整数変数および整数変数に対する数値演算は、デフォルトの数値タイプ（実数）に対する演算よりもはるかに高速に実行されます。

これらの規則については、下記のセクションを参照してください。

■ **コンパイラ命令：**いつ、どこでコンパイラ命令を記述するかについて説明します。

■ **タイプ定義ガイド：**4th Dimensionのデータベースをコンパイルする際に起こりうる様々な種類の障害について説明します。

■ **シンタックスの詳細：**4th Dimensionのコマンドについての追加情報を提供します。

■ **最適化のヒント：**コンパイルモードで起動したアプリケーションをスピードアップするためのヒントを提供します。

例題

(1)以下は、コンパイラ用の基本的な変数宣言の例です。

```
C_BLOB(vxMyBlob) `プロセス変数vxMyBlobはBLOBタイプの変数として宣言される
C_BOOLEAN(<>On Windows) `インタープロセス変数<>OnWindowsはBoolean
                                        タイプの変数として宣言される
C_DATE($vdCurDate) `ローカル変数$vdCurDateは日付タイプの変数として
                                                宣言される
C_GRAPH(vg1 ; vg2 ; vg3) `3つのプロセス変数vg1、vg2、vg3はグラフタイプの
                                        変数として宣言される
```

(2)以下の例では、プロジェクトメソッド「OneMethodAmongOthers」は3つの引数を宣言します。

```
`「OneMethodAmongOthers」プロジェクトメソッド
` OneMethodAmongOthers(実数 ; 整数 { ; 倍長整数})
` OneMethodAmongOthers(合計 ; 割合 { ; 比率})
C_REAL($1) ` 実数タイプの第1引数
C_INTEGER($2) ` 整数タイプの第2引数
C_LONGINT($3) ` 倍長整数タイプの第3引数
```

(3)以下の例では、プロジェクトメソッド「Capitalize」は文字列引数を受け付け、文字列の結果を返します。

```
`「Capitalize」プロジェクトメソッド
` Capitalize(文字列) → 文字列
` Capitalize(元文字列) → 大文字に変換された文字列
C_STRING(255 ; $0 ; $1)
$0:=Uppercase(Substring($1 ; 1 ; 1))+Lowercase(Substring($1 ; 2))
```

(4)以下の例では、プロジェクトメソッド「SEND PACKETS」はテキスト引数の変数番号が後ろにある時間引数を受け付けます。

```
`「SEND PACKETS」プロジェクトメソッド
` SEND PACKETS(時間 ; テキスト { ; テキスト2... ; テキストN})
` SEND PACKETS(ドキュメント参照番号 ; データ { ; データ2... ; データN})
C_TIME($1)
```

```
C_TEXT(${2})
C_LONGINT($vlPacket)
For($vlPacket ; 2 ; Count parameters)
    SEND PACKET($1 ; ${$vlPacket})
End for
```

(5)以下の例では、プロジェクトメソッド「COMPILER_Param_Predeclare28」はコンパイラ用の別のプロジェクトメソッドのシンタックスを事前に宣言します。

```
`「COMPILER_Param_Predeclare28」プロジェクトメソッド
C_REAL(OneMethodAmongOthers ; $1) `OneMethodAmongOthers(実数;整数{;倍長整数})
C_INTEGER(OneMethodAmongOthers ; $2) ` ...
C_LONGINT(OneMethodAmongOthers ; $3) ` ...
C_STRING(Capitalize ; 255 ; $0 ; $1) ` Capitalize(文字列) → 文字列
C_TIME(SEND PACKETS ; $1) `SEND PACKETS(時間;テキスト{;テキスト2 ; テキストN})
C_TEXT(SEND PACKETS ; ${2}) ` ...
```

参照

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

コンパイラコマンドの使用

変数のデータタイプ

4th Dimensionの変数には次の3種類があります。

■ ローカル変数

■ プロセス変数

■ インタープロセス変数

変数の詳細な性質については「変数」の節を参照してください。プロセス変数とインタープロセス変数はコンパイラにとって構造的には同じです。

コンパイラがどのプロセスにどの変数を使用するかを知ることはできないので、プロセス変数の使用にはインタープロセス変数よりも注意が必要です。新規プロセスの開始時にすべての変数は複製されます。プロセス変数はプロセスごとに個別の値を持つことができますが、タイプはデータベース全体を通して同じです。

変数タイプ

すべての変数にはタイプがあります。「データタイプ」の節に記述されるとおり、変数には以下12の異なるタイプがあります。

■ ブール

■ 文字列

■ 日付

■ 整数

■ 倍長整数

■ グラフ

■ 時間

■ ピクチャ

■ 数値（または実数）

■ ポインタ

■ テキスト

■ BLOB

また配列には以下の9タイプがあります。

■ ブール値

■ 文字列値

■ 日付値

■ 整数値

■ 倍長整数値

■ ピクチャ値

■ 実数値

■ ポインタ値

■ テキスト値

シンボルテーブルの作成

インタプリタモードでは、変数は2つ以上のデータタイプを持つことが可能です。これは、コードをコンパイルするのではなく解釈するためです。4th Dimensionはそれぞれのステートメントを別々に解釈し、そのコンテキストを理解します。しかしコンパイルの場合は状況が異なります。インタプリタが一行ずつ実行処理するのに対して、コンパイルの際はデータベース全体を一度に処理します。

コンパイラの処理方法は以下のとおりです。

■ データベース内のオブジェクトを体系的に解析します。ここでいうオブジェクトとはデータベース、プロジェクト、フォーム、トリガオブジェクトメソッドです。

■ オブジェクトを調べ、データベース内で使われる各変数のデータタイプを決定し、変数やメソッドのテーブル（シンボルテーブル）を生成します。

■ すべての変数のデータタイプが確定すると、データベースの翻訳（コンパイル）を行います。ただし、各変数のデータタイプが確定できなければデータベースをコンパイルすることはできません。

同じ変数名で異なる2つのデータタイプが見つかった場合、どちらか1つが優先して採用されることはありません。オブジェクトのタイプを設定してメモリアドレスを割り当てるために、コンパイラはオブジェクトについての名前やデータタイプなど正確に認識する必要があります。コンパイラはデータタイプからそのサイズを判断します。コンパイルされたデータベースごとにマップが作られ、各変数の名前（もしくは識別子)、ロケーション（もしくはメモリアドレス)、変数が占める容量（データタイプによってわかるもの）を記録します。このマップを「シンボルテーブル」と呼びます。環境設定のオプションでコンパイル時にこのテーブルをファイル形式で作成するかどうかを選ぶことができます。このマップは、コンパイラメソッドの自動生成にも使用されます。

変数のタイプ設定

コンパイラは変数の判別基準を守り、以下の方法でタイプを設定します。

■ 変数のタイプが設定されていない場合、自動的にコンパイラが代わりにタイプを設定します。曖昧でない場合は可能な限り、使用目的から変数タイプを設定します。

例えば、

```
V1:=True
```

と記述した場合、変数＜V1＞のデータはブールタイプになります。

同様に以下の場合、

```
V2:= "これは、例題です"
```

と記述すると＜V2＞はテキストタイプの変数になります。

また、上の例ほど明確でない場合でも以下のように変数のデータタイプを設定することができます。

```
V3:= V1      ` V3はV1と同じタイプです。
V4:= 2*V2    ` V4はV2と同じタイプです。
```

また、4th Dimensionのコマンドやメソッドに対するコールからも変数のデータタイプは決定されます。例えばブールタイプと日付タイプの引数をメソッドに渡した場合、コンパイラは呼び出されたローカル変数$1および$2にブールタイプと日付タイプを割り当てます。

推測からデータタイプが決定される場合、環境設定で指定されていない限りは整数、倍長整数、文字列のような制限のあるデータタイプが割り当てられることはありません。デフォルトとして常に最も広範囲なタイプが割り当てられます。

例えば、

```
Number=4
```

と記述した場合、ここでは4が整数であっても他の状況では4.5になる可能性もあるので、"Number"には"実数"のデータタイプが割り当てられます。変数のタイプを整数、倍長整数または文字列にする場合は"コンパイラコマンド"を使います。これらのデータタイプはメモリ占有率が少ないため、処理速度が早くなります。

すでに変数タイプを設定済みで、定義の整合性も完全であると確信した場合、この作業を繰り返さないようコンパイラに指示することができます。定義が徹底されておらず、不完全な場合、コンパイル終了時にエラーメッセージが表示され、必要な変更を行うよう求められます。

■ コンパイラコマンドを使用すると、変数タイプを明示的に定義することができます。使用方法は次の通りです。

```
C_BOOLEAN(Var)
```

この命令で変数"Var"という変数を作り、そのタイプはブールであることをコンパイラに指示します。

アプリケーションでコンパイラコマンドが使用されていれば、4D Compilerがタイプを推測するという作業をしなくて済みます。

コンパイラコマンドは、代入や用途から得られた結果より優先されます。コンパイラコマンド**C_INTEGER**で定義した変数は、実際には**C_LONGINT**で定義したものと同じです。これらは実際には、-2147483648から+2147483647までの倍長整数です。

コンパイラコマンドが必要な場合

コンパイラコマンドは以下のような場合に便利です。

■ コンパイラで前後関係から変数のデータタイプを決定できない場合

■ 使用目的からタイプを決定したくない場合。またコンパイラコマンドを使用するとコンパイル時間が短縮できる場合

曖昧になる場合

コンパイラで変数タイプを決定できないこともあります。このような場合、4D Compilerからは必ずエラーメッセージが出力されます。4D Compilerでデータタイプを決定できない場合は、主に3つの原因があります。

■ データタイプが複数の場合

■ コンパイラが決定したタイプが曖昧な場合

■ タイプを判断する決め手がない場合

データタイプが複数の場合

データベース内で、変数が異なるタイプで再度定義された場合、コンパイラはエラーと判断します。このエラーは簡単に訂正できます。

コンパイラは最初に見つけたタイプを採用し、2回目以降は1回目に割り当てたデータタイプを使用します。

簡単な例を示します。

```
Variable:=True    `メソッドA
Variable:="月は緑色"        `メソッドB
```

"メソッドA"が"メソッドB"の前にコンパイルされると、ステートメント"Variable:="月は緑色""は、"Variable"に対するデータタイプの変更と見なされます。コンパイラは、データタイプが再定義されていて、修正が必要であることをユーザに示します。ほとんどの場合、2番目の変数の名前を変更すればエラーは修正できます。

コンパイラが決定したタイプが曖昧な場合

4D Compilerが決定したオブジェクトのタイプを4D Compiler自身がそのオブジェクトにとって適切なタイプはないと判断することがあります。この場合、コンパイラコマンドで、変数タイプを明確に指定しなければなりません。

以下はアクティブオブジェクト用のデフォルト値を使用した例です。

フォーム内で、コンボボックス、ポップアップメニュー、タブコントロール、ドロップダウンリスト、メニュー／ドロップダウンリスト、およびスクロールエリアのデフォルト値を割り当てることができます。これはプロパティリストの「データソース」テーマ内にあるデフォルト値用の「編集」ボタンを使用して設定することができます（詳細は、『4th Dimensionデザインリファレンス』マニュアルを参照してください）。デフォルト値は、オブジェクトの名前である配列内に自動的にロードされます。オブジェクトをメソッド内で使用しない場合、4D Compilerはテキスト配列として定義しますが、動作上、表示の初期化を行わなければならない場合には、次のようにコーディングします。

```
Case of
    ¥ (Form event=On Load)
        MyPopUp:=2
        ...
End case
```

この場合4D Compilerは"MyPopup"を実数として定義しようとしますが、すでにテキスト配列と定義しており、この2つのタイプの不一致によりデータタイプを決定できなくなります。この場合、フォームメソッドあるいはコンパイラメソッドで明示的に配列を定義する必要があります。

```
Case of
    ¥ (Form event=On Load)
        ARRAY TEXT(MyPopUp;2)
        MyPopUp:=2
        ...
End case
```

タイプを判断する決め手がない場合

変数の宣言なしに使用されていて、前後関係からデータタイプを決定できないような状況です。こうなると、コンパイラにとっての決め手はコンパイラコマンドしかありません。こうした状況は、主に次の4つのコンテキストにおいて起こります。

■ ポインタを使用する場合

■ 複数のシンタックスコマンドを使用する場合

■ データタイプが異なるオプション引数が付いたコマンドを使用する場合

■ URL経由で呼び出された4Dメソッドを使用する場合

ポインタを使用する場合

ポインタは、自分自身のデータタイプを返すことはできますが、指している変数のデータタイプを返すことができません。次のような場合、

```
Var1:=5.2`(1)
Pointer:=->Var1`(2)
Var2:=Pointer-> `(3)
```

(2)で"Pointer"が指す変数のタイプは定義されていますが、"Var2"のタイプは定義されていません。4D Compilerはコンパイルの過程でポインタを認識できますが、それがどのタイプの変数を指しているかを知る手段がありません。そのため、"Var2"のデータタイプを判定することができません。この場合、例えば次のようなコンパイラコマンドが必要になります。

```
C_REAL(Var2)
```

複数のシンタックスコマンドを使用する場合

Year of関数の場合、考えられるデータタイプは日付だけですが、もっと複雑な場合もあります。次の例をご覧ください。

GET FIELD PROPERTIESコマンドは2つのシンタックスを受け入れます。

```
GET FIELD PROPERTIES(tableNo;fieldNo;type;length;index)
GET FIELD PROPERTIES(fieldPointer;type;length;index)
```

複数のシンタックスを持つコマンドを使用する場合、コンパイラは、どのシンタックスと引数が使用されているのかを判定することができません。コマンドの引数がシンタックスに対応するタイプに設定されていない場合は、データベース内の任意の場所で、コンパイラコマンドを使用してタイプを設定する必要があります。

オプション引数付きのコマンドの場合

様々なデータタイプの何種類かのオプション引数付きコマンドを使用する場合、4DCompilerはどのオプション引数が使用されたのかを決定することができません。以下の例をご覧ください。

GET LIST ITEMコマンドには倍長整数とブールの2つのオプション引数があります。このため、コマンドは以下のいずれかのように使用されます。

```
GET LIST ITEM(List;itemPos;itemRef;itemText;sublist;expanded)
```

または、

```
GET LIST ITEM(List;itemPos;itemRef;itemText;expanded)
```

データベースの中で変数が定義されておらず、またその使用方法においてもタイプが明確ではない場合には、コマンドに渡された変数のタイプを決定するのにコンパイラコマンドを使用しなければなりません。

URL経由で呼び出されたメソッドの場合

URL経由で呼び出される4Dメソッドを書く場合、メソッド中で$1を使用しなくても、テキスト変数$1を定義しなければなりません。

```
C_TEXT($1)
```

4D Compilerは、メソッドがURL経由で呼び出されることを知ることができません。

コンパイルの時間の短縮

データベースで使用する変数がすべて明示的に定義されていれば、4D Compilerでタイプ定義を調べる必要はありません。この場合、オプションを設定してメソッドの翻訳フェーズだけを行うように指定できます。この操作により、コンパイル時間を半分以下に短縮することができます。

コードの最適化

コンパイラコマンドを使用することで、メソッドの処理速度を上げることができます。簡単な例として、カウンタにローカル変数を使用した場合、変数のタイプを設定しないと、4D Compilerはその変数を実数と見なします。倍長整数に設定すると、コンパイルしたデータベースの効率が良くなります。なぜなら、例えばWindowsの場合、実数データがメモリを8バイト使用するのに対し、倍長整数にすると、4バイトしか使用しないからです。8バイトのカウンタをインクリメントする場合は当然、4バイトの場合より時間がかかります。この件についての詳細は「最適化のためのヒント」の節を参照してください。

コンパイラコマンドをどこに記述するか

コンパイラコマンドは、コンパイラに変数のタイプ付けをまかせるかどうかによって以下2通りに処理方法が異なります。

コンパイラでタイプ定義される変数

コンパイルで変数タイプを調べたり、またタイプ定義を行ったりする場合は簡単にコンパイラコマンドの記述場所を決めることができます。作業方法に応じて以下のいずれかの方法を選んでください。

- ローカル変数、プロセス変数、インタープロセス変数に応じて、変数が初めて使用されるメソッドやオブジェクトメソッドでコンパイラコマンドを使用します。コンパイラコマンドは、必ず変数が初めて使われる場所、つまり一番初めに実行されるはずのメソッドで使用するようにしてください。コンパイル時、4D Compilerは4th Dimensionで作成した順序でメソッドを処理します。エクスプローラで表示される順番ではない点に注意してください。
- コンパイラコマンドで定義するプロセス変数とインタープロセス変数は、**On Startup**データベースメソッドまたは**On Startup**データベースメソッドから呼び出されるメソッドにまとめてください。ローカル変数については、使用するメソッドの一番初めにコンパイラコマンドをまとめてください。

開発者がタイプ定義する変数

4D Compilerにタイプ定義をチェックさせたくない場合、コンパイラコマンドを識別できるようなコードを4D Compilerに与える必要があります。従来は以下の方法で行います。

プロセス変数、インタープロセス変数についてのコンパイラコマンドおよび引数を、名前の先頭が"COMPILER"で始まる1個あるいは複数個のメソッドに代入します。デフォルトにより、5種類のコンパイラメソッドが自動生成され、変数、配列、メソッド引数ごとに命令をまとめます（詳細は、『4th Dimensionデザインリファレンス』マニュアルを参照してください)。

注：引数定義のためのシンタックスは次のとおりです。

Directive(methodName;parameter)

なお、このシンタックスはインタプリタモードでは実行できません。

特定の引数

■ データベースメソッドによって受け取られた引数
これらの引数が明確に定義されていなければ、4D Compilerがタイプを決定します。定義する場合は、そのデータベースメソッド内で行わなければなりません。この引数定義は、COMPILERメソッド内には書き込みができません。

例：**On Web Connection**が6つのテキスト引数"$1"から"$6"までを受け取った場合、データベースメソッドの最初に、**C_TEXT**($1;$2;$3;$4;$5$6)を記述する必要があります。

■ トリガ
トリガの結果である引数"$0"（倍長整数）は、引数が明確に定義されていなければ、4D Compilerがタイプを決定します。定義する場合は、トリガ自身の中で行う必要があります。この引数定義は、COMPILERメソッド内には書き込みができません

■ "On Drag OVer"フォームイベントを受け入れるオブジェクト
"On Drag OVer"フォームイベントの結果である引数＜$0＞（倍長整数）は、引数が明確に定義されていなければ、4D Compilerがタイプを決定します。定義する場合は、オブジェクトメソッドの中で行う必要があります。

この引数定義は、COMPILERメソッド内には書き込みができません。

注： 4D Compilerは引数"$0"を初期化しません。したがって、"On Drag Over"フォームイベントを使用したら、直ちに"$0"を初期化しなければなりません。

例えば、

```
C_LONGINT($0)
If(Form event=On Drag Over)
    $0:=0
    ...
    If($DataType=Is Picture)
        $0:=-1
    End if
...
End if
```

C_STRINGコンパイラコマンド

C_STRINGでは、最長の文字列を表す引数も受け入れるため、他の命令とは異なるシンタックスが使われます。

```
C_STRING(length;var1{;var2;...;varN})
```

C_STRINGは、固定長の文字列を扱うので、文字列の最大の長さを指定する必要があります。コンパイルするデータベースでは、変数ではなく定数を使用して文字列の長さを指定しなければなりません。

例えば、インタプリタでは、以下のシーケンスを使用することができます。

```
TheLength:=15
C_STRING(TheLength;TheString)
```

4th Dimensionは"TheLength"を解析し、**C_STRING**コマンドで、"TheLength"をその値に置き換えます。しかし、このコマンドでは変数タイプの設定に特定の代入ステートメントが考慮されていないため、"TheLength"の値が15であることはわかりません。文字列の長さがわからなくては、シンボルテーブルにその文字列用のスペースを確保できませんから、文字列の長さを定義する際はコンパイルを念頭において定数を使用する必要があります。

例えば、ステートメントをこのように使用します。

```
C_STRING(15;TheString)
```

次のようなコマンドで定義する場合の固定長の文字配列についても同じです。

```
ARRAY STRING(length;arrayName;size)
```

配列の文字列の長さを示す引数は、定数でなければなりません。

ただし、文字列の長さは、コンパイラコマンドで4D定数または16進法定数に指定することができます。

例えば、

```
C_STRING(4DConstant;TheString)
ARRAY STRING(4DConstant;TheArray;2)
C_STRING(0x000A;TheString)
ARRAY STRING(0x000A;TheArray;2)
```

文字フィールドの長さ（最大80文字まで）と、固定長文字変数を混同しないでください。**C_STRING**コマンド、または**ARRAY STRING**コマンドで定義できる文字列のサイズは1から255までです。

注：このコマンドのシンタックスでは、1行で同じ長さの変数を複数定義することができます。異なる長さの文字列を複数定義する場合は、別の行で行ってください。

タイプの矛盾として見なされない場合

コンパイラコマンドによって、曖昧なデータタイプや文字の長さは除外されます。特定の厳密さは要求されますが、4D Compilerが全ての不一致を受け入れないというわけではありません。例えば、整数と定義した変数に実数値を代入する、あるいは10文字の文字列として定義した変数に30文字の文字列を代入すると、4D Compilerはこれをタイプの不一致と見なさず、コンパイラコマンドに応じた値を代入します。

例えば以下のように記述した場合、

```
C_INTEGER(vInteger)
vInteger:=2.5
```

タイプの矛盾として扱われるとコンパイルされませんが、4D Compilerは上記の例をタイプの不一致として見なさず、代わりに数値の整数部分を切り上げます（2.5ではなくて3になります）。

次は、文字列処理の例です。以下のように記述した場合、

```
C_STRING(10;MyString)
MyString:="It is a beautiful day"
```

4D Compilerは文字列定数の最初の10文字、"It is a be"だけを代入します。

参照

Optimization Hints、Syntax Details、Typing Guide

タイプ設定ガイド

この節では変数タイプに生じる矛盾の主な原因と問題を解決するための方法について説明します。

単純変数におけるタイプの矛盾

単純なデータタイプの矛盾には以下のパターンがあります。

■ 2種類の用途による矛盾

■ 用途とコンパイラコマンドの矛盾

■ 暗黙のタイプ変更による矛盾

■ 2つのコンパイラコマンドによる矛盾

■ 2種類の用途による矛盾

最も単純なタイプの矛盾は、1つの変数名が2つの異なるオブジェクトを示す場合です。

例えば、以下のように書き、

```
Variable:=5
```

また、同じアプリケーション内で次のように書いたとします。

```
Variable:=True
```

この2つのステートメントはタイプの矛盾を引き起こします。ほとんどの場合、どちらか一方の変数名を変えることで問題は解決できます。

用途とコンパイラコマンドの矛盾

例えば、以下のように書き、

```
Variable:=5
```

また、同じアプリケーション内で、次のように書いたとします。

```
C_BOOLEAN(Variable)
```

コンパイラコマンドが最初に処理されるので、"Variable"はブールタイプに設定されますが、"Variable:=5"というステートメントがあるため、これをタイプの矛盾と見なします。変数名かコンパイラコマンドのどちらかを変更することで、問題は解決します。

1つの式に異なるタイプの変数を使うと、それをコンパイラはタイプの不一致として指摘します。簡単な例を紹介します。

```
Bool:=True    ` Boolのタイプはブールです。
C_INTEGER(<>Integer)      `コンパイラコマンドによる倍長整数の定義
<>vInteger:=3` 代入値はコンパイラの指定と同じタイプ
Var:= <>Integer+Bool          `データタイプが一致しない変数を使用した演算
```

暗黙のタイプ変更による矛盾

関数の中には戻り値のデータタイプが明確に決まっているものがあります。このような関数の戻り値を不注意に異なったタイプの変数に代入すると、データタイプの矛盾が起ります。

例えば、インタプリタで以下のように書いた場合、

```
IdenNo:=Request("Identification Number")`IdenNoのデータタイプはテキストです。
If(OK=1)
      IdenNo:=Num(IdenNo)                `IdenNoのデータタイプは実数です。
QUERY([Contacts]Id=IdenNo)
End if
```

この例では、3行目でタイプの矛盾が生じます。これを解決するためには変数の動作を制御する方法が考えられます。異なる名前の中間変数を作成する必要がある場合や、次の例のようにメソッドの構造を変更することで修正できる場合もあります。

```
IdenNo:=Num(Request("Identification Number")) `IdenNoのデータタイプは実数です。
If(OK=1)
      QUERY([Contacts]ID=IdentNo)
End if
```

2つのコンパイラコマンドによる矛盾

同じ変数に対して2つの異なるコンパイラコマンドを使用すると、タイプの再定義になります。例えば、1つのデータベースで次のように書いたとします。

```
C_BOOLEAN(Variable)
C_TEXT(Variable)
```

この場合、コンパイラによって矛盾が検出され、エラーファイルに出力されます。通常、どちらか一方の変数名を変えることで問題は解決します。

C_STRINGコマンドを使用する場合、最大の文字長を変更することでデータタイプの矛盾が生じることがあるので、注意してください。例えば、次のように書くと、コンパイラはタイプの矛盾と見なします。

```
C_STRING(5;MyString)
```

```
MyString:="Hello"
C_STRING(7;MyString)
MyString:="Flowers"
```

文字列タイプの変数を定義する際には、十分なサイズのエリアが必要ですが、デフォルトによりコンパイラは短い方を採用してしまいます。解決するには、文字列に最大値を与えるためのコンパイラコマンドを使用することができます。以下のように書くことができます。

```
C_STRING(7; String)
String:="Flowers"
String:="Hello"
```

注：以下のようにC_STRING(7; String)と2回書いても命令は受け入れられますが、2回書く必要はありません。

```
C_STRING(7; String)
String:="Flowers"
C_STRING(7; String)
String:="Hello"
```

ローカル変数についての注意

ローカル変数におけるデータタイプの矛盾は、プロセス変数やインタープロセス変数とほとんど同じものです。唯一の違いは、指定されたメソッドの中のみタイプが一貫していればよいという点です。

プロセス変数やインタープロセス変数では、データベースの全体のレベルでタイプの矛盾が起きましたが、ローカル変数の場合は、メソッドレベルで矛盾が起きます。

例えば、同じメソッドの中で、

```
$Temp:="Flowers"
```

と書き、次に、

```
$Temp:=5
```

と書くことはできませんが、メソッドM1というメソッドには、

```
$Temp:="Flowers"
```

と書き、メソッドM2という別のメソッドの中に、

```
$Temp:=5
```

と書くことはできます。これは、ローカル変数の範囲がメソッドそのもので、データベース全体ではないためです。

配列の矛盾

タイプの矛盾に配列の要素数は無関係です。コンパイル前のデータベースと同様、配列も動的に管理されます。配列の要素数はメソッドを通して変更可能で、配列用に最大値を定義する必要はありません。このため、要素数を0にすることも、追加や消去、もしくは内容の削除を行うことも可能です。

コンパイルを前提にデータベースを作る場合は、以下の原則に従ってください。

■ 配列要素のデータタイプを変えないこと

■ 配列の次元数を変えないこと

■ 文字列では、文字の長さを変えないこと

配列要素のデータタイプの変更

整数として定義された配列は、データベース全体を通して整数配列でなければなりません。データベース内で、ブールタイプなどの要素を使用することはできません。

例えば、

```
ARRAY INTEGER(MyArray;5)
ARRAY BOOLEAN(MyArray;5)
```

と書いた場合、"MyArray"のタイプを決定できません。どちらか一方の配列名を変えてください。

配列の次元数の変更

コンパイル前のデータベースでは、配列の次元数を変更することができます。コンパイラでシンボルテーブルを作成する際、1次元配列と2次元配列は別々に管理されます。このため、1次元配列を2次元配列として、もしくは2次元配列を1次元配列として再定義することはできません。

したがって、同じデータベースの中で以下のように定義することはできません。

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray1;10;10)
```

ただし、同じアプリケーションの中に次のステートメントを書くことはできます。

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray2;10;10)
```

データベース内で配列の次元数を変更することはできませんが、配列のサイズを変更できます。2次元配列の1つの配列のサイズを変更するには次のように書きます。

ARRAY BOOLEAN(MyArray;5)
ARRAY BOOLEAN(MyArray;10)

注：2次元配列は実際には複数の1次元配列から構成されています。詳細は、「2次元配列」の節を参照してください。

固定長文字列の場合

文字列には固定長文字列と同じ規則が適用されます。

次のように書いた場合、

ARRAY STRING(5;MyArray;10)
ARRAY STRING(10;MyArray;10)

コンパイラにより長さの不一致が検出されます。これは最大の文字列の長さを定義することで簡単に解決できます。短い方の文字列は、コンパイラが自動的に処理します。

暗黙のタイプ変更

COPY ARRAYコマンド、**LIST TO ARRAY**コマンド、**ARRAY TO LIST**コマンド、**SELECTION TO ARRAY**コマンド、**SELECTION RANGE TO ARRAY**コマンド、**ARRAY TO SELECTION**コマンド、または**DISTINCT VALUES**コマンドの使用により、データタイプの要素、次元数、もしくは文字列や文字の長さが（意図的にもしくは無意識的に）変更されてしまう場合があります。その場合、前に述べた3つの状況のいずれかに該当します。

コンパイラはエラーメッセージを出力します。通常、修正が必要な点は非常に明確です。

暗黙の配列のタイプ変更の例は、「シンタックスの詳細」の節を参照してください。

ローカル配列

ローカル配列（これを生成したメソッドによってのみ見ることができる配列）が含まれるデータベースをコンパイルする場合、使用前に4th Dimensionで明示的にこの配列を定義する必要があります。明示的に配列を定義するとは、**ARRAY REAL**コマンドや**ARRAY INTEGER**コマンドなどを使い、配列を定義することをいいます。メソッドにより10要素のローカル整数配列を生成する場合、メソッドの中に次の行が必要です。

ARRAY INTEGER($MyArray;10)

フォーム内で作成された変数のタイプ

フォーム内で作成された変数タイプ（例えば、ボタン、ドロップダウンリストボックスなど）は、すべてプロセスもしくはインタープロセス変数です。

■ インタプリタのデータベースでは、フォーム変数のデータタイプは重要ではありませんが、コンパイルされたアプリケーションの場合は重要です。ただし、規則は非常に明確です。

■ コンパイラコマンドでフォーム変数のタイプを定義します、もしくは4D Compilerにより環境設定のコンパイルでデフォルトタイプを割り当てるように設定することができます。

デフォルトで実数とされる変数

次のフォーム変数は、デフォルトとして実数タイプとして設定されます。

■ チェックボックス

■ 3Dチェックボックス

■ ボタン

■ ハイライトボタン

■ 透明ボタン

■ 3Dボタン

■ ピクチャボタン

■ ボタングリッド

■ ラジオボタン

■ 3Dラジオボタン

■ ラジオピクチャ

■ ピクチャメニュー

■ 階層式ポップアップメニュー

■ 階層式リスト

■ ルーラ

■ ダイアル

■ サーモメータ

注：ルーラ、ダイアルおよびサーモメータのフォーム変数については、例えば環境設定のボタンタイプに倍長整数を選んだとしても、常に実数タイプに設定されます。

上記いずれかの変数に対して、データタイプの矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

グラフ変数

グラフエリアは自動的にグラフ（倍長整数）のデータタイプに設定されるので、タイプの不一致が生じることはありません。もしデータタイプの矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

プラグインエリア変数

プラグインエリアは常に倍長整数タイプに設定されるので、タイプの不一致が生じることはありません。もしデータタイプの矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

デフォルトでテキストとされる変数

テキストタイプに設定されるフォーム変数は以下のとおりです。

- 入力不可変数
- 入力可変数
- ドロップダウンリスト
- メニュー、ドロップダウンリスト
- スクロールエリア
- ポップアップメニュー
- コンボボックス
- タブコントロール

これらの変数は2つのカテゴリーに分類できます。

- **単純変数：**入力可、入力不可の変数
- **表示変数：**ドロップダウンリスト、メニュー/ドロップダウンリスト、スクロールエリア、ポップアップメニュー、コンボボックスおよびタブコントロール

■ 単純変数
デフォルトのデータタイプはテキストです。メソッドやオブジェクトメソッドで使用される場合は、ユーザが選択したデータタイプが割り当てられます。同じ名前で異なるタイプの変数が存在する場合以外は、データタイプの矛盾が起こることはありません。

■ 表示変数
変数はフォーム内で配列を表示するために使用されます。フォームエディタにデフォルト値が設定されている場合、配列を定義するための**ARRAY STRING**コマンドや**ARRAY TEXT**コマンドを使い、対応する変数を明確に定義する必要があります。

ポインタ

ポインタを使うと、4th Dimensionツールの強力で多彩な機能を活用することができます。コンパイル後もポインタの利点をそのまま活用できます。1つのポインタでデータタイプの異なる変数を指定することができます。1つの変数に異なるデータタイプを代入して矛盾を引き起こさないようにしてください。また、ポインタで参照する変数のデータタイプを変更しないよう注意してください。

この問題の例を示します。

```
Variable:=5.3
Pointer:=->Variable
Pointer->:=6.4
Pointer->:=False
```

この例では、ポインタで参照する変数のタイプは実数です。これにブール値を代入しているので、データタイプの矛盾が起こります。1つのメソッド内で、異なった目的のためにポインタを使う場合には、参照先の変数タイプが定義されていることを確認してください。

ポインタの正しい使い方の例を示します。

```
Variable:=5.3
Pointer:=->Variable
Pointer->:=6.4
Bool:=True
Pointer:=->Bool
Pointer->:=False
```

ポインタは、常に参照するオブジェクトとの関連で定義されています。このため、ポインタによるデータタイプの矛盾をコンパイラが検知することはできません。矛盾があっても、変数設定フェーズやコンパイルフェーズでは、エラーメッセージが出力されません。ただし、ポインタに関係する矛盾を見つける方法がまったくないわけではありません。コンパイラの環境設定の"範囲チェック"オプションでポインタの使用状況を確認することができます（詳細は、『4th Dimensionデザインリファレンス』マニュアルを参照してください）。

プラグインコマンド

概要

コンパイルの際、コンパイラはデータベースで使用されるプラグインコマンドのパラメータのタイプや数値などについての定義を解析します。呼び出しがメソッドと一致していれば、その階層でタイプの矛盾が生じる危険はありません。

Windowsのプラグインを「WIN4DX」フォルダ（ディレクトリ）に、Macintoshのプラグインを「MAC4DX」フォルダに入れます。これらフォルダは4th Dimensionに許可されている次のいずれかの場所に置きます（詳細は、『4th Dimensionインストールガイド』を参照してください）。

■ ストラクチャファイルと同階層のフォルダ内

■ Windowsでは実行可能なアプリケーションの隣、またはMacintoshではソフトウェアパッケージと同階層のフォルダ内

■ コンピュータのアクティブな4Dのシステムフォルダの中

コンパイラは、これらのファイルを複製しませんが、これらを分析して、これらのルーチンの適切な定義を決定します。

プラグインが同じ階層にない場合、4D Compilerは「ファイルを開く」ダイアログボックスを表示してプラグインの場所を確認します。

マルチプラットフォームコンパイル

Windows版4D Compilerでのプラットフォーム独立型コンパイル

4D Server（Windows）に4D Client（Macintosh）から接続する場合、プラグインはWindows版を使用します。

4D Client（Macintosh）から4D Server（Windows）にある4Dプラグインを使用するには以下のように行います。

1. Macintosh版プラグインをMacintoshマシンにインストールする。

2. 4D Transporterを使用してMacintosh版プラグインをWindows用にトランスポートする。

3. Windows上に「Mac4DX」という名前の新しいフォルダを作成する。

4. トランスポートされた2つのファイル（「Plug-inName.4CX」ファイルもしくは「Plug-inName.4DX」ファイルおよび「Plug-inName.RSR」ファイル）をMacintosh上からWindows上の「Mac4DX」フォルダにコピーする。

5. 「Mac4DX」フォルダをデータベースの「Win4DX」フォルダと同じ階層に置く。

Macintosh版4D Compilerでのプラットフォーム独立型コンパイル

4D Server（Macintosh）に4D Client（Windows）から接続する場合、プラグインはWindows版を使用します。

4D Client（Windows）から4D Server（Macintosh）にある4Dプラグインを使用するには以下のように行います。

1. Windows版プラグイン（「Plug-inName.4DX」ファイルおよび「Plug-inName.RSR」ファイルを含む）をMacintoshマシンにインストールする。

2. Macintosh上に「Win4DX」という名前の新しいフォルダを作成する。

3. 「Win4DX」フォルダにWindowsの「Plug-inName.4DX」ファイルおよび「Plug-inName.RSR」ファイルをコピーする。

4. 「Win4DX」フォルダをデータベースの「Mac4DX」フォルダと同じ階層に置く。

暗黙の引数を受け取るプラグインコマンド

特定のプラグイン（例えば4D Write）は、暗黙に4th Dimensionコマンドを呼び出すコマンドを実行します。例えば4D Writeの場合、**WR ON EVENT**コマンドのシンタックスは次のようになります。

WR ON EVENT(area;event;eventMethod)

最後の引数は、4th Dimensionで作成したメソッドの名前です。このメソッドはイベントが受け取られるたびに4D Writeによって呼び出され、自動的に次の引数を受け取ります。

引数	タイプ	説明
$0	倍長整数	戻り値
$1	倍長整数	4D Writeエリア

$2	倍長整数	Shiftキー
$3	倍長整数	Altキー(Windows); Optionキー(MacOS)
$4	倍長整数	Ctrlキー(Windows), Commandキー (MacOS)
$5	倍長整数	イベントのタイプ
$6	倍長整数	引数<イベント>の値

コンパイラがこれらの引数の存在を認識し、これらを考慮するためには、引数をコンパイラコマンド、またはメソッド内での使用法によってタイプ設定しなければなりません。メソッドで使用する場合には、タイプを明確に推測できるように使用する必要があります。

4Dコンポーネント

4th Dimensionと4D Insiderを使い、4Dコンポーネントの作成および管理が可能です。4Dコンポーネントは4Dオブジェクトのライブラリのようなもので、それぞれのオブジェクトには可視性や更新の可否を示す属性（プライベート、プロテクト、パブリック）が割り当てられます。4Dコンポーネントの管理に関する詳細については4D Insiderのドキュメントを参照してください。原則として、4Dコンポーネントの開発者はコンポーネントがコンフリクトを起こさずにコンパイルできるかを確かめるべきです。それでも問題が生じる場合はあります。コンポーネントに属するオブジェクトが原因でコンパイルエラーが発生した場合、4D Compilerはオブジェクトの属性に従って、以下のような情報を表示します。

■ **プライベート：**4D Compilerは4Dコンポーネントの名前を表示しますが、関連するオブジェクトの名前は表示しません。

■ **プロテクトおよびパブリック：**4D Compilerは他のデータベースオブジェクトと同様に、オブジェクト名を表示します（標準の動作）。

ローカル変数$0...$Nの処理および引数の受け渡し

ローカル変数の処理はこれまでに述べた規則に従います。他の変数と同様、メソッドの実行中にデータタイプを変更することはできません。ここでは、タイプの矛盾を起こす可能性のある2つのケースについて検証します。

■ タイプ変更が必要な場合。ポインタの使用によりデータタイプの矛盾を避けることができます。

■ 引数の間接参照が必要な場合

ポインタの使用でタイプの矛盾を避ける

変数タイプを変更することはできませんが、ポインタを使用して異なるデータタイプを参照することはできます。例えば、1次元配列のメモリサイズを返す関数を考えてみてください。この場合、メモリサイズを数字で表せないテキスト配列とピクチャ配列を除き、実数の結果のみを返します（「配列とメモリ」の節を参照してください）。

また、テキスト配列とピクチャ配列については、文字列の結果を返します。この関数の引数は、メモリサイズを調べようとする配列へのポインタです。この操作を行うには2つの方法があります。

■ ローカル変数のデータタイプを気にせずに使用する方法
このメソッドはインタプリタでしか動作しません。

■ ポインタを使用してインタプリタもしくはコンパイルモードで処理する方法

■ MemSize機能（インタプリタモードのみ）*Macintoshを使用する場合の例

```
$Size:=Size of array($1->)
$Type:=Type($1->)
Case of
    ¥($Type=Real array)
        $0:=8+($Size*10) ` $0は実数
    ¥($Type=Integer array)
        $0:=8+($Size*2)
    ¥($Type=LongInt array)
        $0:=8+($Size*4)
    ¥($Type=Date array)
        $0:=8+($Size*6)
    ¥($Type=Text array)
        $0:=String(8+($Size*4))+("+Sum of text lengths") ` $0はテキスト
    ¥($Type=Picture array)
        $0:=String(8+($Size*4))+("+Sum of picture sizes") ` $0はテキスト
    ¥($Type=Pointer array)
        $0:=8+($Size*16)
    ¥($Type=Boolean array)
        $0:=8+($Size/8)
End case
```

このメソッドでは、"$0"のデータタイプが"$1"の配列によって異なるため、コンパイルできません。

■ MemSize機能（インタプリタモード、コンパイルモード）*Macintoshを使用する場合の例

次にポインタを使用する方法を示します。

```
$Size:=Size of array($1->)
$Type:=Type($1->)
VarNum:=0
Case of
        ¥($Type=Real array)
                VarNum:=8+($Size*10) ` VarNum is a Real
        ¥($Type=Integer array)
                VarNum:=8+($Size*2)
        ¥($Type=LongInt array)
                VarNum:=8+($Size*4)
        ¥($Type=Date array)
                VarNum:=8+($Size*6)
        ¥($Type=Text array)
                VarText:=String(8+($Size*4))+("+Sum of text lengths")
        ¥($Type=Picture array)
                VarText:=String(8+($Size*4))+("+Sum of picture sizes")
        ¥($Type=Pointer array)
                VarNum:=8+($Size*16)
        ¥($Type=Boolean array)
                VarNum:=8+($Size/8)
End case
If(VarNum#0)
        $0:=->VarNum
Else
        $0:=->VarText
End if
```

2つの関数は、以下の点が違っています。

■ 1番目の関数では、結果が変数であること

■ 2番目の関数では、結果が変数へのポインタであること。結果はポインタで簡単に参照できます。

引数の参照

4D Compilerは、引数の間接参照の機能をサポートしています。インタプリタでは、引数のデータタイプと数値を自由に設定できます。データタイプの矛盾を起こさない限り(呼ばれた側でセットされていない引数を使用しないこと)、コンパイルモードでも同様に自由に設定することができます。タイプの矛盾の元になるので、間接参照する引数は、すべて同じデータタイプにしてください。間接参照を上手に使うために、間接参照する引数は他の引数を渡した後に配置するようにします。

メソッド内で、間接参照は"${$i}"のように表示します。"$i"は数値変数です。"${$i}"を"間接参照引数（generic parameter）"と呼びます。

以下は間接参照の例です。数値を合計し、引数として渡されたフォーマットに編集して返すような関数を考えてください。合計される元の数値は、メソッドが呼ばれるたびに変わります。このメソッドでは数値と編集フォーマットを引数としてメソッドに渡さなければなりません。

この関数は、以下のようにして呼び出します。

```
Result:=MySum("##0.00";125.2;33.5;24)
```

この場合、数値を合計し、指定されたフォーマットに編集して"182.70"が返されます。関数の引数は正しい順序で渡してください。最初にフォーマット、次に値です。

以下はMySum関数です。

```
$Sum:=0
For($i;2;Count parameters)
    $Sum:=$Sum+${$i}
End for
$0:=String($Sum;$1)
```

この関数は次の方法で呼び出すことができます。

```
Result:=MySum("##0.00";125,2;33,5;24)
Result:=MySum("000";1;18;4;23;17)
```

他のローカル変数と同様、間接参照引数はコンパイラコマンドで定義する必要がありません。ただし、曖昧になりそうな場合や最適化のために必要な場合は以下のシンタックスを使用することができます。

```
C_INTEGER(${4})
```

このコマンドは、4番目以降に間接参照されたすべての引数のデータタイプが整数であることを意味します。$1、$2、$3には、いかなるデータタイプも使用できますが、間接参照で$2を使用した場合のデータタイプは間接参照引数になります。このため、実数であってもデータタイプは整数と見なされます。

注：コンパイラはこのコマンドを変数設定フェーズで使います。定義に使用する数値は変数ではなく、定数でなくてはなりません。

予約変数と定数

4th Dimensionの変数および定数には、コンパイラによってデータタイプおよび識別子が割り当てられているものがあります。このため、ユーザはこれらの変数名や定数を、新しい変数やメソッド、関数、プラグインコマンドに使用することはできません。インタプリタモードで行うのと同じ方法で、これらの値を検証し使用することができます。

システム変数

4th Dimensionのシステム変数および対応するデータタイプの完全なリストを紹介します。

変数	タイプ
OK	倍長整数
Document	文字列（255）
FldDelimit	倍長整数
RecDelimit	倍長整数
Error	倍長整数
MouseDown	倍長整数
KeyCode	倍長整数
Modifiers	倍長整数
MouseX	倍長整数
MouseY	倍長整数
MouseProc	倍長整数

クイックレポート変数

クイックレポートで計算用のカラムを作成する際、最初のカラムに変数C1を２番目にはC2を、そして3番目のカラムにはC3をというように4th Dimensionが自動的に変数を作成します。この処理はユーザには見えません。

これらの変数をメソッド内で使用する場合、他の変数と同様C1、C2...Cnはタイプ変更することができませんので、留意してください。

定数

4th Dimensionで定義されている定数については本マニュアルに完全なリストの記載があります。4Dの定数は、デザインモードのエクスプローラでも表示されます。

参照

Optimization Hints、Syntax Details、Using Compiler Directives.

シンタックスの詳細

4D Compilerは、4th Dimensionの通常のコマンドシンタックスにもとづいてコマンド処理を行います。この点に関して、コンパイルのためにデータベースを特に変更する必要はありません。この節では、こうしたコマンドについての注意点や詳細について項目別に説明します。

- 変数のデータタイプを決定付けるようなコマンドは、データタイプ矛盾の原因になることがあります。
- コマンドの中には複数のシンタックスを持つものがあり、どのシンタックスが最適なのか知っておくと役に立ちます。

文字列

Ascii(character)

インタプリタでは、Ascii関数に渡す文字列が空でもデータが入っていても構いませんが、コンパイル後は空の文字列を渡すことができません。**Ascii**関数の引数が変数の場合に空の文字列を渡していても、コンパイル中にエラーを見つけることはできません。

コミュニケーション

SEND VARIABLE（変数）
RECEIVE VARIABLE（変数）

上記2つのコマンドは変数をディスクに保存または読み込む場合に使用されます。引数は変数です。コマンドから受け取る変数のタイプは、常に渡した時と同じタイプでなければなりません。変数のリストをファイルに送る場合には、誤ってデータタイプを変えてしまう恐れがあるので、リストの先頭に送る変数のデータタイプを指定することをおすすめします。そうすると、変数を受け取る際には、常にタイプが返されることになります。**RECEIVE VARIABLE**コマンドを呼び出したら、**Case of**文を使用して、次から受け取るデータを処理できます。

例題

```
SET CHANNEL(12;"File")
    If(OK=1)
        $Type:=Type([Client]Total_TO)
        SEND VARIABLE($Type)
        For($i;1;Records in selection)
            $Send_TO:=[Client]Total_TO
            SEND VARIABLE($Send_TO)
        End for
    End if
    SET CHANNEL(11)
    SET CHANNEL(13;"MyFile")
    If(OK=1)
        RECEIVE VARIABLE($Type)
        Case of
            ¥($Type=Is String Var)
                RECEIVE VARIABLE($String)
                    `プロセス中の変数を受領
            ¥($Type=Is Real)
                RECEIVE VARIABLE($Real)
                    `プロセス中の変数を受領
            ¥($Type=Is Text)
                RECEIVE VARIABLE($Text)
                    `プロセス中の変数を受領
        End case
    End if
SET CHANNEL(11)
```

ストラクチャへのアクセス

Field（フィールドポインタ）またはField（テーブル番号；フィールド番号）

Table（テーブルポインタ）または（テーブル番号）または（フィールドポインタ）

2つのコマンドは、与えられた引数によって、戻り値のデータタイプが違います。

■ ポインタを与えると、**Table**関数は数値を返します。

■ 数値を与えると、**Table**関数はポインタを返します。

コンパイラでは、これらの関数から結果のデータタイプを決定できません。このような場合は、コンパイラコマンドを使用して明確に定義してください。

ドキュメント

Open document、**Append document**、**Create Document**の関数によって返される参照ドキュメントのデータタイプは時間タイプです。

演算

Mod（value;divider）

4th Dimensionでは25を3で割った余りを求める場合、次の2通りの方法があります。

```
Variable:=Mod(25;3)
```

あるいは

```
Variable:=25%3
```

4D Compilerはこの2つの式を区別します。**Mod**関数はすべての数値に使用できますが、%演算子は整数と倍長整数にしか使用できません。%演算子の演算数が、倍長整数データタイプの範囲を越えた場合には、返される結果が誤りであることが多くなります。

例外処理

■ **IDLE**

■ **ON EVENT CALL**(Method{; ProcessName})

■ **ABORT**

ON EVENT CALL

IDLEコマンドは例外処理を行うために4th Dimensionに追加されました。**ON EVENT CALL**コマンドを用いる場合は、必ずこの**IDLE**コマンドも使用してください。このコマンドはイベント管理命令として定義することができます。

システムイベント（マウスクリックやキー操作など）は、4th Dimensionのカーネルにしか検知できません。ほとんどの場合、カーネルコールはコンパイル後のコードそのものによって起動され、ユーザに対しては透過的です。

例えば、4th Dimensionがイベント待ちループなどの中でイベントの受信を待っている場合はカーネルコールがありません。

Windowsの場合の例

```
      `MouseClickメソッド
If(MouseDown=1)
```

```
        <>vTest:=True
        ALERT("マウスがクリックされました")
    End if
        `Waitメソッド
    <>vTest:=False
    ON EVENT CALL("MouseClick")
    While(<>vTest=False)
        `イベント待ちループ
    End while
    ON EVENT CALL("")
```

この場合、IDLEコマンドを次のように追加します。

```
        `Waitメソッド
    <>vTest:=False
    ON EVENT CALL("MouseClick")
    While(<>vTest=False)
        IDLE
        `イベントを検知するカーネルコール
    End while
    ON EVENT CALL("")
```

ABORT

このコマンドは、エラー処理プロジェクトメソッド内でのみ使用してください。これは4th Dimensionで使用した場合とまったく同じように動作しますが、**EXECUTE**、**APPLY TO SELECTION**、**APPLY TO SUBSELECTION**コマンドから呼び出されたメソッド内の場合は例外です。このような状況は避けた方がよいでしょう。

配列

4D Compilerが配列のデータタイプを決める際に使用する4th Dimensionコマンドは、以下の7種類です。

- **COPY ARRAY**(source;destination)
- **SELECTION TO ARRAY**(field;array)
- **ARRAY TO SELECTION**(array;field)
- **SELECTION RANGE TO ARRAY**(start;end;field;array)
- **LIST TO ARRAY**(list;array{; itemRefs})
- **ARRAY TO LIST**(array;list{; itemRefs})
- **DISTINCT VALUES**(field;array)

COPY ARRAY

COPY ARRAYコマンドは2個の配列タイプの引数を使用します。引数の一方がどこにも定義されていないと、4D Compilerは定義されている方のデータタイプから未定義の配列のデータタイプを決定します。

この場合、以下のように処理されます。

■ 最初の引数が定義されている場合：2番目の配列には、最初の配列のデータタイプが適用されます。

■ 2番目の引数が定義されている場合：最初の配列に2番目の配列のデータタイプが適用されます。

4D Compilerはデータタイプを厳密にチェックするので、**COPY ARRAY**コマンドは同じタイプの配列間で行わなければなりません。このため、整数と倍長整数と実数、あるいは、テキスト配列と文字配列で文字列の長さが一定でない場合等のように、タイプの似ている配列間のコピーをする際には要素を1つずつコピーする必要があります。

例えば、整数タイプの配列要素を実数タイプの配列にコピーする場合、次のように行います。

```
$Size:=Size of array(ArrInt)
ARRAY REAL(ArrReal;$Size)
        `実数配列と整数配列のサイズを同じにする。
For($i;1;$Size)
        ArrReal{$i}:=ArrInt{$i}
                `それぞれの要素をコピーする。
End for
```

処理中に配列の次元数を変更することはできませんので、注意してください。1次元配列を2次元配列にコピーすると、エラーメッセージが出力されます。

SELECTION TO ARRAY、ARRAY TO SELECTION、DISTINCT VALUES、SELECTION RANGE TO ARRAY

4th Dimensionのインタプリタモードと同様、これら4つのコマンドでは配列の定義は要求されません。タイプが定義されていない配列にはコマンドで指定したフィールドのデータタイプが割り当てられます。

以下のように記述すると、"MyArray"は整数配列になります（"IntField"が整数フィールドの場合）。

```
SELECTION TO ARRAY([MyTable]IntField;MyArray)
```

配列が定義されている場合は、フィールドと同じデータタイプになっているかどうか確認してください。整数、倍長整数、実数は似ていますが、同じタイプではありません。

ただし、テキストや文字のデータタイプでは、多少許容範囲が広くなります。デフォルトにより、文字タイプのフィールドを使用するコマンドで配列をあらかじめ定義せずに引数として使った場合、配列に割り当てられるデフォルトのデータタイプはテキストです。あらかじめ配列を文字またはテキストタイプとして宣言されている場合、ユーザの指定が適用されます。テキストタイプのフィールドについても同様で、宣言されたタイプが優先されます。

SELECTION TO ARRAY、**SELECTION RANGE TO ARRAY**、**ARRAY TO SELECTION**、**DISTINCT VALUES**コマンドは、1次元の配列でしか使用できません。

SELECTION TO ARRAYコマンドにはもう１つのシンタックスがあります。

SELECTION TO ARRAY（table;array）

この場合、変数配列は倍長整数になります。**SELECTION RANGE TO ARRAY**コマンドについても同様です。

LIST TO ARRAY、ARRAY TO LIST

LIST TO ARRAYおよび**ARRAY TO LIST**コマンドに使用できる配列は次の２種類だけです。

■ 1次元の文字配列

■ 1次元のテキスト配列

このコマンドの場合、引数に使う配列をあらかじめ宣言する必要はありません。デフォ

ルトでは、宣言されていない配列は、テキスト配列になります。配列があらかじめテキストか文字タイプで宣言されていない場合、ユーザの指定したタイプになります。

配列に関連したコマンドでのポインタの使用

ポインタについての節で説明したように、配列を定義するコマンドの引数にポインタ参照が使われていると、4D Compilerにはタイプの矛盾を発見できません。

Pointer->が配列を表示するところで、下記のように記述した場合、"Pointer->"が配列だとすると、フィールドと配列のタイプが同じかどうかチェックできません。

SELECTION TO ARRAY([Table]Field;Pointer->)

フィールドと配列のタイプの矛盾が起こらないようプログラマが気をつけるべきです。ポインタで参照する配列は、必ずタイプを定義してください。4D Compilerは、引数にポインタを使用している配列定義ステートメントを見つけると、警告メッセージを出力します。警告メッセージはこの種の矛盾を見つける際に役立ちます。

ローカル配列

データベースで、ローカル配列（定義されたメソッド内のみで有効な配列）を使用している場合は、使用前に明確に宣言しておく必要があります。ローカル配列を定義するには、**ARRAY REAL**、**ARRAY INTEGER**など、配列を定義するコマンドを使用します。

例えば、プロシージャで10個の要素を持つローカルな整数配列を作る場合、次のようなコマンドを使用前に定義しておきます。

```
ARRAY INTEGER($MyArray;10)
```

ランゲージコマンド

■ **Get pointer**(varName)

■ **Type**(object)

■ **EXECUTE**(statement)

■ **TRACE**

■ **NO TRACE**

Get pointer

Get Pointer関数は、渡された引数のポインタを返します。

ポインタの配列を初期化する場合、配列の各要素はそれぞれ与えられた変数を表します。例えばV1、V2、...V12というような12個の変数の場合、以下のように書くことができます。

```
ARRAY POINTER(Arr;12)
Arr{1}:=->V1
Arr{2}:=->V2
...
Arr{12}:=->V12
```

また、以下のように書くこともできます。

```
ARRAY POINTER(Arr;12)
For($i;1;12)
    Arr{$i}:=Get pointer("V"+String($i))
```

```
End for
```

この処理が終了すると、各要素が変数<Vi>を指すポインタの配列ができます。この2つの書き方は、両方ともコンパイルできますが、他の場所で変数V1...V12のタイプが明らかにされていないと、コンパイラはデータタイプを決定できません。そのため、このような変数は別の場所で明示的に使用するか、または定義する必要があります。

明示的に変数を定義する方法は、2通りあります。

■ コンパイラコマンドを使用してV1...V12を定義するには以下のように記述します。

```
C_LONGINT(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

■ メソッドでV1...V12に値を代入するには以下のように記述します。

```
V1:=0
V2:=0
...
V12:=0
```

タイプ設定（オブジェクト）

コンパイルされたデータベースの変数のデータタイプは1つだけなので、この関数は不必要に思えるかも知れません。しかし、ポインタを使用する際、この関数が役立ちます。例えば、ポインタが参照する変数のデータタイプを知りたい場合、ポインタの性質上、指されているオブジェクトがわかりにくい場合があります。

EXECUTE

EXECUTEコマンドは、インタプリタモードでは有効ですが、コンパイルモードではその利点を活かすことができません。コンパイルモードでは、引数として**EXECUTE**コマンドに渡された時点でメソッド名が解釈されるため、4D Compilerの利点を活用できない上に、引数のシンタックスチェックもできません。また、引数にローカル変数を使用することもできません。

EXECUTEコマンドは、複数のステートメントに置き換えることができます。以下に2つの例を挙げます。

例題

```
i:= FormFunc
EXECUTE("INPUT FORM(Form"+String(i)+")")
```

これは、以下ように書き換えることができます。

```
i:=FormFunc
```

```
VarForm:="Form"+String(i)
INPUT FORM(VarForm)
```

また次の例では、

```
$Num:=SelPrinter
EXECUTE("Print"+$Num)
```

ここでは、**EXECUTE**コマンドを**Case of**関数に置き換えることができます。

```
Case of
    ¥ ($Num=1)
        Print1
    ¥ ($Num=2)
        Print2
    ¥ ($Num=3)
        Print3
End case
```

EXECUTEコマンドは常に置き換えが可能です。実行するメソッドはデータベースのプロジェクトメソッドのリストから選択されたもので、その数には限りがあります。このため、**EXECUTE**コマンドは必ず**Case of**文や他のコマンドで置き換えることができます。さらに、コードの実行速度は**EXECUTE**コマンドよりも速くなります。

TRACEおよびNO TRACE

これらのコマンドはデバッグ処理の段階で使用します。コンパイルされたデータベースでは機能しません。4D Compilerはこれらのコマンドを無視するので、メソッド内に残しておいても問題ありません。

変数

■ **Undefined**(variable)

■ **SAVE VARIABLES**(document;variable1{; variable2...})

■ **LOAD VARIABLES**(document;variable1{; variable2...})

■ **CLEAR VARIABLE**(variable)

Undefined

4D Compilerではコンパイルモードの変数は必ず定義されます。データタイプは、コンパイルが完了するまでに設定されます。このため、引数が渡されると毎回**Undefined**関数はFalse（偽）を返します。

注：アプリケーションがコンパイルモードで実行されているかどうかはCompiled application関数を呼び出して確認してください。

SAVE VARIABLES、LOAD VARIABLES

インタプリタでは、**LOAD VARIABLE**コマンドの実行後にUndefined関数を使用して変数が未定義かどうか調べることにより、ドキュメントファイルの存在の有無をチェックできます。コンパイル後はこの方法を使用できません。**Undefined**関数が常に**False**（偽）を返すからです。

このテストはインタプリタでもコンパイル後でも次のようにして実行できます。

1. どの変数にとっても無効な値で、ロードする変数を初期化します。
2. LOAD VARIABLEコマンドを実行した後、ロードした変数のうちの1つを初期値と比較します。

メソッドは以下のようになります。

```
Var1:="xxxxxx"
        `"xxxxxx"はLOAD VARIABLEによって返されない値
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LOAD VARIABLE("Document";Var1;Var2;Var3;Var4)
If(Var1="xxxxxx")
        ` ドキュメントが見つからなかった
        ...
Else
        ` ドキュメントが見つかった
...
End if
```

CLEAR VARIABLE

インタプリタモードでは、この関数には次の2つのシンタックスがあります。

CLEAR VARIABLE（変数）
CLEAR VARIABLE（"a"）

コンパイル後のデータベースでは、1番目のシンタックス**CLEAR VARIABLE**（変数）は変数を再度初期化します（数値は0に、文字列やテキストは空にする等）。理由は、コンパイル後は未定義の変数が存在しないからです。したがって、コンパイル後はテキスト、ピクチャ、BLOB、配列タイプの変数を除き、**CLEAR VARIABLE**コマンドで変数のメモリを解放することはできません。

配列の場合、**CLEAR VARIABLE**コマンドは、要素数が0の新しい配列の定義を意味します。

整数配列の場合、**CLEAR VARIABLE**（配列）は以下の2つのいずれかの式と同じ結果になります。

```
ARRAY INTEGER(MyArray;0)
      `1次元配列である場合
ARRAY INTEGER(MyArray;0;0)
      `2次元配列である場合
```

2番目のシンタックス**CLEAR VARIABLE**("a")は、コンパイラが名前ではなくアドレスで変数にアクセスするため4D Compilerで使用することはできません。

特定のコマンドで使用されるポインタ

以下のコマンドには、共通する特徴が1つあります。これらのコマンドでは、最初の任意の引数＜テーブル＞を受け入れ、2番目の引数をポインタにすることが可能です。

ADD TO SET
APPLY TO SELECTION
COPY NAMED SELECTION
CREATE EMPTY SET
CREATE SET
CUT NAMED SELECTION
DIALOG
EXPORT DIF
EXPORT SYLK
EXPORT TEXT
GOTO RECORD
GOTO SELECTED RECORD
GRAPH TABLE QUERY
IMPORT DIF
IMPORT SYLK
IMPORT TEXT
INPUT FORM
LOAD SET
LOCKED ATTRIBUTES
ORDER BY
ORDER BY FORMULA
OUTPUT FORM
PAGE SETUP
Print form
PRINT LABEL
QR REPORT
QUERY
QUERY BY FORMULA
QUERY SELECTION
SELECTION BY FORMULA
REDUCE SELECTION
RELATE MANY
REMOVE FROM SET

コンパイルモードでは、任意の引数＜テーブル＞を扱うことが可能です。しかし、これらのコマンドに最初に渡された引数がポインタの場合、4D Compilerはポインタが何を参照しているのかわかりません。このため、この引数はテーブルポインタとして扱われるので注意が必要です。

QUERYコマンドのケースで考えてみましょう。**QUERY**コマンドのシンタックスは以下のとおりです。

```
QUERY({table{;formula{;*}}}
```

この式の最初の引数の要素はフィールドでなければなりません。

次のように書いた場合、4D Comilerは2番目の要素内でフィールドを表す記号を探します。

```
QUERY(PtrField->=True)
```

"="の記号が見つかった場合、これを処理するための関数式を判別できないため、エラーメッセージが出力されます。

曖昧にならないようにするためには、以下のいずれかのように記述することができます。

```
QUERY(PtrTable->;PtrField->=True)
```

または、

```
QUERY([Table];PtrField->=True)
```

定数

独自の4DK#リソース（定数）を作成する場合は、必ず数値のタイプを倍長整数（L）もしくは実数（R）に、文字列のタイプを（S）に設定してください。その他のタイプでは警告メッセージが表示されます。

参照

Optimization Hints、Typing Guide、Using Compiler Directives.

最適化のヒント

"良いプログラムを作成する"ための決定的な方法を説明するのは難しいことですが、良い構造を持つプログラムの利点を再度ここで強調します。4th Dimensionは構造化プログラミングが可能であり、この能力はプログラミングの大きな助けになります。

構造化されたデータベースとそうでないデータベースとでは、コンパイルに費やす労力は同じでも結果は大きく異なります。例えば、n個のオブジェクトに対する共通メソッドは、同じステートメントで書かれたn個のオブジェクトメソッドより、インタプリタモードでもコンパイルモードでも、はるかに良い結果をもたらすことでしょう。つまり、プログラミングの質がコンパイルされたコードの品質にも影響を与えるのです。

4th Dimensionで実行することにより、コードを段階的に改善します。4D Compilerを頻繁に使用することにより、間違いを訂正するフィードバックを得て、最も効果的な解決方法に到達できます。

この節では、単純な繰り返しの作業にかかる時間を短縮するためのアドバイスや秘訣を紹介します。

コードに関するコメント

プログラミングテクニックによっては、コードが他の人にとって理解しづらいものもあります。また、自分で修正する場合でも時が経つと分からなくなることもあります。このため、詳細にわたるコメントをメソッドに入れることをお勧めします。コメントが多すぎるとインタプリタデータベースでは実行が遅くなりますが、コンパイルされたデータベースにはまったく影響しません。

コンパイラコマンドの使用によるコードの最適化

コンパイラコマンドを使用すると、コードの実行速度がかなり速くなります。用途から変数のタイプを決定する場合、一番広い範囲をカバーできるタイプを設定します。例えば、変数のタイプを"Var:=5"というステートメントで定義する場合、整数のみ使用されているのにもかかわらず、コンパイラはタイプを実数に設定します。

数値変数

環境設定で数値のタイプが設定されていない場合、デフォルトとして（コンパイラコマンドにより設定されたタイプではない）数値変数には、実数が割り当てられます。実数の場合、倍長整数より計算が遅いので、数値変数が常に倍長整数タイプだと分かっている場合は、コンパイラコマンド**C_INTEGER**もしくは**C_LONGINT**で変数を定義すると効果的です。

例えば、ループのカウンタは常に整数として定義しておくとよいでしょう。4th Dimensionの関数の中には整数を返すものがあります（**Ascii**関数等）。4D Compilerは、このような関数の結果を未定義の変数に代入する場合も、変数のタイプを整数ではなく実数にします。変数が別のタイプの値に使用されないことが明らかな場合は、必ずコンパイラコマンドで変数を宣言してください。

ここで簡単な例を示します。指定された範囲内のランダムな数を返す関数です。

```
$0:=Mod(Random;($2-$1+1))+$1
```

このように記述されていると、4D Compilerは"$0"のタイプを整数や倍長整数ではなく実数に設定してしまうので、メソッドにコンパイラコマンドを使用してください。

```
C_LONGINT($0)
$0:=Mod(Random;($2-$1+1))+$1
```

メソッドの戻り値に使用するメモリスペースも少なく、プロシージャの実行速度も速くなります。

もう1つの例を紹介します。2つの変数を倍長整数として定義します。

```
C_LONGINT($var1;$var2)
```

そして、他の2つの変数の合計が3つ目のタイプの定義されていない変数に返されます。

```
$var3:=$var1+$var2
```

4D Compilerは、3つ目の変数"$var3"を実数とします。結果を倍長整数としたい場合は、倍長整数として明示的に定義しなければなりません。

注：コンパイルモードでの計算結果は、計算結果を受け取る変数のタイプではなく、計算に使用される数値のデータタイプであることに注意してください。

下記の例では、変数が倍長整数として計算されています。

```
C_REAL($var3)
C_LONGINT($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

コンパイルモードおよびインタプリタモードの両方で、$var3は-2147483648となります。

しかし、下記の例では、最適化のために4D Compilerは、1の値を整数と見なします。

```
C_REAL($var3)
C_LONGINT($var1)
$var1:=2147483647
$var3:=$var1+1
```

コンパイルモードでは、倍長整数として計算されるため、"$var3"は"-2147483648"となります。インタプリタモードでは、実数として計算されるため、"$var3"は"2147483648"となります。

ボタンは倍長整数として定義できる具体的なケースです。

文字

環境設定で特にタイプが指定されていない場合、文字変数のデフォルトのタイプとしてテキストが割り当てられます。

例えば、以下のように書いた場合、"MyString"のタイプを4D Compilerはテキストとして定義します。

```
MyString:="Hello"
```

この変数を頻繁に使用するのであれば、**C_STRING**コマンドで定義することをお勧めします。テキスト変数を処理するより、長さが決まっている文字タイプの変数を処理する方がはるかに速いからです。コンパイラコマンドの動きを決めるこのルールを覚えておいてください。

文字の値を比較したい場合、文字自身についてというよりも、そのASCII値としての比較をしてください。通常の文字比較では、その過程で読み分け記号等の文字に代わるすべての文字が考慮されてしまいます。

その他のヒント

この節は、2次元配列、フィールド、およびポインタについてのヒントを記述します。

2次元配列

2次元配列は、2番目の次元が1番目の次元より大きい方が実行効率が上がります。

例えば、次のように定義された配列は、

```
ARRAY INTEGER(Array;5;1000)
```

次のような配列より効率が高くなります。

```
ARRAY INTEGER(Array;1000;5)
```

フィールド

フィールドを使用して複数の演算を行う場合、変数にフィールドの値を代入して計算をしたほうが、直接フィールドで計算するより効率が良くなります。以下のメソッドをご覧ください。

```
Case of
    ¥ ([Client]Dest="New York City")
        Transport:="Messenger"
    ¥ ([Client]Dest="Puerto Rico")
        Transport:="Air mail"
```

```
        ¥ ([Client]Dest="Overseas")
                Transport:="Express mail service"
        Else
                Transport:="Regular mail service"
End case
```

このメソッドは以下のように記述すると実行速度が速くなります。

```
$Dest:=[Client]Dest
Case of
        ¥ ($Dest="New York City")
                Transport:="Messenger"
        ¥ ($Dest="Puerto Rico")
                Transport:="Air mail"
        ¥ ($Dest="Overseas")
                Transport:="Express mail service"
        Else
                Transport:="Regular mail service"
End case
```

ポインタ

フィールドの場合と同じように、ポインタ参照よりも変数を使用する方が速くなります。ポインタ参照される変数で何回も計算する場合、値を変数に格納すると時間を節約できます。

例えば、ポインタ"MyPtr"がフィールドや変数を指しており、その値を使用して一連のテストをする場合は以下のように記述することができます。

```
Case of
        ¥ (MyPtr-> = 1)
                Sequence 1
        ¥ (MyPtr-> = 2)
                Sequence 2
        ...
End case
```

このテストは、以下のように記述することで実行速度が上がります。

```
Temp:=MyPtr->
Case of
        ¥ (Temp= 1)
                Sequence 1
        ¥ (Temp= 2)
```

```
            Sequence 2
        ...
    End case
```

ローカル変数

コードを作成する場合は、できるだけローカル変数を使用してください。ローカル変数には、次の利点があります。

- ローカル変数は、データベースのスペースを多く必要としません。ローカル変数は、メソッド中で使用されると作成され、メソッドの実行が終わると破棄されます。
- 生成されるコードは、ローカル変数（特に倍長整数）のために最適化されます。これはループカウンタに便利です。

参照

Syntax Details、Typing Guide、Using Compiler Directives

エラーメッセージ

ここでは、4D Compilerが出力するメッセージについて説明します。メッセージは、以下の数種類になります。

- 警告メッセージ
- エラーメッセージ
- 範囲チェックメッセージ

警告メッセージ

このメッセージは、コンパイルの過程で出力されます。ここでは、各メッセージを問題のあるコードの例と共に示します。

「COPY ARRAYコマンド中にポインタが存在します」

```
COPY ARRAY(Pointer->;Array)
```

「SELECTION TO ARRAYコマンド中にポインタが存在します」

```
SELECTION TO ARRAY(Pointer->;MyArray)
SELECTION TO ARRAY([MyTable]MyField;Pointer->)
```

「ARRAY TO SELECTIONコマンド中にポインタが存在します」

```
ARRAY TO SELECTION(Pointer->;[MyTable]MyField)
```

「LIST TO ARRAYコマンド中にポインタが存在します」

LIST TO ARRAY(List;Pointer->)

「ARRAY TO LISTコマンド中にポインタが存在します」

ARRAY TO LIST(Pointer->;List)

「配列定義コマンド中にポインタが存在します」

ARRAY REAL(Pointer->;5)

ARRAY REAL(Array; Pointer->)と書いた場合、このメッセージは出力されません。配列の次元数はデータタイプに影響を与えないからです。ポインタで参照する配列は、事前に定義しておく必要があります。

「DISTINCT VALUESコマンド中にポインタが存在します」

DISTINCT VALUES(Pointer->;Array)

「Undefined関数は使用しないでください」

If(**Undefined**(Variable))

コンパイルしたデータベース中のUndefinedコマンドは、常にFalse（偽）を返します。

「このメソッドは、パスワードにより保護されています」

「フォームXページ目の自動動作ボタンの名前がありません」

タイプ障害を避けるため、すべてのボタンに名前が必要です。

「ポインタの参照先を文字として処理します」

Pointer->[[2]]:="a"

「文字列のインデックスを数値タイプとして処理します」

文字型変数等[[Pointer->]]:="a"

「配列のインデックスを実数タイプとして処理します」

ALERT(MyArray{Pointer->})

「プラグインコマンドの呼び出しでパラメータが不足しています」

WR SET FONT(Area)

エラーメッセージ

このメッセージはコンパイルの処理段階で生成されます。データベースのコンパイルされたデータベースを作成するために表示されるエラーを修正するかどうかはユーザが判断します。各メッセージを問題のあるコードの例と共に示します。

メッセージは次のグループに分けて説明します。

■ 変数タイプ設定

■ シンタックス

■ 引数

■ 演算子

■ プラグインコマンド

■ 総合エラー

変数タイプ設定

「変数タイプが異なるため代入できません」

```
MyReal:=12.3`MyRealは実数
MyBoolean:=True    `MyBooleanはブール
MyReal:=MyBoolean` ブールは実数に代入できません
```

「文字列の長さは変更できません」

```
C_STRING(3;MyString)
C_STRING(5;MyString)
```

「配列の次元数は変更できません」

```
ARRAY TEXT(MyArray;5;5)
ARRAY TEXT(MyArray;5)
```

「フォームの配列変数タイプが一致しません

```
ARRAY INTEGER(MyArray)
```

「配列定義コマンドに次元数がありません」

```
ARRAY INTEGER(MyArray)
```

「変数が必要です」

```
COPY ARRAY(MyArray;"")
```

「定数が必要です」

```
C_STRING(Variable;MyString)
```

「変数タイプが不明です。この変数はメソッドで使用されています」

変数タイプが決められません。コンパイラコマンドが必要です。

「定数タイプが無効です」

```
OK:="The weather is nice"
```

「メソッドM1が不明です」

この行は存在しないメソッド、もしくは存在しなくなったメソッドを呼び出しています。

「誤ったフィールドの使用」

```
MyDate:=Add to date(BooleanField;1;1;1)
```

「文字列の長さは255文字（バイト）までです」

```
C_STRING(325;MyString)
```

「変数Variableはメソッドではありません」

```
Variable(1)
```

「変数Variableは配列ではありません」

```
Variable{5}:=12
```

「結果が式と一致しません」

```
Text:="Number"+Num(i)
```

「変数タイプが不適切です」

```
Integer:=MyDate*Text
```

「変数$Iのタイプを固定文字から実数に変更できません」

```
$i:="3"
$($i):=5
```

「配列のインデックスが数値ではありません」

```
IntArray{"3"}:=4
```

「変数Variableのタイプをテキストから配列に変更できません」

```
C_TEXT(Variable)
COPY ARRAY(TextArray;Variable)
```

「変数Variableのタイプをテキストから実数に変更できません」

```
Variable:=Num(Variable)
```

「配列MyBooleanのタイプをブールタイプの配列から実数タイプの変数に変更できません」

```
Variable:=MyBoolean
```

「配列IntArrayのタイプを整数タイプの配列からテキストタイプの配列に変更できません」

```
ARRAY TEXT(IntArray;12)
```

“IntArray”が他の所で整数配列として定義されている場合。

「ポインタタイプ以外の変数をポインタとして参照しています」

```
Variable->:=5
```
"Variable"のタイプがポインタではない場合。

「変数Var1のタイプをテキストから数値に変更できません」

```
Var1:=3.5
```

「フィールドの使い方に誤りがあります」

```
Variable:=[MyTable]MyField
```
[MyTable]MyFieldは日付フィールドで"Variable"は数値です。

シンタックス

「ポインタタイプ以外の変数をポインタとして参照しています」

```
Variable:=Num("The weather is nice")->
```
この関数を参照することは使用できません。

「シンタックスエラー」

```
If(Boolean)
End for
```

「"}"がありません」

文中の右カッコ"}"の数が左カッコ"{"の数より少ない場合。

「"{"がありません」

文中の左カッコ"{"の数が右カッコ"}"の数より少ない場合。

「")"がありません」

文中の右カッコ")"の数が左カッコ"("の数より少ない場合。

「"("形のカッコがありません」

文中の左カッコ"("の数が右カッコ")"の数より少ない場合。

「フィールドが必要です」

```
If(Modified(Variable))
```

「変数が必要です」

```
C_INTEGER([MyTable]MyField)
```

「定数が必要です」

```
C_INTEGER(${"3"})
```

「";"が必要です」

COPY ARRAY(Array1 Array2)

「文字参照記号"]]"が足りません」

MyString[[3:="a"

「文字参照記号"[["が足りません。]

MyString 3]]:="a"

「ここではサブテーブルを使えません」

ARRAY TO SELECTION(Array;Subtable)

「If文への引数はブールタイプでなければなりません」

If(Pointer)

「式が複雑すぎます」

命令文を短く分割してください。

「メソッドが複雑すぎます」

"Case of...End"または"If...End if"構造が多すぎます。

「フィールドが不明です」

使用メソッドがおそらく他のデータベースからコピーされたもので、存在しないフィールドへの参照が式に含まれています。

「テーブルが不明です」

使用メソッドがおそらく他のデータベースからコピーされたもので、存在しないテーブルへの参照が式に含まれています。

「ポインタが参照する式に誤りがあります」

Pointer:=->Variable+3

「文字列インデックスの使い方に誤りがあります」

MyReal[[3]]

または

MyString[[Variable]]

"Variable"が数値変数以外の場合。

引数

「この式の結果を引数としてこのメソッドもしくはコマンドに代入することはできません」

```
MyMethod(Num(MyString))
```
MyMethodの引数にブール式が必要な場合。

「このメソッドに渡す引数が多すぎます」

```
DEFAULT TABLE(Table;Form)
```

「この値を引数としてこのメソッドもしくはコマンドに代入することはできません」

```
MyMethod(3+2)
```
MyMethodの引数にブール式が必要な場合。

「結果のタイプが一致しません」

```
C_INTEGER($0)
$0:=False
```

「変数のタイプが一致しません」

```
C_INTEGER(${3})
For($i;3;5)
    ${$i}:=String($i)
End for
```

「このコマンドに引数は不要です」

```
SHOW TOOL BAR(MyVar)
```

「このコマンドには1つ以上の引数が必要です」

```
DEFAULT TABLE
```

「MyStringを引数としてメソッドに代入することはできません」

```
MyMethod(MyString)
```
MyMethod引数にブール式が必要な場合。

「引数"$1"のタイプが呼ぶ側のメソッドと呼ばれる側のメソッドで一致していません」

```
Calculate("3+2")
```
メソッド"Calculate"の中でコンパイラコマンド**C_INTEGER**($1)が記述されている場合。

「COPY ARRAYコマンドの引数に変数が含まれています」

```
COPY ARRAY(Variable;Array)
```

「変数$1のタイプを数値からテキストに変更できません」

```
$1:=String($1)
```

「配列を引数として使用できません」

```
ReInit(MyArray)
```
配列をメソッドに代入する場合は、配列にポインタを渡してください。

演算子

「演算と変数タイプが一致しません」

```
Bool2:=Bool1+True
```

ブール値のフィールドを加算することはできません。

「演算子>は不要です」

```
QUERY(MyTable;[MyTable]MyField=0;>)
```

「これらの変数タイプを比較することはできません」

```
If(Number=Picture2)
```

「この変数タイプに-(マイナス)符号を付けることはできません」

```
Boolean:=-False
```

プラグインコマンド

「プラグインコマンド"PExt"の定義が正しくありません」

「プラグインコマンドに対する引数が足りません」

「プラグインコマンドに対する引数が多すぎます」

「プラグインコマンドの変数の定義が正しくありません」

総合エラー

「同じ名前のメソッドが複数あります」

データベースをコンパイルするには、プロジェクトメソッドすべてに異なる名前をつける必要があります。

「内部エラーNo.xx」

このメッセージが出力されたら4D社テクニカルサポートに問い合わせ、エラー番号を教えてください。

「変数Variableのタイプが不明です。この変数はM1メソッドで使用されています」

変数のタイプが判断できません。コンパイラコマンドを使用してください。

「オリジナルのメソッドが壊れています」

オリジナルのストラクチャでメソッドが壊れています。該当するメソッドを削除するか置き換えてください。

「4Dのコマンドではありません」

メソッドが壊れています。

「フォームに配置した変数のタイプは変更できません」

フォーム内のグラフタイプの変数に"OK"といった名前をつけると、このメッセージが出力されます。

「関数と変数が同じ名前です：Name」

関数か変数いずれかの名前を変えてください。

「メソッドと変数が同じ名前です：Name」

メソッドか変数いずれかの名前を変えてください。

「プラグインコマンドと変数が同じ名前です：Name」

プラグインコマンドか変数いずれかの名前を変えてください。

範囲チェックメッセージ

これらのメッセージはコンパイルされたデータベースの実行中に4th Dimensionのエラーメッセージウインドウに表示されます。

「結果が変数の範囲を越えました」

```
MyArray{17} := 2.3
```

上記のステートメントが実行された場合、MyArrayは要素数が5つの配列です。このメッセージはMyArray{17}にアクセスしようとすると表示されます。

「ゼロによる割算が発生しました」

```
Var1:=0
Var2:=2
Var3:=Var2/ Var1
```

「引数がありません」

カレントプロシージャには引数が3つしか渡されていないのに、ローカル変数“$4”を使用しているような場合にこのメッセージが表示されます。

「ポインタが初期化されていません」

```
MyPointer->:=5
```

MyPointerがまだ初期化されていない場合。

「代入先が小さすぎます」

```
C_STRING(MyString1;5)
C_STRING(MyString2;10)
```

```
MyString2:="Flowers"
MyString1:= MyString2
```

「文字参照エラー」

```
i:=-30
MyString[[i]]:=MyString2
```

「引数に渡された文字列が空です」

```
MyString[[1]]:= ""
```

「ゼロによる割算が発生しました」

```
Var1:=0
Var2:=2
Var3:=Var2% Var1
```

「EXECUTE内での無効な引数」

```
EXECUTE("MyMethod(MyAlpha)")
```

MyMethod引数に英数字以外のタイプが必要な場合。

「変数へのポインタがコンパイラには未知のものです」

```
MyPointer:= Get pointer("Variable")
MyPointer:= "MyString"
```

変数がデータベース内で明確に宣言されていない場合。

「ポインタを使用して再タイプ設定を試行しました」

```
Boolean:=Pointer->
```

ポインタが整数タイプのフィールドを指す場合。

「ポインタの誤った使用」

```
Character:=StringVar[[Pointer]]
```

ポインタが数値を指さない場合。

参照

Optimization Hints、Syntax Details、Typing Guide、Using Compiler Directives

C_BLOB

C_BLOB({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_BLOBコマンドは、指定されたそれぞれの変数をBLOBタイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_BLOB(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_BLOB(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド

C_BOOLEAN

C_BOOLEAN({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_BOOLEANコマンドは、指定されたそれぞれの変数をブール変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_BOOLEAN(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_BOOLEAN(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters

C_DATE

C_DATE({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_DATEコマンドは、指定されたそれぞれの変数を日付変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_DATE(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_DATE(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters

C_GRAPH

C_GRAPH({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_GRAPHコマンドは、指定されたそれぞれの変数をグラフ変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_GRAPH(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_GRAPH(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド

C_INTEGER

C_INTEGER({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

注：このコマンドは、以前のデータベースとの互換性を保つために4th Dimensionに残されています。実際には、4Dとコンパイラは整数を倍調整数へと内部的にタイプ変換します。

例えば

```
C_INTEGER($MyVar)
$TheType:=Type($MyVar) `$TheType = 9(Is Longint)
```

説明

C_INTEGERコマンドは、指定されたそれぞれの変数を整数タイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_INTEGER(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_INTEGER(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters、C_LONGINT、C_REAL.

C_LONGINT

C_LONGINT({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_LONGINTコマンドは、指定されたそれぞれの変数を倍長整数タイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_LONGINT(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_LONGINT(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters、C_INTEGER、C_REAL

C_PICTURE

C_PICTURE({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_PICTUREコマンドは、指定されたそれぞれの変数をピクチャタイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方を4D Compiler用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_PICTURE(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_PICTURE(${5})」宣言は、4Dと4D Compilerに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters

C_POINTER

C_POINTER({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_POINTERコマンドは、指定されたそれぞれの変数をポインタタイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方を4D Compiler用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_POINTER(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_POINTER(${5})」宣言は、4Dと4D Compilerに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters

C_REAL

C_REAL({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_REALコマンドは、指定されたそれぞれの変数を実数タイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_REAL(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_REAL(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters、C_INTEGER、C_LONGINT

C_STRING

C_STRING({メソッド ;}サイズ; 変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
サイズ	数値	→	文字列の長さ
変数	変数または${...}	→	定義する変数の名前

説明

C_STRINGコマンドは、指定されたそれぞれの変数を文字列変数としてキャストします。

引数＜サイズ＞は、定義した変数に納められる文字（列）の最大の長さを指定します。文字は最大255バイトまでです。処理速度が問題となる場合、テキスト変数ではなくできるだけ文字（列）変数を使用してください。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_STRING(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_STRING(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters、C_TEXT

C_TEXT

C_TEXT({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_TEXTコマンドは、指定されたそれぞれの変数をテキストタイプの変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_TEXT(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_TEXT(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters、C_STRING

C_TIME

C_TIME({メソッド ;}変数 {; 変数2 ; …; 変数N})

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	変数または${...}	→	定義する変数の名前

説明

C_TIMEコマンドは、指定されたそれぞれの変数を時間変数としてキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ（$0、$1、$2等）またはその両方をコンパイラ用に事前に定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_TIME(${...})」を使用すると、パラメータ群がメソッドの最後のパラメータ群である場合に、不定数のパラメータを同一タイプとして宣言できます。例えば、「C_TIME(${5})」宣言は、4Dとコンパイラに対して、5番目のパラメータから始め、そのメソッドがそのタイプの不定数のパラメータを受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

例題

前節での例を参照してください。

参照

コンパイラコマンド、Count parameters

IDLE

IDLE

説明

IDLEコマンドは、コンパイラと一緒に使用する目的だけに作成されたコマンドです。このコマンドは、コンパイルしたデータベースにおいてのみ使用され、制御を4th Dimensionエンジンに戻す命令が、そのデータベース中のメソッドにない場合に使用します。例えば、ループ内に全くコマンドを含まない**For**ループを持ったメソッドを実行している場合に、**ON EVENT CALL**コマンドで割り込みメソッドをインストールしていてもループを抜け出すことはできません。また、他のアプリケーションに切り替えることができません。このような場合、**IDLE**コマンドを挿入して、4th Dimensionによりイベントがトラップされるようにします。割り込みを起こしたくない場合は、**IDLE**コマンドを削除します。

例題

以下の例は、**IDLE**コマンドを呼び出さないかぎり、コンパイルしたデータベースでループから抜け出すことができません。

```
` Do Something プロジェクトメソッド
ON EVENT CALL("EVENT METHOD")
<>vbWeStop:=False
MESSAGE("実行中..."+Char(13)+"どれかキーを押すと実行を停止します。")
Repeat
    IDLE ` 4Dコマンドを含まない処理を実行する
Until(<>vbWeStop)
ON EVENT CALL("")

` EVENT METHOD プロジェクトメソッド
If(Undefined(KeyCode))
    KeyCode:=0
End if
If(KeyCode#0)
    CONFIRM("処理を停止しても良いですか？")
    If(OK=1)
        <>vbWeStop:=True
    End if
End if
```

参照

コンパイラコマンド、ON EVENT CALL

データベースメソッド

データベースメソッドは、通常のセッションイベントが発生すると、4th Dimensionが自動的に実行するメソッドです。



データベースメソッドを作成するかまたは開いて編集する方法は、以下の通りです。

1.「エクスプローラ」ウインドウを開く。

2.「メソッド」ページを選択する。

3.「データベースメソッド」テーマを拡げる。

4. メソッド名をダブルクリックする。

または、以下のように実行します。

1. メソッドを選択する。

2. enterキーまたはreturnキーを押す。

データベースメソッドは、他のメソッドと同じ方法で編集します。

データベースメソッドは、他のメソッドから呼び出すことはできません。データベースメソッドは、作業セッション中のある時点で、4th Dimensionから自動的に起動されます。以下の表は、データベースメソッドの実行の概要を示しています。

データベースメソッド	4th Dimension	4D Server	4D Client
On Startup	○、1回	×	○、1回
On Exit	○、1回	×	○、1回
On Web Connection	○、複数	○、複数	×
On Backup Startup	○、複数	○、複数	○、複数
On Backup Shutdown	○、複数	○、複数	○、複数
On Server Startup	×	○、1回	×
On Server Shutdown	×	○、1回	×
On Server Open Connection	×	○、複数	×
On Server Close Connection	×	○、複数	×

各データベースメソッドに関する詳細は、以下の節を参照してください。

■ 「On Startup」データベースメソッド

■ 「On Exit」データベースメソッド

■ 「On Web Connection」データベースメソッド

■ 「On Backup Startup」データベースメソッド

■ 「On Backup Shutdown」データベースメソッド

■ 「On Server Startup」データベースメソッド（『4D Serverリファレンス』を参照）

■ 「On Server Shutdown」データベースメソッド（『4D Serverリファレンス』を参照）

■ 「On Server Open Connection」データベースメソッド（『4D Serverリファレンス』を参照）

■ 「On Server Close Connection」データベースメソッド（『4D Serverリファレンス』を参照）

「On Startup」データベースメソッド

「On Startup」データベースメソッドは、データベースを開くと呼び出されます。

この状態は、以下のような4D環境で発生します。

■ 4th Dimension

■ 4D Client（クライアント側で、接続が4D Serverにより受け付けられた後）

■ 4D Runtime

■ コンパイルし、4D Runtime Volume Licenseを組み込んだ4Dアプリケーション

注：「On Startup」データベースメソッドは、4D Serverによって起動されることはありません。

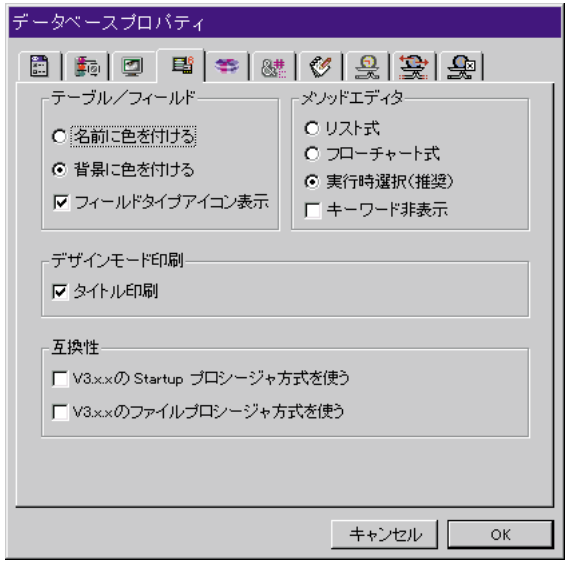
「On Startup」データベースメソッドは、4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、このデータベースメソッドをユーザが呼び出すことはできません。「On Startup」データベースメソッド内から作業を呼び出して実行するには、プロジェクトメソッドの場合と同様に、サブルーチンを使用します。

「On Startup」データベースメソッドは、以下のような処理に最適です。

- 作業セッション全体で使用するインタープロセス変数を初期化する。
- データベースを開いた時にプロセスを自動的に開始する。
- 以前の作業セッション中にこの目的で保存された初期設定やシステム定義をロードする。
- 条件が一致しない（システムリソースがない等の）場合には、明示的に**QUIT 4D**コマンドを呼び出して、データベースを開かないようにする。
- データベースを開く度に自動的に実行したい他の動作を実行する。

4th Dimensionバージョン3との互換性

データベースメソッドは、バージョン6で導入された新しいメソッドです。バージョン3の4th Dimensionでは、データベースを開いた時に4Dが自動的に実行するメソッドは「Startup」プロシージャだけでした。バージョン3のデータベースをバージョン6に変換し、新しい「On Startup」データベースメソッドの機能を活用したい場合には、下図の「データベースプロパティ」ダイアログボックスにある「旧バージョンのStartupプロシージャ方式を使用する」チェックボックスを選択解除する必要があります。このプロパティは「Startup」メソッドと「On Startup」データベースメソッドの切り替えにだけ影響を与えます。このプロパティを非選択にせずに、例えば「On Exit」データベースメソッドを追加すると、この後者のデータベースメソッドが4Dによって起動されます。



「On Exit」データベースメソッド

「On Exit」データベースメソッドは、データベースを終了すると呼び出されます。

この状態は、以下のような4D環境で発生します。

■ 4th Dimension

■ 4D Client（クライアント側で、接続が4D Serverにより受け付けられた後）

■ 4D Runtime

■ コンパイルし、4D Runtime Volume Licenseを組み込んだ4Dアプリケーション

注：「On Exit」データベースメソッドは、4D Serverによっては起動されません。

「On Exit」データベースメソッドは、4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、このデータベースメソッドをユーザが呼び出すことはできません。「On Exit」データベースメソッド内から作業を呼び出して実行するには、プロジェクトメソッドの場合と同様に、サブルーチンを使用します。

データベースは、以下のうちいずれかの状態になると終了します。

■ ユーザが「ユーザ」モードまたは「デザイン」モードの「ファイル」メニューから「終了」メニューを選択した場合

■ **QUIT 4D**コマンドへの呼び出しが実行された場合

■ 4Dのプラグインソフトから**QUIT 4D**エントリポイントへの呼び出しが実行された場合

データベースの終了がどのような方法で実行されたかに関わらず、4Dは以下のような処理を実行します。

■ 「On Exit」データベースメソッドがない場合には、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。

■ 「On Exit」データベースメソッドがある場合には、4Dは新しく作成されたローカルプロセスの中でこのメソッドの実行を開始します。したがって、このデータベースメソッドを使用し、インタープロセス間通信を通じて、（データ入力を）終了、または処理の実行を中止しなければならないことを、他のプロセスに通知することができます。4Dは、いずれ終了するということに注意してください。「On Exit」データベースメソッドは、必要なクリーンアップや終了の処理を実行することができますが、中断処理を拒否することができず、ある時点で終了することになります。

「On Exit」データベースメソッドは、以下のような処理を実行するには最適です。

■ データベースを開いた時に自動的に開始されたプロセスを停止する。

■ 「On Startup」データベースメソッドの以下のセッションの開始時に再使用される初期設定やシステム定義を（ローカルに、ディスク上に）保存する。

■ データベースを終了する度に実行したい処理が他にあれば、それを実行する。

注：「On Exit」データベース・メソッドがローカル／クライアントプロセスであるのを忘てはいけません。それはデータ・ファイルにアクセスすることができません。

このように、「On Exit」データベースメソッドが検索またはソートを実行すると、終了しようとしている4D Clientは「フリーズ」して、実際には終了しません。

クライアントがアプリケーションを終了するとき、ユーザがデータにアクセスする必要があるならば、データ・ファイルにアクセスすることができる「On Exit」データベースメソッド内から、新しいグローバルなプロセスを作成します。

すると、新しいプロセスが「On Exit」データベースメソッド実行の終わりの前に、正常終了します（例によって、インタープロセス変数を使うことによって）。

▼ 以下の例では、作業セッション中に発生する重要なイベントを把握し、その内容説明を“ジャーナル”という名前のテキストドキュメントに書き込むために使用します。

■ 「On Startup」データベースメソッドは、すべての使用プロセスにデータベースが実行されているかどうかについて通知するインタープロセス変数「<>vbQuit4D」を初期化します。このデータベースメソッドは、ジャーナルファイルがまだない場合には、これを作成します。

```
`「On Startup」データベースメソッド
C_TEXT(<>vtIPMessage)
C_BOOLEAN(<>vbQuit4D)
<>vbQuit4D:=False
If(Test path name("ジャーナル") # Is a document)
        $vhDocRef:=Create document("ジャーナル")
        If(OK=1)
                CLOSE DOCUMENT($vhDocRef)
        End if
End if
WRITE JOURNAL("セッションのオープン")
```

▼「WRITE JOURNAL」プロジェクトメソッドは、他のメソッドのサブルーチンとして使用され、受け取った情報をジャーナルファイルに書き込みます。

```
`「WRITE JOURNAL」プロジェクトメソッド
` WRITE JOURNAL(テキスト)
` WRITE JOURNAL(イベントの内容)
C_TEXT($1)
C_TIME($vhDocRef)
While(Semaphore("$Journal"))
        DELAY PROCESS(Current process ; 1)
End while
$vhDocRef:=Append document("ジャーナル")
If(OK=1)
        PROCESS PROPERTIES(Current process ; $vsProcessName ; $vlState ;
                                                vlElapsedTime ; $vbVisible)
        SEND PACKET($vhDocRef;String(Current date)+Char(9)+String(Current time)
                                                +Char(9)
                        +String(Current process)+Char(9)+$vsProcessName+
                                                Char(9)+$1+Char(13))
        CLOSE DOCUMENT($vhDocRef)
End if
CLEAR SEMAPHORE("$Journal")
```

ドキュメントファイルは、毎回開いて閉じられることに注意が必要です。さらに、ドキュメントファイルへのアクセス保護としてセマフォを利用していることにも注意が必要です。2つのプロセスがジャーナルファイルに同時にアクセスすることを防ぐためです。

▼「M_ADD_RECORDS」プロジェクトメソッドは、「カスタム」メニューから「レコード追加」メニュー項目が選択されると実行されます。

```
`「M_ADD_RECORDS」プロジェクトメソッド
MENU BAR(1)
```

```
Repeat
    ADD RECORD([テーブル1] ; *)
    If(OK=1)
        WRITE JOURNAL("テーブル1：" + "追加レコード#"
                                    +String(Record number([テーブル1]))
    End if
Until((OK=0) | <>vbQuit4D)
```

このメソッドは、ユーザが最後のデータ入力をキャンセルするか、またはデータベースを終了するまでループします。

▼[テーブル1]の入力フォームには、「On Outside Call」イベントの処理手順も含まれています。したがって、プロセスがデータ入力中であっても、ユーザは現在のデータ入力を保存するかまたは保存しないで、スムーズに終了できます。

```
` [テーブル1];"入力"フォームのフォームメソッド
Case of
    ¥(Form event=On Outside Call)
        If(<>vtIPMessage="終了")
            CONFIRM("このレコードで行った変更を保存したいですか？")
            If(OK=1)
                ACCEPT
            Else
                CANCEL
            End if
        End if
End case
```

▼「M_QUIT」プロジェクトメソッドは、「ファイル」メニューから「終了」が選択されると実行されます。

```
` 「M_QUIT」プロジェクトメソッド
$vlProcessID:=New process("DO_QUIT" ; 32*1024 ; "$DO_QUIT")
```

このメソッドには、ある仕掛けがあります。**QUIT 4D**コマンドが呼び出されると、このコマンドは即座に有効になります。したがって、呼び出しが実行されているプロセスは、データベースが実際に終了されるまでの間は「停止モード」になります。このプロセスは、データ入力が発生するプロセスのうちのいずれかである可能性があるため、**QUIT 4D**コマンドへの呼び出しは、この目的にだけ開始されるローカルプロセス内で実行されます。「DO_QUIT」メソッドは、以下のようになります。

```
` 「DO_QUIT」プロジェクトメソッド
CONFIRM("終了してもいいですか？")
If(OK=1)
    WRITE JOURNAL("データベースの終了")
```

```
            QUIT 4D
                  `QUIT 4Dは即座に有効になるため、それ以降のコード行はすべて
                                                        実行されません。
                  `...
      End if
```

▼最後に、すべての開いているユーザプロセスに対して「ただちに終了するように」と通知する「On Exit」データベースメソッドは、以下のようになります。このメソッドは、<>vbQuit4Dを**True**に設定し、データ入力を実行しているユーザプロセスに、プロセス間メッセージを送信します。

```
      `「On Exit」データベースメソッド
      <>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 JOURNAL("セッションのクローズ")
```

注："ML_..."または"M_..."で始まる名前を持つプロセスは、「新規プロセス開始」プロパティを選択したメニューによって開始されます。この例では、そのようなプロセスは、「レコード追加」メニューが選択された時に開始されたプロセスです。

「(**Current time**-$vhStart)>=!!00:01:00!!」という判定式により、データベースメソッドは「他のプロセスを待っている」状態を終了します。他のプロセスがただちに反応しない場合には、ループを繰り返します。

▼ 次に示しているのは、データベースによって作成されたジャーナルファイルの代表的な例です。

```
97/8/21  15:47:25  1  ユーザ/カスタムメニュープロセス  セッションのオープン
97/8/21  15:55:43  5  ML_1                          テーブル1：レコード追加#23
97/8/21  15:55:46  5  ML_1                          テーブル1：レコード追加#24
97/8/21  15:55:54  6  $DO_QUIT                      データベースの終了
97/8/21  15:55:58  7  $xx                           セッションのクローズ
```

注：$xxという名前は、「On Exit」データベースメソッドを実行するために4Dが開始したローカルプロセスの名前です。

ADD RECORD

ADD RECORD({テーブル} {; }{*})

引数	タイプ		説明
テーブル	テーブル	→	データ入力を実行するテーブルまたは省略した場合、デフォルトテーブル
*	*	→	スクロールバーの表示しない

説明

ADD RECORDコマンドは、データベースの<テーブル>に対して（引数<テーブル>を省略した場合、デフォルトテーブル）新しいレコードを追加します。

ADD RECORDコマンドは、新しいレコードを作成し、それをカレントプロセスのカレントレコードとして設定し、カレント入力フォームを表示します。「カスタム」モードにおいて、ユーザが新しいレコードを受け入れると、新しいレコードがカレントセレクションにある唯一のレコードになります。

次の図は、データ入力に使用する一般的なフォームです。



このフォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションのアスタリスク（*）を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

ADD RECORDコマンドは、ユーザがレコードを保存するか、あるいは取り消すまでフォームを表示します。ユーザが複数のレコードを追加する場合は、新しいレコードごとに1回ずつコマンドを実行しなければなりません。

ユーザが「登録」ボタンをクリック、または“enter”キーを押した場合、あるいは**ACCEPT**コマンドが実行されると、レコードが保存されます。

ユーザが「キャンセル」ボタンをクリックするか、キャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合、または**CANCEL**コマンドが実行されると、レコードは保存されません（キャンセルされます）。

ADD RECORDコマンドの実行後、システム変数OKにはレコードが保存されると1が、キャンセルされると0が代入されます。

注：キャンセルされた場合でも、レコードはメモリ上に残されたままとなります。カレントレコードポインタが変更される前にSAVE RECORDコマンドを実行すれば、レコードは保存されます。

例題

1. 次の例は、データベースに新しいレコードを追加する際によく使われるループです。

```
INPUT FORM([顧客] ; "顧客入力") ` 顧客テーブルの入力フォームをセット
Repeat          ` ユーザにキャンセルされるまでループ
      ADD RECORD([顧客];*)      ` 顧客テーブルへ新しいレコードを追加
Until(OK=0)  ` ユーザがキャンセルし、OK=0になるまで
```

2. 次の例は、顧客データを検索し、その検索結果により、2つのステートメントうちの1つを実行します。顧客が全く見つからない場合は、**ADD RECORD**コマンドで新しい顧客を追加します。少なくとも1つの顧客が見つかった場合には、最初のレコードが表示され、**MODIFY RECORD**コマンドでこれらのレコードを修正することができます。

```
READ WRITE([顧客])
INPUT FORM([顧客] ; "入力1")       ` 入力フォームを設定
v番号:=Num(Request("顧客番号を入力してください。"))        ` 顧客番号を獲得
If(OK=1)
      QUERY([顧客] ; [顧客]番号=v番号) ` 顧客を検索する
      If(Records in selection([顧客])=0)    ` 顧客が見つからない場合
            ADD RECORD([顧客])        ` 新規の顧客を追加
```

```
            Else
                If(Not(Locked([顧客])))
                    MODIFY RECORD([顧客]) ` レコードを修正
                    UNLOAD RECORD([顧客])
                Else
                    ALERT("このレコードは現在使用中です")
                End if
            End if
        End if
```

参照

ACCEPT、CANCEL、CREATE RECORD、MODIFY RECORD、SAVE RECORD

MODIFY RECORD

MODIFY RECORD({テーブル} {; }{*})

引数	タイプ		説明
テーブル	テーブル	→	データ入力を実行するテーブルまたは省略した場合、デフォルトテーブル
*	*	→	スクロールバーの表示しない

説明

MODIFY RECORDコマンドは、＜テーブル＞（引数＜テーブル＞を省略した場合、デフォルトテーブル）のカレントレコードを修正するために使用します。**MODIFY RECORD**コマンドは、カレントプロセスにレコードがまだロードされていない場合にレコードをロードし、カレント入力フォームにレコードを表示します。カレントレコードがなければ、**MODIFY RECORD**コマンドは何も行いません。また、**MODIFY RECORD**コマンドはカレントセレクションに影響を与えません。

フォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションのアスタリスク（*）を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

MODIFY RECORDコマンドを使用するには、必ずカレントレコードは読み込み／書き込み可能であり、またレコードはロックされていてはいけません。入力フォームにカレントセレクションのレコード内を移動するためのボタンを設定すると、ユーザはこれらのボタンをクリックしてレコードを修正したり他のレコードへ移動することができます。

ユーザが「登録」ボタンをクリック、または“enter”キーを押した場合、あるいは**ACCEPT**コマンドが実行されると、レコードが保存されます。

ユーザが「キャンセル」ボタンをクリックするか、キャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合、または**CANCEL**コマンドが実行されると、レコードは保存されません（キャンセルされます）。

MODIFY RECORDコマンドの実行後、システム変数OKにはレコードが保存されると1が、キャンセルされると0が代入されます。

注：キャンセルされても、レコードはメモリ上に残されます。カレントレコードポインタが変更される前にSAVE RECORDコマンドを実行すれば、レコードは保存されます。

MODIFY RECORDコマンドを使用しても、レコードのデータが全く変更されなければ、レコードが修正されたとは解釈されず、レコードが受け付けられても再保存されることはありません。また、変数の変更、チェックボックスのチェック、ラジオボタンの選択も修正にはなりません。データ入力またはメソッドによりフィールドのデータが変更された場合にのみレコードが保存されます。

例題

ADD RECORDコマンドの例題を参照してください。

参照

ADD RECORD、Locked、Modified record、READ WRITE、UNLOAD RECORD

ADD SUBRECORD

ADD SUBRECORD(サブテーブル ; フォーム ; {*})

引数	タイプ		説明
サブテーブル	サブテーブル名	→	データ入力を実行するサブテーブル
フォーム	文字列	→	データ入力に使用するフォーム
*	*		スクロールバーの表示しない

説明

ADD SUBRECORDコマンドは、＜フォーム＞を使用して＜サブテーブル＞に対して新しいサブレコードを追加します。**ADD SUBRECORD**コマンドは、メモリ上に新しいサブレコードを作成して、それをカレントサブレコードとし、＜フォーム＞を表示します。必ず、カレント親レコードが存在しなければなりません。プロセスのカレント親レコードが存在しない場合には、**ADD SUBRECORD**コマンドは何も行いません。＜フォーム＞は、必ず＜サブテーブル＞に属するフォームでなければなりません。

ユーザが「登録」ボタンをクリックしたり、テンキー上の“enter”キーを押したり、または**ACCEPT**コマンドが実行された場合は、サブレコードはメモリに格納されます。サブレコードの追加後、このサブレコードを保存するために親レコードを明示的に保存しなければなりません。

サブレコードは、「キャンセル」ボタンがクリックされたり、キャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合、または**CANCEL**コマンドが実行された場合には、サブレコードはメモリに格納されません。

ADD SUBRECORDコマンドの実行後、システム変数OKにはサブレコードが追加されると1が、キャンセルされると0が代入されます。

フォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションのアスタリスク（*）を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

例題

▼以下の例は、プロジェクトメソッドの一部です。これは、[社員]テーブルの[社員]子供サブテーブルにサブレコードを追加します。複数の子供に関するデータは、[社員]子供というサブテーブルに保存します。サブレコード（[社員]子供）の内容を保存するには、親レコード（[社員]）を保存する必要があるという点に注意してください。

```
ADD SUBRECORD([社員]子供 ; "子供追加")
If(OK=1)        ` レコード登録されているか？
```

```
            SAVE RECORDS([社員])    ` [社員]レコード保存
        End if
```

参照

ACCEPT、CANCEL、MODIFY SUBRECORD、SAVE RECORD

MODIFY SUBRECORD

MODIFY SUBRECORD(サブテーブル ; フォーム ; {*})

引数	タイプ		説明
サブテーブル	サブテーブル名	→	データ入力を実行するサブテーブル
フォーム	文字列	→	データ入力に使用するフォーム
*	*		スクロールバーの表示しない

説明

MODIFY SUBRECORDコマンドは、＜フォーム＞を使用して、修正するカレントサブレコードを表示します。フォームは必ず＜サブテーブル＞に属していなければなりません。

親テーブルのカレントレコードは必ず存在している必要があります。プロセスのカレント親レコードが存在しない場合には、**MODIFY SUBRECORD**コマンドは何も行いません。さらに、カレントサブレコードが存在しない場合にも、**MODIFY SUBRECORD**コマンドは何も行いません。

ユーザが「登録」ボタンをクリックしたり、テンキー上の“enter”キーを押したり、または**ACCEPT**コマンドが実行された場合は、サブレコードはメモリに格納されます。サブレコードの修正後、このサブレコードを保存するために親レコードを明示的に保存しなければなりません。

サブレコードは、「キャンセル」ボタンがクリックされたり、キャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合、または**CANCEL**コマンドが実行された場合には、サブレコードはメモリに格納されません。

MODIFY SUBRECORDコマンドの実行後、システム変数OKにはサブレコードの変更が行われると1が、キャンセルされると0が代入されます。

フォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションのアスタリスク（*）を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

参照

ACCEPT、ADD SUBRECORD、CANCEL、SAVE RECORD

DIALOG

DIALOG({テーブル ; }フォーム)

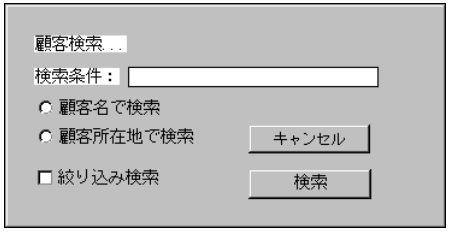
引数	タイプ		説明
テーブル	テーブル名	→	フォームの属するテーブルまたは省略した場合、デフォルトテーブル
フォーム	文字列	→	ダイアログとして表示するフォーム

説明

DIALOGコマンドは、ユーザに対して<フォーム>を提示します。このコマンドは、処理を実行する際の任意選択項目等のように、変数を用いてユーザから情報を取得したり、あるいはユーザに情報を交換するために使用します。

Open window関数で開いたモーダルウインドウ（形式的ウインドウ）にフォームを表示するのが最も一般的な使用方法です。

一般的なダイアログの例を次に示します。



表示したり取得しなければならない情報がより複雑で、**ALERT**コマンドや**CONFIRM**コマンド、**Request**関数で処理できない場合は、これらのコマンドの代わりに**DIALOG**コマンドを使用してください。

注：4th Dimensionの「環境設定」にあるオプションを用いて（「互換性」ページ）、ダイアログボックスのフィールドへのデータ入力を禁止することができます（これにより、変数へのデータ入力のみに制限する）。この制約は、以前のバージョンの4th Dimensionでの動作に相当します。

DIALOGコマンドは、**ADD RECORD**コマンドや**MODIFY RECORD**コマンドと異なり、カレントの入力フォームを使用しません。必ず引数<フォーム>に使用するフォームを指定してください。また、ボタンを省略するとデフォルトボタンパネルは表示されません。この場合は、Escape（Windows）キーまたはEsc（Mac OS）キーを使用してのみフォームを終了することができます。

ユーザが「登録」ボタンをクリックしたり、テンキー上の“enter”キーを押したり、または**ACCEPT**コマンドが実行された場合は、ダイアログが受け付けられます。

「キャンセル」ボタンがクリックされたり、キャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合、または**CANCEL**コマンドが実行された場合には、ダイアログはキャンセルされます。

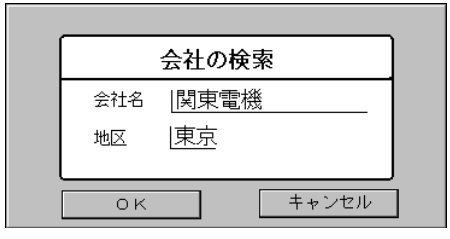
DIALOGコマンドの実行後、システム変数OKにはダイアログが受け付けられると1が、キャンセルされると0が代入されます。

例題

▼次の例は、検索条件を入力するための**DIALOG**コマンドの使用方法を示しています。変数“v名称”と“v地区”を含むカスタムフォームが表示され、ユーザは検索条件を入力することができます。

```
OPEN WINDOW(10 ; 40 ; 370 ; 220)        ` モーダルウインドウを開く
DIALOG([会社] ; "検索フォーム")  ` 検索ダイアログを表示する
CLOSE WINDOW    ` ウインドウを閉じる
If(OK=1)        ` ダイアログが受け入れられると...
        QUERY([会社] ; [会社]会社名=v名称 ; *)
        QUERY([会社] ; & ; [会社]地区=v地区)
End if
```

以下の図に、表示されたカスタムダイアログボックスを示します。



参照

ACCEPT、ADD RECORD、CANCEL、Open window

Modified

Modified(フィールド) → ブール

引数	タイプ		説明
フィールド	フィールド	→	テストするフィールド
戻り値	ブール	←	フィールドが変更された場合にTrue、それ以外はFalse

説明

Modified関数は、データ入力時にプログラムから＜フィールド＞に値が割り当てられたり、または値が編集された場合に、“True（真）”を返します。**Modified**関数はフォームメソッドのみ（またはフォームメソッドから呼ばれたサブルーチン）で使用されなければいけません。

データ入力時には、ユーザがフィールドを編集した後（元の値が変更されたかどうかに関わらず）、別のフィールドへ移動してそのフィールドを離れた場合にフィールドが修正されたものと認識されます。“tab”キーでフィールドを移動しただけでは、**Modified**関数は“True（真）”にならない点に注意してください。**Modified**関数を“True（真）”にするためには、必ずフィールドを編集しなければなりません。

メソッドの実行時には、フィールドに値が割り当てられると（異なる値かどうかに関係なく）、フィールドが編集されたものと解釈されます。

いずれの場合でも、フィールドの値が実際に変更されたかどうかを調べるには、**Old**関数を使用します。

注：Modified関数はあらゆるタイプのフィールドに対して適用できますが、この関数をOld関数と組み合わせて使用する場合には、Old関数の制約に注意してください。詳細についてはOld関数の説明を参照してください。

Modified関数は、組み込みエリアのフィールドで使用することはできません。

データ入力時には、**Modified**関数をフォームメソッドで使用するよりも、オブジェクトメソッドで処理を実行する方が簡単です。フィールドが修正される度に、**On Data Change**イベントでオブジェクトメソッドが送信されるため、フォームメソッドで**Modified**関数を使用したのと同じ意味を持つからです。

注：処理を正常に実行するため、Modified関数はフォームメソッドまたは、フォームメソッドから呼び出されるメソッド内でのみ使用するほうが良いでしょう。

例題

1. 次の例は、フィールドの“[注文]数量”または“[注文]価格”が変更されたかどうかを判定します。どちらかが変更されると、“[注文]合計”フィールドを計算し直します。

```
` ユーザがいずれかのフィールドを変更した場合
If((Modified([注文]数量)) |(Modified([注文]価格)))
[注文]合計:=[注文]数量 * [注文]価格
End if
```

数量フィールドと価格フィールドのオブジェクトメソッドに2行目を使用しても同じことを実行できます。

2. テーブル[任意のテーブル]のレコードを選択し、次にフィールド[任意のテーブル]重要が修正される可能性がある複数のサブルーチンを呼び出しますが、レコードの保存は行いません。メインのメソッドの終わりで、**Modified**関数を使用してレコードを保存する必要があるかどうかを調べます。

```
` レコードがカレントレコードとして選択される
` 次にサブルーチンを使い、処理を実行する
（何か処理をする）
（何か他の処理をする）
` ...
` レコードを保存する必要があるかどうかを調べるためにフィールドをテストする
If(Modified([任意のテーブル]重要))
      SAVE RECORD([任意のテーブル])
End if
```

参照

Old

Old

Old(フィールド) → 文字列、数値、日付、ブール、時間

引数	タイプ		説明
フィールド	フィールド	→	古い内容を得るフィールド
戻り値	表現式	←	元のフィールドの値

説明

Old関数は、プログラムから値が割り当てられたり、データ入力で修正される前に<フィールド>に納められていた値を返します。

ユーザがテーブルのカレントレコードを変更するたびに、4Dは新しいカレントレコードがメモリーにロードされた時点での複製“イメージ”をメモリ上に作成し、管理します(最適化の理由から、4Dはテキスト、ピクチャ、BLOBタイプのフィールドを無視します)。レコードを修正する際には、レコードの実際のイメージを使って作業を実行するのであって、この複製イメージを使うわけではありません。カレントレコードが再度変更されると、このイメージは破棄されます。

注：Modified関数はいつもPUSH RECORD関数とPOP RECORDコマンドの実行の後は“True（真）”を返します。

Old関数はこの複製イメージの値を返します。すなわち、既存のレコードに対しては、ディスク上に保存されているフィールドの値を返すということです。新しく作成されたレコードの場合には、**Old**関数はそのフィールドタイプに応じた空の値を返します。例えば、<フィールド>が文字フィールドの場合は空の文字列（ヌル）を返します。また、<フィールド>が数値フィールドの場合はゼロ（0）を返します。日付フィールドの場合は!00.00.00!を返します。時間フィールドの場合は!!00:00:00!!を返します。ブールフィールドの場合に、**Old**関数は“False（偽）”を返します。

Old関数は、<フィールド>がメソッドによって修正された場合にも、データ入力時にユーザによって修正された場合にも関係なく機能します。

Old関数は、テキストフィールド、ピクチャフィールド、BLOBフィールドに関しては適用されません。その他のフィールドタイプに関しては、サブフィールドも含めて適用されますが、サブテーブルのフィールドそのものに対して適用しても意味がありません。

フィールドの元の値を復元するには、**Old**関数から返された値を割り当てます。

参照

Modified

Current date

Current date {(*)} → 日付

引数	タイプ		説明
*	*	→	サーバから現在の日付を返す
戻り値	日付	←	現在の日付

説明

Current date関数は、使用しているオペレーションシステム（OS）のシステムクロックの日付を現在の日付として返します。

4D Server：4D Clientマシン上でこの関数を実行する場合に引数＜*＞を指定すると、サーバの現在日付が返されます。

例題

1. 以下の例は、現在の日付を警告ボックスに表示します。

```
ALERT("今日は、"+String(Current date)+"です。")
```

2. 国際市場に向けたアプリケーションを作成する場合に、使用している4Dのバージョンが日付フォーマット“MM/DD/YYYY（US版)”や“DD/MM/YYYY（フランス版)”に対応するかどうかを知る必要があるとします。これはデータ入力フィールドのカスタマイズのために知っておくと便利な情報です。

以下のプロジェクトメソッドにより目的の処理を行うことができます。

` Sys date formatグローバル関数
` Sys date format -> 文字列
` Sys date format -> デフォルトの4Dデータフォーマット
C_STRING(31;$0;$vsDate;$vsMDY;$vsMonth;$vsDay;$vsYear)
C_LONGINT($1;$vlPos)
C_DATE($vdDate)

` 月、日、年の値がすべて異なる日付の値を取得

```
$vdDate:=Current date
Repeat
	$vsMonth:=String(Month of($vdDate))
	$vsDay:=String(Day of($vdDate))
	$vsYear:=String(Year o($vdDate)%100)
	If(($vsMonth=$vsDay) |($vsMonth=$vsYear) |($vsDay=$vsYear))
		vOK:=0
		$vdDate:=$vdDate+1
	Else
		vOK:=1
	End if
Until(vOK=1)
$0:="" ` 関数の結果を初期化
$vsDate:=String($vdDate)
$vlPos:=Position("/";$vsDate) ` 文字列“../../..”から最初のセパレータ“/”を探す
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` 日付から最初の桁を取り出す
` 最初の区切り文字“/”とともに最初の桁を取り除く
$vsDate:=Substring($vsDate;$vlPos+1)
Case of
	¥($vsMDY=$vsMonth) ` 月を表わす桁
		$0:="MM"
	¥($vsMDY=$vsDay) ` 日を表わす桁
		$0:="DD"
	¥($vsMDY=$vsYear) ` 年を表わす桁
		$0:="YYYY"
End case
$0:=$0+"/" ` 関数の結果の組み立て開始
$vlPos:=Position("/";$vsDate) ` 文字列“../..”から2番目のセパレータを探す
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` 日付から以下の桁を取り出す
` 文字列を日付の最後の桁まで減らす
$vsDate:=Substring($vsDate;$vlPos+1)
Case of
	¥($vsMDY=$vsMonth) ` 月を表わす桁
		$0:=$0+"MM"
	¥($vsMDY=$vsDay) ` 日を表わす桁
		$0:=$0+"DD"
	¥($vsMDY=$vsYear) ` 年を表わす桁
		$0:=$0+"YYYY"
End case
$0:=$0+"/" ` 関数の結果の組み立てを続ける
Case of
```

```
        ¥($vsDate=$vsMonth) ` 月を表わす桁
                $0:=$0+"MM"
        ¥($vsDate=$vsDay) ` 日を表わす桁
                $0:=$0+"DD"
        ¥($vsDate=$vsYear) ` 年を表わす桁
                $0:=$0+"YYYY"
    End case
    ` この時点で、$0 は“MM/DD/YYYY”または“DD/MM/YYYY”または…
```

参照

Date Operators、Day of、Month of、Year of

Day of

Day of(日付) → 数値

引数	タイプ		説明
日付	日付	→	日を取り出す日付
戻り値	数値	←	日

説明

Day of関数は、＜日付＞から日を取り出して返します。

注：Day of関数は1から31までの値を返します。日付から曜日を取得するには、Day number関数を使用します。

例題

▼ 以下の例は、**Day of**関数の使用方法を示しています。結果は変数“v結果”に代入されます。変数“v結果”に代入される内容についての説明がコメントされています。

```
v結果:=Day of(!97.07.21!)    ` v結果に21を代入
v結果:=Day of(Current date)        ` v結果に現在の日付の日を代入
```

参照

Day number、Month of、Year of

Month of

Month of(日付) → 数値

引数	タイプ		説明
日付	日付	→	月を取り出す日付
戻り値	数値	←	月を表わす数値

説明

Month of関数は、＜日付＞から月を取り出して返します。

4th Dimensionでは以下の定数があらかじめ定義されています。

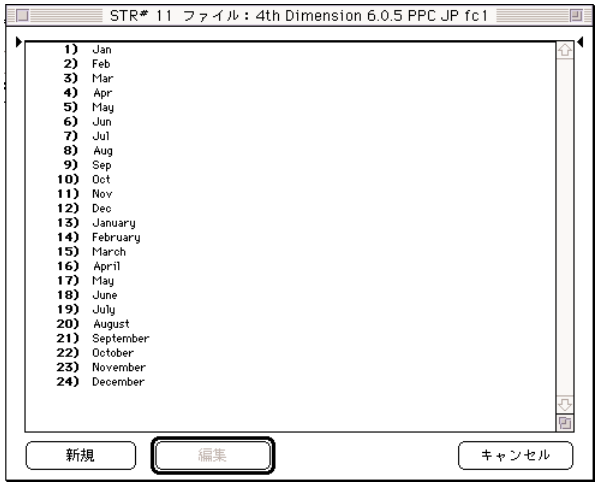
定数	タイプ	数値
January	倍長整数	1
February	倍長整数	2
March	倍長整数	3
April	倍長整数	4
May	倍長整数	5
June	倍長整数	6
July	倍長整数	7
August	倍長整数	8
September	倍長整数	9
October	倍長整数	10
November	倍長整数	11
December	倍長整数	12

例題

1. 以下の例は、**Month of**関数の使用方法を示しています。結果は変数“v結果”に代入されます。変数“v結果”に代入される内容についての説明がコメントされています。

```
v結果:=Month of(!97.07.21!)          ` v結果に7を代入
v結果:=Month of(Current date)        ` v結果に現在の日付の月を代入
```

2. 4th Dimensionのリソース“STR#”ID=11には、月の名称が納められています。



以下のプロジェクトメソッドは日付の月の名称を返します。

```
` Month name of プロジェクトメソッド
` Month name of(Date) -> 文字列
` Month name of(Date) -> 月の名称
$0:=Get indexed string(11;12+Month of($1))
```

以下のプロジェクトメソッドは日付の月の略称を返します。

```
` Month abbr of プロジェクトメソッド
` Month abbr of(Date) -> 文字列
` Month abbr of(Date) -> 月の名称
$0:=Get indexed string(11;Month of($1))
```

参照

Day of、Year of

Year of

Year of(日付) → 数値

引数	タイプ		説明
日付	日付	→	年を取り出す日付
戻り値	数値	←	日付の年を表わす数値

説明

Year of関数は、＜日付＞から年を取り出して返します。

例題

▼ 以下の例は、**Year of**関数の使用方法を示しています。結果は変数“v結果”に代入されます。変数“v結果”に代入される内容についての説明がコメントされています。

```
v結果:=Year of(!97.07.21!)  ` v結果に1997を代入
v結果:=Year of(!2092.07.21!)  ` v結果に2092を代入
v結果:=Year of(Current date)  ` v結果に現在の日付の年を代入
```

参照

Day of、Month of

Day number

Day number(日付) → 数値

引数	タイプ		説明
日付	日付	→	曜日に対応する数値を得る日付
戻り値	数値	←	曜日を表わす番号

説明

Day number関数は、＜日付＞に対応するの曜日を数値として返します。

注：＜日付＞が空の場合に、Day number関数は2を返します。

4th Dimensionでは以下の定数があらかじめ定義されています。

定数	タイプ	数値
Sunday	倍長整数	1
Monday	倍長整数	2
Tuesday	倍長整数	3
wednesday	倍長整数	4
Thursday	倍長整数	5
Friday	倍長整数	6
Saturday	倍長整数	7

注：Day number関数は1から7までの値を返します。日付から日を取り出すには、Day of関数を使用します。

例題

▼ 以下の例は、現在の日付を文字列にして返す関数です。

```
$曜日:=Day number(Current date)`$曜日に現在の曜日に対応する数値を代入
Case of
    ¥($曜日=1)
        $0:="Sunday"
    ¥($曜日=2)
        $0:="Monday"
    ¥($曜日=3)
        $0:="Tuesday"
    ¥($曜日=4)
        $0:="wednesday"
    ¥($曜日=5)
        $0:="Thursday"
```

```
        ¥($曜日=6)
                $0:="Friday"
        ¥($曜日=7)
                $0:="Saturday"
    End case
```

参照

Day of

Add to date

Add to date(日付 ; 年 ; 月 ; 日) → 日付

引数	タイプ		説明
日付	日付	→	日、月、年を追加する日付
年	数値	→	日付に追加する年
月	数値	→	日付に追加する月
日	数値	→	日付に追加する日
戻り値	日付	←	返される日付

説明

Add to date関数は、引数＜日付＞に＜年＞、＜月＞、＜日＞を追加して、その結果の日付を返します。

通常、任意の日付に日を追加する場合、日付演算子を使用しますが、**Add to date**関数は（“+”日付加算演算子を使用する場合のように）1ケ月の日数やうるう年の取り扱い方法を気にすることなく月や年をすぐに追加することができます。

例題

▼ 以下の例は、翌年の同じ日を計算します。

$vdInOneYear:=**Add to date**(**Current date** ; 1 ; 0 ; 0)

▼ 以下の例は、翌月の同じ日を計算します。

$vdNextMonth:=**Add to date**(**Current date** ; 0 ; 1 ; 0)

▼ 以下の例は、「$vdTomorrow:=Current date+1」と同じ結果を示します。

$vdTomorrow:=**Add to date**(**Current date** ; 0 ; 0 ; 1)

参照

Date Operators

Date

Date(日付文字列) → 日付

引数	タイプ		説明
日付文字列	文字列	→	文字列日付に変換する文字列
戻り値	日付	←	日付

説明

Date関数は＜日付文字列＞を解釈し、日付に変換して返します。

＜日付文字列＞は通常の日付のフォーマットの規則に従って、YY.MM.DD（年.月.日）の順に指定します。年は2桁でも4桁でも構いません。また、月は1桁か2桁の数値です。年に2桁の数値を指定すると、**Date**関数は19を前に付け加えます（**SET DEFAULT CENTURY**コマンドでこのデフォルトを変更していない場合）。セパレータには、スラッシュ（/）、スペース、ピリオド（.）、コンマ（,）の文字を使用します。

Date関数は＜日付文字列＞が有効な日付であるかどうかのチェックは行いません。無効な日付（“94/13/35”のような）が渡されると、**Date**関数は無効な日付を返します。しかし、＜日付文字列＞が日付として解釈されない場合（例えば“94/aa/12”）、空の日付の値が返されます。

＜日付文字列＞が有効な日付かどうかを調べるのはユーザの判断次第です。

例題

1. 以下の例は、「リクエスト」ダイアログボックスを使用して、ユーザから日付の入力を得ます。入力された文字列を日付に変換し、変数“ReqDate”に格納します。

```
ReqDate:=Date(Request("日付を入力してください:";String(Current date)))
If(OK=1)
        ` ReqDateに格納された日付を使った処理
End if
```

1. 以下の例は、文字列“97.06.10”を日付に変換します。

```
vDate:=Date("97.06.10")
```

Current time

Current time {(*)} → 時間

引数	タイプ		説明
*	*	→	サーバから現在の時間を返す
戻り値	時間	←	現在の時間

説明

Current time関数は、システムクロックの現在の時間を返します。

現在の時間は、常に!!00:00:00!! から!!23:59:59!!の間の値です。**Current time**関数から返される時間を文字列に変換する場合は、**String**関数または**Time string**関数を使用します。

4D Server：4D Clientマシン上でこの関数を実行する場合に引数＜*＞を指定すると、サーバの現在時間が返されます。

例題

1. 以下の例は、処理時間の計測方法を示しています。“Long Operation”は、計測の対象となるプロジェクトメソッドです。

```
$開始時間:=Current time    ` 開始時間を保存
Long operation             ` 計測対象の操作
ALERT("処理の所要時間は" + String(Current time - $開始時間))` 所要時間を表示
```

2. 以下の例は、現在の時間より時、分、秒を取り出します。

```
$現在時間:=Current time
ALERT("現在の時："+String($現在時間//3600))
ALERT("現在の分："+String(($現在時間//60)%60))
ALERT("現在の秒："+String($現在時間%60))
```

参照

Milliseconds、String、Tickcount、Time Operators

Time string

Time string(秒) → 文字列

引数	タイプ		説明
秒	数値	→	午前0時からの秒数
戻り値	文字列	←	24時間形式の時間を表わす文字列

説明

Time string関数は、＜秒＞を24時間制で表現する文字列に変換して返します。

文字列は“HH:MM:SS”のフォーマットです。

1日の秒数は86,400秒ですが、この値を越えると、**Time string**関数は、時間、分、秒を加算し続けます。例えば、**Time string**（86401）は、文字列“24:00:01”を返します。

例題

▼以下の例は、“46800秒は13:00:00です。”というメッセージを警告ボックスに表示します。

```
ALERT("46800秒は" + Time string(46800) + "です。")
```

参照

String、Time

Time

Time(時間文字列) → 時間

引数	タイプ		説明
時間文字列	文字列	→	時間を表す文字列
戻り値	時間	←	時間文字列で指定した時間

説明

Time関数は、＜時間文字列＞に指定した時間に相当する時間式を返します。

＜時間文字列＞には“HH:MM:SS”の形式で、24時間以内の値を指定します。

例題

▼以下の例は、警告ボックスに“1:00P.M.=13時0分”というメッセージを表示します。

```
ALERT("1:00 P.M. = " + String(Time("13:00:00") ; 4))
```

参照

String、Time string

Tickcount

Tickcount → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	コンピュータが起動されてから経過したTick数（1/60秒）

説明

Tickcount関数は、コンピュータが起動されてから経過したTick数（60分の1秒）を返します。

注：Tickcount関数は、倍長整数型の値を戻します。

例題

▼ **Milliseconds**関数の例を参照してください。

参照

Current time、Milliseconds

Milliseconds

Milliseconds → 倍長整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	倍長整数	←	コンピュータが起動されてから経過したミリ秒数

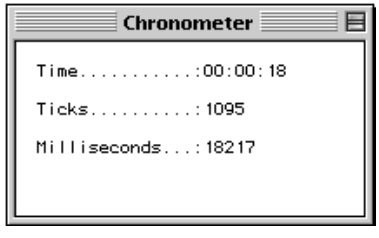
説明

Milliseconds関数は、コンピュータが起動されてから経過したミリ秒（1000分の1秒）を返します。

例題

▼ 以下の例は、"Chronometer"ウインドウを1分間表示します。

```
Open window(100 ; 100 ; 300 ; 200 ; 0 ; "Chronometer")
$vhTimeStart:=Current time
$vlTicksStart:=Tickcount
$vrMillisecondsStart:=Milliseconds
Repeat
    GOTO XY(2;1)
    MESSAGE("Time.....:"+String(Current time -$vhTimeStart))
    GOTO XY(2;3)
    MESSAGE("Ticks.....:"+String(Tickcount -$vlTicksStart))
    GOTO XY(2;5)
    MESSAGE("Milliseconds...:"+String(Milliseconds -$vrMillisecondsStart))
Until((Current time -$vhTimeStart) >=?00:01:00?)
CLOSE WINDOW
```



参照

Current time、Tickcount

SET DEFAULT CENTURY

SET DEFAULT CENTURY(世紀 { ; ピボット年度})

引数	タイプ		説明
世紀	数値	→	2桁年度の日付入力における デフォルトの世紀（マイナス1）
ピボット年度	数値	→	2桁年度の日付入力におけるピボット年度

説明

SET DEFAULT CENTURYコマンドは、2桁の年度で日付を入力する場合に4Dにより使用されるデフォルトの世紀を指定することができます。

年数が2桁の日付データの入力値の変換は、ピボット年度の値により決まります。

■ 年数がピボット年度と等しいかそれより大きい場合、現在のデフォルトの世紀が使われます。

■ 年数がピボット年度より小さい場合、次の世紀（現在のデフォルトから導かれる）が使われます。

デフォルトでは、4Dは世紀を20世紀にセットし、30がピボット年度となります。例えば、

■ 97.01.25は、1997年1月25日を意味します。

■ 30.01.25は、1930年1月25日を意味します。

■ 29.01.25は、2029年1月25日を意味します。

■ 07.01.25は、2007年1月25日を意味します。

この既定値を変更するには**SET DEFAULT CENTURY**コマンドを実行します。新しいデフォルトの世紀数だけ、または世紀数とピボット年度を渡すことで、コマンドは即座に反映されます。

新しいデフォルトにひとつ前の世紀数だけを渡した場合、年数が2桁の日付データの入力値は今世紀に変換されます。

SET DEFAULT CENTURY(20) `デフォルトの世紀を21世紀に切り替える

この場合、

■ 97.01.25は、2097年1月25日を意味します。

■ 07.01.25は、2007年1月25日を意味します。

また、オプションでピボット年度を指定できます。

例えば、ピボット年度が1995である以下のコードを呼び出すと、

SET DEFAULT CENTURY(19 ; 95)　　`入力した年度が95より小さい場合は
デフォルト世紀を21世紀に切り替える

この場合、

■ 97.01.25は、1997年1月25日を意味します。

■ 07.01.25は、2007年1月25日を意味します。

注：このコマンドは、日付が2桁年度で入力された場合にのみ4Dの日付解釈方法に影響を与えます。

4桁年度の日付入力では、

■ 1997.01.25は、1997年1月25日を意味します。

■ 2097.01.25は、2097年1月25日を意味します。

■ 1907.01.25は、1907年1月25日を意味します。

■ 2007.01.25は、2007年1月25日を意味します。

このコマンドは、データ入力にのみ影響を与えます。データの保存および計算等では無効です。

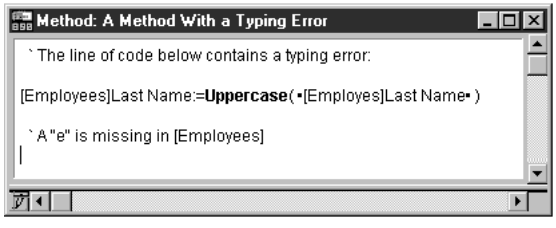
なぜデバッガを使用するか?

メソッドを開発し、テストする際には、エラーを発見し、修正することが重要です。

言語を使用する場合におこりうるエラーには、タイプミス、シンタックス（構文）エラー、環境エラー、設計やロジックのエラー、ランタイムエラー等、いくつかの種類があります。

タイプミス

タイプミスは、「メソッド」エディタによって検出され、黒丸（・）によって示されます。メッセージはメソッドウインドウの上のインフォメーションエリアに表示されます。次のウインドウは、タイプミスを表しています。メッセージはメソッドウインドウの上のインフォメーションエリアに表示されます。



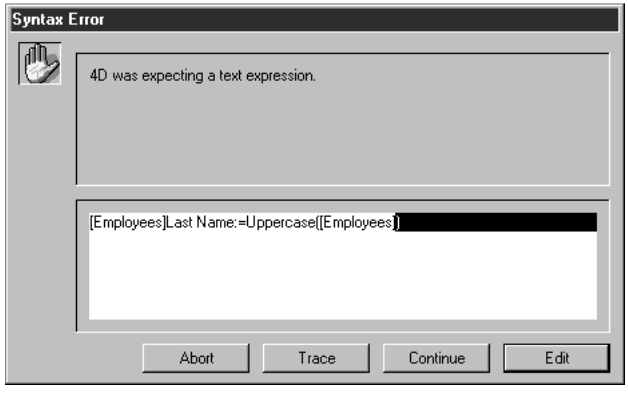
注：コメントは、このマニュアルで説明するために手動で入力しています。4Dがエラーの発生位置で挿入するのは（・）だけです。

このようになった場合には、タイプミスを修正し（テンキー上で）enterキーを押して、修正内容を確認します。

「メソッド」エディタに関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

シンタックス（構文）エラー

シンタックス（構文）エラーは、メソッドを実行する際に検出されます。シンタックスエラーが発生すると、次ページの図のように「シンタックスエラー」ウインドウが表示されます。

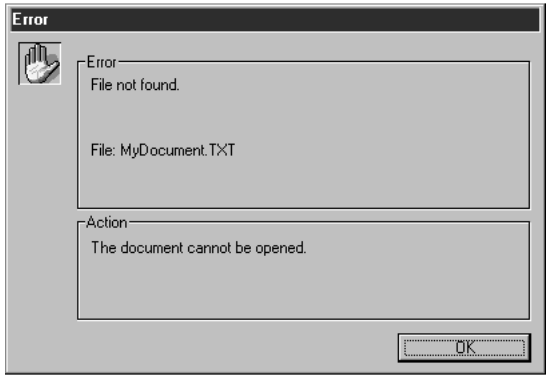


このウインドウでのエラーは、テキスト式が対象となっている**Uppercase**関数にテーブル名が渡されていることです。このウインドウとボタンの詳細については、後述の「「シンタックスエラー」ウインドウ」の節を参照してください。

環境エラー

まれに、配列やBLOBを作成するために十分なメモリがない場合があります。

ディスク上のドキュメントにアクセスした場合にドキュメントが存在しないか、または既に他のアプリケーションによって表示されている可能性があります。このような場合には次のように「エラー」ウインドウが表示され、エラーの内容と実行できなかった操作が表示されます。



このようなエラーは、作成したコードやコードの作成方法が直接の原因となって発生するわけではありません。ただ単に「思わしくない状況が発生した」ことが原因となって発生します。ほとんどの場合、これらのエラーは、**ON ERR CALL**コマンドを使用してインストールした「エラー検出」メソッドによって簡単に対応できます。詳細については、**ON ERR CALL**コマンドの説明個所を参照してください。

設計エラーとロジックエラー

これらは通常、発見することが最も難しいタイプのエラーです。検出するには、デバッガを使用します。

これまでに説明しているエラーは、タイプミスを除いて、「設計エラーとロジックエラー」という範疇に該当します。例は次の通りです。

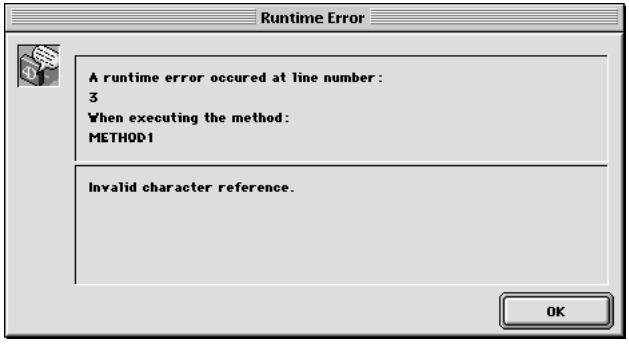
- まだ初期化されていない変数を用いようとしたため、シンタックスエラーが発生する場合があります。
- 引数の中で正しい値を取得していないサブルーチンによって名前が受け取られたドキュメントを開こうとしたため、環境エラーが発生する場合があります。この場合には、実際に中断が発生しているコードの一部分が、問題の原因とは異なっている場合があることに注意が必要です。

設計やロジックのエラーには、次のような場合もあります。

- **SAVE RECORD**コマンドを呼び出す際に、対象となるレコードがロックされているかどうかを最初にテストしなかったために、レコードが正しく更新されない。
- オプション引数を追加した状態がテストされていないため、メソッドが想定通りに動作しない。

ランタイムエラー

コンパイルモードでは、インタプリタモードでは決して見られない次のようなエラーが発生する場合があります。



これは、「**Position**関数が文字列の長さを超えるような文字にアクセスしようとしている」という意味です。

問題の原因を迅速に発見するには、メソッドの名前と行番号を記録し、ストラクチャファイルのインタプリタ版を再び開いて、メソッドの指定された行を確認します。

エラーが発生した場合の対応

エラーは発生するものです。相当行数（数百行程度）のプログラム・コードをエラーが発生しないように作成できることは、非常にまれです。むしろ、エラーに対応または修正するほうが普通です。

4th Dimensionはマルチタスク対応アプリケーションなので、ウインドウを切り替えるだけで、すばやくメソッドを編集し、実行することができます。そのたびメソッド全体を再実行する必要がないため、失敗やエラーを非常に迅速に修正できます。また、デバッガを使用すると、エラーを迅速に検出することができます。

エラー検出の際によくある初歩的な失敗は、「シンタックスエラー」ウインドウの「アボート」ボタンをクリックし、「メソッド」エディタに戻り、コードを表示して原因を確認しようとすることです。これは絶対に止めてください。常にデバッガを使用すれば、相当の時間と労力を節減することができます。

■ 予想していないシンタックスエラーが発生した場合には、デバッガを使用します。

■ 環境エラーが発生した場合には、デバッガを使用します。

■ その他どのようなタイプのエラーが発生した場合でも、デバッガを使用します。

ほとんどの場合、デバッガは、エラーが発生した理由を知るために必要な情報を表示します。この情報があれば、エラーの修正方法はわかります。

Tips：デバッガの使用法を数時間費やして学習し、実際に試しておけば、エラーの原因を究明しなければならなくなった時に何日分、何週間分もの時間と労力をかけずにすむことなります。

デバッガを使用するもう1つの理由は、コードの作成です。いつも以上に複雑なアルゴリズムを作成してしまう場合があります。達成感こそありますが、コーディングが正しいかどうか、テストする前でもまったく確かではありません。見当もつかないまま実行するのではなく、コードの最初で**TRACE**コマンドを使用します。その後、コードをステップごとに実行して動作を制御し、予想通りエラーが発生するかどうかを確認します。完全主義的な考え方では、このような方法は望ましくないかもしれませんが、現実主義的な方法が報われる場合もあります。いずれにしても、デバッガを使用してください。

一般的な結論

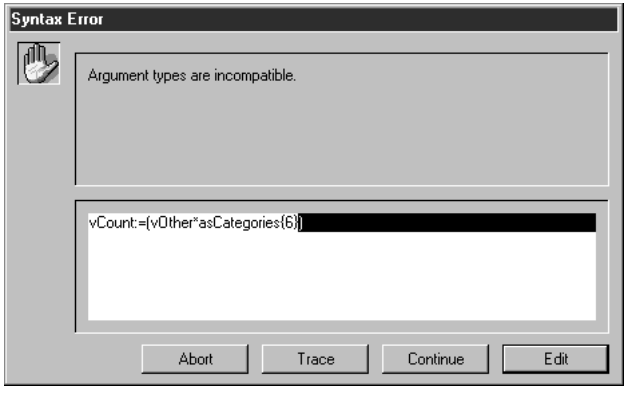
エラーが出たら、デバッガを使用する。

「シンタックスエラー」ウインドウ

「シンタックスエラー」ウインドウは、メソッドの実行が一時停止されると表示されます。メソッドは、次の2つのうちいずれかの理由で一時停止されます。

- それ以上メソッドの実行ができないような構文エラーがあるため、4th Dimensionが実行を一時停止します。
- メソッドが実行されている間に、「Alt+クリック（Windows）」または「option+クリック（Macintosh）」を押すことによってユーザーの割り込みを実行します。

次の図は、「シンタックスエラー」ウインドウです。



「シンタックスエラー」ウインドウ上部のテキストエリアには、エラーの内容が表示されています。下部のテキストエリアには、エラーが発生した時に実行されていた行が表示され、エラーが発生した部分が強調表示されています。

ウインドウの一番下の部分には、「アボート」、「トレース」、「続行」、「編集」の4つのボタンがあります。

- 「アボート」：メソッドは一時停止され、メソッドの実行を開始する前の状態に戻ります。イベントに対してフォームメソッドまたはオブジェクトメソッドが実行されている場合には、いずれの場合にも停止され、フォームに戻ります。メソッドが「カスタム」モードから実行されている場合には、「カスタム」モードに戻ります。

■「**トレース**」：「トレース／デバッガ」モードに入り、「デバッグ」ウインドウが表示されます。現在の行が一部実行されている場合には、「トレース」ボタンを数回クリックする必要も出てきます。行の実行が終了すれば、「デバッグ」ウインドウが表示されます。

■「**続行**」：実行は継続されます。エラーが発生した行は、その位置によっては一部実行される場合があります。その場合には、慎重に実行を継続してください。エラーが原因で、メソッドが正常に実行できない場合があるためです。通常は、継続しようとは思わないでしょう。**SET WINDOW TITLE**コマンド等のように、コードの残りの部分の実行やテストの妨げにならない単純な呼び出しでエラーが発生している場合には、「続行」ボタンをクリックしても構いません。このようにすれば、より重要なコーディングに集中し、些細なエラーは後で修正することができます。

■「**編集**」：すべてのメソッドの実行は一時停止されます。4th Dimensionは「デザイン」モードに切り替わります。エラーが発生したメソッドが「メソッド」エディタで表示され、エラーを修正することができます。このオプションは、エラーの内容がすぐ判明し、これ以上調査しなくても修正できる場合に使用します。

デバッガ

デバッガという用語は、バグという用語に由来しています。メソッド内のバグとは、除去すべき間違いのことです。エラーが発生した場合、またはメソッドの実行を監視する必要がある場合には、デバッガを使用します。デバッガでは、メソッドをステップごとにゆっくり確認してメソッドの情報を検証できるため、バグを発見するために役立ちます。このようにメソッドをステップごとに確認する処理は、トレースと呼ばれます。

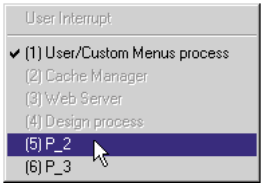
次のような方法を使用して、「デバッグ」ウインドウでメソッドを表示し、トレースすることができます。

■「シンタックスエラー」ウインドウで「トレース」ボタンをクリックした場合。

■ **TRACE**コマンドを使用した場合。

■ メソッドが実行されている間に、デバックボタンをクリックした場合（ユーザーモードで）。

■ メソッドが実行されている間に「Alt+Shift+右クリック（Windows）」または「control+option+command+クリック（Macintosh）」を実行した場合。この時、ポップアップメニューからのプロセスのトレースを選択します。

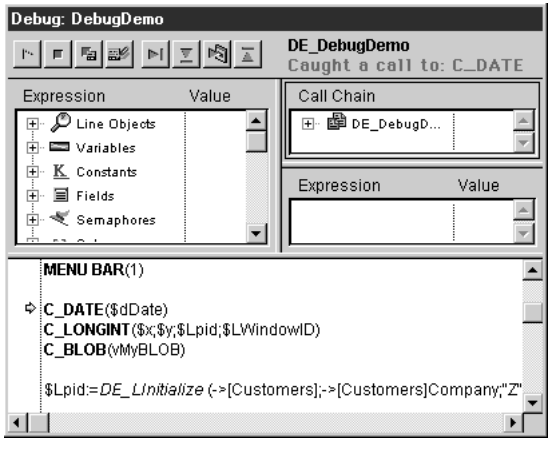


■ ランタイムエクスプローラのプロセスページを選択した後、トレースボタンをクリックした場合（後述の「表示されないプロセスまたはコードを実行していないプロセスのトレース」の節を参照してください）。

■ 「メソッドエディタ」ウインドウまたはランタイムエクスプローラのブレークおよびキャッチページで「ブレークポイント」を作成または編集した場合。

注：データベースにパスワードのシステムがある場合には、その構造へのアクセス特権を持つグループに所属する設計者やユーザだけが、メソッドをトレースできます。

次に示すのは、「デバッグ」ウインドウです。



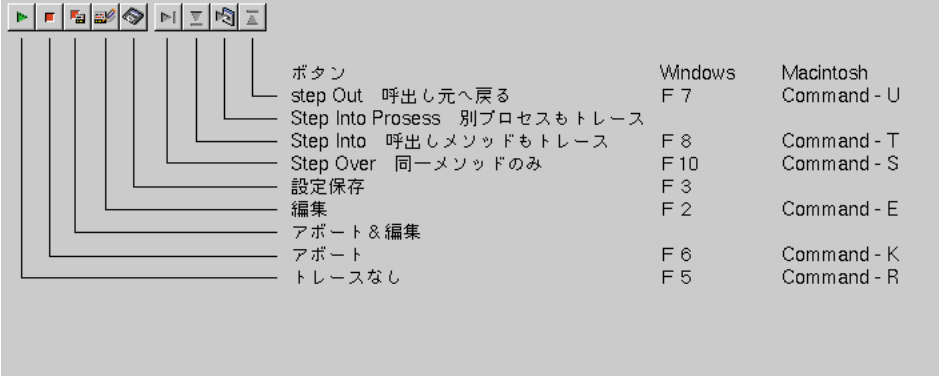
必要に応じて「デバッグ」ウインドウを移動、またはその内部にあるウインドウ枠のサイズを変更することができます。

新しいデバッグウインドウの表示には、同じセッション内で表示された最後のデバッグウインドウと同じ構成（ウインドウのサイズと位置、分割線の配置および「カスタム・ウォッチ」エリアの内容）を使用します。

4th Dimensionは、マルチタスク環境です。複数のユーザプロセスを実行した場合には、それぞれのプロセスを個別にトレースできます。プロセスそれぞれについて1つの「デバッグ」ウインドウを表示することができます。

「実行コントロール」ツールバーボタン

「デバッグ」ウインドウの上部にある「実行コントロール」ツールバーには、8個のボタンがあります。



「トレースなし」	F5 or command+r
「アボート」	F6 or command+k
「アボート＆編集」	
「編集」	F2 or command+e
「設定保存」	F3
「Step Over（同一メソッドのみ）」	F10 or command+s
「Step Into（呼び出しメソッドもトレース）」	F8 or command+t
「Step Into Prosess（別プロセスもトレース）」	
「Step Out（呼び出し元へ戻る）」	F7 or command+u

「トレースなし」ボタン

トレースが一時停止され、通常のメソッドの実行が再開されます。

注：Shift+F5またはShiftを押しながら「トレースなし」ボタンをクリックすると、実行が再開されます。この操作により、カレントプロセスでの次のTRACEコマンド呼び出しも無効になります。

「アボート」ボタン

メソッドは一時停止され、メソッドの実行を開始する前の状態に戻ります。イベントに対して実行しているフォームメソッドまたはオブジェクトメソッドをトレースしている場合には、いずれの場合にも停止され、フォームに戻ります。「カスタム」モードから実行しているメソッドをトレースしていた場合には、「カスタム」モードに戻ります。

「アボート&編集」ボタン

「アボート」ボタンががクリックされたかのように、メソッドは一時停止されます。さらに、必要に応じて、4th Dimensionは「デザイン」モードプロセスを開いて前面に表示し、「メソッドエディタ」ウインドウを開いて、「アボート&編集」ボタンがクリックされた時点で実行していたメソッドを表示します。

Tips：このボタンの使用は、コードにどのような変更が必要か、メソッドのテストを実行するには、その変更がいつ必要かがわかっている場合に行ってください。変更が完了すると、メソッドを再実行します。

「編集」ボタン

「編集」ボタンをクリックすると、「アボート&編集」ボタンをクリックした場合と同じ動作が実行されますが、現在の実行をアボートしません。メソッドの実行はその時点で一時停止されます。必要に応じて、4th Dimensionは「デザイン」モードプロセスを開いて前面に表示し、「編集」ボタンがクリックされた時点で実行されていたメソッドを「メソッドエディタ」ウインドウで表示します。

重要：このメソッドを修正することはできますが、「デバッグ」ウインドウで現在トレースされているメソッドのインスタンスに対しては、この修正内容は表示されず、実行もされません。メソッドがアボートされるか、または正常に終了した後、このメソッドが次に実行される時に修正が反映されます。つまり、メソッドへの変更を有効にするには、メソッドを再ロードしなければなりません。

Tips：このボタンは、コードに必要な変更内容がわかっている場合や、変更が実行やトレースの対象となるコードの残りの部分の妨げにならない場合に使用します。

Tips：オブジェクトメソッドは、各イベントごとに再ロードされます。オブジェクトメソッドをトレースしている場合には（つまり、ボタンのクリックに対して）、フォームを終了する必要はありません。オブジェクトメソッドを編集し、変更内容を保存し、フォームに戻って再実行することができます。フォームメソッドのトレースや変更の際に、フォームメソッドを再ロードするには、フォームを終了し、再び表示しなければなりません。フォームを大規模にデバッグする場合のコツは、（デバッグの対象となっている）コードを、フォームメソッドからのサブルーチンとして使用しているプロジェクトメソッドに入力することです。このようにすれば、このサブルーチンがフォームメソッドから呼び出されるたびに再ロードされるため、フォームをトレースし、編集し、再テストしている間もフォームを使用することができます。

「設定保存」ボタン

現在のデバッグウインドウの構成（ウインドウのサイズと位置、分割線の配置および「カスタム・ウォッチ」エリアの内容）を、データベースが開かれるたびにデフォルトで使用できるように保存することができます。

これらの内容は、データベースのストラクチャファイルに保存されます。

「Step Over（同一メソッドのみ）」ボタン

現在のメソッド行（プログラムカウンタと呼ばれる黄色い矢印で示されている行）が実行され、デバッガは次の行に移動します。「ステップ（同一メソッドのみ）」ボタンはサブルーチンや関数に移動することはなく、現在トレースの対象となっているメソッドのレベルにとどまります。サブルーチンや関数呼び出しもトレースしたい場合には、「ステップ（呼び出しメソッドもトレース）」ボタンを使用します。

「Step Into（呼び出しメソッドもトレース）」ボタン

別のメソッド（サブルーチンまたは関数）を呼び出す行が実行される場合にこのボタンを使用すると、呼び出されているメソッドが「デバッグ」ウインドウに表示され、このメソッドをステップごとに実行できます。

「デバッグ」ウインドウの「メソッド連鎖」エリアでは、新しいメソッドが現在（トップ）のメソッドになります。別のメソッドを呼び出していない行が実行される場合には、このボタンは「ステップ（同一メソッドのみ）」ボタンと同等に動作します。

「Step Into Prosess（別プロセスもトレース）」ボタン

新しいプロセスを作成する行が実行される場合（**New Process**コマンドをコールした場合）にこのボタンを使用すると、新しい「デバッグ」ウインドウが表示され、新しく作成されたプロセスのプロセスメソッドをトレースすることができます。新しいプロセスを作成しない行が実行される時には、このボタンは「ステップ（同一メソッドのみ）」ボタンと同等に動作します。

「Step Out（呼び出し元へ戻る）」ボタン

サブルーチンや関数をトレースする場合にこのボタンをクリックすると、現在トレースされているメソッド全体を実行し、呼び出し元メソッドに戻ることができます。「デバッグ」ウインドウはメソッド連鎖中の前のメソッドに戻ります。現在のメソッドがメソッド連鎖中の最後のメソッドである場合には、「デバッグ」ウインドウが閉じられます。

「実行コントロール」ツールバーについて

「実行コントロール」ツールバーの右側には、デバッガから次のような情報が表示されます。

■ 現在トレースしているメソッドの名前（黒で表示されます）

■ 「デバッグ」ウインドウが表示される原因となった問題（赤で表示されます）

ここで示されているウインドウの例を使用すると、次のような情報が表示されています。

■ 現在トレースされているメソッドは「De_DebugDemo」メソッドです。

■ 「デバッグ」ウインドウが表示されているのは、C_DATEコマンドへの呼び出しが検出され、このコマンドは検出の対象となったコマンドのうちの1つであるためです。

デバッガとメッセージが表示される理由の一部は、次の通りです（赤で表示されます）。

■ **TRACEコマンド：TRACE**コマンドへの呼び出しが実行されたため。

■ **ブレークポイントに到達：**一時的ブレークポイントまたは永続的ブレークポイントが発見されたため。

■ **ユーザ割り込み：**「Alt+Shift+右クリック（Windows）」または「control+option+command+クリック（Macintosh）」を使用するか、あるいは「デザイン」モードの「ランタイムエクスプローラ」の「プロセス」ページのトレースボタンをクリックしたため。

■ **次のコールを捕まえました：**コマンド名：検出の対象となった4Dコマンドへの呼び出しが実行位置にあるため。

■ **新規プロセスへステップしています：**「ステップ（新規プロセスもトレース）」ボタンを使用したため、新しく作成されたプロセス用に開かれた「デバッグ」ウインドウよりこのメッセージが表示される。

「デバッグ」ウインドウの各エリア

「デバッグ」ウインドウは、前述の「実行コントロール」ツールバーとサイズ変更可能な次の4つのエリアから構成されます。

■ 「ウォッチ」エリア

■ 「メソッド連鎖」エリア

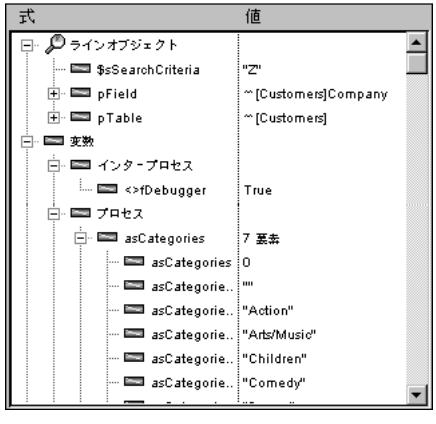
■ 「カスタム・ウォッチ」エリア

■ 「ソースコード」エリア

最初の3つのエリアでは、操作が簡単な階層型リストを使用して、関連するデバッグ情報を表示します。4番目の「ソースコード」エリアは、トレースされているメソッドのソースコードを表示します。それぞれのエリアには、デバッグ作業を支援する独自の機能があります。マウスを使用して、「デバッグ」ウインドウだけではなく、各エリアも垂直および水平方向にサイズを変更することができます。さらに、最初の3つのエリアには、表示する2つのカラムの間に点線による区切り線が含まれています。マウスを使用して、この点線を移動し、必要に応じて水平方向にカラムのサイズを変更することができます。

「ウォッチ」エリア

「ウォッチ」エリアは、「デバッグ」ウインドウの左上隅の「実行コントロール」ツールバーの下に表示されます。



「ウォッチ」エリアには、システム、4D環境、および実行環境について役立つ一般情報が表示されます。

「式」欄には、オブジェクトや式の名前が表示されます。「値」欄には、オブジェクトや式に対応する現在の値が表示されます。

エリア右側にある値をクリックすると、そのオブジェクトの値を変更できる場合には、オブジェクトの値を修正できます。

複数レベルを対象とする階層リストは、メインレベルでテーマごとにまとめられています。テーマは、次の通りです。

- ラインオブジェクト
- 変数
- 定数
- フィールド
- セマフォ
- セット
- プロセス
- 命名セレクション
- インフォメーション
- キャッシュの統計

テーマによっては、各項目に1つまたは複数のサブレベルがある場合もあります。テーマ名の隣にあるリストノード（アイコン）をクリックすると、テーマが拡大、または縮小します。テーマが拡大されている場合には、そのテーマにある項目は見えています。テーマに複数レベルの情報がある場合には、各項目の隣にあるリストノードをクリックすると、そのテーマで提供されているすべての情報を調べることができます。

どの時点でも、テーマ、テーマサブリスト（あれば）、テーマ項目を「カスタム・ウォッチ」エリアにドラッグ＆ドロップすることができます。

キャッシュの統計：4Dのキャッシュ内にロードされているテーブル、インデックスページおよび命名セレクションの利用に関しての統計を表示します。

インフォメーション：現在のデフォルトテーブル（あれば）等、一般的な情報を表示します。このテーマからの表現式を修正することはできません。

命名セレクション：カレント（現在トレースの対象となっている）プロセスで定義されているプロセス命名セレクションを一覧表示します。また、インタープロセス命名セレクションも一覧表示します。各命名セレクションでは、「値」欄にレコード数およびテーブル名が表示されます。命名セレクションを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

プロセス：作業セッションを開始してから起動されたプロセスを一覧表示します。「値」欄に、それぞれのプロセスの現在の状態（実行中、一時停止等）および使用した時間が表示されます。このテーマからの表現式を修正することはできません。

セット：カレント（現在トレースの対象となっている）プロセスで定義されているセットを一覧表示します。インタープロセスセットも一覧表示します。各セットでは、「値」欄にレコードの数とテーブル名が表示されます。セットを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

セマフォ：現在設定されているローカルセマフォおよびグローバルセマフォを一覧表示します。各セマフォでは、「値」欄にセマフォを設定するプロセスの名前が表示されます。セマフォを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

テーブルとフィールド：データベース内にあるテーブルやフィールドを一覧表示しますが、サブフィールドは表示しません。各テーブル項目では、「値」欄にカレントプロセスのカレントセクションのサイズは勿論、ロックされたレコードナンバー（テーブルアイテムは拡張される）も表示されます。各フィールド項目では、カレントレコードがある場合には、「値」欄にカレントレコードのフィールドの値（ピクチャ、サブテーブル、BLOBは除く）が表示されます。このテーマでは、フィールドの値を修正することはできますが（ただし、取り消しはできません）、テーブル情報を修正することはできません。

定数：「エクスプローラ」ウインドウの「定数」ページのように、4th Dimensionが提供する定義済み定数を表示します。このテーマからの表現式を修正することはできません。

変数：このテーマは、次のサブテーマから構成されます。

- **インタープロセス変数：**この時点で使用されているインタープロセス変数のリストを表示します。インタープロセス変数を使用していない場合には、このリストは空白の場合があります。インタープロセス変数の値は、変更することができます。
- **プロセス変数：**カレントプロセスで使用されている変数のリストを表示します。このリストが空白であることはほとんどありません。プロセス変数の値は、修正することができます。
- **ローカル変数：**現在トレースの対象となっているメソッド（ソースコードエリアに表示されているメソッド）で使用されているローカル変数のリストを表示します。ローカル変数を使用していない場合や、ローカル変数がまだ作成されていない場合には、このリストは空白の場合があります。ローカル変数の値は、修正することができます。
- **パラメータ（引数）：**メソッドが受け取ったパラメータのリストを表示します。現在トレースの対象となっているメソッド（ソースコードエリアに表示されているメソッド）にパラメータが渡されていない場合には、このリストは空白の場合があります。パラメータの値は、修正することができます。

■ セルフポインタ：オブジェクトメソッドをトレースしている場合には、現在のオブジェクトへのポインタを表示します。この値を修正することはできません。

注：文字列変数、テキスト変数、数値変数、日付変数、および時間変数は、修正することができます。つまり、キーボードを使用して値を入力できる変数は、修正することができます。

他の変数と同様に、配列は、その設定範囲によって、インタープロセス配列サブテーマ、プロセス配列サブテーマ、およびローカル配列サブテーマで表示されます。デバッガは各配列に階層レベルをつけて表示します。このため、配列要素の値がある場合にはこれを取得、または変更することができます。デバッガは要素ゼロを含む最初の100個の要素を表示します。「値」欄には、配列名ごとのサイズが表示されます。配列を作成した後、最初のサブ項目には、現在選択されている要素番号、次に要素ゼロ、その次に他の要素（100個まで）が表示されます。文字列配列、テキスト配列、数値配列、および日付配列は、修正することができます。選択された要素番号、要素ゼロ、および他の要素（100個まで）も、修正することができます。配列のサイズを修正することはできません。

注：個別の配列要素も含め、項目はいつでも「ウォッチ」エリアから「カスタム・ウォッチ」エリアへドラッグ＆ドロップすることができます。

ラインオブジェクト

このテーマには、次のようなオブジェクトや式の値が表示されます。

■ 実行されるコードの行（プログラムカウンタにより、「ソースコード」エリア内で黄色の矢印でマークされている行）で使用されている。

■ コードの前の行で使用されている。

コードの前の行とは実行直後の行であるため、「ラインオブジェクト」テーマでは、その行が実行される前または後の現在の行のオブジェクトや式が表示されます。例えば、次のメソッドを実行した場合を想定します。

```
TRACE
a:=1
b:=a+1
c:=a+b
      `…
```

1. 「デバッグ」ウインドウで「ソースコード」エリアのプログラムカウンタを「a:=1」の行にセットします。この時点では「ラインオブジェクト」テーマには、次のように表示されています。

   a：　未定義

変数aが表示されているのは、実行の対象となっている（ただし、まだ初期化されていない）行で使用されているためです。

2. 1行先に移動すると、プログラムカウンタは「b:=a+1」の行に設定されます。この時点では「ラインオブジェクト」テーマには、次のように表示されています。

```
a：　1
b：　未定義
```

変数aが表示されているのは、ちょうど実行され、数値1を割り当てられたばかりの行で使用されているためです。また、変数aが表示されているのは、変数bへの割り当て式として実行される行でも使用されているためでもあります。変数bが表示されているのは、実行の対象となっている（ただし、まだ初期化されていない）行で使用されているためです。

3. 再び1行先に移動すると、プログラムカウンタは「c:=a+b」の行に設定されます。この時点では「ラインオブジェクト」テーマには、次のように表示されています。

```
c：　未定義
a：　1
b：　2
```

変数cが表示されているのは、実行の対象となっている（ただし、まだ初期化されていない）行で使用されているためです。変数aとbが表示されているのは、1つ前の行で使用され、この行で実行されているからです。

「ラインオブジェクト」テーマは、とても便利なツールです。ある行が実行される度に「カスタム・ウォッチ」エリアに式を入力することなく、「ラインオブジェクト」テーマによって表示される値を検証することができます。

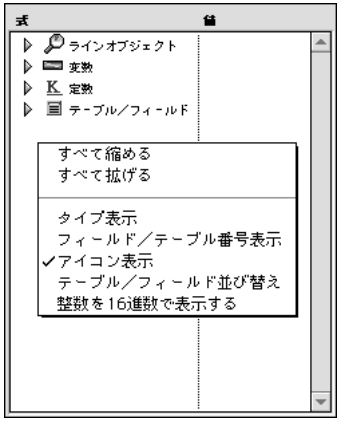
「コンテキスト」メニュー

「ウォッチ」エリアの「コンテキスト」メニューで追加オプションを提供します。このメニューは、次のように表示します。

■ Windowsでは、マウスの右ボタンを使用して「ウォッチ」エリア内の任意の位置をクリックする。

■ Macintoshでは、「ウォッチ」エリアの任意の位置で「control+クリック」を実行する。

「ウォッチ」エリアの「コンテキスト」メニューが、次のように表示されます。



■「すべて縮める」：「ウォッチ」エリアの階層リストのすべてのレベルを縮小します。

■「すべて拡げる」：「ウォッチ」エリアの階層リストのすべてのレベルを拡張します。

■「タイプ表示」：それぞれのオブジェクトのオブジェクトタイプを（適切な場合に）表示します。

■「フィールド／テーブル番号表示」：「フィールド」のそれぞれのテーブルまたはフィールドの番号を表示します。テーブル番号やフィールド番号を用いて作業している場合、または、**Table**関数や**Field**関数を使用し、ポインタを用いて作業している場合、このオプションは非常に便利です。

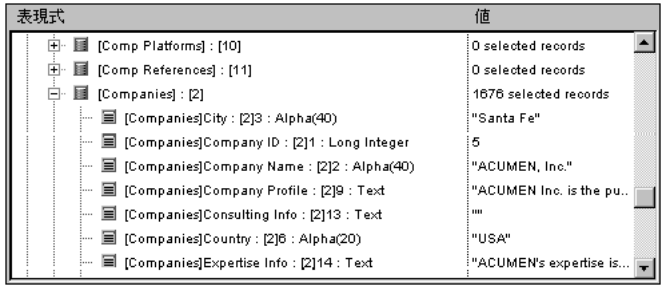
■「アイコン表示」：それぞれのオブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、あるいは「タイプ表示」オプションを使用することにして、このオプションをオフにすることもできます。

■「テーブル／フィールド並び替え」：テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。

■「整数を16進数で表示する」：通常、数字は10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。

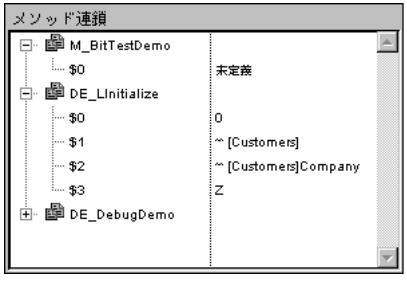
注：数値を16進法で入力するには、0x（ゼロの後にx）とタイプし、その後に16進数を続けます。

次の図は「ウォッチ」エリアですべてのオプションが選択された状態を示しています。



「メソッド連鎖」エリア

1つのメソッドによって他のメソッドも呼び出される場合があります。さらにそれらがその他のメソッドを呼び出す場合もあります。このため、デバッグ処理中には、メソッドの連鎖、つまり呼び出しチェーンを表示しておくと非常に便利です。「メソッド連鎖」エリアは「デバッグ」ウインドウの上部右側にあり、この便利な機能を提供しています。このエリアは、階層リストを使用して表示されます。次の図は、「メソッド連鎖」エリアの例を示しています。



■ それぞれのメインレベルの項目は、メソッドの名前です。最も上にある項目は、現在トレースされているメソッド、次のメインレベルの項目は呼び出し元のメソッド（現在トレースしているメソッドを呼び出したメソッド)、その次の項目は呼び出し元のメソッドの呼び出し元メソッド、等のように続きます。この例では、メソッド「M_BitTestDemo」がトレースされています。このメソッドは、メソッド「DE_LInitialize」によって呼び出され、これは「DE_DebugDemo」によって呼び出されています。

■ 「メソッド連鎖」エリアにあるメソッドの名前をダブルクリックすると、呼び出し元のメソッドに移動し、そのソースコードが「ソースコード」エリアに表示されます。このようにすると、呼び出し元のメソッドが呼び出されたメソッドへの呼び出しをどのように実行したか、すばやく確認することができます。このようにして、呼び出しチェーンのあらゆる段階を検証することができます。

■ メソッド名の隣にあるノードアイコンをクリックすると、メソッドのパラメータ（$1, $2...）およびオプションの関数結果（$0）のリストが拡張、または縮小されます。値はエリアの右側に表示されます。

右側の矢印にある値をクリックすると、パラメータや関数の結果の値を変更することができます。この図では、次の通りです。

1. 「M_BitTestDemo」メソッドは、まだどのパラメータも受け取っていません。

2. 「M_BitTestDemo」メソッドの$0は現在未定義です。これは、メソッドが$0に値を割り当てていないためです（メソッドがこの割り当てをまだ実行していないか、メソッドがサブルーチンであり、関数ではないことが原因です）。

3. 「DE_LInitialize」メソッドは、「DE_DebugDemo」メソッドから3つのパラメータを受け取ります。$1はテーブル[Customers]へのポインタ、$2はフィールド[Customers]Companyへのポインタ、$3は値が"Z"の英数字パラメータです。

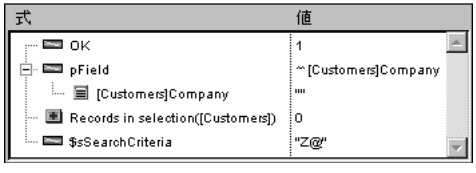
■ メソッドにパラメータリストを割り当てれば、パラメータや関数の結果を「カスタム・ウォッチ」エリアにドラッグ＆ドロップすることができます。

「カスタム・ウォッチ」エリア

「メソッド連鎖」エリアのすぐ下にあるのは、「カスタム・ウォッチ」エリアです。このエリアは、式を評価するために使用します。フィールド、変数、ポインタ、演算、組み込み関数、カスタム定義関数等、値を戻すものなら何でも、どのようなタイプの式でも評価できます。

テキスト形式で表示できる式であれば、どのような式でも評価することができます。ピクチャやBLOBのフィールドや変数は、対象になりません。一方、デバッガは、割り当てられた階層リストを使用して、配列やポインタを表示できるようにします。BLOBの内容を表示するには、**BLOB to text**等のようなBLOBコマンドを使用します。

次の例では、2つの変数、1つのフィールドポインタ変数、組み込み関数の結果、演算の項目のうちいくつかが示されています。



新しい式の挿入

次のように「カスタム・ウォッチ」エリアに式を追加して、評価することができます。

■ 「ウォッチ」エリアからオブジェクトまたは式をドラッグ＆ドロップします。

■ 「メソッド連鎖」エリアからオブジェクトまたは式をドラッグ＆ドロップします。

■ 「ソースコード」エリアで、評価できる式をクリックします。

空の式を作成するには、「カスタム・ウォッチ」エリアの空白スペースの任意の位置をダブルクリックします。すると、新しい式“新規式”が追加された後、編集モードになり、編集することができます。結果を戻す4th Dimensionのフォーミュラを入力できます。

フォーミュラを入力した後、enterまたはreturnキーを押して（またはエリアの任意の位置をクリックして)、式を評価します。

式を変更するには、その式をクリックして選択し、再びクリックすると（またはテンキー上のenterを押す）編集モードになります。

式が必要でなくなった場合には、その式をクリックして選択し、Backspaceキーまたはdeleteキーを押します。

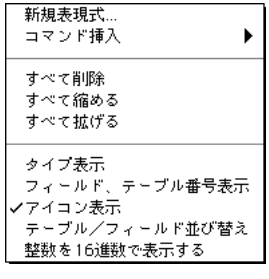
「カスタム・ウォッチ」エリアの「コンテキスト」メニュー

式を入力し、編集する場合には、「カスタム・ウォッチ」エリアの「コンテキスト」メニューを使用すると、4th Dimensionの「フォーミュラ」エディタにアクセスできて便利です。実際には、この「コンテキスト」メニューには、他にもオプションがあります。

このメニューを表示するには、次のように実行します。

■ Windowsでは、マウスの右ボタンを使用して、「カスタム・ウォッチ」エリアの任意の位置をクリックする。

■ Macintoshでは、「カスタム・ウォッチ」エリアの任意の位置で「control+クリック」を実行する。



■「新規表現式...」：新しい式を入力し、（以下の図のような）4th Dimensionの「フォーミュラ」エディタを表示して、新しい式を編集できるようにします。



「フォーミュラ」エディタに関する詳細は、『4th Dimensionユーザリファレンス』を参照してください。

■**「コマンド挿入」**：この階層メニュー項目は、「フォーミュラ」エディタを使用せずにコマンドを新しい式として挿入するためのショートカットです。

■**「すべて削除」**：現在あるすべての式を削除します。

■**「すべて縮める／すべて拡げる」**：階層型リストを使用して評価が実行されたすべての式（ポインタ、配列等）を縮小、または拡大します。

■**「タイプ表示」**：（適切な場合に）各オブジェクトのオブジェクトタイプを表示します。

■**「フィールド、テーブル番号表示」**：「フィールド」にあるそれぞれのテーブルやフィールドの番号を表示します。**Table**関数や**Field**関数を使用してテーブル番号やフィールド番号、ポインタを用いた作業を行っている場合、このオプションは非常に便利です。

■**「アイコン表示」**：各オブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、あるいは「タイプ表示」オプションを使用することにして、このオプションをオフにすることもできます。

■**「テーブル／フィールド並べ替え」**：テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。

■「整数を16進数で表示する」：通常、数字は10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。

注：数値を16進法で入力するには、0x（ゼロの後にx）とタイプし、その後に16進数を続けます。

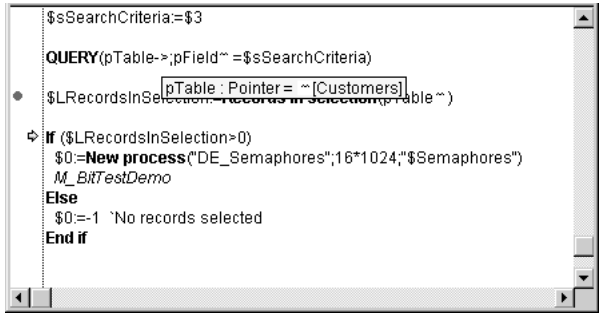
「ソースコード」エリア

「ソースコード」エリアには、トレースされているメソッドのソースコードが表示されます。

■ メソッドが長すぎてテキストエリアに収まらない場合には、スクロールするとメソッドの他の部分も表示できます。

■ 評価できる式（フィールド、変数、ポインタ、配列等）にマウスポインタを移動すると、「ツールチップ」が表示され、オブジェクトや式の現在の値とその宣言型が示されます。

次の図は「ソースコード」エリアを示しています。



「ツールチップ」が表示されているのは、マウスポインタが変数「pTable」の上に置かれているためで、その表示によると、この変数は[Customers]テーブルへのポインタです。

ソースコードエリア内でテキストの部分を選択することが可能になりました。この場合、選択されたテキストの上にカーソルを移動すると、選択されたオブジェクトの値をTipsとして表示します。

変数名またはフィールドをクリックすると自動的に選択されます。

Tips：「ソースコード」エリアで（評価できる）式をクリックすると、式またはオブジェクトが「カスタム・ウォッチ」エリアにコピーされます。選択されたテキストは、「カスタム・ウォッチ」エリア内に、次の方法で置くことができます。

■ 単純にドラッグ&ドロップを行なうことによって。選択されたテキストをクリックし、それを「カスタム・ウォッチ」エリア内へドラッグ&ドロップ。

■ WindowsではCtrlキーを、またはMacintoshではcommandのキーを押しながら、選択されたテキストをクリックすることによって。

■ WindowsではCtrlキーを押しながらD、またはMacintoshではcommandキーを押しながらDのキーのコンビネーション。

プログラムカウンタ

「ソースコード」エリアの左マージンにある黄色の矢印（上図を参照してください）は、実行される次の行を表しています。この矢印は、“プログラムカウンタ”と呼ばれます。プログラムカウンタは、常に実行寸前の行を表示します。

デバッグのために、メソッドのプログラムカウンタの位置を呼び出しチェーン（実際に実行されるメソッド）の先頭に変更することができます。そのためには、黄色の矢印をクリックして目的の行まで上下にドラッグします。

警告：この機能は、十分注意して使用してください。

プログラムカウンタを順方向に移動しても、スキップした行をデバッガがすばやく実行しているわけではありません。

同様に、プログラムカウンタを逆方向に移動しても、既に実行された行の結果をデバッガが逆方向に実行しているわけではありません。

プログラムカウンタを移動するということは、デバッガにその位置からのトレースや実行を追跡するように指示するというだけに過ぎません。すべての現在の設定内容、フィールド、変数等には、プログラムカウンタの移動による影響はありません。

プログラムカウンタを移動する例は、次の通りです。例えば、次のようなコードをデバッグすると仮定します。

```
`…
If(条件)
        DO SOMETHING
Else
        DO SOMETHING ELSE
End if
```

プログラムカウンタは、行「If(条件)」に設定されています。1ステップ先に進むと、プログラムカウンタが行「DO SOMETHING ELSE」に移動していることがわかります。別の分岐にある行を実行しようとしていたため、これは思いがけないことです。この場合には、“条件”という式が次のステップに影響を与えるような演算を実行していなければ、プログラムカウンタを行「DO SOMETHING」に戻します。これで、コードの当初目的としていた部分を続けてトレースすることができます。

デバッガのブレークポイントの設定

デバッグの過程において、コードのトレースを一部スキップすることが必要な場合があります。デバッガでは、特定の位置までコードを実行する方法がいくつか提供されます。

- ステップごとの処理中に、「ステップ（呼び出しメソッドもトレース）」ボタンではなく「ステップ（同一メソッドのみ）」ボタンをクリックすることができます。プログラムカウンタ行で呼び出されているサブルーチンや関数の実行を避けたい場合には、この方法が役立ちます。

- 間違ってサブルーチンの処理を始めてしまった場合には、「ステップ（呼び出し元へ戻る）」ボタンをクリックすると、そのサブルーチンを実行した後直接、呼び出し元のメソッドに戻ることができます。

- **TRACE**コマンド呼び出しをある位置で指定した場合には、「トレースなし」ボタンをクリックすると、その**TRACE**コマンド呼び出しまでの実行を再開始することができます。

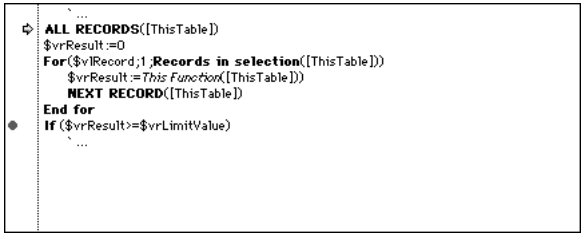
次に、プログラムカウンタを行「**ALL RECORDS**([This Table])」に設定し、次のようなコードを実行していると想定してみます。

```
`...
ALL RECORDS([This Table])
$vrResult:=0
For($vlRecord ; 1 ; Records in selection([This Table]))
        $vrResult:= this Function([This Table])
        NEXT RECORD([This Table])
End for
If($vrResult >= $vrLimitValue)
        `...
```

ここでの目的は、**For**ループが終了した後で「$vrResult」の値を評価することです。コードがこの位置まで実行されるにはかなりの処理時間がかかるため、現在の実行をアボートしたくない場合には、**TRACE**コマンド呼び出しを行「**If**($vrResult..」の前に挿入するようにメソッドを編集する必要があります。

ループのステップ処理を実行することも1つの方法ですが、「This Table」テーブルに何百件ものレコードが入っている場合には、この処理に一日費やすことになります。このような状況では、デバッガのブレークポイントを使用できます。ブレークポイントは、「ソースコード」エリアの左マージンをクリックすると挿入できます。

例えば、次の例では、行「**If**($vrResult...」のレベルで「ソースコード」エリアの左マージンをクリックします。



このようにすると、その行にブレークポイントが挿入されます。ブレークポイントは赤色の点で表されます。次に、「トレースなし」ボタンをクリックします。

このようにすると、ブレークポイントで示された行まで、通常の実行が再開されます。ブレークポイントで示された行は実行されずに、トレースモードへ戻ります。この例では、ループ全体は連続して正常に実行されています。そのため、ブレークポイントに到達した時には、マウスボタンを「$vrResult」の上に移動して、その値をループの終了地点で評価する必要があります。

プログラムカウンタの位置を越えてブレークポイントを設定し、「トレースなし」ボタンをクリックすると、トレースされた部分のメソッドをスキップすることができます。

赤色のブレークポイントは、永続的ブレークポイントです。このブレークポイントは、一度作成するとそのまま残ります。データベースを終了し、後で再び表示すると、ブレークポイントは残っています。

永続的ブレークポイントを消去する方法は、次の2通りです。

- 永続的ブレークポイントを使用した後、赤色の点をクリックすると、ブレークポイントは消えます。

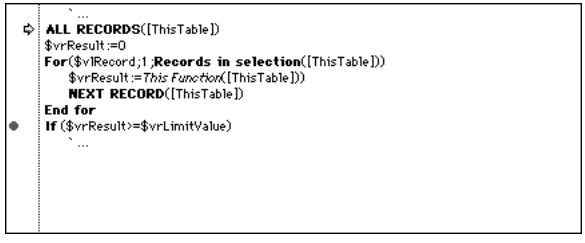
- 永続的ブレークポイントをまだ使用する場合には、これを残しておきたい場合もあります。永続的ブレークポイントを編集すると、一時的に使用不可能にすることができます。編集方法については、「ブレークポイント」の節で説明しています。

ブレークポイント

「ソースコード」エリアの節で説明しているように、ブレークポイントは、ブレークしたいコード行と同じレベルで、「ソースコード」エリアの左マージンかメソッドエディターウインドウをクリックして設定します。

注：メソッドエディタ内で直接ブレークポイントの挿入、変更、削除をすることができます。もちろん、ブレークポイントはメソッドエディタとランタイムエクスプローラの間で（デバッガも同様に）お互いに影響し合っています。実際、ブレークポイントはこれらの3つのエディタ内で定義することができます。

次の図では、ブレークポイントは「**If**($vrResult>=$vrLimitValue)」に設定されています。



赤色の点を再びクリックすると、ブレークポイントは削除されます。

ブレークポイントの編集

「ソースコード」エリア、またはメソッドエリアウインドウの左マージンで「Alt+クリック（Windows)」または「option+クリック（Macintosh)」を実行すると、コードの行単位で「ブレークポイントプロパティ」ウインドウにアクセスできます。

■ 既存のブレークポイントをクリックすると、そのブレークポイントについてのウインドウが表示されます。

■ ブレークポイントが設定されていない行をクリックすると、デバッガはブレークポイントを作成し、新しく作成されたブレークポイントに関するウインドウを表示します。

「ブレークポイントプロパティ」ウインドウは、次の図の通りです。



プロパティは、次の通りです。

「場所」：メソッドの名前とブレークポイントが設定されている行番号を示します。この情報を変更することはできません。

「タイプ」：デフォルトでは、デバッガは、永続的ブレークポイントを作成します。永続的ブレークポイントは、「デバッグ」ウインドウの「ソースコード」エリアで赤色の点で示されます。一時的ブレークポイントを作成するには、「一時的」オプションを選択します。一時的ブレークポイントは、メソッド中で一度だけブレークしたい場合に役立ちます。一時的ブレークポイントは、「デバッグ」ウインドウの「ソースコード」エリアで緑色の点で示されます。「Alt+Shift（Windows）」キーまたは「option+shift（Macintosh）」キーを押しながら左マージンをクリックして、「ソースコード」エリアに一時的ブレークポイントを直接設定することもできます。

注：一時的ブレークポイントはデバッガだけで設定できます。

「以下に当てはまる場合にブレークする」：TrueまたはFalseを返す4th Dimensionフォーミュラを入力することによって、条件付きブレークポイントを作成することができます。例えば、「**Records in selection**([aTable])=0」の条件を満たす場合に限った行でブレークさせる場合には、このフォーミュラを入力すると、デバッガがこのブレークポイントを検出した際に、テーブル[aTable]のレコードが選択されていない場合に限ってブレークが発生します。フォーミュラの構文が不確かな場合には、「構文検査」ボタンをクリックします。

「ブレーク前のスキップ回数」：ループ構造（While、Repeat、For）内、またはループから呼び出されているサブルーチンや関数内のコード行にブレークポイントを設定することができます。例えば、現在調査している問題は、少なくともループを200回繰り返すまでは発生しないことがわかっているものとします。このような場合には200と入力すると、ブレークポイントは201回目の繰り返しからアクティブになります。

「ブレークポイント無効」：永続的ブレークポイントが現在は必要でないものの、後で必要になるかもしれない場合には、ブレークポイントを編集して一時的に使用不可能にしておくことができます。無効にされたブレークポイントは、「デバッグ」ウインドウの「ソースコード」エリアおよびメソッドエディタ内、ランタイムエクスプローラにおいて、点（·）ではなくダッシュ記号（-）で表示されます。

「デバッグ」ウインドウ内でブレークポイントを作成、または編集することができます。さらに「デザイン」モードの「ブレークリスト」ウインドウを使用して、既存のブレークポイントを編集することもできます。詳細については、次節を参照してください。

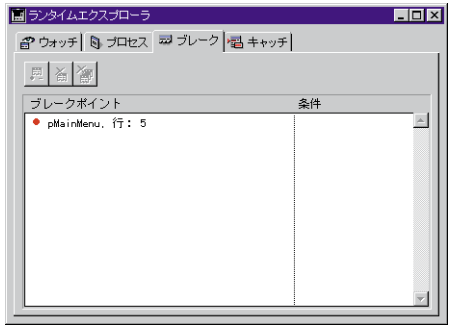
ブレークリスト

ブレークリストは「デバックウインドウ」又は「メソッドエディタ」で作成したブレークポイントを管理することが出来る「ランタイムエクスプローラ」のページです。

ブレークリストのページを開くには、

1. 他のモードで作業している場合は、「デザイン」モードに移動する。
2. 「ツール」メニューから「ランタイムエクスプローラ」を選択する。

   「ランタイムエクスプローラ」はフローティングパレットから表示できます。その場合、フローティングパレットは通常前面に表示されたままです。フローティングパレット内にランタイムエクスプローラを表示するには、Windowsでは、「Ctrl＋Shift＋F9」、Macintoshでは、「command＋shift＋F9」を押します。または、shiftキーを押しながら、ツールメニューからランタイムエクスプローラを選択します（デザインモードのみ）。
3. ブレイクリストを表示するには、ブレイクタブコントロールをクリックする。



ブレイクリストは2つの欄から構成されています。

■ 左の欄には、ブレイクポイントの有効／無効状況と、メソッド名とブレークポイントが設定されている行番号（「デバッグ」ウインドウ又はメソッドエディタを使用）が表示されます。

■ 右の欄には、ブレークポイントに関連する条件（ある場合）が表示されます。

このウインドウを使用するには

■ ブレークポイントの条件を設定する

■ それぞれブレークポイントに有効、無効又は削除を設定する

■ ブレークポイントをダブルクリックして定義されたメソッドが表示しているメソッドエディタウインドウをオープンする。

しかし、ウインドウから永続的ブレークポイントを新しく追加することはできません。永続的ブレークポイントはデバッグウインドウかメソッドエディタからのみ設定できます。

▼ ブレークポイントの条件の設定

ブレークポイントの条件を設定するには

1. 右の欄の任意の場所をクリックする。

2. ブール値を返す4Dフォーミュラ（式、またはコマンドコールやプロジェクトメソッド）を入力する。

注：条件を削除するにはフォーミュラを削除します。

▼ ブレークポイントを削除する

ブレークポイントを削除するには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。

2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。

3. deleteキーを押す、または削除ボタンをクリックする。

注：ブレークポイントをすべて削除するには、すべて削除ボタンをクリックするか、またはコンテキストメニューから「すべて削除」を選択します。

Tips：

・中断コマンドやブレークポイントに条件を追加すると、処理速度が低下します。これは、例外条件になるたびに条件が評価されるためです。一方で、条件を追加することにより、デバッグ処理は速くなります。これは、条件を満たさない場合は自動的にスキップされるためです。

・中断コマンドやブレークポイントを無効にすると、削除するのと同じ効果があります。実行中、デバッガはこの項目に対して時間をほとんど割きません。項目を無効にする利点は、再度その項目が必要となった場合に再作成しなくてもいいという点です。

参照：ブレークポイント、中断コマンド、デバッガ、ソースコードエリア、なぜデバッガを使用するか?

コマンドの中断（コマンドのキャッチ）

コマンド中断リスト（コマンド・キャッチ・リスト）は4Dのコマンド呼び出しブレーク（中断）を付け加えることが出来る「ランタイムエクスプローラ」のページです。

コマンドの中断は、任意の場所にブレークポイントを設定することなく、大きな範囲でトレースを行える便利な方法です。例えば、いくつかのプロセスを実行した後に、削除すべきでないレコードが削除されてしまった場合には、**DELETE RECORD**や**DELETE SELECTION**といったコマンドの処理を中断することにより、調査の範囲を狭めることができます。これらのコマンドが呼び出されるたびに、問題のレコードが削除されたかどうかを調べ、コードの誤った箇所を突き止めることができます。

少し経験を積めば、ブレークポイントとコマンドの中断とを組み合わせて使用できるようになります。

コマンド中断ページを開くには

1. デザインモードにする。
2. 実行メニューからランタイムエクスプローラを選択する。ランタイムエクスプローラは、フローティングパレットとして表示することができます。（フローティングパレットにすると、常に前面に表示されます）Shiftキーを押しながら、実行メニューからランタイムエクスプローラを選択する。またはWindowsでは、Ctrlキーを押しながらShiftキーとF9キーを押す。Macintoshでは、commandキーを押しながらshiftキーとF9キーを押す。
3. キャッチタブコントロールをクリックすると、処理中断コマンドリストが表示される。

このページは実行中に中断するコマンドをリスト表示します。2つの欄から構成されています。

■ 左の欄には、中断コマンドの有効／無効状況と、コマンド名が表示されます。

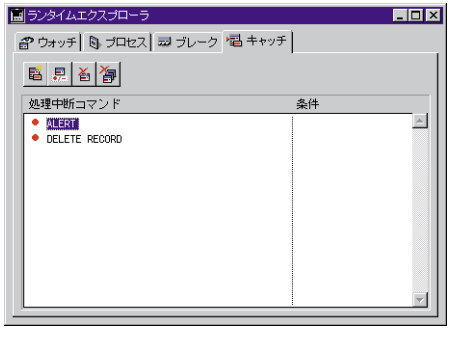
■ 右の欄には、中断コマンドに関連する条件（ある場合）が表示されます。

中断するコマンドを新しく追加する

新しくコマンドを追加するには次の手順に従ってください。

1. 「ブレークリスト」メニューから「新規ブレークポイント追加」ボタンををクリックする。または、処理中断コマンドリスト上でマウス（Windowsの場合は左マウスボタン）をダブルクリックする。

   ともに、リストにコマンド名が新しく追加され、デフォルトとして**ALERT**コマンドと表示されます。このコマンド名は編集できる状態になっています。



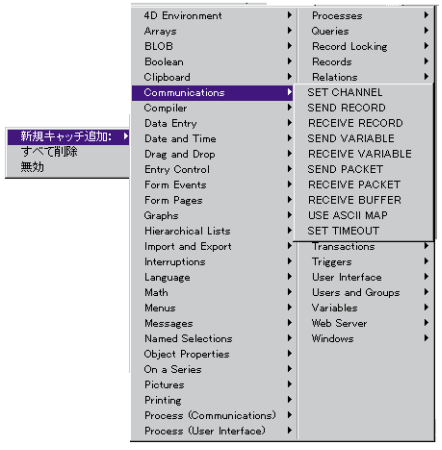
2. 中断しようとするコマンド名を入力する。

3. enterキーまたはreturnキーを押し、選択を確定する。

または、

1. 右マウスボタンをクリック（Windows）またはcontrolキーを押しながらクリック（Macintosh）する。

コンテキストメニューが表示されます。



2. 「新規ブレークポイント追加」を選択し、コマンドテーマとコマンド名のサブメニューから目的のコマンドを選択する。

選択したコマンド名が新しくリストに追加されます。

中断コマンド名を編集する

中断コマンドの名前を編集するには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。
2. 編集モードと選択モードとを切り替えるには、enterキーまたはreturnキーを押す。
3. コマンド名を入力または修正する。
4. 変更を有効にするには、enterキーまたはreturnキーを押す。

入力した名前が既存の4Dコマンドではない場合、項目は修正前の値に設定されます。新しく追加した場合は**ALERT**コマンドに戻ります。

中断コマンドを無効／有効にする

中断コマンドを無効、あるいは有効にするには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。
2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。

3. 「有効／無効」ボタンをクリックするか、またはコンテキストメニューから「有効／無効」を選択する。

ショートカット：リストの各項目は、点（丸）をクリックして無効、あるいは有効にすることができます。無効になった場合には、点がダッシュ（-）に変わります。

中断コマンドを削除する

中断コマンドを削除するには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。

2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。

3. Deleteキーを押す、または削除ボタンをクリックする。

注：中断コマンドをすべて削除するには、すべて削除ボタンをクリックするか、またはコンテキストメニューから「すべて削除」を選択します。

中断コマンドに条件を設定する

中断コマンドに条件を設定するには次の手順に従ってください。

1. 右の欄の項目をクリックする。

2. ブール値を返す4Dフォーミュラ（式、またはコマンドコールやプロジェクトメソッド）を入力する。

注：条件を削除するにはフォーミュラを削除します。

「ブレークポイント」エリア

「ブレークポイント」エリアには「デバッグ」ウインドウで作成した永続的ブレークポイントだけが表示されます。「処理中断コマンド」エリアとは異なり、このエリアで永続的ブレークポイントを新しく追加することはできません。永続的ブレークポイントは「デバッグ」ウインドウでのみ作成することができます。

ブレークポイントを有効／無効にする

ブレークポイントを有効／無効にするには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。

2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。

3. 「ブレークリスト」メニュー、またはコンテキストメニューから「有効／無効」を選択する。

ショートカット：リストの各項目は、点（丸）をクリックして無効、あるいは有効にすることができます。無効になった場合には、点がダッシュ（-）に変わります。

ブレークポイントを削除する

ブレークポイントを削除するには次の手順に従ってください。

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。

2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。

3. deleteキーを押す、または「ブレークリスト」メニューから「削除」を選択する。

注：ブレークポイントをすべて削除するには、「ブレークリスト」メニュー、またはコンテキストメニューから「すべて削除」を選択します。

ブレークポイントに条件を設定する

ブレークポイントに条件を設定するには次の手順に従ってください。

1. 右の欄の項目をクリックする。

2. ブール値を返す4Dフォーミュラ（式、またはコマンドコールやプロジェクトメソッド）を入力する。

注：条件を削除するにはフォーミュラを削除します。

Tips：

・中断コマンドやブレークポイントに条件を追加すると、処理速度が低下します。これは、例外条件になるたびに条件が評価されるためです。一方で、条件を追加することにより、デバッグ処理は速くなります。これは、条件を満たさない場合は自動的にスキップされるためです。

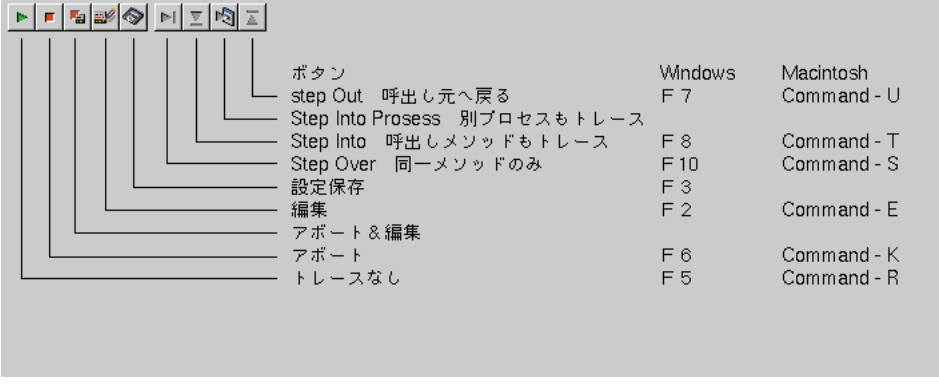
・中断コマンドやブレークポイントを無効にすると、削除するのと同じ効果があります。実行中、デバッガはこの項目に対して時間をほとんど割きません。項目を無効にする利点は、再度その項目が必要となった場合に再作成しなくてもいいという点です。

デバッガのショートカット

この節では、「デバッグ」ウインドウで提供されているすべてのショートカットをリストしています。

「実行コントロール」ツールバーでのボタンショートカット

次の図では、「デバッグ」ウインドウの左上隅にある8個のボタンのショートカットを示しています。



「トレースなし」	F5 or command+r
「アボート」	F6 or command+k
「アボート＆編集」	
「編集」	F2 or command+e
「設定保存」	F3
「Step Over（同一メソッドのみ）」	F10 or command+s
「Step Into（呼び出しメソッドもトレース）」	F8 or command+t
「Step Into Prosess（別プロセスもトレース）」	
「Step Out（呼び出し元へ戻る）」	F7 or command+u

■ shift押しながらF5、およびshiftを押しながらトレースなしボタンをクリック、実行を再開始します。さらに、現在のプロセスの次の**TRACE**コマンド呼び出しをすべて使用不可能にします。

「ウォッチ」エリアでのキーボードショートカット

■「ウォッチ」エリアで「マウスの右ボタンをクリック（Windows）」するか、または「control+クリック（Macintosh）」を実行すると、「ウォッチ」エリアの「コンテキスト」メニューがプルダウンされます。

■「ウォッチ」エリア内の項目をダブルクリックすると、その項目が「カスタム・ウォッチ」エリアにコピーされます。

「メソッド連鎖」エリアでのキーボードショートカット

■「メソッド連鎖」エリアでメソッドの名前をダブルクリックすると、メソッドが、「ソースコード」エリアの呼び出しチェーンにある呼び出しに対応する行に表示されます。

「カスタム・ウォッチ」エリアでのキーボードショートカット

■「カスタム・ウォッチ」エリアで「マウスの右ボタンをクリック（Windows）」するか、または「control+クリック（Macintosh）」を実行すると、「カスタム・ウォッチ」エリアの「コンテキスト」メニューがプルダウンされます。

■「カスタム・ウォッチ」エリア内でダブルクリックすると、新しい表現式が作成されます。

「ソースコード」エリアでのキーボードショートカット

■左マージンをクリックすると、ブレークポイントが設定される（永続的ブレークポイントの場合）、またはブレークポイントが削除されます。

■「Alt+Shift+クリック（Windows）」または「option+shift+クリック（Macintosh）」により、一時的ブレークポイントが設定されます。

■「Alt+クリック（Windows）」または「option+クリック（Macintosh）」により、「ブレークポイントプロパティ」ウインドウが表示されます。

■テキストを選択し、「カスタム・ウォッチ」エリアにドラック&ドロップします。

■Ctrlキー（Windows）、またはcommandキー（Macintosh）を押しながら、選択されたテキストをクリックすることによって。

■ Ctrlキー（Windows）、またはcommandキー（Macintosh）を押しながらDのキーのコンビネーションを使うことによって。

全エリア共通のキーボードショートカット

■ どのペインでも項目が選択されていない場合にenterキーを押すと、1行ずつ進みます。

■ 項目の値が選択されている場合には、矢印キーを使用してリスト内を移動します。

■ 項目が編集されている場合には、矢印キーを使用してカーソルを移動します。「Ctrl+a／x／c／v（Windows）」、または「command+a／x／c／v（Macintosh）」を「編集」メニューの「すべてを選択／切り取り／コピー／貼り付け」メニューへのショートカットとして使用します。

ドラッグ&ドロップ

この章では、「ルーチン」エディタの「Drag&Drop」テーマ内にあるドラッグ&ドロップコマンドについて説明します。

DRAG AND DROP PROPERTIES **Drop position**

4th Dimensionでは、フォーム内でのオブジェクト間で組み込みのドラッグ&ドロップ機能を導入しています。同一のウインドウ内でも、別のウインドウへも、あるオブジェクトから別のオブジェクトへドラッグ&ドロップすることができます。

4th Dimensionでは、デスクトップや別のアプリケーションとの組み込みドラッグ&ドロップ機能はありません。ただし、この機能は4Dのパートナーによって開発されたプラグインで提供されています。

注：まず最初に、ドラッグ&ドロップアクションがある点から別の点までいくつかのデータを「移動させる」ものであると想定します。次に、ドラッグ&ドロップが操作のメタファにもなれることを確認します。

ドラッグ可能およびドロップ可能なオブジェクトプロパティ

あるオブジェクトから別のオブジェクトにドラッグ&ドロップを実行するには、「オブジェクトプロパティ」ウインドウでそのオブジェクト用の「ドラッグ可」プロパティを選択する必要があります。ドラッグ&ドロップ処理では、ドラッグされるオブジェクトがソース（送信元）オブジェクトになります。

あるオブジェクトをドラッグ&ドロップ処理の送信先にするには、「オブジェクトプロパティ」ウインドウでそのオブジェクト用の「ドロップ可」プロパティを選択する必要があります。ドラッグ&ドロップ処理では、データを受け取る（ドロップされた）オブジェクトが送信先オブジェクトになります。

デフォルトでは、新しく作成されたオブジェクトはドラッグもドロップもできません。これらのプロパティを設定するかどうかはユーザに任されています。

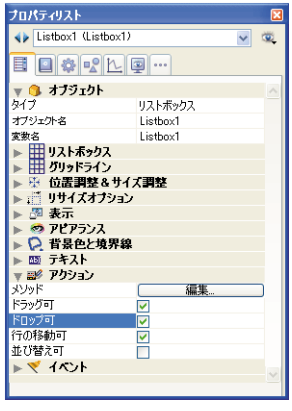
入力フォームまたはダイアログのフォームにあるすべてのオブジェクトは、ドラッグまたはドロップの対象にできます。配列の個別の要素（例えば、スクロール可能エリア）または階層リストの項目はドラッグ&ドロップができます。また逆に、配列の個別の要素や階層リストの項目に対してオブジェクトをドラッグ&ドロップすることもできます。ただし、出力フォームのディテイルエリアからオブジェクトをドラッグ&ドロップすることはできません。

4th Dimensionではアクティブオブジェクトの任意のタイプ（フィールドまたは変数）を対象オブジェクトとしても送信先オブジェクトとしても使用できるため、ドラッグ&ドロップのユーザインタフェースは簡単に作成できます。例えば、ボタンのドラッグ&ドロップができます。

注：「ドラッグ可」になっているボタンをドラッグするには、Altボタン（WIndows）またはOptionボタン（Mac OS）を押す必要があります。3Dタイプのボタンだけは直接ドラッグすることができます。

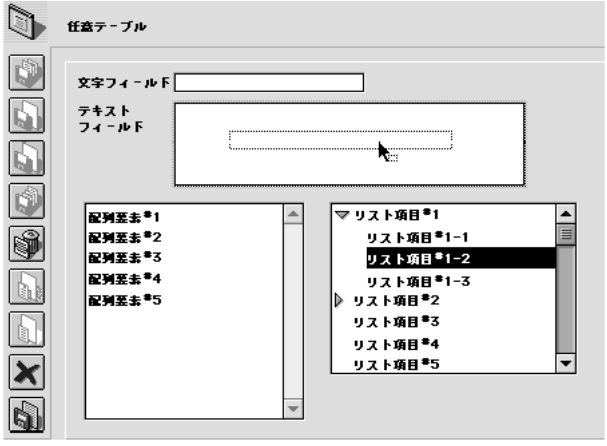
注：ドラッグとドロップの両方ができるオブジェクトは、ユーザが禁止しない限り、それ自体にもドロップできます。詳細については、以下の説明を参照してください。

以下の図は、選択したオブジェクトに対して「プロパティリスト」ウインドウで「ドラッグ可」プロパティと「ドロップ可」プロパティを設定した状態を示しています。



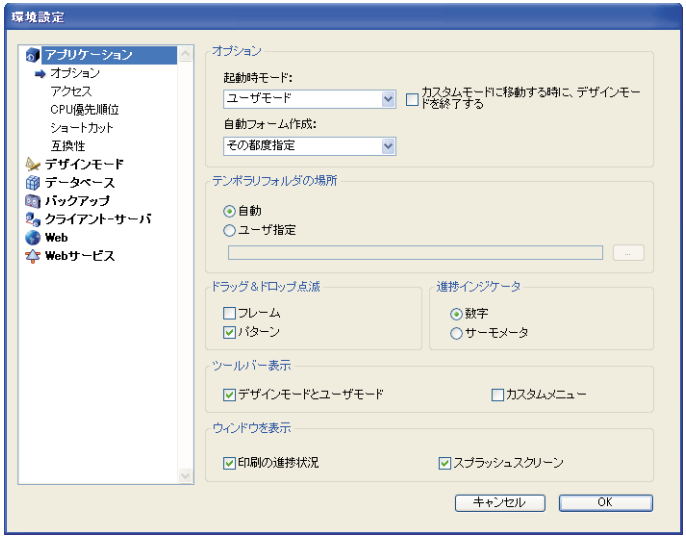
ドラッグ&ドロップのユーザインタフェース処理

4th Dimensionはドラッグ&ドロップ機能のユーザインタフェース部分に対応しています。ドラッグ可能なオブジェクトをクリックしてから、マウスをドラッグすると、4th Dimensionがオブジェクトをドラッグします。4th Dimensionはマウスの動きに従って点線の矩形を画面に表示することでこの操作を反映します。以下の図では、階層リストの項目がテキストフィールド上にドラッグされています。



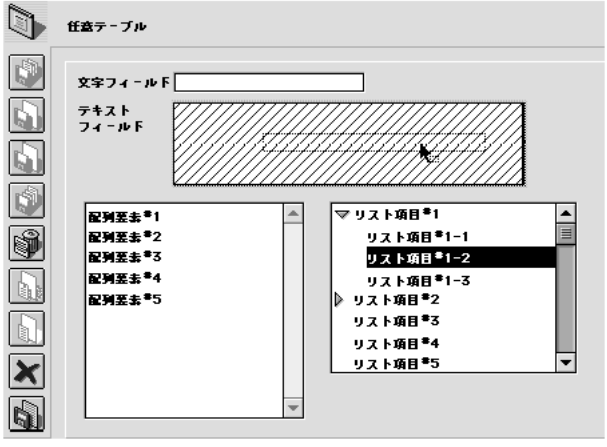
テキストフィールドエリアの周りに反転したグレーのフレームのハイライト（点滅）があることに注目してください。このハイライトは、送信先オブジェクト（上の図では、テキストフィールド）を示します。この箇所でマウスボタンを放すと、4th Dimensionは、ユーザがハイライトされた送信先オブジェクトにドラッグしたオブジェクトをドロップしたいものと想定します。

「環境設定」ダイアログボックスの「アプリケーション->オプション」では、以下の図のように送信先オブジェクトのドラッグ&ドロップ時のハイライト（点滅）をフレームまたはパターン（または、その両方）に設定することができます。



デフォルトのハイライト（点滅）は、「フレーム」です。これは、オブジェクトの周りを囲むグレーの反転した矩形です。色付きの背景または色付きのオブジェクトフレームを使用する場合、このハイライト（点滅）を使用するとわかりにくくなります。この場合は、別の手段として、「パターン」ハイライトを使用します。これは、以下に示すように送信先オブジェクトを斜線パターンで塗りつぶします。

以下の図は、テキストフィールドにドラッグされる階層リストの項目です。



以下では、配列要素がその配列にドラッグされています。



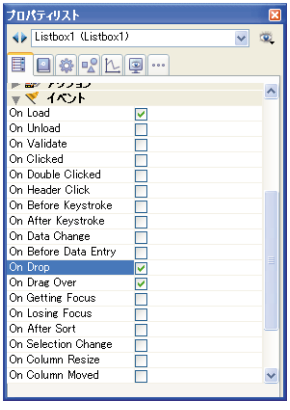
また、ハイライトの両方のタイプを選択することもできます。

注：送信先オブジェクトが配列（スクロール可能エリア）または階層リストの場合には、送信先オブジェクトのハイライトは、要素または項目に「従い」ます。

ドラッグ&ドロップのプログラムによる処理

4th Dimensionはドラッグ&ドロップのユーザインタフェース部分を実行します。そのプログラム部分の実行はユーザに任されています。それを実行するために、4th Dimensionは2つのフォームイベント、「**On Drag Over**」と「**On Drop**」を用意しています。両方のイベントとも送信先オブジェクトに送信されます。ドラッグ&ドロップの処理中、対象オブジェクトのオブジェクトメソッドはまったく関係しません。

On Drag Overと**On Drop**を受け付けるために、送信先オブジェクトは以下の図で示すように、これらの2つのイベントを「プロパティリスト」ウインドウでアクティブにする必要があります。



On Drag Overイベント

On Drag Overイベントは、マウスポインタがオブジェクトの上に移動した時に、繰り返し送信先オブジェクトに送信されます。このイベントの応答として、開発者は通常、以下のことを行います。

■ **DRAG AND DROP PROPERTIES**コマンドを呼び出します。これによって、対象オブジェクトに関する情報を得られます。

■ （オブジェクトメソッドが現在実行されている）送信先オブジェクトと対象オブジェクトの両方の性質とタイプに応じて、ドラッグ&ドロップの受け付けまたは拒否を行います。

ドラッグを受け付けるには、送信先オブジェクトメソッドが0（ゼロ）を返す必要があるため、「$0:=0」と記述します。ドラッグを拒否するには、オブジェクトメソッドが-1（マイナス1）を返す必要があるため、「$0:=-1」と記述します。**On Drag Over**イベント中に、4th Dimensionはこのオブジェクトメソッドを関数として扱います。結果が返されない場合には、4th Dimensionはドラッグが受け付けられたものと認識します。

ドラッグを受け付けた場合には、送信先オブジェクトがハイライトされます。ドラッグを拒否した場合には、送信先オブジェクトはハイライトされません。ドラッグを受け付けることは、ドラッグされたデータが送信先オブジェクトに挿入されるという意味にはなりません。これは、単にマウスボタンをこの場所で離したときに、ドラッグされたデータが送信先オブジェクトによって受け付けられることを意味するだけです。

ドロップ可能なオブジェクトに対して開発者が**On Drag Over**イベントを処理しない場合には、そのオブジェクトは、ドラッグされたデータの性質やタイプに関係なく、すべてのドラッグ処理に対してハイライトされます。

On Drag Overイベントは、ドラッグ&ドロップ処理の最初の段階を制御する手段です。ドラッグされたデータが送信先オブジェクトと互換性のあるタイプかどうかをテストでき、また、ドラッグの受け付けや拒否をできるだけでなく、4th Dimensionがあなたの判断に基づいて送信先オブジェクトをハイライトした（またはしなかった）ため、この処理が行われたことを同時にユーザに通知することができます。

On Drag Overイベントはマウスの移動に従って、現在の送信先オブジェクトに対して繰り返し送信されるため、このイベントのコード処理は短く、短期間で実行できるようにしてください。

警告：ドラッグ&ドロップがプロセス間のドラッグ&ドロップである場合、つまり、対象オブジェクトが送信先オブジェクトのプロセス（ウインドウ）とは異なるプロセス（ウインドウ）にある場合は、On Drag Overイベントに対する送信先オブジェクトのオブジェクトメソッドは対象プロセスのコンテキスト内（対象オブジェクトのプロセス）で実行され、送信先オブジェクトのプロセスでは実行されません。これはそのような実行が発生した場合に限ります。この種の実行の長所については、この章の最後で説明します。

On Dropイベント

On Dropイベントはマウスポインタが送信先オブジェクトに対して離されたときにそのオブジェクトに一度送信されます。このイベントはドラッグ&ドロップ処理の第2段階であり、ユーザアクションの応答として処理を実行します。

このイベントは、**On Drag Over**イベント中にドラッグが受け付けられなかった場合には、オブジェクトに送信されません。オブジェクトに対して**On Drag Over**イベントを処理し、ドラッグを拒否した場合には、**On Drop**イベントは発生しません。つまり、**On Drag Over**イベント中に対象オブジェクトと送信先オブジェクト間のデータタイプの互換性をテストして、有効なドロップを受け付けた場合には、**On Drop**中にデータの再テストをする必要はありません。データが送信先オブジェクトに対して適切であることは既にわかっているためです。

4th Dimensionのドラッグ&ドロップを実現する上での興味深い点は、必要なことは何でもできるというところです。以下の例を参照してください。

■ 階層リストの項目がテキストフィールドに対してドロップされる場合には、テキストフィールドの最初、最後、または途中にリスト項目のテキストを挿入できます。

■ フォームには2つの状態のピクチャボタンが含まれており、それぞれごみ箱が空であるか、いっぱいであるかを表わすものとします。オブジェクトをそのボタンにドロップすることは、(ユーザインタフェースの立場からすると)「ごみ箱にドラッグされドロップされたオブジェクトを削除すること」です。この時、ドラッグ&ドロップはある場所から別の場所にデータを移動しません。その代わりにアクションを実行します。

■ フローティングウインドウからフォーム内のオブジェクトに配列要素をドラッグすることは、「このウインドウで、データベースに格納されている顧客をリストしているフローティングウインドウからドラッグ&ドロップした名前を持つ顧客レコードを表示せよ」という意味になります。

■ その他

上記の例からもおわかりのように、4th Dimensionのドラッグ&ドロップインタフェースはユーザが考えた任意のユーザインタフェースメタファを実現できるフレームワークです。

ドラッグ&ドロップのコマンド

DRAG AND DROP PROPERTIESコマンドは、以下を返します。

■ ドラッグされたオブジェクト（フィールドまたは変数）へのポインタ

■ ドラッグされたオブジェクトが配列要素またはリスト項目の場合には、配列要素番号または項目番号。

■ 対象プロセスのプロセス番号

Drop positionコマンドは、送信先オブジェクトが配列（スクロール可能エリア）または階層リストの場合には、目的の要素またはリスト項目の項目位置を示す要素番号を返します。

RESOLVE POINTERや**Type**のようなコマンドは、ソースオブジェクトの属性やタイプを調べる際に有効です。

ドラッグされたデータをコピーする目的でドラッグ＆ドラッグ操作が行われた場合、これらのコマンドの機能は関わっているプロセスの数によって変わります。

■ ドラッグ＆ドロップが1つのプロセスに限定されている場合、これらのコマンドを使い、適当な動作を実行します（つまり、ソースオブジェクトを送信先オブジェクトに割り当てるだけです）。

■ ドラッグ＆ドロップがプロセス間のドラッグ＆ドロップである場合、ドラッグされたデータへのアクセスには注意が必要です。つまり、ソースプロセスからデータのインスタンスにアクセスしなくてはなりません。ドラッグされたデータが変数の場合、**GET PROCESS VARIABLE**を使用して正しい値を取得します。ドラッグされたデータがフィールドの場合、これら2つのプロセスではテーブルのカレントレコードが異なる可能性が高いということに留意してください。したがって、正しいレコードにアクセスする必要があります。

2番目のケースについては解決策が複数あります。

■ 送信先のオブジェクトメソッドに対する**On Drag Over**イベントがソースプロセスのコンテキスト内で実行される場合、フィールドデータやレコード番号をインタープロセス変数にコピーし、**On Drop**イベントでこれを再利用します。

■ **On Drop**イベントでプロセス間通信を開始して、必要なデータを取得することができます。

ドラッグ&ドロップがデータを移動するためではなく、特定の処理のためのユーザインタフェースメタファである場合には、何でも希望することを実行できます。

参照

Drop position、DRAG AND DROP PROPERTIES、Form event

Drop position

Drop position → 整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	送信先配列要素番号、リスト項目位置、あるいは、最後の配列要素または最後のリスト項目を超えてドロップが発生した場合は-1

説明

Drop position関数は、オブジェクトがドラッグされ、ドロップされた配列要素番号またはリスト項目位置を返します。

一般に、配列または階層リストに対して発生したドラッグ&ドロップイベントを処理している時に**Drop position**関数を使用します。

送信先オブジェクトが配列の場合には、この関数は配列要素の番号を返します。送信先オブジェクトが階層リストの場合には、この関数はリスト項目の位置を返します。どちらの場合も、ソースオブジェクトが最後の配列要素または最後のリスト項目を超えてドロップされた場合には、この関数は-1を返します。

ドラッグ&ドロップイベントでははく、配列や階層リストに対して発生したイベントを処理している間に**Drop position**関数を呼び出すと、この関数は-1を返します。

重要：フォームオブジェクトは、その「ドロップ可」プロパティが選択されている場合にはドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag OverまたはOn Dropあるいはその両方に対して、これらのイベントを処理するためにアクティブにする必要があります。

例題

▼ **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照

Drop position、DRAG AND DROP PROPERTIES、Form event

DRAG AND DROP PROPERTIES

DRAG AND DROP PROPERTIES(対象オブジェクト;対象要素;対象プロセス)

引数	タイプ		説明
対象オブジェクト	ポインタ	←	ドラッグ&ドロップされた対象オブジェクトのポインタ
対象要素	数値	←	ドラッグされた配列要素番号、または、ドラッグされた階層リスト項目、あるいは、対象オブジェクトが配列でもリストでもない場合には-1
対象プロセス	数値	←	対象プロセス番号

説明

DRAG AND DROP PROPERTIESコマンドにより、オブジェクトに対する**On Drag Over**イベントまたは**On Drop**イベントが発生した時に対象オブジェクトについての情報を取得することができます。

一般に、**On Drag Over**イベントまたは**On Drop**イベントが発生する（送信先オブジェクト）オブジェクトのオブジェクトメソッド（または、呼び出すサブルーチンの1つ）の内部から**DRAG AND DROP PROPERTIES**コマンドを使用します。

重要：フォームオブジェクトは、その「ドロップ可」プロパティが選択されている場合にはドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag OverまたはOn Dropあるいはその両方に対して、これらのイベントを処理するためにアクティブにする必要があります。

このコマンドの呼び出しの後には、以下が発生します。

■ 引数<対象オブジェクト>は、対象オブジェクト（ドラッグ&ドロップされたオブジェクト）へのポインタです。このオブジェクトは、送信先オブジェクト（**On Drag Over**イベントまたは**On Drop**イベントが発生するオブジェクト）または異なるオブジェクトである可能性がある点に注意してください。同一のオブジェクト間でデータのドラッグとドロップを実行することは、配列や階層リストでは便利です。これは、ユーザが配列またはリストを手作業でソートできる簡単な方法です。

■ ドラッグ&ドロップされたデータが配列要素（配列になる対象オブジェクトの要素）である場合には、引数<対象要素>はその配列の要素番号を返します。それ以外の場合には、ドラッグ&ドロップされたデータがリスト項目（階層リストになる対象オブジェクトの項目）である場合には、引数<対象要素>はそのリスト項目の位置を返します。上記以外の場合、つまりソースオブジェクトが配列でも階層リストでもない場合には、引数<対象要素>は-1を返します。

■ ドラッグ&ドロップ処理はプロセス間でも発生します。引数＜対象プロセス＞は、対象オブジェクトが属するプロセス番号と同じです。この引数の値をテストすることが重要です。同一プロセス内のドラッグ&ドロップに応答するのは、単に対象データから送信先オブジェクトにコピーするだけです。一方で、プロセス間のドラッグ&ドロップを扱う場合、**GET PROCESS VARIABLE**コマンドを使用して対象プロセスのオブジェクトのインスタンスから対象データを取得します。対象オブジェクトがフィールドの場合、プロセス間通信により対象プロセスの値を取得するか、またはOn Drag Overイベントに応答しながらその特殊なケースを処理しなければなりません（後述）。

しかし通常は、対象変数（配列およびリスト）からデータ入力エリア（フィールドまたは変数）へ向けてドラッグ&ドロップユーザインタフェースをインプリメントします。

ドラッグ&ドロップイベントがないところで**DRAG AND DROP PROPERTIES**コマンドを呼び出すと、引数＜対象オブジェクト＞はヌル値を返し、＜対象要素＞は-1を、＜対象プロセス＞は0を返します。

Tips：4th Dimensionはドラッグ&ドロップのグラフィカルな部分を自動的に処理します。次にユーザは適切な方法でイベントに応答する必要があります。以下の例では、イベントに応答してドラッグされたデータをコピーしています。別の方法として洗練されたユーザインタフェースをインプリメントできます。例えば、フローティングウインドウから配列要素をドラッグ&ドロップすることにより、送信先ウインドウ（送信先オブジェクトが存在するウインドウ）に構造化されたデータ（例: 対象配列要素によって特定されたレコードのいくつかのフィールド）を入れるような処理です。

On Drag Overイベント中に送信先オブジェクトがドラッグ&ドロップ処理を受け付けるかどうかを対象オブジェクトのタイプや性質（または他の理由）に従って判断するには、**DRAG AND DROP PROPERTIES**コマンドを使用します。ドラッグ&ドロップを受け付ける場合には、オブジェクトメソッドは「$0:=0」を返す必要があります。ドラッグ&ドロップを受け付けない場合には、オブジェクトメソッドは「$0:=-1」を返す必要があります。ドラッグ&ドロップを受け付けたか、拒否したかは、画面に反映されます。つまり、オブジェクトがドラッグ&ドロップ処理の送信先となる場合にはハイライトされ、送信先にならない場合にはハイライトされません。

Tips：On Drag Overイベント中に、送信先オブジェクトのオブジェクトメソッドは、対象オブジェクトのプロセスのコンテキスト内で実行されます。プロセス間ドラッグ&ドロップの対象オブジェクトがフィールドである場合には、このイベントの機会を利用して対象データをインタープロセス変数にコピーできます。そうすることによって、後で、On Dropイベント中に、ドラッグされたフィールドの値を取得するために対象プロセスとのプロセス間通信を開始する必要がなくなります。プロセス間ドラッグ&ドロップに対象オブジェクトとして変数が含まれる場合には、On Dropイベント中にGET PROCESS VARIABLEコマンドを使用できます。

例題

1. データベースフォームの中には、スクロールエリアがあって、そのスクロールエリアのある部分から別の部分へドラッグ&ドロップするだけで要素の順序を手作業で変えたい場合があります。それぞれの状況に応じて特定のコードを書くよりも、これらのスクロールエリアの任意のものを処理する汎用プロジェクトメソッドを作成することができます。この場合には、以下のようなコードを作成できます。

```
`配列自体のドラッグ&ドロップ　プロジェクトメソッドの処理
`配列自体のドラッグ&ドロップ（ポインタ）→ブール値
`配列自体のドラッグ&ドロップ（->配列）→配列自体のドラッグ&ドロップだったもの
Case of
    ¥(Form event=On Drag Over)
        DRAG AND DROP PROPERTIES($vpSrcObj ; $vlSrcElem ;
                                                    $vlPID)
        If($vpSrcObj=$1)
            `配列からそれ自体に対するドラッグ&ドロップである場合には
                                                    それを受け付ける
            $0:=0
        Else
            $0:=-1
        End if
    ¥(Form event=On Drop)
        `ドラッグ&ドロップ対象オブジェクトについての情報を取得する
        DRAG AND DROP PROPERTIES($vpSrcObj ; $vlSrcElem ; $vlPID)
        `送信先の配列要素番号を取得する
        $vlDstElem:=Drop position
        `配列要素が配列要素自体にドロップされなかった場合
        If($vlDstElem # $vlSrcElem)
            ` 配列の要素0にドラッグされた要素を保存する
            $1->{0}:=$1->{$vlSrcElem}
            `ドラッグされた配列要素を削除する
            DELETE ELEMENT($1-> ; $vlSrcElem)
            `送信先要素がドラッグされた要素を超える場合
            If($vlDstElem>$vlSrcElem)
                `送信先の配列要素番号から1引く
                $vlDstElem:=$vlDstElem-1
            End if
            `ドラッグ&ドロップが最後の要素を超えて発生した場合
            If($vlDstElem=-1)
                ` 送信先の配列要素番号を配列の最後で新しい要素に
                                                    設定する
```

```
                $vlDstElem:=Size of array($1->)+1
            End if
                `この新しい配列要素を挿入する
            INSERT ELEMENT($1-> ; $vlDstElem)
                `配列の要素ゼロに先ほど保存した値を設定する
            $1->{$vlDstElem}:=$1->{0}
            `要素は新しく選択された配列の要素になる
            $1->:=$vlDstElem
        End if
End case
```

このプロジェクトメソッドの作成後、以下のように使用します。

```
` anArrayスクロールエリアのオブジェクトメソッド
Case of
    `...
    ¥(Form event=On Drag Over)
        $0:=配列自体のドラッグ&ドロップ(Self)
    ¥(Form event=On Drop)
        配列自体のドラッグ&ドロップ(Self)
` ...
End case
```

2. データベースフォームの中にテキスト入力エリアがあり、そのエリアに各種コピー元からドラッグ&ドロップしたい場合があります。それぞれの状況に応じて特定のコードを書くのではなく、これらのテキスト入力エリアの任意のものを処理する汎用プロジェクトメソッドを作成することができます。この場合には、以下のようなコードを作成できます。

```
` テキストエリアへのドロップ処理　プロジェクトメソッド
` テキストエリアへのドロップ処理（ポインタ）
` テキストエリアへのドロップ処理（->テキスト変数または文字列変数）
Case of
` ドラッグ＆ドロップの受け入れや拒否のためにこのイベントを使用
    ¥(Form event=On Drag Over)
    ` 拒否するために$0を初期化
        $0:=-1
        ` ドラッグ＆ドロップするオブジェクトの情報を取得
        DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
        ` この例題では、あるオブジェクトからそれ自身へのドラッグ＆ドロップは
                                                    許可しない
        If($vpSrcObj $1)
        ` ドラッグされたデータタイプを取得
            $vlSrcType:=Type($vpSrcObj->)
```

```
            Case of
               ¥($vlSrcType=Is Alpha Field)
                  ` 英数字フィールドならOK
                  $0:=0
                  ` IP変数へ値をコピー
                  <>vtDraggedData:=$vpSrcObj->
               ¥($vlSrcType=Is Text)
                  ` テキストタイプのフィールドや変数はOK
                  $0:=0
                  RESOLVE POINTER($vpSrcObj;$vsVarName;$vlTableNum;
                                                        $vlFieldNum)
                  ` フィールドの場合
                  If(($vlTableNum>0) &($vlFieldNum>0))
                  ` IP 変数へ値をコピー
                     <>vtDraggedData:=$vpSrcObj->
                  End if
               ¥($vlSrcType=Is String Var)
                  ` 文字列変数はOK
                  $0:=0
               ¥(($vlSrcType=String array) |($vlSrcType=Text array))
                  ` 文字列配列およびテキスト配列はOK
                  $0:=0
               ¥(($vlSrcType=Is LongInt) |(lSrcType=Is Real)
                  If(Is a list($vpSrcObj->))
                  ` 階層リストはOK
                     $0:=0
                  End if
            End case
         End if
      ` このイベントを使用して実際のドラッグ＆ドロップ動作を実行
         ¥(Form event=On Drop)
            $vtDraggedData:=""
            ` ドラッグ＆ドロップするオブジェクトの情報を取得
            DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
            RESOLVE POINTER($vpSrcObj;$vsVarName;$vlTableNum;$vlFieldNum)
            ` フィールドの場合
            If(($vlTableNum>0) &($vlFieldNum>0))
            ` イベント中にIP 変数セットを取得
               $vtDraggedData:=<>vtDraggedData
            Else
            ` ドラッグされた変数タイプを取得
```

```
$vlSrcType:=Type($vpSrcObj->)
Case of
` 配列の場合
   ¥(($vlSrcType=String array) |($vlSrcType=Text array))
      If($vlPID Current process)
      ` この変数の対象プロセスのインスタンスから要素を読み込む
         GET PROCESS VARIABLE($vlPID;
         $vpSrcObj->{$vlSrcElem};$vtDraggedData)
      Else
      ` 配列要素をコピー
         $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
      End if
   ¥(($vlSrcType=Is LongInt) |( vlSrcType=Is Real)
      ` 階層リストの場合
      If(Is a list($vpSrcObj->))
      ` 別プロセスのリストの場合
         If($vlPID Current process)
         ` 別プロセスのリストリファレンスを取得
            GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vlList)
         Else
            $vlList:=$vpSrcObj->
         End if
         ` 位置を把握したアイテムのテキストを取得
         GET LIST ITEM($vlList;$vlSrcElem;$vlItemRef;$vsItemText)
         $vtDraggedData:=$vsItemText
      End if
   Else
   ` 文字列変数またはテキスト変数
      If($vlPID Current process)
         GET PROCESS VARIABLE($vlPID;$vpSrcObj->;
                                          $vtDraggedData)
      Else
         $vtDraggedData:=$vpSrcObj->
      End if
   End case
End if
` 実際にドロップするものがある場合（対象オブジェクトは空の可能性がある）
If($vtDraggedData "")
` テキスト変数の長さが32,000バイトを超えないか調べる
   If((Length($1->)+Length($vtDraggedData))<=32000)
      $1->:=$1->+$vtDraggedData
```

```
            Else
                BEEP
                ALERT("テキストが長すぎるため、ドラッグ＆ドロップを完了
                                                    できませんでした。")
            End if
        End if
    End case
```

このプロジェクトメソッドの作成後、以下のように使用します。

```
` [任意のテーブル]テキストフィールド オブジェクトメソッド
Case of
` ...
    ¥(Form event=On Drag Over)
        $0:=テキストエリアへのドロップ処理(Self)
    ¥(Form event=On Drop)
        テキストエリアへのドロップ処理(Self)
` ...
End case
```

参照

Drop position、DRAG AND DROP PROPERTIES、Form event

ACCEPT

ACCEPT

説明

ACCEPTコマンドは、以下の目的のために入力フォーム内で使用します。

■ **ADD RECORD**、**MODIFY RECORD**、**ADD SUBRECORD**、**MODIFY SUBRECORD**を使って開始されたデータ入力による新規または修正レコードやサブレコードを受け入れるため

■ **DIALOG**コマンドで表示されたフォームを受け入れるため

■ **DISPLAY SELECTION**や**MODIFY SELECTION**コマンドを使って、レコードセレクションを表示しているフォームを終了するため

ACCEPTコマンドは、ユーザが“enter”キーを押した場合と同じ動作を実行します。フォームが受け付けられた後、システム変数OKは1に設定されます。

一般に**ACCEPT**コマンドは、メニューを選択した結果として実行されます。また、「動作なし」属性ボタンのオブジェクトメソッド内でもよく使用されます。

また、**Open window**関数におけるオプションの「クローズボックス」メソッド内でもよく使用されます。あるウインドウにコントロールメニューボックス（Macintosh版では、クローズボックス）がある場合、**ACCEPT**や**CANCEL**コマンドはコントロールメニューボックスがクリックされたり、または「閉じる」メニューが選択された際に実行されるメソッドの中で呼び出されます。

ACCEPTコマンドは、待ち行列を作成することはできません。あるイベントに対して、メソッド内で2つの**ACCEPT**コマンドを続けて実行しても、1つの**ACCEPT**コマンドを実行したのと同じ効果しか得られません。

参照

CANCEL

CANCEL

CANCEL

説明

CANCELコマンドは、フォームやオブジェクトのメソッド（またはサブルーチン）で以下の目的のために使用されます。

- **ADD RECORD**、**MODIFY RECORD**、**ADD SUBRECORD**、**MODIFY SUBRECORD**を使って開始されたデータ入力による新規または修正レコードやサブレコードをキャンセルするため
- **DIALOG**コマンドで表示されたフォームをキャンセルするため
- **DISPLAY SELECTION**や**MODIFY SELECTION**コマンドを使って、レコードセレクションを表示しているフォームを終了するため
- **Print form**関数を使用して実行しようとするフォーム印刷をキャンセルするため（後述）。

データ入力中に、**CANCEL**コマンドは、ユーザがキャンセルキーコンビネーション（Windowsでは「Ctrl+.（ピリオド）」キー、Macintoshでは「command+.（ピリオド）」キー）を押した場合と同じ動作を実行します。

一般に**CANCEL**コマンドは、メニューコマンドの結果として実行されます。また、「動作なし」属性ボタンのオブジェクトメソッド内でもよく使用されます。

また、**Open window**関数におけるオプションの「クローズボックス」メソッド内でもよく使用されます。あるウインドウにコントロールメニューボックス（Macintosh版では、クローズボックス）がある場合、**ACCEPT**や**CANCEL**コマンドはコントロールメニューボックスがクリックされたり、または「閉じる」メニューが選択された際に実行されるメソッドの中で呼び出されます。

CANCELコマンドは、待ち行列を作成することができません。あるイベントに対してメソッド内で2つの**CANCEL**コマンドを続けて実行しても、1つの**CANCEL**コマンド実行をした場合と同じ効果しか得られません。

最後に、**Print form**関数を使用する際に、このコマンドを「On Printing Detail」フォームイベントで使用することができます。この状況において、**CANCEL**コマンドは実行直前の印刷を一時中断し、次のページに印刷を再開します。この仕組みを利用して、スペースが不足している場合やページブレークが必要な場合のフォーム印刷を管理することができます。

注：この操作は、PAGE BREAK(*) コマンドとは異なります。PAGE BREAK(*)コマンドは、印刷待ちであるすべてのフォームをキャンセルします。

例題

SET PRINT MARKERコマンドの例題を参照してください。

参照

ACCEPT、PAGE BREAK、Print form

システム変数とセット

CANCELコマンドを実行すると（フォームまたは印刷がキャンセルされ)、システム変数OKには0が代入されます。

Keystroke

Keystroke → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	ユーザによって入力された文字

説明

Keystroke関数は、ユーザがフィールドや入力可能エリアに入力した文字を返します。

通常、**On Keystroke**イベントを操作中に、フォームやオブジェクトメソッド内で**Before Keystroke**関数を呼び出します。キーストロークイベントを検出するには、**Form event**関数を使用します。

注：Keystrokeはサブフォームでは機能しません。

ユーザが実際に入力した文字を他の文字と置き換えるには、**FILTER KEYSTROKE**コマンドを使用します。

注：新しく入力する文字同様に、編集中の入力可能エリアの現在値に応じ、“即効で”処理を行いたい場合、以下のことを覚えていてください。つまり、画面に表われているテキストは、編集中のエリア用のデータソースフィールドや変数の値にはまだなっていません。そのエリア用のデータ入力が有効になってから、データソースフィールドや変数に入力値が割り当てられます（例として、他のエリアへの図表化、ボタンのクリック等）。したがって、データ入力を変数に“投影”し、この投影した値を使用して動作を実行するかどうかはユーザ次第です。特有の機能を実行するために、現在のテキスト値を知る必要がある場合は、上記を実行する必要があります。

Get edeted text関数も使うことができます。

以下のことを実行する場合に**Keystroke**関数を使用します。

■ カスタマイズされた方法で文字をフィルタする場合

■ データ入力フィルタを使用しない方法でデータ入力を振い分ける場合

■ 動的ルックアップをインプリメントする場合

例題

1. **FILTER KEYSTROKE**コマンドの例題を参照してください。

2. **On Before Keystroke**イベントを処理する際、カレントテキストエリア（カーソルが置かれている）の編集処理を実行するのであって、このエリア用のデータソース（フィールドまたは変数）の“未来値”を処理するわけではありません。以下の“キーストローク処理”プロジェクトメソッドを使用すれば、どのテキストエリアのデータ入力も2番目の変数に投影できます。ユーザはこの2番目の変数を使用して、そのエリアに文字を入力しながら機能を実行できます。最初の引数として、そのエリアのデータソースへのポインタを渡し、それから第2引数として投影変数へのポインタを渡します。メソッドは投影変数内のテキストエリアの新しい値を返し、その値が最後に入力した文字が挿入される前の値と異なる場合はTrueを返します。

```
`「キーストローク処理」プロジェクトメソッド
`キーストローク処理（ポインタ ; ポインタ）→ ブール
`キーストローク処理（->ソースエリア ; -> 現在値）→ 新しい値
`入力可能エリア内のテキスト選択範囲を取得する
C_POINTER($1;$2)
C_TEXT($vtNewValue)
` 入力エリア内のテキスト選択範囲を取得
GET HIGHLIGHT($1-> ; $vlStart ; $vlEnd)
` 現在値で作業を開始する
$vtNewValue:=$2->
` 押されたキーや入力した文字に応じて、適切な処理が実行される
Case of
    `Backspace(delete) キーを押された
    ¥(Ascii(Keystroke)=Backspace )
        `選択した文字またはテキストカーソルの左にある文字が削除される
        $vtNewValue:=Substring($vtNewValue ; 1 ; $vlStart-1-
                Num($vlStart=$vlEnd))+Substring($vtNewValue ; $vlEnd)
        `受け付け可能な文字が入力された
    ¥(Position(Keystroke ; "abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
        If($vlStart#$vlEnd)
            `1文字以上の文字が選択され、キーストロークがこれらを
                                                無効にする
            $vtNewValue:=Substring($vtNewValue ; 1 ; $vlStart-1)
                        +Keystroke+Substring($vtNewValue ; $vlEnd)
        Else
            `テキスト選択がテキストカーソル
            Case of
                `テキストカーソルは現在、テキストの冒頭
                ¥($vlStart<=1)
```

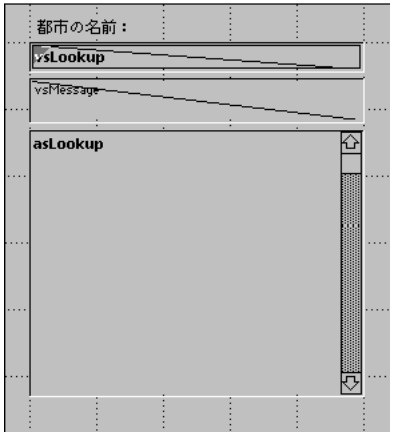
```
                                    `テキストの冒頭に文字を挿入する
                                    $vtNewValue:=Keystroke+$vtNewValue
                                        `テキストカーソルは現在、テキストの
                                                                末尾にある
                            ¥($vlStart>=Length($vtNewValue))
                                    `テキストの末尾に文字を加える
                                    $vtNewValue:=$vtNewValue+Keystroke
                        Else
                            `テキストカーソルはテキスト内の任意の場所にあり、
                            `そこに新しい文字を挿入する
                            $vtNewValue:=Substring($vtNewValue ; 1 ; $vlStart-1)
                                +Keystroke+Substring($vtNewValue ; $vlStart)
                        End case
                End if
                `矢印キーが押された
                `キーストロークを受け付けるだけで何も行わない
            ¥(Ascii(Keystroke)=Left Arrow Key )
            ¥(Ascii(Keystroke)=Right Arrow Key )
            ¥(Ascii(Keystroke)=Up Arrow Key )
            ¥(Ascii(Keystroke)=Down Arrow Key )
        Else
            `文字、アラビア数字、スペース、ダッシュ以外の文字を受け付けない
            FILTER KEYSTROKE("")
        End case
        `値が現在、前と異なっている場合
        $0:=($vtNewValue#$2->)
            `以下のキーストローク操作用の値を返す
        $2->:=$vtNewValue
```

このプロジェクトメソッドをアプリケーションに追加した後、以下のようにこのメソッドを使用することができます。

```
    `「MyObject」 入力可能エリアのオブジェクトメソッド
    Case of
        ¥(Form event=On Load)
            MyObject:=""
            MyShadowObject:=""
        ¥(Form event=On Before Keystroke)
            If(キーストローク処理(->MyObject ; ->MyShadowObject))
                `MyShadowObject 内に格納されている値を使用しての適当
                                                        な処理を実行
            End if
```

End case

以下のフォームの一部を検証してみましょう。



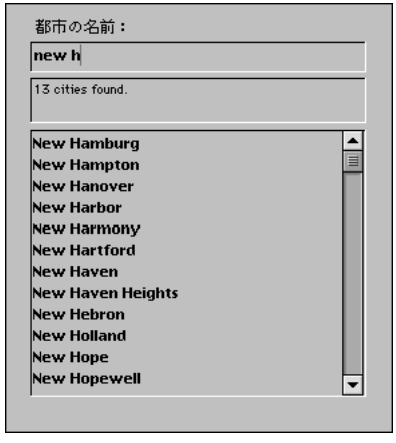
これは、入力可能エリア「vsLookup」、入力不可エリア「vsMessage」、スクロールエリア「asLookup」オブジェクトで構成されています。「vsLookup」内に文字を入力しながら、このオブジェクト用のメソッドは[米国郵便番号]テーブルに対するクエリを実行します。これによりユーザは都市名の最初の文字を入力するだけで米国の都市を見つけることができます。

下記に「vsLookup」入力可能エリアのオブジェクトメソッドを示します。

```
`「vsLookup」入力可能エリアのオブジェクトメソッド
Case of
	¥(Form event=On Load)
		vsLookup:=""
		vsResult:=""
		vsMessage:="検索したい都市の最初の文字を入力してください。"
		CLEAR VARIABLE(asLookup)
	¥(Form event=On Before Keystroke )
		If( キーストローク処理(->vsLookup ; ->vsResult))
			If(vsResult#"")
				QUERY([米国郵便番号];[米国郵便番号]都市
				                                =vsResult+"@")
				MESSAGES OFF
				DISTINCT VALUES([米国郵便番号]都市 ; asLookup)
				MESSAGES ON
				$vlResult:=Size of array(asLookup)
				Case of
					¥($vlResult=0)
```

```
                        vsMessage:="都市はありません。"
                    ¥($vlResult=1)
                        vsMessage:="1つの都市を見つけました"
                Else
                    vsMessage:=String($vlResult)+" 都市が見つかりました"
                End case
            Else
                DELETE ELEMENT(asLookup ; 1 ; Size of array(asLookup))
                vsMessage:="検索したい都市の最初の文字を入力してください。"
            End if
        End if
    End case
```

以下の図は、「ユーザ」モードでフォームを表示したものです。



4th Dimensionのプロセス間通信機能を利用すると、これと同じようにユーザインターフェースを構築して、プロセスとの通信を実行するフローティングウインドウで検索機能を提供し、そのウインドウでレコードの一覧や編集を実行することができます。

参照

FILTER KEYSTROKE、Form event、Get edited text

FILTER KEYSTROKE

FILTER KEYSTROKE(フィルタ文字)

引数	タイプ		説明
フィルタ文字	文字列	→	フィルタしたキーストローク文字またはキーストロークをキャンセルするには空の文字列

説明

FILTER KEYSTROKEコマンドを使用すれば、ユーザがフィルードや入力可能エリアに入力した文字を引数<フィルタ文字>の最初の文字と置き換えることができます。

空の文字列を受け渡すと、キーストロークはキャンセルされ無視されます。

通常、**On Before Keystroke**フォームイベントの処理中に、フォームやオブジェクトメソッド内で**FILTER KEYSTROKE**コマンドをコールします。キーストロークイベントを見つけるには、**Form event**関数を使用します。実際のキーストロークを得るには、**Keystroke**関数を使用します。

注：FILTER KEYSTROKEコマンドを使用すれば、ユーザが入力した文字をキャンセル、または他の文字と置き換えることができます。一方、特定のキーストロークに対して1文字以上の文字を挿入したい場合は、以下のことを覚えていてください。つまり、画面に表れているテキストは、編集中のエリア用のデータソースフィールドや変数の値にはまだなっていません。エリア用のデータ入力が有効になってから、データソースフィールドや変数に入力値が割り当てられます。したがって、データ入力を変数に投影し、この投影した値を使用して動作を行い、そして入力可能エリアの再割り当てを実行するかどうかはユーザ次第となります（下記の例題を参照してください）。

以下のことを実行する場合に**FILTER KEYSTROKE**コマンドを使用します。

■ カスタマイズされた方法で文字をフィルタする場合

■ データ入力フィルタを使用しない方法でデータ入力をフィルタする場合

■ 動的ルックアップを実施する場合

警告：FILTER KEYSTROKEコマンドを呼び出してからKeystroke関数を呼び出すと、実際に入力した文字の代わりにこのコマンドに渡した文字が返されます。

例題

1. 以下のコードを使用します。

```
`「MyObject」入力可能エリアのオブジェクトメソッド
Case of
	¥(Form event=On Load )
		MyObject:=""
	¥(Form event=On Before Keystroke )
		If(Position(Keystroke ; "0123456789")>0)
			FILTER KEYSTROKE("*")
		End if
End case
```

「MyObject」エリアに入力した数字は、すべてアスタリスク文字に変換されます。

2. 以下のコードは、「パスワード入力」エリアの働きを実施します。このエリア内では、入力した文字がすべて（画面上は）ランダム文字群に置き換えられます。

```
`「vsPassword」入力可能エリアのオブジェクトメソッド
Case of
	¥(Form event=On Load )
		vsPassword:=""
		vsActualPassword:=""
	¥(Form event=On Before Keystroke )
		キーストローク処理(->vsPassword ; ->vsActualPassword)
		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
```

データ入力が有効になったら、変数「vsActualPassword」内でユーザが入力した正確なパスワードを取り出します。

注：「キーストローク処理」プロジェクトメソッドは、Keystroke関数の例題の項に示されています。

3. アプリケーション内には、いくつかの文を入力できるテキストエリアがあります。また、このアプリケーションには、データベースを通してよく使用される用語の辞書テーブルが含まれています。テキストエリアを編集しながら、あるテキストエリアで選択された文字群をもとにして、辞書の見出し項目の取得や挿入をすぐに行いたい場合、以下の2つの方法があります。

■ 関連キーを持つボタンを準備する

または

■ テキストエリア編集中に特定のキーストロークを捉える

この例題は、ヘルプキーにもとづいて2番目の方法を実現しています。

上記の説明にあるように、テキストエリア編集中に、データ入力を有効にした後、このエリア用のデータソースに入力値が割り当てられます。このエリアの編集中に、辞書の見出し項目を取り出す、またはテキストエリアに挿入するためには、データ入力を投影することが必要となります。第1、第2の引数として入力可能エリアと投影変数へのポインタを渡します。そして第3の引数として“禁止”文字の文字列を渡します。キーストロークをどのように扱った場合でも、このメソッドは元のキーストロークを返します。“禁止”文字とは、ユーザが入力可能エリアに挿入されたくない文字また特殊文字として扱いたい文字を指します。

```
` 「投影キーストローク」プロジェクトメソッド
` 投影キーストローク（ポインタ ; ポインタ ; 文字）→ 文字列
` 投影キーストローク（->ソースエリア;->現在値;フィルタ）→元のキーストローク
C_STRING(1;$0)
C_POINTER($1;$2)
C_TEXT($vtNewValue)
C_STRING(255;$3)
` 元のキーストロークを返す
$0:=Keystroke
    `入力可能エリア内のテキスト選択範囲を取得
GET HIGHLIGHT($1-> ; $vlStart ; $vlEnd)
    `現在値で作業を開始
$vtNewValue:=$2->
    `押されたキーや入力された文字に応じて、適当な動作を実行
Case of
    `Backspace(delete) キーを押した場合
    ¥(Ascii($0)=Backspace )
        `選択した文字またはテキストカーソルの左にある文字を削除
        $vtNewValue:= Delete text($vtNewValue;$vlStart;$vlEnd)
        `矢印キーが押された
        `キーストロークを受け付けるだけで何も行わない
```

```
        ¥(Ascii($0)=Left Arrow Key )
        ¥(Ascii($0)=Right Arrow Key )
        ¥(Ascii($0)=Up Arrow Key )
        ¥(Ascii($0)=Down Arrow Key )
        ` 受け付け可能な文字が入力された
        ¥(Position($0 ; $3)=0)
        $vtNewValue:= Insert text($vtNewValue ; $vlStart ; $vlEnd ; $0)
    Else
        ` 文字が受け付けられない
        FILTER KEYSTROKE("")
    End case
    ` 以下のキーストローク処理用の値を返す。
    $2->:=$vtNewValue
    このメソッドでは以下の2つのサブメソッドを使用しています。
    `「Delete text」プロジェクトメソッド
    ` Delete text(文字 ; 倍長整数 ; 倍長整数) → 文字列
    ` Delete text(->テキスト ; 選択開始 ; 選択最終) → 新しいテキスト
    C_TEXT($0;$1)
    C_LONGINT($2;$3)
    $0:=Substring($1 ; 1 ; $2-1-Num($2=$3))+Substring($1 ; $3)
    `「Insert text」プロジェクトメソッド
    ` Insert text(文字列 ; 倍長整数 ; 倍長整数 ; 文字列) → 文字列
    ` Insert text(->ソーステキスト ; 選択開始 ; 選択最終 ;挿入テキスト) ->新しいテキスト
    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
```

これらのプロジェクトメソッドをプロジェクトに追加したら、これを以下のように使用することができます。

```
`「vsDescription」入力可能エリアのオブジェクトメソッド
Case of
	¥(Form event=On Load )
		vsDescription:=""
		vsShadowDescription:=""
		`特殊キーとして扱う「禁止」文字のリストを作成
		`（この例題では、ヘルプキーのみがフィルタされる）
		vsSpecialKeys:=Char(HelpKey)
	¥(Form event=On Before Keystroke )
		$vsKey:=投影キーストローク(->vsDescription ; ->vsShadowDescription
                                                        ;vsSpecialKeys)
		Case of
			¥(Ascii($vsKey)=Help Key )
			`Helpキーが押された場合に処理を実行する
			`この例題では、辞書の見出し項目が検索され挿入される
				辞書検索(->vsDescription ; ->vsShadowDescription)
		End case
End case
```

「辞書検索」プロジェクトメソッドを次に示します。この目的は投影変数を使用して、編集中の入力可能エリアを再割り当てすることです。

```
`「辞書検索」プロジェクトメソッド
`辞書検索（ポインタ ; ポインタ）
`辞書検索（-> 入力可能エリア ; ->投影変数）
`入力可能領域内のテキスト選択範囲を取得してください。
C_POINTER($1;$2)
C_LONGINT($vlStart;$vlEnd)
` 入力可能エリア内のテキスト選択範囲を取得
GET HIGHLIGHT($1-> ; $vlStart ; $vlEnd)
	`選択したテキストまたはテキストカーソルの左にある単語を取得
$vtHighlightedText:= Get highlighted text($2-> ; $vlStart ; $vlEnd)
	`検索すべきものがあるか？
If($vtHighlightedText#"")
		`テキスト選択がテキストカーソルで行われる場合、
		`その選択はテキストカーソルの後に続く単語から始まります。
	If($vlStart=$vlEnd)
		$vlStart:=$vlStart-Length($vtHighlightedText)
	End if
		`最初に使用可能な辞書見出しを検索します。
	QUERY([辞書];[辞書]見出し=$vtHighlightedText+"@")
		`1つあった場合
```

```
        If(Records in selection([辞書])>0)
            `あれば、投影テキスト内に挿入します。
            $2->:= Insert text($2-> ; $vlStart ; $vlEnd ; [辞書]見出し)
            `投影テキストを編集中の入力可能テキストにコピーします。
            $1->:=$2->
            `挿入した辞書見出しのすぐ後に選択を設定します。
            $vlEnd:=$vlStart+Length([辞書]見出し)
            HIGHLIGHT TEXT(vsComments ; $vlEnd ; $vlEnd)
        Else
            `辞書テーブルに合致する見出しがない場合。
            BEEP
        End if
    Else
    ` 反転表示されたテキストがない場合
    BEEP
    End if
```

「Get highlight text」メソッドを次に示します。

```
    `「Get highlighted text」プロジェクトメソッド
    ` Get highlighted text(文字 ; 倍長整数 ; 倍長整数 ) -> 文字列
    ` Get highlighted text(テキスト ; 選択開始 ; 選択最終) -> 反転表示されたテキスト
    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
```

参照

Form event、Keystroke

GOTO AREA

GOTO AREA({*;}オブジェクト)

引数	タイプ		説明
*	*	→	指定している場合=オブジェクトはオブジェクト名（文字列） 省略している場合=オブジェクトはフィールドまたは変数
オブジェクト	オブジェクト	→	オブジェクト名（*が指定されている場合）または、フィールドか変数（*が省略されている場合）

説明

GOTO AREAコマンドは、データ入力オブジェクト＜オブジェクト＞をフォームのアクティブエリアとして選択するために使用されます。これはフィールドや変数をユーザがクリック、あるいはタブで移動することと同じです。

オプション引数＜*＞を指定した場合は、＜オブジェクト＞にオブジェクト名（文字列）を指定します。オプション引数＜*＞を省略した場合は、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなくフィールドまたは変数の参照を渡します（フィールドまたは変数のみ）。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」を参照してください。

注：このコマンドは入力フォームでのみ有効です。サブフォームの入力エリアには、このコマンドは無効です。

例題

(1)**GOTO AREA**の2通りの使い方

```
GOTO AREA([Personnel]Name)  ` フィールドまたは変数の参照
GOTO AREA(*; “MyVar”)  `オブジェクト名での参照
```

(2)**REJECT**コマンドの例を参照してください。

参照

REJECT

REJECT

REJECT {(フィールド)}

引数	タイプ	説明
フィールド	フィールド	取り消すフィールド

説明

REJECTコマンドには、2つの形式があります。第1の形式は、引数がありません。これは、データ入力全体を取り消し、ユーザは強制的にフォーム上にとどまります。第2の形式は、＜フィールド＞だけを取り消し、ユーザは強制的にそのフィールド上にとどまります。

注：このコマンドを使用する前に、4th Dimension内に組み込まれているデータチェックツールを使用してください。

REJECTコマンドの第1の形式では、完全でないレコードをユーザが受け入れないようにします。**REJECT**を使用しなくても同じ効果を得ることができます。つまり、フィールドが正しく入力された後、「動作なし」ボタンと数値キーパッド上の“enter”キーの組み合わせに、**ACCEPT**コマンドと**CANCEL**コマンドを使用してレコードの受け入れや取り消しを行います。**REJECT**コマンドの第1の形式よりも、この2番目の手法を利用することをお勧めします。

第1の形式を使用する場合には、**REJECT**コマンドを実行してユーザがレコードを受け入れないようにしますが、通常これはレコードが完全ではなかったり、不正確な入力が行われたために行います。ユーザがレコードを受け入れようとすると、**REJECT**コマンドの実行によりレコードが受け入れられません。そしてこのレコードはフォーム上に表示されたままになります。したがって、ユーザはレコードが受け入れられるか、あるいは取り消されるまでデータの入力を続行しなければなりません。

この形式の**REJECT**コマンドを実行するのに最適な場所は、数値キーパッド上の“enter”キーを関連キーとして持つ「登録」ボタンのオブジェクトメソッドです。この場合のデータのチェックは、レコードが受け入れられた場合にのみ実行されます。ユーザは“enter”キーを押して、データをチェックしないで済ませることはできません。

REJECTコマンドの第2の形式は、引数＜フィールド＞を用いて実行します。カーソルは、データ入力（フィールド）エリアにとどまります。この形式は、ユーザが正しい値を入力するように強制します。この場合、フィールドの修正の直後に**REJECT**コマンドを使用しなければなりません。**Modified**関数を使用して、修正されたかどうかを判定します。データ入力エリアのオブジェクトメソッドで、**REJECT**コマンドを使用することもできます。このコマンドは、サブフォームエリア上のフィールドには何の影響も与えません。

REJECTコマンドのいずれかの形式をフォームメソッドまたはフォームのオブジェクトメソッドに記述しなければなりません。あるテーブルのサブフォームの詳細フォームに**REJECT**コマンドを使用する場合は、詳細フォームのフォームメソッドまたはオブジェクトメソッドに記述します。

取り消されたフィールドのデータを選択するために、**HIGHLIGHT TEXT**コマンドを使用することができます。

例題

1. 次は銀行取引レコードの例です。「登録」ボタンのオブジェクトメソッドで第1の形式の**REJECT**コマンドを使用する方法を示しています。数値パッドキー上の“enter”キーは、「登録」ボタンの対応キーとして設定します。つまり、ユーザがレコードを受け入れるために“enter”キーを押した場合でも、このボタンのオブジェクトメソッドが実行されるようにします。取引が小切手で行われる場合には、小切手番号があるはずです。小切手番号がない場合は、ここのチェックで取り消します。

```
Case of
    ¥(([処理]種別="小切手") &([処理]小切手番号=""))｀小切手に番号がない場合
        ALERT("小切手番号を入力してください。")｀ユーザに警告する
        REJECT｀入力を取り消す
        GOTO AREA([処理]小切手番号)｀“小切手番号”フィールドに戻る
End case
```

2. 以下の例は、フィールド“[従業員]給与”のオブジェクトメソッドの一部です。このオブジェクトメソッドは、フィールド“[従業員]給与”を調べて200万円以下の場合には取り消しします。「フォーム」エディタでフィールドの最小値を指定しても、同じ処理を実現することができます。

```
If([従業員]給与 < 2000000)
    ALERT("給与を200万円以上にしてください。")
    REJECT([従業員]給与)
End if
```

参照

ACCEPT、CANCEL、GOTO AREA

EDIT ITEM

EDIT ITEM ({*; } オブジェクト{; 項目})

引数	タイプ		説明
*	*	→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、またはフィールドまたは変数（“*”を省略した場合）
項目	数値	→	項目番号

説明

このコマンドを使用すると、引数＜オブジェクト＞に指定したリストや配列において、カレント項目または引数＜項目＞で指定した番号の項目を編集することができます。つまり、選択した項目は変更され、入力された文字で項目内容がすべて置き換えられます。

任意の引数“*”を渡すと、引数＜オブジェクト＞がオブジェクト名であることを示します（この場合、＜オブジェクト＞には文字列を渡す）。この引数を渡さない場合、引数＜オブジェクト＞がフィールド、または変数であることを示します。この場合、文字列ではなく、フィールドや変数の参照番号を渡します。

このコマンドは次の入力可オブジェクトに対して適用されます。

- 階層リスト
- リストボックス（後述の「リストボックス」の節を参照）
- サブフォーム（この場合、＜オブジェクト＞にはオブジェクト名（サブフォーム）だけを渡すことが可能）
- **MODIFY SELECTION**や**DISPLAY SELECTION**コマンドで表示されたリストフォーム

リストではない入力可オブジェクトに対してこのコマンドを使用すると、**GOTO AREA**コマンドと同じ動作を行います。リストや配列が空であるか非表示の場合、コマンドは何も行いません。また、リストや配列が入力不可の場合、コマンドは指定された項目を選択するだけで、編集モードには移行しません。

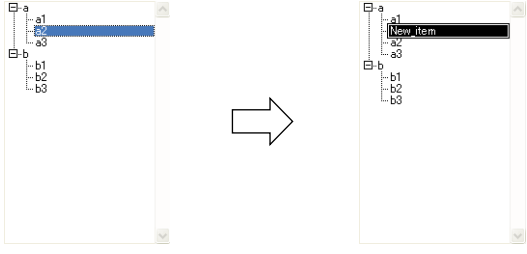
任意の引数＜項目＞を使用すると、編集モードに変更する項目の位置（階層リスト）または行番号（リストボックス、リストフォーム、“複数選択”モードのサブフォーム）を指定することができます。この引数を渡さない場合、コマンドは＜オブジェクト＞のカレント項目に対して適用されます。カレント項目が存在しなければ、＜オブジェクト＞の1番目の項目を編集モードに変更します。

注：階層リストにおいて、EDIT ITEMコマンドは自動的にリストをリドロウします。したがって、EDIT ITEMコマンドを使用する場合は、REDRAW LISTコマンドを呼び出すべきではありません。

例題

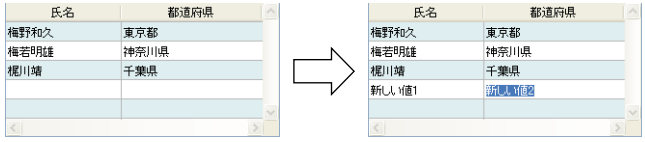
(1) このコマンドは、階層リストに新しい項目を作成する際に特に役立ちます。コマンドが呼び出されると、リストへ追加または挿入された最後の項目が自動的に入力可能となり、ユーザによる動作は必要ありません。
次のコードは、既存のリストに新規項目を挿入するボタンのメソッドとして使用することができます。デフォルトとして表示される“New_item”というテキストは、自動的に変更を受け入れられる状態になります。

```
vlUniqueRef:=vlUniqueRef+1
INSERT LIST ITEM(hList;*;"New_item";vlUniqueRef)
EDIT ITEM(*;"MyList")
```



(2) リストボックスに2つのカラムが存在し、それぞれの変数名は“配列1”と“配列2”であるものとします。次の例題は、リストボックスに新しい行を挿入し、配列2の新規項目を強調表示します。

```
$vlRowNum:=Size of array(xsNames)+1
INSERT LISTBOX ROW(*;"MyListbox";$vlRowNum)
xsNames{$vlRowNum}:="新しい値1"
xsAdr1{$vlRowNum}:="新しい値2"
EDIT ITEM(xsAdr1;$vlRowNum)
```



(3)次の例題を使用して、サブセレクション中の最終サブレコードの1番目のフィールドを編集モードに変更することができます。

```
LAST SUBRECORD([Children])
EDIT ITEM(*;"Subform")
```

参照

GOTO AREA、INSERT LIST ITEM、SET LIST ITEM

ODBC LOGIN

ODBC LOGIN({データエントリ{; ユーザ名{; パスワード}}})

引数	タイプ		説明
データエントリ	文字列	→	ODBCマネージャにおけるデータソースエントリの名前
ユーザ名	文字列	→	データソースに登録されたユーザの名前
パスワード	文字列	→	データソースに登録されたユーザのパスワード

説明

ODBC LOGINコマンドを使用すると、外部のODBCデータソースへ接続することができます。

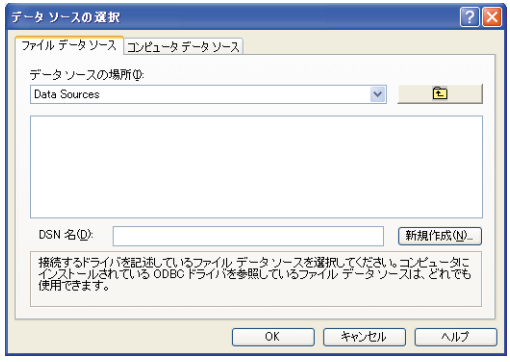
注：ODBC（Open DataBaseConnectivity）スタンダードでは、ライブラリの標準機能（Standard functions）が定義されています。これらの機能により4th Dimensionのようなアプリケーションから SQLを使用してODBCに対応するすべてのデータソース（データベース、スプレッドシートなど）にアクセスすることができるようになりました。

引数＜データエントリ＞には、ODBCドライバマネージャにおいて入力されたデータソースの名前を納めます。

＜ユーザ名＞には、外部データソースへの接続を許可されたユーザの名前を納めます。例えば、Oracleを使用している場合のユーザ名は“Scott”です。

＜パスワード＞には、外部データソースへの接続を許可されたユーザのパスワードを納めます。例えば、Oracleを使用している場合のパスワードは“tiger”です。

これらの引数はオプションです。引数をまったく渡さない場合は、コマンドにより「ODBCマネージャ」ダイアログボックスが表示され、外部データソースを選択することができます。



このコマンドの有効範囲は、各プロセスです。つまり、異なる2つの接続を行いたい場合、2つのプロセスを作成してそれぞれのプロセス内で各接続を実行しなくてはなりません。

例題

▼ この命令は、「ODBCドライバ」マネージャウインドウを表示します。

```
ODBC LOGIN
```

▼ この命令は、名前とパスワードに“Scott”と“tiger”を使用して、“MyOracle”という名前の外部データソースへ接続します。

```
ODBC LOGIN("MyOracle";"Scott";"tiger")
```

参照

ODBC LOGOUT

システム変数とセット

接続が正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC LOGOUT

ODBC LOGOUT

引数　　タイプ　　説明

このコマンドに引数は必要ありません。

説明

ODBC LOGOUTコマンドは、カレントプロセスの接続をクローズします（該当する場合）接続が行われていない場合、コマンドは何も行いません。

参照

ODBC LOGIN

システム変数とセット

ログアウトが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて、このエラーを遮ることができます。

ODBC SET OPTION

ODBC SET OPTION (オプション; 値)

引数	タイプ		説明
オプション	倍長整数	→	設定するオプションの番号
値	倍長整数	→	オプションの新しい値

説明

ODBC SET OPTIONコマンドを使用して、＜オプション＞に渡したオプションの＜値＞を変更します。

＜オプション＞には、「External Data Source」テーマにある次の定数のうちいずれかを納めることができます。

定数	説明とその値
OBDC Asynchronous(1)	0= 同期接続（デフォルト値） 1（または0以外の値）=非同期接続
ODBC Max Rows(2)	結果のグループにおける最大行数（プレビューで使用）
ODBC Max Data Length(3)	返されるデータの最大長
ODBC Query Timeout(4)	ODBC EXECUTEコマンドの実行時に応答を待機する最大タイムアウト この値が考慮されるためには、接続をオープンする前にこの値を設定しなくてはなりません。 値：時間（秒単位） 省略値：タイムアウトなし
ODBC Connection Timeout(5)	ODBC LOGINコマンドの実行時の最大タイムアウト 可能な値：時間（秒単位） 省略値：タイムアウトなし コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

参照

ODBC GET OPTION

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC GET OPTION

ODBC GET OPTION(オプション; 値)

引数	タイプ		説明
オプション	倍長整数	→	オプションの番号
値	倍長整数	←	オプションの値

説明

ODBC GET OPTIONコマンドは、＜オプション＞に渡したオプションの現在の＜値＞を返します。

各種オプションとその関連する値についての詳細は、前述の**ODBC SET OPTION**コマンドを参照してください。

参照

ODBC SET OPTION

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC EXECUTE

ODBC EXECUTE(sql命令 {;バインドオブジェクト1; ...;バインドオブジェクト_n})

引数	タイプ		説明
SQL命令	テキスト	→	実行するSQLコマンド
バインドオブジェクト1...n	変数¦フィールド	←	結果を受け取る（必要な場合）

説明

ODBC EXECUTEコマンドは、SQLコマンドを実行してその結果を4Dオブジェクト（配列、変数、フィールド）にバインドするために使用します。

このコマンドを実行するには、カレントプロセス内で有効な接続が必要になります。

引数＜SQL命令＞には、実行するSQLコマンドを代入し、その結果は＜バインドオブジェクト＞へ返されます。各変数はカラムの並び順でバインドされます。つまり、リモートカラムのうち余ったカラムがあれば、それは切り捨てられます。

引数として4Dフィールドが＜バインドオブジェクト＞に渡された場合、コマンドはレコードを作成してそれを自動的に保存します。4Dフィールドは同じテーブルに属していなければなりません（テーブル1のフィールドとテーブル2のフィールドを同じ呼び出しで渡すことはできません）。別々のテーブルのフィールドを渡すと、エラーが生成されます。

引数＜バインドオブジェクト＞に4D配列や変数を渡す場合、処理されるデータのタイプを検証するため、コマンドを呼び出す前にこれらを宣言するようお勧めします。必要に応じて配列は自動的にリサイズされます。

4D変数を使用すると、一度に一つのレコードが取り出されます。

例題

1. この例題では、外部データソースのempテーブルのenameカラムを取得します。結果は、4Dフィールドである“[従業員]名前”に保存されます。4Dレコードは自動的に作成されます。

```
SQLStmt:="SELECT ename FROM emp"
ODBC EXECUTE(SQLStmt;[従業員]名前)
ODBC LOAD RECORD(ODBC All Records)
```

2. レコード作成を検証するため、トランザクション内にコードを組み込み、操作が十分であることが分かった場合にのみ確定することができます。

```
ODBC LOGIN("mysql";"root";"")
SQLStmt:="SELECT alpha_field FROM app_testTable"
START TRANSACTION
ODBC EXECUTE(SQLStmt;[Table 2]Field1)
While(Not(ODBC End Selection))
        ODBC LOAD RECORD
                ... `データ確定用のコードをここに記述する
End while
VALIDATE TRANSACTION`トランザクションの確定
```

3. この例題では、外部データソースempテーブルのenameカラムを取得します。その結果は、配列aNameに保存されます。レコードは10件ずつ取り出します。

```
ARRAY STRING(30;tNames;20)
SQLStmt:="SELECT ename FROM emp"
ODBC EXECUTE(SQLStmt;aName)
While(Not(ODBC End Selection))
        ODBC LOAD RECORD(10)
End while
```

4. この例題では、外部データソースの特定のIDに対し（WHERE句）、empテーブルのenameカラムとjobカラムを取得します。その結果は、4D変数vNameとvJobに保存されます。最初のレコードだけを要求します。

```
SQLStmt:="SELECT ename, job FROM emp WHERE id= 3"
ODBC EXECUTE(SQLStmt; vName; vJob)
ODBC LOAD RECORD
```

参照

ODBC LOAD RECORD

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC End selection

ODBC End selection→ブール

引数	タイプ		説明
このコマンドに引数は必要ありません。			
戻り値	ブール	←	結果セットの限界に達した

説明

ODBC End selection関数は、結果セットの限界に達したかどうかを判定するために使用します。

例題

▼次のコードは、以下のパラメータを用いて外部データソース（Oracle）へ接続します。

```
C_TEXT(vName)
ODBC LOGIN("TestOracle";"scott";"tiger")
If(OK=1)
    ODBC EXECUTE("SELECT ename FROM emp";vName)
    While(Not(ODBC End selection))
        ODBC LOAD RECORD(0)
    End while
    ODBC LOGOUT
End if
```

このコードは、Empという名前のテーブルに保存されたemp名（ename）を4D変数であるvNameに返します。

ODBC LOAD RECORD

ODBC LOAD RECORD{(レコード数)}

引数	タイプ		説明
レコード数	整数	→	ロードするレコードの数

説明

ODBC LOAD RECORDコマンドは、現在の接続において開かれたODBCソースからのレコードを1件以上4th Dimension内に取り込みます。

任意の引数＜レコード数＞を使用し、取り出すレコード数を設定します。

■ この引数を省略すると、コマンドはデータソースのカレントレコードを取り出します。この方法は、一度に1レコードを受け取るループ中のデータ取得に対応します。

■ ＜レコード数＞に整数値を渡すと、コマンドは＜レコード数＞件のレコードを取り出します。

■ 「ODBC All Records」定数（または値-1）を渡すと、コマンドはテーブルの全レコードを取り出します。

注：最後の2つの引数は、取得したデータが配列や4Dフィールドに関連付けられている場合にのみ意味を持ちます。

参照

ODBC CANCEL LOAD、ODBC EXECUTE

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC CANCEL LOAD

ODBC CANCEL LOAD

引数	タイプ	説明

このコマンドに引数は必要ありません。

説明

ODBC CANCEL LOADコマンドは現在の**SELECT**リクエストを終了してパラメータを初期化します。

このコマンドを使用して、**ODBC LOGIN**コマンドにより開始された同一接続内（つまり同一カーソル）において、複数の**SELECT**リクエストを実行します。

例題

▼ この例題では、同一接続内で2つのリクエストが実行されます。

```
C_BLOB(MyBlob)
C_TEXT(MyText)
ODBC LOGIN("mysql";"root";"")
SQLStmt:="SELECT blob_field FROM app_testTable"
ODBC EXECUTE(SQLStmt;MyBlob)
While(Not(ODBC End selection))
        ODBC LOAD RECORD
End while
        `カーソルのリセット
ODBC CANCEL LOAD
SQLStmt:="SELECT Name FROM Employee"
ODBC EXECUTE(SQLStmt;MyText)
While(Not(ODBC End selection))
        ODBC LOAD RECORD
End while
```

参照

ODBC LOAD RECORD、ODBC LOGIN.

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC SET PARAMETER

ODBC SET PARAMETER(オブジェクト; パラメータタイプ)

引数	タイプ		説明
オブジェクト	4Dオブジェクト	→	使用される4Dオブジェクト（変数、配列、フィールド）
パラメータタイプ	倍長整数	→	パラメータのタイプ

説明

ODBC SET PARAMETERコマンドを使用すると、4D変数や配列、フィールドをODBCリクエストで使用することができます。

ODBCリクエストで4th Dimensionオブジェクトを使用する

4th Dimensionでは、2通りの方法で4DオブジェクトをODBCリクエストに挿入することができます。つまり、直接関連付ける方法と、**ODBC SET PARAMETER**を使用してパラメータを指定する方法です。

■ **直接関連付ける方法：**このモードでは、リクエストテキスト内の“<<”と“>>”記号の間に、使用する4Dオブジェクト（変数、配列、フィールド）の名前を挿入する必要があります。

例：INSERT INTO emp(empno,ename) VALUES(<<vEmpno>>,<<vEname>>)

この例では、リクエストの実行中に4D変数であるvEmpnoとvEnameの値でパラメータが置き換えられます。このソリューションは、4Dフィールドと配列を使用した場合でも有効です。

この使いやすいシンタックスには、SQL規格に準拠していないという難点が1つあり、またexitパラメータを使用することもできません。この問題を改善するために、**ODBC SET PARAMETER**コマンド（二番目の方法）を使用することができます。

■ **パラメータを指定する方法：**この方法では、**ODBC SET PARAMETER**コマンドを使用して、ODBCリクエスト内で使用したい4Dオブジェクトを指定することができます。

■ 引数<オブジェクト>には、リクエストで使用される4Dオブジェクト（変数、配列、フィールド）を渡します。

■ 引数<パラメータタイプ>には、パラメータのSQLタイプを渡します。任意の値を渡すか、「External Data Source」テーマにある次の定数のいずれかを使用することができます。

定数	タイプ	値
OBDC Param In	倍長整数	1
OBDC Param In Out	倍長整数	2
OBDC Param Out	倍長整数	4

SQLリクエスト内におかれた“?”記号が4Dオブジェクトの値で置き換えられます（標準のシンタックス）。リクエストに複数の“?”記号が含まれている場合、**ODBC SET PARAMETER**コマンドを複数回呼び出す必要があります。4Dオブジェクトの値は、コマンドの実行順に合わせてリクエスト内で順次割り当てられます。

例題

1. この例題を使用して、関連付けられた4D変数を直接呼び出すODBCリクエストを実行します。

```
C_TEXT(MyText)
C_LONGINT(MyLongint)
ODBC LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable(alpha_field, longint_field) VALUES
                                                    (<<MyText>>, <<MyLongint>>)"
For(vCounter;1;10)
      MyText:="Text"+String(vCounter)
      MyLongint:=vCounter
      ODBC EXECUTE(SQLStmt)
End for
```

2. 上記と同じ例題ですが、**ODBC SET PARAMETER**コマンドが使用されています。

```
C_TEXT(MyText)
C_LONGINT(MyLongint)
ODBC LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable(alpha_field, longint_field) VALUES(?,?)"
For(vCounter;1;10)
      MyText:="Text"+String(vCounter)
      MyLongint:=vCounter
      ODBC SET PARAMETER(MyText;ODBC Param In)
      ODBC SET PARAMETER(MyLongint;ODBC Param In)
      ODBC EXECUTE(SQLStmt)
End for
```

3. この例題を使用して、関連付けられた4D配列を直接使用するODBCリクエストを実行します。

```
ARRAY TEXT(MyTextArray;10)
ARRAY LONGINT(MyLongintArray;10)
```

```
For(vCounter;1;Size of array(MyTextArray))
    MyTextArray{vCounter}:="Text"+String(vCounter)
    MyLongintArray{vCounter}:=vCounter
End for
ODBC LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable(alpha_field, longint_field) VALUES
                                        (<<MyTextArray>>, <<MyLongintArray>>)"
ODBC EXECUTE(SQLStmt)
```

4. この例題を使用して、関連付けられた4Dフィールドを直接使用するODBCリクエストを実行します。

```
ALL RECORDS([Table 2])
ODBC LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable(alpha_field, longint_field) VALUES
                                        (<<[Table 2]Field1>"+">, <<[Table 2]Field2>)"
ODBC EXECUTE(SQLStmt)
```

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

ODBC GET LAST ERROR

ODBC GET LAST ERROR(エラーコード;エラーテキスト;ODBCエラー;SQL Serverエラー)

引数	タイプ		説明
エラーコード	倍長整数	←	エラーコード
エラーテキスト	倍長整数	←	エラーテキスト
ODBCエラー	テキスト	←	ODBCエラーコード
SQLServerエラー	テキスト	←	SQLサーバのネイティブエラーコード

説明

ODBC GET LAST ERRORコマンドは、ODBCコマンドの実行中に発生した最後のエラーに関連する情報を返します。エラーの発生箇所としては、4th Dimensionアプリケーション、ネットワーク、ODBCソースなどが考えられます。

通常、このコマンドは**ON ERR CALL**コマンドを用いて設定されたエラー処理用メソッド内から呼び出されます。

■ 引数<エラーコード>はエラーコードを返します。

■ 引数<エラーテキスト>はエラーテキストを返します。

最後の2つの引数は、ODBCソースからのエラーの場合にのみ代入されます。それ以外の場合は、空のテキストが返されます。

■ 引数<ODBCエラー>はODBCエラーコード（SQL state）を返します。

■ 引数<SQLServerエラー>はSQLサーバのネイティブエラーコードを返します。

参照

ON ERR CALL

ODBC IMPORT

ODBC IMPORT(ソーステーブル{; プロジェクト}{; *})

引数	タイプ		説明
ソーステーブル	文字列	→	ODBCデータソース内のテーブル名
プロジェクト	BLOB	→	読み込むプロジェクトの内容
		←	読み込むプロジェクトの新しい内容（“*”を渡した場合）
*	*	→	読み込み用ダイアログボックスの表示とプロジェクトの更新

説明

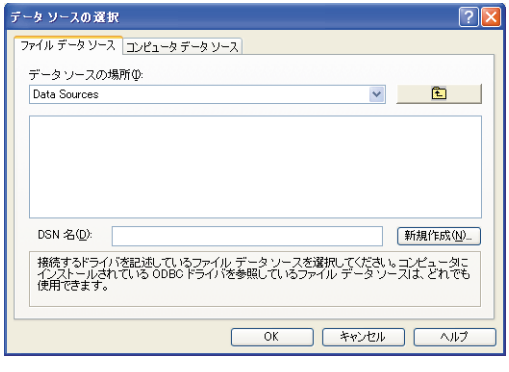
ODBC IMPORTコマンドを使用すると、外部ODBCソースのテーブル＜ソーステーブル＞からデータが読み込まれます。接続パラメータ（ソース名、ユーザとパスワード）は、BLOB＜プロジェクト＞に納められます。

注：

・プロジェクトには、すべての読み込みパラメータ（特に、データソース、対象のテーブルやフィールド）が格納されます。ODBC読み込み用ダイアログボックスでこれらのパラメータを設定した後、必要に応じてこれらのパラメータをディスク上のフィールドに保存することができます。詳細については、『ユーザリファレンス』マニュアルを参照してください。

・ODBC読み込み用ダイアログボックスに生成されたプロジェクトは、4th Dimensionの標準読み込み用ダイアログボックス（Standard import dialogue box）やコマンドに対応していません。

任意の引数＜プロジェクト＞を渡さない場合、**ODBC IMPORT**コマンドはODBCデータソースを選択するダイアログボックスを表示します。



ソースを選択すると、4th DimensionのODBC読み込み用ダイアログボックスが現れ、操作の設定を行うことができます。2つのダイアログボックスのいずれかで「キャンセル」ボタンをクリックすると、実行は中断されてシステム変数OKに0が代入されます。

引数＜プロジェクト＞に有効なODBC読み込みプロジェクトを格納したBLOBを渡すと、ユーザによる操作を必要とせずに直接読み込みが実行されます。これを行うには、**DOCUMENT TO BLOB**コマンドを使用し、あらかじめディスク上に保存されたプロジェクトを引数＜プロジェクト＞に渡したフィールドやBLOB変数内にロードしておく必要があります。

また、空の引数＜プロジェクト＞と任意の引数＜*＞を用いて**ODBC IMPORT**コマンドを呼び出した後で、引数＜プロジェクト＞をBLOBフィールド（後述）に保存することもできます。一方、このソリューションにより、データファイルを用いてプロジェクトを保存し、他方ではディスクからBLOBへのプロジェクトの保存フェーズを回避することができます。

注：空のプロジェクトの定義については、EXPORT DATAコマンドで例を参照してください。

任意の引数＜*＞を指定した場合、＜プロジェクト＞に指定された設定（存在する場合）を用いて、ODBC読み込み用のダイアログボックスが表示されます。これにより、定義済みのプロジェクトを使用しながらも、いくつかのパラメータを変更することができます。さらに、その際はダイアログボックスのクローズ後に、“新規”プロジェクトのパラメータが引数＜プロジェクト＞へ格納されます。この後、それをBLOBフィールドやディスクファイルなどに保存することができます。

参照

ODBC EXPORT

システム変数とセット

（データソースを選択する際、もしくは読み込みを設定する際）2つのうち、いずれかのダイアログボックスで「キャンセル」ボタンをクリックすると、システム変数OKには0が代入されます。読み込みが正しく実行されると、システム変数OKには1が代入されます。

ODBC EXPORT

ODBC EXPORT(ソーステーブル{; プロジェクト}{; *})

引数	タイプ		説明
ソーステーブル	文字列	→	ODBCデータソース内のテーブル名
プロジェクト	BLOB	→	書き出すプロジェクトの内容
		←	書き出すプロジェクトの新しい内容（“*”を渡した場合）
*	*	→	書き出し用ダイアログボックスの表示とプロジェクトの更新

説明

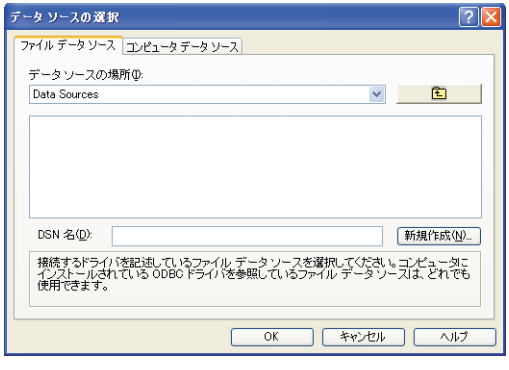
ODBC EXPORTコマンドを使用すると、外部ODBCソースのテーブル＜ソーステーブル＞へデータが書き出されます。接続パラメータ（ソース名、ユーザとパスワード）は、BLOB＜プロジェクト＞に納められます。

注：

・プロジェクトには、すべての書き出しパラメータ（特に、データソース、書き出されるテーブルやフィールド）が格納されます。ODBC書き出し用ダイアログボックスでこれらのパラメータを設定した後、必要に応じてこれらのパラメータをディスクファイルに保存することができます。詳細については、『ユーザリファレンス』マニュアルを参照してください。

・ODBC書き出し用ダイアログボックスに生成されたプロジェクトは、4th Dimensionの標準書き出し用ダイアログボックス（Standard import dialogue box）やコマンドに対応していません。

任意の引数＜プロジェクト＞を渡さない場合、**ODBC EXPORT**コマンドはODBCデータソースを選択するダイアログボックスを表示します。



ソースを選択すると、4th DimensionのODBC書き出し用ダイアログボックスが現れ、操作の設定を行うことができます。2つのダイアログボックスのいずれかで「キャンセル」ボタンをクリックすると、実行は中断されてシステム変数OKに0が代入されます。

引数＜プロジェクト＞に有効なODBC書き出しプロジェクトを格納したBLOBを渡すと、ユーザによる操作を必要とせずに直接書き出しが実行されます。これを行うには、**DOCUMENT TO BLOB**コマンドを使用し、あらかじめディスク上に保存されたプロジェクトを引数＜プロジェクト＞に渡したフィールドやBLOB変数内にロードしておく必要があります。

また、空の引数＜プロジェクト＞と任意の引数＜*＞を用いて**ODBC EXPORT**コマンドを呼び出した後で、引数＜プロジェクト＞をBLOBフィールド（後述）に保存することもできます。一方、このソリューションにより、データファイルを用いてプロジェクトを保存し、他方ではディスクからBLOBへのプロジェクトの保存フェーズを回避することができます。

注：空のプロジェクトの定義については、EXPORT DATAコマンドで例を参照してください。

任意の引数＜*＞を指定した場合、＜プロジェクト＞に指定された設定（存在する場合）を用いて、ODBC書き出し用のダイアログボックスが表示されます。これにより、定義済みのプロジェクトを使用しながらも、いくつかのパラメータを変更することができます。さらに、その際はダイアログボックスのクローズ後に、“新規”プロジェクトのパラメータが引数＜プロジェクト＞へ格納されます。この後、それをBLOBフィールドやディスクファイルなどに保存することができます。

読み込みが正常に実行されると、システム変数OKに1が代入されます。

参照

ODBC IMPORT

参照

システム変数とセット

（データソースを選択する際、もしくは書き出しを設定する際）2つのうち、いずれかのダイアログボックスで「キャンセル」ボタンをクリックすると、システム変数OKには0が代入されます。書き出しが正しく実行されると、システム変数OKには1が代入されます。

Form event

Form event → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	フォームイベント番号

説明

Form event関数は、発生したばかりのフォームイベントのタイプを示す数値を返します。通常、ユーザはフォームやオブジェクトメソッド内から**Form event**関数を使用します。

4th Dimensionには事前に定義された定数（「Form Events」テーマ内にあります）が用意されており、**Form event**関数から返される値と比較することができます。

一部のイベントは汎用であり（任意のタイプのオブジェクトに対して生成される）、それ以外のイベントは特定タイプのオブジェクトに限定されています。

汎用イベント

次のイベントは任意のフォームまたはオブジェクトに対して生成されます。

定数	値	説明
On Load	1	フォームは表示または印刷されようとしています。
On Unload	24	フォームは終了または解放されようとしています。
On Validate	3	レコードのデータ入力が有効となりました。
On Clicked	4	オブジェクトがクリックされました。
On Double Clicked	13	オブジェクトがダブルクリックされました。
On Before Keystroke	17	フォーカスを持つオブジェクト内に文字が入力されようとしています。イベント内で**GET edited text**関数を実行しようとするとオブジェクト内の入力前のテキストを返します。

On After Keystroke	28	フォーカスを持つオブジェクト内に文字が入力されています。イベント内で**GET edited text**関数を実行すると、オブジェクトの最後に入力された文字を返します。
On Getting Focus	15	フォームオブジェクトがフォーカスを獲得しました。
On Losing Focus	14	フォームオブジェクトがフォーカスを失いました。
On Activate	11	フォームのウインドウが最前列のウインドウになりました。
On Deactivate	12	フォームのウインドウはもはや最前列のウインドウではありません。
On Outside Call	10	フォームが**CALL PROCESS**コマンド呼び出しを受け取りました。
On Drop	16	データがオブジェクト上にドロップされました。
On Drag Over	21	データがオブジェクト上にドロップされる可能性があります。
On Mouse Enter	35	マウスカーソルはオブジェクトのグラフィックエリア内に入りました。
On Mouse Move	37	マウスカーソルはオブジェクトのグラフィックエリア内を移動しました（少なくとも1ピクセル）
On Mouse Leave	36	マウスカーソルはオブジェクトのグラフィックエリアから出ました。
On Menu Selected	18	メニュー項目が選択されました。
On Data Change	20	オブジェクトデータが変更されました。
On Plug in Area	19	プラグインエリアが自身の実施すべきオブジェクトメソッドを要求しました。
On Header	5	フォームのヘッダエリアが印刷／表示されようとしています。
On Printing Detail	23	フォームのディテイルエリアが印刷されようとしています。
On Printing Break	6	フォームのブレークエリアの1つが印刷されようとしています。
On Printing Footer	7	フォームのフッタエリアが印刷されようとしています。
On Close Box	22	ウインドウのクローズボックスがクリックされました。
On Display Detail	8	レコードがリスト内に表示されようとしています。
On Open Detail	25	レコードがダブルクリックされ、ユーザは入力フォームに移ります。
On Close Detail	26	ユーザは入力フォームから出力フォームへ戻ります。
On Selection Change	31	・リストボックス：行または列の現在の選択範囲が変更されました。 ・リスト上のレコード：リストフォームまたはサブフォーム上で、カレントレコードまたは行の現在の選択範囲が変更されました。

		・階層リスト：クリックまたはキーストロークに続けて、リスト上の選択範囲が変更されました。
On Load Record	40	リストへの入力の際、修正中にレコードがロードされました（ユーザがレコード行をクリックし、フィールドが編集モードに移行）。
On Timer	27	**SET TIMER**コマンドで定義されたTick数が経過しました。
On Resize	29	フォームウインドウのサイズが変更されました。

リストボックス

次のイベントは、リストボックスに対してのみ生成されます。

定数	値	説明
On Before Data Entry	41	リストボックスのセルが編集モードへ移行しようとしています。
On Column Moved	32	ユーザのドラッグ＆ドロップにより、リストボックスの列が移動されました。
On Row Moved	34	ユーザのドラッグ＆ドロップにより、リストボックスの行が移動されました。
On Column Resize	33	リストボックスの列幅が変更されました。
On Header Click	42	リストボックスの列ヘッダでクリックが行われました。
On After Sort	30	リストボックスの列で標準の並び替えが実行されたところです。

3Dボタン

次のイベントは3Dボタンに対してのみ生成されます。

定数	値	説明
On Long Click	39	3Dボタンがクリックされ、マウスボタンが一定時間押されたままです。
On Arrow Click	38	3Dボタンの“矢印”エリアがクリックされました。

階層リスト

次のイベントは階層リストに対してのみ生成されます。

定数	値	説明
On Expand	43	クリックまたはキーストロークによって、階層リストの項目が展開されました。
On Collapse	44	クリックまたはキーストロークによって、階層リストの項目が折り畳まれました。

イベントとメソッド

フォームイベントが発生すると、4th Dimensionは以下のようなアクション（動作）を実行します。

- まず、4th Dimensionは、フォームの各オブジェクトをブラウズし、対応するオブジェクトイベントプロパティが選択されているオブジェクトのオブジェクトメソッドを呼び出します。

- 次に、対応するフォームイベントプロパティが選択されている場合、フォームメソッドを呼び出します。

オブジェクトメソッドは（それが存在する場合）一定の順序で呼び出されることはありません。一般的な法則としては、常にオブジェクトメソッドはフォームメソッドより先に呼び出されるというだけです。

オブジェクトがサブフォームである場合、サブフォームのリストフォームのオブジェクトメソッドが呼び出され、それからリストフォームのフォームメソッドが呼び出されます。この後、4Dは親フォームのオブジェクトメソッドを呼び出します。つまり、オブジェクトがサブフォームである場合、4Dはサブフォームオブジェクト内のオブジェクトとフォームメソッドに対して同じ一般法則を適用します。

On Loadイベントと**On Unload**イベントを除き、発生したイベントに対応するイベントプロパティがフォームプロパティ上で選択されていない場合でも、同じイベントプロパティが選択されているオブジェクトのオブジェクトメソッドは呼び出されます。つまり、フォームレベルでイベントを有効あるいは無効にしても、オブジェクトのイベントプロパティに影響を与えないということです。

ひとつのオブジェクト内に含まれるイベント数は、イベントの性質によって異なります。

- **On Load**イベント：**On Load**オブジェクトイベントプロパティが選択されているフォームの（すべてのページの）オブジェクトは、そのオブジェクトメソッドが呼び出されます。さらに、On Loadフォームイベントプロパティが選択されている場合には、そのフォームのフォームメソッドが呼び出されます。

- **On Activate**または**On Resize**イベント：オブジェクトメソッドは呼び出されません。その理由は、このイベントが特定のオブジェクトに適用されるのではなく、フォーム全体に適用されるからです。したがって、On Activateフォームイベントプロパティが選択されている場合、フォームメソッドだけが呼び出されます。

- **On Drag Over**イベント：**On Drag Over**オブジェクトイベントプロパティが選択されている場合、イベント内に含まれるドロップ可能なオブジェクトのオブジェクトメソッドだけが呼び出されます。フォームメソッドは呼び出されません。

■ **On Timer**イベント：このイベントは、フォームメソッドで前もって**SET TIMER**コマンド呼び出しが実行されている場合にのみ発生します。**On Timer**フォームイベントプロパティが選択されている場合、フォームメソッドだけがこのイベントを受け取り、オブジェクトメソッドは呼び出されません。

警告：On Drag Overイベント中は、他のイベントとは異なり、オブジェクトメソッドは、ドラッグ&ドロップされるオブジェクトのプロセスコンテキスト内で実行され、ドラッグ&ドロップ送信先オブジェクトのプロセスコンテキスト内では実行されません。詳細は、DRAG AND DROP PROPERTIESコマンドとDrop position関数を参照してください。

■ フォームに対し、「**On Mouse Enter**」、「**On Mouse Move**」、「**On Mouse Leave**」の各イベントが選択されている場合、各フォームオブジェクトごとにこれらのイベントが生成されます。あるオブジェクトに対してこれらのイベントが選択されている場合、そのオブジェクトに対してのみイベントが生成されます。重なり合ったオブジェクトが存在する場合は、最上位のレベルから下層まで順番に、イベントを処理可能なオブジェクトのうち、最初に現れたオブジェクトよってイベントが生成されます。**SET VISIBLE**コマンドを用いて非表示にしたオブジェクトでは、これらのイベントが生成されません。オブジェクトの入力時、それ以外のオブジェクトは、マウスの位置に応じ、このタイプのイベントを受け取ります。

■ リスト上のレコード：**MODIFY SELECTION**や**DISPLAY SELECTION**により表示されたリストフォームおよびサブフォームにおいて、メソッドの呼び出し順序とフォームイベントの順序は次のようになります。

ヘッダエリアの各オブジェクトに対して：
「On Header」イベントが指定されたオブジェクトメソッド
「On Header」イベントが指定されたフォームメソッド
各レコードに対して：
詳細エリアの各オブジェクトに対して：
「On Display Detail」イベントが指定されたオブジェクトメソッド
「On Display Detail」イベントが指定されたフォームメソッド

「**On Display Detail**」イベントおよび「**On Header**」イベントから、ダイアログボックスを表示する4Dコマンドを呼び出すことはできません。これを行うとシンタックスエラーが発生します。

特に関係するコマンドは、**ALERT**、**DIALOG**、**CONFIRM**、**Request**、**ADD RECORD**、**MODIFY RECORD**、**DISPLAY SELECTION**、**MODIFY SELECTION**です。

以下の表は、オブジェクトメソッドとフォームメソッドを各イベントタイプごとに呼び出す方法を要約したものです。

定数	オブジェクトメソッド	フォームメソッド	対象オブジェクト
On Load	はい	はい	全オブジェクト
On Unload	はい	はい	全オブジェクト
On Validate	はい	はい	全オブジェクト
On Clicked	はい (+) (*)	はい	関連するオブジェクトのみ
On Double Clicked	はい (+) (*)	はい	関連するオブジェクトのみ
On Before Keystroke	はい (+) (*)	はい	関連するオブジェクトのみ
On After Keystroke	はい (+) (*)	はい	関連するオブジェクトのみ
On Getting Focus	はい (++) (*)	はい	関連するオブジェクトのみ
On Losing Focus	はい (++) (*)	はい	関連するオブジェクトのみ
On Activate	なし	はい	なし
On Deactivate	なし	はい	なし
On Outside Call	なし	はい	なし
On Drop	はい (+++) (*)	はい	関連するオブジェクトのみ
On Drag Over	はい (+++) (*)	なし	関連するオブジェクトのみ
On Mouse Enter	はい	はい	全オブジェクト
On Mouse Move	はい	はい	全オブジェクト
On Mouse Leave	はい	はい	全オブジェクト
On Menu Selected	なし	はい	なし
On Data Change	はい (++++) (*)	はい	関連するオブジェクトのみ
On Plug in Area	はい	はい	関連するオブジェクトのみ
On Header	はい	はい	全オブジェクト
On Printing Details	はい	はい	全オブジェクト
On Printing Break	はい	はい	全オブジェクト
On Printing Footer	はい	はい	全オブジェクト
On Close Box	なし	はい	なし
On Display Details	はい	はい	全オブジェクト
On Open Details	なし	はい	なし
On Close Details	なし	はい	なし
On Resize	なし	はい	なし
On Selection Change	はい(**)	はい	関連するオブジェクトのみ
On Load Record	なし	はい	なし
On Timer	なし	はい	なし
On Before Data Entry	はい(リストボックス)	なし	関連するオブジェクトのみ
On Column Moved	はい(リストボックス)	なし	関連するオブジェクトのみ
On Row Moved	はい(リストボックス)	なし	関連するオブジェクトのみ
On Column Resize	はい(リストボックス)	なし	関連するオブジェクトのみ
On Header Click	はい(リストボックス)	なし	関連するオブジェクトのみ
On After Sort	はい(リストボックス)	なし	関連するオブジェクトのみ

On Long Click	はい(3Dボタン)	はい	関連するオブジェクトのみ
On Arrow Click	はい(3Dボタン)	はい	関連するオブジェクトのみ
On Expand	はい(階層リスト)	なし	関連するオブジェクトのみ
On Collapse	はい(階層リスト)	なし	関連するオブジェクトのみ

+クリック可能な場合
++キーボード入力可能な場合
+++ドロップ可能な場合
++++変更可能な場合

(*)の印の付いた箇所に関する詳細は、後述の「イベント、オブジェクト、プロパティ」の節を参照してください。

(**)リストボックス、階層リスト、サブフォームタイプのオブジェクトだけが、このイベントをサポートします。

重要：あらゆるイベントに対して、対応するイベントのプロパティがそのフォームやオブジェクト用に選択されている場合は、そのフォームやオブジェクトのメソッドが呼び出されることを覚えていください。「デザイン」モードで（「フォームプロパティ」ウインドウを使用して）イベントを無効にすると、メソッドを呼び出す回数をかなり縮小でき、これによりフォームの実行速度を著しく最適化することができるというメリットがあります。

警告：オブジェクトとそれが属するフォームの両方に対してOn LoadとOn Unloadイベントイベントが有効である場合、オブジェクトに対してOn LoadとOn Unloadイベントが生成されます。これらのイベントがオブジェクトに対してのみ有効となる場合、これは起こりません。この2つのイベントはフォームレベルでも有効でなくてはなりません。

イベント、オブジェクト、プロパティ

イベントが、その性質とプロパティに応じて、オブジェクト用に実際に生じうる場合、オブジェクトメソッドが呼び出されます。以下の節では、ユーザが様々なオブジェクトタイプを操作する際に一般的に使用するイベントを詳しく説明します。

「フォーム」エディタの「プロパティ」リストには、選択したオブジェクトやフォームに対応するイベントだけが表示される点に留意してください。

クリック可能なオブジェクト

クリック可能なオブジェクトは、主にマウスで操作します。これには、以下のものが含まれます。

■ ブール入力可能フィールドや変数

■ ボタン、デフォルトボタン、ラジオボタン、チェックボックス、ボタングリッド

■ 3Dボタン、3Dラジオボタン、3Dチェックボックス

■ ポップアップメニュー、階層ポップアップメニュー、ピクチャメニュー

■ ドロップダウンリスト、メニューまたはドロップダウンリスト

■ スクロールエリア、階層リスト、リストボックス

■ 透明ボタン、ハイライトボタン、ラジオピクチャ

■ サーモメータ、ルーラ、ダイアル（これらは、スライダーオブジェクトとも呼ばれます）

■ タブコントロール

■ スプリッター

On Clickedか**On Double Clicked**オブジェクトのイベントプロパティが、これらオブジェクトの1つに対して選択されると、ユーザはオブジェクト内またはオブジェクト上のクリックを見つけて操作することができるようになります。この場合、状況に応じて**On Clicked**や**On Double Clicked**を返す**Form event**関数を使用します。

これらのイベントがオブジェクトのひとつに対して選択されている場合、ユーザがオブジェクトをダブルクリックすると**On Clicked**イベントに続いて**On Double Clicked**イベントが発生します。

すべてのオブジェクトに対し、マウスボタンが離されると、**On Clicked**イベントが発生します。ただし、いくつかの例外があります。

■ **透明ボタン：**クリックされると、すぐに**On Clicked**イベントが発生します。マウスボタンがリリースされるまで待機しません。

■ **スライダーオブジェクト（サーモメータ、ルーラ、ダイアル）：**ユーザがコントロールをスライドしている最中にオブジェクトメソッドを呼び出す必要があることを表示フォームが示している場合、クリックされるとすぐに**On Clicked**イベントが生じます。

注：キーボードでアクティブにできるオブジェクトもあります。例えば、チェックボックスがフォーカスを獲得すると、スペースバーを使用してこれを入力できます。このような場合でも、On Clickedイベントは発生します。

警告：コンボボックスはクリック可能なオブジェクトとは考えられていません。コンボボックスは、入力可能なテキストエリアとして扱う必要があります。この入力可能なテキストエリアに関連するドロップダウンリストはデフォルトの値を提供します。結果として、ユーザはOn Before KeystrokeとOn After KeystrokeとOn Data Changeイベントを通じて、コンボボックス内でデータ入力操作を実行することになります。

キーボード入力可能なオブジェクト

キーボード入力可能なオブジェクトとは、キーボードを使用してユーザがデータを入力するオブジェクトのことであり、ユーザはそのオブジェクト用に**On Before Keystroke**イベントと**On After Keystroke**イベントを見つけることにより、最下位レベルでデータ入力を振るい分けます（この時**Get edited text**関数を用いるとよいでしょう）。これには、以下のものが含まれます。

■ すべての入力可能なフィールドオブジェクト（ピクチャ、サブテーブル、BLOBは除く）

■ すべての入力可能な変数（ピクチャ、BLOB、ポインタ、配列は除く）

■ コンボボックス

■ リストボックス

これらオブジェクトの1つに対し、**On Before Keystroke**かまたは**On After Keystroke**オブジェクトイベントプロパティが選択されると、ユーザはこのオブジェクト内でキーストロークを検出し、処理することができます。この場合、**On Before Keystroke**か**On After Keystroke**イベントをそれぞれ返す**Form event**関数を使用します（詳細は**Get edited text**関数を参照してください）。

注：

・POST KEYコマンドは「On Before Keystroke」と「On After Keystroke」イベントを生成します。

・階層リストは入力可オブジェクトですが、「On Before Keystroke」と「On After Keystroke」イベントを生成しません（後述の「階層リスト」の節も参照してください）。

変更可能なオブジェクト

変更可能なオブジェクトはデータソースを持っており、このデータソースの値はマウスやキーボードで変更できます。ただし、これらは、実際には**On Clicked**イベントを通じて操作するユーザインタフェースのコントロールとは考えられていません。これには、以下のものが含まれます。

■ すべての入力可能なフィールドオブジェクト（サブテーブル、BLOBは除く）

■ すべての入力可能な変数（BLOB、ポインタ、配列は除く）

■ コンボボックス

■ 外部オブジェクト（外部オブジェクト用のデータ入力全体は4D Extensionで受け付けられます）

■ 階層リスト

■ リストボックス

これらのオブジェクトが**On Data Change**イベントを受け取ります。これらのオブジェクトの1つに対し、**On Data Change**オブジェクトイベントプロパティが選択された後、ユーザはデータソースの値の変更を見つけて操作することができます。この場合、**On Data Change**を返す**Form event**関数を使用します。4Dによって内部でオブジェクトに関連する変数が更新されると、すぐにイベントが生成されます（例：通常、オブジェクトの入力エリアのフォーカスが失われた場合）。

タブ使用可能なオブジェクト

ユーザがタブを使用してタブ使用可能なオブジェクトに移動、またはこれをクリックすると、タブ使用可能なオブジェクトはフォーカスを獲得します。フォーカスを取得しているオブジェクトは（キーボード上でタイプされた）文字を受け取ります。受け取る文字は、メニューやボタンのようなオブジェクトへのアクセラレータ（Windows）、またはショートカット（Macintosh）以外のものになります。

以下のものを除き、オブジェクトはすべてタブ使用可能です。

- 入力不可能なフィールドや変数
- ボタン（Macintoshで使用）
- ボタングリッド
- 3Dボタン、3Dラジオボタン、3Dチェックボックス
- ポップアップメニュー、階層ポップアップメニュー
- メニューまたはドロップダウンリスト（Macintoshで使用）
- ピクチャメニュー
- スクロール可能エリア
- 透明ボタン、ハイライトボタン、ラジオピクチャ
- グラフ
- 外部オブジェクト（外部オブジェクト用の全データ入力は、4D Extensionで受け付けられません）
- タブコントロール
- スプリッター

タブ使用可能なオブジェクトに対し、**On Getting Focus**、または**On losing Focus**オブジェクトイベントプロパティが選択された後、ユーザはフォーカスの変更を見つけて操作することができるようになります。この場合、状況に応じて**On Getting Focus**または**On losing Focus**を返す**Form event**関数を使用します。

3Dボタン

3Dボタンを使用して、高度なグラフィックインタフェースを設置することができます(3Dボタンの説明は、『デザインリファレンス』マニュアルを参照してください)。一般的なイベントに加え、2つの特定イベントを使用して、これらのボタンを管理することができます。

■ **On long click：**このイベントは、3Dボタンがクリックを受け取り、マウスボタンが一定時間押したままの状態である場合に生成されます。理論上、このイベントが生成されるための時間の長さは、システム環境設定で定義したダブルクリックを区切る最長時間と同じです。
このイベントは、“前世代”の3Dボタン（つまり、背景オフセットスタイル）やポップアップメニューが付属する3Dボタンの矢印エリアを除き、あらゆるスタイルの3Dボタン、3Dラジオボタン、3Dチェックボックスに対して生成されます。

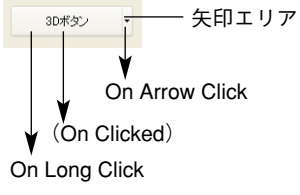
通常、このイベントは、ボタンクリックが長く行われた場合にポップアップメニューを表示するために使用されます。ユーザが“長いクリック”のタイムアウト前にマウスボタンを離すと、**On click**イベントが生成されます（このイベントが有効である場合)。

■ **On arrow click：**3Dボタンスタイルのなかには、ポップアップメニューに関連付けられ、矢印を表示できるものがあります。その矢印をクリックすると、選択用のポップアップメニューが表示され、最初のボタンアクションに関連した追加アクションを設定することができます。
4th Dimensionでは、**On arrow click**イベントを用いて、このタイプのボタンを管理することができます。ユーザが“矢印”をクリックすると、このイベントが生成されます（マウスボタンが押された直後)。

- ■ ポップアップメニューが“分離”タイプの場合、矢印のあるボタン部分でクリックが行われた場合にのみ、このイベントが生成されます。
- ■ ポップアップメニューが“リンク”タイプの場合、ボタンの任意の場所でクリックが行われると、このイベントが生成されます。このタイプのボタンでは、**On long click**イベントが生成されないということに注意してください。

3Dボタン例（“ツールバーボタン”スタイル）

分離型のポップアップメニューの場合



リンク型のポップアップメニューの場合



「なし」、「ツールバーボタン」、「ベベル」、「角の丸いベベル」、「Office XP」といった3Dボタン、3Dラジオボタン、3Dチェックボックススタイルは、「ポップアップメニューあり」プロパティに対応します。

リストボックス

7つのフォームイベントを使用して、リストボックスの各種機能を管理することができます。

■ **On Before Data Entry**：このイベントは、リストボックスのセルが編集される直前に生成されます（入力カーソルが表示される前）。例えば、このイベントを使用すると、開発者はユーザが表示モードか編集モードかによって、異なるテキストを表示することができます。

■ **On Selection Change**：このイベントは、リストボックスの行や列のカレント選択項目が変更されるたびに生成されます。また、このイベントはレコードリストや、階層リストにおいても生成されます。

■ **On Column Moved**：このイベントは、ユーザがドラッグ＆ドロップを使い、リストボックスの列を移動すると生成されます。ドラッグ＆ドロップで列を最初の位置に戻した場合には、このイベントは生成されません。**MOVED LISTBOX COLUMN NUMBER**コマンドは、列の新しい位置を返します。

■ **On Row Moved**：このイベントは、ユーザがドラッグ＆ドロップを使い、リストボックスの行を移動すると生成されます。ドラッグ＆ドロップで行を最初の位置に戻した場合には、このイベントは生成されません。

■ **On Column Resize**：このイベントは、リストボックスの列幅が変更されると生成されます（マウスを使用、またはプログラム上で**SET LISTBOX COLUMN WIDTH**コマンドを使用）。

■ **On Header Click**：このイベントは、リストボックスの列ヘッダ上でクリックが行われると生成されます。この場合、Self関数を使用して、クリックされた列ヘッダを調べることができます。列や列ヘッダ上で右クリック（Windows）またはCtrl+クリック（Mac OS）を行うと、「**On Clicked**」イベントが生成されます。
リストボックスにおいて、「並び替え可」プロパティが選択されている場合、変数$0に値“0”または“-1”を渡すことにより、標準的なカラムソートを許可するかどうかを決定することができます。

  ■ $0が0の場合、標準の並び替えが実行されます。

  ■ $0が-1の場合、標準の並び替えは実行されず、ヘッダには並び替えを示す矢印が表示されません。この場合でも、開発者は4th Dimensionの配列管理用コマンドを使用し、独自の並び替え条件に基づいて列のソートを行うことができます。

リストボックスに対して、「並び替え可」プロパティが選択されていない場合、変数$0は使用されません。

■ **On After Sort**：このイベントは、標準の並び替えが実行された直後に生成されます（ただし、「**On Header Click**」イベントで$0に-1が返された場合、このイベントは生成されません）。このメカニズムは、ユーザが前回実行した並び替えの方向を保存する際に役立ちます。このイベントにおいて、**Self**関数は並び替えが行われた列の変数へのポインタを返します。

階層リスト

一般的なイベントに加えて、3つの特定のイベントを使用し、階層リスト上で実行されたユーザアクションを処理することができます。

■ **On selection change**：マウスクリックやキーストロークの後、階層リストの選択項目が変更されるたびに、このイベントが生成されます。
また、このイベントは、リストボックスオブジェクトや、レコードリストにおいても生成されます。

■ **On expand**：マウスクリックやキーストロークを用いて階層リストの項目が展開されるたびに、このイベントが生成されます。

■ **On collapse**：マウスクリックやキーストロークを用いて階層リストの項目が縮められるたびに、このイベントが生成されます。

これらのイベントは、相互に排他的ではありません。階層リストに対し、連続して生成される場合があります。

キーストローク後（順に）

イベント	状況
On data change	項目が編集された
On expand/ On collapse	→や←の矢印キーを用いてサブリストのオープン／クローズ
On selection change	新しい項目の選択
On clicked	キーボードを用いたリストのアクティブ化

マウスクリック後（順に）

イベント	状況
On data change	項目が編集された
On expand/ On collapse	展開／縮小アイコンを用いてのサブリストのオープン／クローズ、または編集不可のサブリスト上でダブルクリック
On selection change	新しい項目の選択
On clicked/ On double clicked	クリックやダブルクリックを用いてリストのアクティブ化

例題

ここで説明される例題は、すべてフォームとオブジェクトのプロパティが適切に選択されていることを前提としています。

1. 以下の例は、[親]テーブルのフォームが画面上に表示される前に、[親]子供サブテーブルのサブレコードのセレクションをソートします。

```
` [親]テーブルのフォームメソッド
Case of
	¥(Form event=On Load)
		ORDER SUBRECORDS BY([親]子供 ; [親]子供'名字 ; >)
		...
End case
```

2. 以下の例は、レコードが変更される日付を（フィールドに）自動的に割り当てるために使用されている**On Validate**イベントを示しています。

```
`任意のフォームメソッド
Case of
		...
	¥(Form event=On Validate)
		[テーブル]最新修正日:=Current date
End case
```

3. 以下の例では、ドロップダウンリストの処理（初期化、ユーザによるクリック、オブジェクトのリリース）がすべてオブジェクトのメソッド内に含まれています。

```
`「asBurgerSize」ドロップダウンリストのオブジェクトメソッド
Case of
	¥(Form event=On Load)
		ARRAY STRING(31 ; asBurgerSize ; 3)
		asBurgerSize{1}:="小"
		asBurgerSize{1}:="中"
		asBurgerSize{1}:="大"
	¥(Form event=On Clicked)
		If(asBurgerSize#0)
			ALERT("サイズ "+asBurgerSize{asBurgerSize}+
				" のハンバーガーを1つ！")
		End if
	¥(Form event=On Unload)
		CLEAR VARIABLE(asBurgerSize)
End case
```

4. 以下の例は、オブジェクトメソッドにおいて、ピクチャ値だけを受け入れるフィールドオブジェクトに対し、ドラッグ&ドロップ操作を受け付けたり、その処理を実行する方法を示しています。

```
`[テーブル]ピクチャ入力可能ピクチャフィールドのオブジェクトメソッド
Case of
   ¥(Form event=On Drag Over)
      `ドラッグ&ドロップ操作が開始され、マウスはカレントフィールド
                                                上にあります。
      `ソースオブジェクトについての情報を獲得してください。
      DRAG AND DROP PROPERTIES($vpSrcObject ; $vlSrcElement ;
                                                $lSrcProcess)
      `オブジェクトメソッドが例外的にプロセスのコンテキスト内で
                                実施されているためユーザ側はソースプロセスの
                              ID番号をテストする必要がないことに注意してください。
      $vlDataType:=Type($vpSrcObject->)
      `ソースデータはピクチャ(フィールド、変数、配列 ) ですか ?
      If(($vlDataType=Is Picture) |($vlDataType=Picture Array))
         `そうであれば、ドラッグを受け付けてください。マウスボタンが押さ
                                れたままであることに注意してください。
         `ドラッグを受け付けている間、4Dはオブジェクトを
                              ハイライトさせ、ユーザはソースデータをオブジェクト上に
                                ドロップできることがわかります。
         $0:=0
      Else
         `ドラッグを受け付けない場合
         $0:=-1
         `この場合、オブジェクトはハイライトされません。
      End if
   ¥(Form event=On Drop)
      `ソースデータがオブジェクト上にドロップされたので、これを
      `そのオブジェクトにコピーする必要があります。
      `ソースオブジェクトの情報を獲得してください。
      DRAG AND DROP PROPERTIES($vpSrcObject ; $vlSrcElement ;
                                                $lSrcProcess)
      $vlDataType:=Type($vpSrcObject->)
      Case of
         `ソースオブジェクトはピクチャ フィールドまたはピクチャ変数です。
         ¥($vlDataType=Is Picture)
            `ソースオブジェクトは同じプロセス
                                （つまり同じウインドウとフォームから）のものか ?
            If($lSrcProcess=Current process)
```

```
            `正しい場合、ソースの値のコピーだけを行う。
            [テーブル]ピクチャ:=$vpSrcObject->
         Else
            ` 違う場合、ソースオブジェクトは変数か？
            If(Is a variable($vpSrcObject))
               `正しい場合、ソースプロセスからその値を獲得する。
               GET PROCESS VARIABLE($lSrcProcess ;
                                    $vpSrcObject-> ; $vgDraggedPict)
               [テーブル]ピクチャ:=$vgDraggedPict
            Else
               `違う場合、ソースプロセスからフィールドの値を獲得する
                     ために CALL PROCESSコマンドを使用してください。
            End if
         End if
         `ソースオブジェクトはピクチャ配列です。
      ¥($vlDataType=Picture Array)
         `ソースオブジェクトは同じプロセス（つまり同じウインドウと
         `フォームから）のものですか？
         If($lSrcProcess=Current process)
            `正しい場合、ソースの値のコピーだけを行ってください。
            [テーブル]ピクチャ:=$vpSrcObject->{$vlSrcElement}
         Else
            `違う場合、ソースプロセスからその値を獲得してください。
            GET PROCESS VARIABLE($lSrcProcess ; $vpSrcObject
               ->{$vlSrcElement} ; $vgDraggedPict)
            [テーブル]ピクチャ:=$vgDraggedPict
         End if
   End case
End case
```

注：On Drag Overと On Dropイベントを操作する方法を示す他の例題に関しては、DRAG AND DROP PROPERTIESコマンドの例題を参照してください。

5. 以下の例は、フォームメソッド用のテンプレートです。ここでは、集計リポートフォームでフォームを出力フォームとして使用している間に発生する可能性のある各イベントを示しています。

```
`集計レポートの出力フォームとして使用されるフォームのメソッド
$vpFormTable:=Current form table
Case of
   ...
   ¥(Form event=On Header)
      `ヘッダエリアが印刷されようとしています。
```

```
            Case of
                ¥(Before selection($vpFormTable->))
                    `最初のブレークヘッダ用のコードはここです。
                ¥(Level =1)
                    `ブレークヘッダレベル1用のコードはここです。
                ¥(Level =2)
                    `ブレークヘッダレベル2用のコードはここです。
                    ...
            End case
        ¥(Form event=On Printing Details)
            `レコードが印刷されようとしています。
            ` 各レコード用のコードはここです。
        ¥(Form event=On Printing Break)
            ` ブレークが印刷されようとしています。
            Case of
                ¥(Level =0)
                    `ブレークレベル0用のコードはここです。
                ¥(Level =1)
                    `ブレークレベル1用のコードはここです。
                    ...
            End case
        ¥(Form event=On Printing Footer)
            If(End selection($vpFormTable->))
                `最後のフッタ用のコードはここです。
            Else
                ` フッタ用のコードはここです。
            End if
    End case
```

6. 以下の例は、イベントを操作するフォームメソッドのテンプレートを示しています。

このイベントは、**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドを使用して表示されるフォームに対して発生する可能性があるイベントです。解説の意味から、フォームウインドウのタイトルバーにイベントの性質を表示します。

```
`任意のフォームメソッド
Case of
    ¥(Form event=On Load)
        $vsTheEvent:="フォームが表示されようとしています。"
    ¥(Form event=On Unload)
        $vsTheEvent:="出力フォームが終了し、画面から消えようとしています。"
    ¥(Form event=On Display Details)
```

```
            $vsTheEvent:="表示しているレコード番号#"+String
                                                (Selected record number([テーブル]))
        ¥(Form event=On Menu Selected)
            $vsTheEvent:="メニュー項目が選択されています。"
        ¥(Form event=On Header)
            $vsTheEvent:="ヘッダエリアが描かれようとしています。"
        ¥(Form event=On Clicked”)
            $vsTheEvent:="レコードがクリックされました。"
        ¥(Form event=On Double Clicked”)
            $vsTheEvent:="レコードがダブルクリックされました。"
        ¥(Form event=On Open Details)
            $vsTheEvent:="レコード番号#"+String(Selected record number
                                    ([テーブル]))+" がダブルクリックされました。"
        ¥(Form event=On Close Details)
            $vsTheEvent:="出力フォームに戻ります。"
        ¥(Form event=On Activate)
            $vsTheEvent:="フォームのウインドウが最前面ウインドウに
                                                                なります。"
        ¥(Form event=On Deactivate)
            $vsTheEvent:="フォームのウインドウはもはや最前面ウインドウでは
                                                            ありません。"
        ¥(Form event=On Menu Selected)
            $vsTheEvent:="メニュー項目が選択されました。"
        ¥(Form event=On Outside call)
            $vsTheEvent:="他からの呼び出しを受信しました。"
    Else
        ˚ $vsTheEvent:="イベントNo."+String(Form event)
    End case
    SET WINDOW TITLE($vsTheEvent)
```

7. **On Before Keystroke**と**On After Keystroke**イベントの操作方法の例題に関しては、**Keystroke**関数と**FILTER KEYSTROKE**コマンドの例題を参照してください。

8. 以下の例は、クリックとダブルクリックをスクロール可能エリアと同じ方法で扱う方法を示しています。

```
    `「asChoices」スクロールエリアのオブジェクトメソッド
    Case of
        ¥(Form event=On Load)
            ARRAY STRING(...; asChoices ;...)
             ...
            asChoices:=0
```

```
        ¥((Form event=On Clicked) |(Form event=On Double Clicked))
            If(asChoices#0)
                `アイテムをクリックすると、ここで何かを行います。
                ...
            End if
            ...
    End case
```

9. 以下の例は、それぞれ異なる応答でクリックとダブルクリックを扱う方法を示しています。選択した要素を管理するために要素ゼロを使用するすることに注意してください。

```
    `「asChoices」スクロールエリアのオブジェクトメソッド
    Case of
        ¥(Form event=On Load)
            ARRAY STRING(...; asChoices;...)
            ...
            asChoices:=0
            asChoices{0}:="0"
        ¥(Form event=On Clicked)
            If(asChoices#0)
                If(asChoices # Num(asChoices))
                    `新規アイテムがクリックされました、ここで何かを
                                                実行します。
                    ...
                    `次回のために新しく選択した配列要素を保存して
                                                ください。
                    asChoices{0}:=String(asChoices)
                End if
            Else
                asChoices:=Num(asChoices{0})
            End if
        ¥(Form event=On Double Clicked)
            If(asChoices#0)
                `新規アイテムがダブルクリックされました、ここで違う作業
                                                を実行します。
            End if
                ...
    End case
```

10. この例題では、フォームメソッド内からステータステキスト情報領域を維持する方法を示しています。この場合、**On Getting Focus**と**On Losing Focus**イベントを使用します。

```
` [交渉];"データ入力" フォームメソッド
Case of
        ¥(Form Event=On Load)
                C_TEXT(vtStatusArea)
                vtStatusArea:=""
        ¥(Form Event=On Getting Focus)
                RESOLVE POINTER(Focus object ; $vsVarName ; $vlTableNum ;
                                                        $vlFieldNum)
                If(($vlTableNum#0) &($vlFieldNum#0))
                        Case of
                                ¥($vlFieldNum=1) `名字フィールド
                                        vtStatusArea:="交渉した人の名字を入力して
                                        ください。これは自動的に大文字になります。"
                                         ...
                                ¥($vlFieldNum=10) ` 郵便番号フィールド
                                        vtStatusArea:="5桁の郵便番号を入力して
                                                ください。これは自動的にチェックされ
                                                        有効になります。"
                                        ...
                        End case
                End if
        ¥(Form Event=On Losing Focus)
                vtStatusArea:=""
                ...
End case
```

11 この例題では、レコードデータ入力に使用されるフォームを使って、クローズウインドウイベントへ応答する方法を示しています。

```
`データ入力フォームのメソッド
$vpFormTable:=Current form table
Case of
                ...
        ¥(Form Event=On Close Box)
                If(Modified record($vpFormTable->))
                        CONFIRM("このレコードは変更されました。
                                                        変更を保存しますか？")
                        If(OK=1)
```

```
                ACCEPT
            Else
                CANCEL
            End if
        Else
            CANCEL
        End if
            ...
End case
```

12.この例題では、データソースの値が変更される度に、テキストや英数字フィールドを大文字にする方法を示しています。

```
` [交渉]名字フィールドのオブジェクトメソッド
Case of
            ...
    ¥(Form event=On Data Change)
        [交渉]名字:= Uppercase(Substring([交渉]名字 ; 1 ; 1))
                +Lowercase(Substring([交渉]名字 ; 2))
            ...
End case
```

参照

CALL PROCESS、Current form table、DRAG AND DROP PROPERTIES、FILTER KEYSTROKE、Get edited text、Keystroke、SET TIMER

Before

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Load**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

Before → ブール

説明

Before実行サイクルを発生させるためには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Load**イベントプロパティを必ず選択してください。

参照

Form event

During

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Clicked**等のイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

During → ブール

説明

During実行サイクルを発生させるためには、「デザイン」モードにおいて、そのフォームあるいはオブジェクトに対して、**On Clicked**等の適当なイベントを必ず選択してください。

参照

Form event

After

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Validate**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

After → ブール

説明

After実行サイクルを発生させるためには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Validate**イベントプロパティを必ず選択してください。

参照

Form event

In header

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Header**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

In header → ブール

説明

In header実行サイクルを発生させるためには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Header**イベントプロパティを必ず選択してください。

参照

During、In break、In footer

In break

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Printing Break**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

In break → ブール

説明

In break実行サイクルを発生させるためには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Printing Break**イベントプロパティを必ず選択してください。

参照

During、In break、In footer

In footer

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Printing Footer**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

In footer → ブール

説明

In footer実行サイクルを発生させるためには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Printing Footer**イベントプロパティを必ず選択してください。

参照

During、In break、In footer

Activated

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Activated**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

Activated → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	実行サイクルがアクティブ化である場合に“True”を返す

説明

Activated関数は、フォームを含むウインドウが最前面のプロセスの最前面のウインドウになると、フォームメソッドにおいて“True”を返します。

フォームの**Activated**フェーズに**TRACE**コマンドまたは**ALERT**コマンドを入れると、無限ループになるため、このような使用方法は避けてください。

注：Activated実行サイクルを発生させるには（V3データベースとの互換性のために）、「デザイン」モードでそのフォームのOn Activatedイベントプロパティを必ず選択してください。データベースを変換した際に、この作業は自動的に実行されます。

参照

Deactivated、Form event

Deactivated

互換性に関する注意

このコマンドは、互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Deactivate**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

Deactivated → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	実行サイクルが非アクティブ化である場合に“True”を返す

説明

Deactivated関数は、フォームを含む最前面プロセスの最前面ウインドウが背後に配置されると、フォームメソッドとオブジェクトメソッドに“True”を返します。

Deactivated実行サイクルを発生させるには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Deactivate**イベントプロパティを必ず選択してください。

参照

Activated、Form event

Outside call

互換性に関する注意

このコマンドは互換性を維持する目的で4Dに残されています。バージョン6からは、**Form event**関数を使用し、**On Outside call**イベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

Outside call → ブール

説明

Outside call実行サイクルを発生させるには、「デザイン」モードでそのフォームあるいはオブジェクトの**On Outside call**イベントプロパティを必ず選択してください。

参照

CALL PROCESS、Form event

Get edited text

Get edited text → Text

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	テキスト	←	入力されているテキスト

説明

このコマンドは、フォームオブジェクトに入力されているテキストを返します。

このコマンドは、主に入力されているテキストを検査するために、新しいフォームイベント**On After Keystroke**と共に使用されます。また、**On Before Keystroke**フォームイベントと共に使用することもできます。

注：新しいフォームイベントであるOn After Keystrokeに対応するために、従来のイベントであるOn KeystrokeはOn Before Keystrokeとなりました。

フォームオブジェクト内でテキストを入力しないコンテキストでは空白を返します。

例題

▼ 下記のメソッドは、入力される文字を自動的に大文字に変換します。

```
If(Form event=On After Keystroke)
    [Trips]Agencies:=Uppercase(Get edited text)
End if
```

▼ テキストフィールドへの入力文字の処理例があります。これは、他のテキストフィールド（ここでは"Words"という名前）に、入力された文章の単語を取り出すというものです。これを実行するには、フィールドのオブジェクトメソッド内に下記のコードを書き込みます。

```
If(Form event=On After Keystroke)
    $RealTimeEntry:=Get edited text
    PLATFORM PROPERTIES($platform)
    If($platform#3) ` Macintosh または Power Macintosh
        Repeat
            $DecomposedSentence:=Replace string
                ($RealTimeEntry;Char(32);Char(13))
```

```
            Until(Position(" ";$DecomposedSentence)=0)
        Else `Windows
        Repeat
            $DecomposedSentence:=Replace string($RealTimeEntry;
                                                Char(32);Char(13)+Char(10))
            Until(Position(" ";$DecomposedSentence)=0)
        End if
        [Example]Words:=$DecomposedSentence
    End if
```

注：この例は、単語がスペース（Char(32)）によって区切られていると仮定しているため、完全なものではありません。完全な解決法としては、すべての単語を抽出するように他のフィルタを付加する必要があります（カンマ、セミコロン、アポストロフィー等の区切り）。

参照

Form event

SET TIMER

SET TIMER(Tick数)

引数	タイプ		説明
Tick数	倍長整数	→	設定するTick数

説明

このコマンドは、現在のプロセスで、指定されたTick数（1Tick=1/60秒）ごとに**On Timer**フォームイベントを発生させます。

このコマンドがフォームを表示していないプロセス内で呼び出されても何も変化はありません。

4DのWebサーバで、このコマンドと**On Timer**フォームイベントを使用し、4Dフォームの再送処理をすることができます。

この方法は、バンド幅を節約しながら「リアルタイム」でHTMLページを更新することができます。この場合のフォームの更新は、実際には自動ではありません。**REDRAW**コマンドを呼び出す必要があります。そうすると、データが変更された場合にのみ**REDRAW**を呼び出すことによって、システムを最適化できます。

JavaScriptを解釈するブラウザは、ページを自動的に書き直すことが可能です。**SET TIMER**によって定義された時間がブラウザによって使用されます。これは数秒間（5が現実的な値です）であり、Webプロセスのタイムアウト時間でなければなりません。下記の2番目の例を参照してください。

On Timerフォームイベント発生を取り消すには、チック数に0をセットした**SET TIMER**をもう一度実行します。

例題

▼ フォームが画面に表示されている時に、コンピュータが3秒毎に警告音を鳴らすようにしたいと仮定します。これを実行するには、下記のようにフォームメソッドを書きます。

```
If(Form event=On Load)
        SET TIMER(60*3)
End if
If(Form event=On Timer)
        BEEP
End if
```

▼ Webサーバが、Webブラウザ上に表示されている4Dフォームを、5秒毎に更新するようにしたいと仮定します。これを実行するには、下記のようにフォームメソッドを書きます。

```
If(Form event=On Load)
        SET TIMER(60*5)
End if
If(Form event=On Timer)
`データが変更されている場合のみ以下の行を実行させるために、
`ここにデータ変更判断文を置くことができます。
        REDRAW([My Table];"My Form")
End if
```

参照

On Timerフォームイベント

Right click

Right click → ブール値

引数	タイプ		説明
この関数には、引数は必要ありません			
戻り値	ブール値	←	右クリックが検出された場合はTrue（真）、それ以外の場合はFalse（偽）

説明

Right click関数は、マウスの右ボタンがクリックされた場合にTrueを返します。

この関数は、「On clicked」フォームイベント内でのみ使用してください。したがって、ストラクチャモードにおいて、このイベントがフォームプロパティや特定のオブジェクトで適切に選択されていることを確認する必要があります。

注：この関数は、WindowsおよびMac OS Xでのみ動作します。Mac OS 9では常にFalseを返します。

参照

Contextual click、Form event

Contextual click

Contextual click → ブール値

引数	タイプ		説明
この関数には、引数は必要ありません			
戻り値	ブール値	←	コンテクスチュアルクリックが検出された場合はTrue（真）、それ以外の場合はFalse（偽）

説明

Contextual click関数は、コンテクスチュアルクリックが行われた場合にTrueを返します。

■ Windowsにおいて、コンテクスチュアルクリックはマウスの右ボタンを使用して行います。

■ Mac OSにおいて、コンテクスチュアルクリックは、Ctrlキーを押しながらクリックして行います。

この関数は、「On clicked」フォームイベント内でのみ使用してください。したがって、ストラクチャモードにおいて、このイベントがフォームプロパティや特定のオブジェクトで適切に選択されていることを確認する必要があります。

例題

このメソッドをスクロールエリアと組み合わせて使用すると、コンテキストメニューを用いて配列要素の値を変更することができます。

```
If(Contextual click)
    If(Pop up menu("True;False")=1)
        myArray{myArray}:="True"
    Else
        myArray{myArray}:="False"
    End if
End if
```

参照

Form event、Right click

GET FORM PROPERTIES

GET FORM PROPERTIES({テーブル ;} フォーム名 ; 幅 ; 高さ {; ページ数 {; 幅変更 {; 高さ変更 {; タイトル }}}})

引数	タイプ		説明
テーブル	テーブル	→	フォームが属しているテーブル 省略されている場合デフォルトテーブル
フォーム名	文字列	→	フォームの名前
幅	倍長整数	←	フォームの幅（ピクセル単位）
高さ	倍長整数	←	フォームの高さ（ピクセル単位）
ページ数	倍長整数	←	フォームのページ数
幅変更	ブール	←	True=幅サイズ変更可 False=ウインドウ幅固定
高さ変更	ブール	←	True=高さサイズ変更可 False=ウインドウ高さ固定
タイトル	テキスト	←	フォームのウインドウタイトル

説明

GET FORM PROPERTIESコマンドは、フォーム引数フォーム名でのプロパティを返します。

幅および高さの引数は、フォームの幅と高さをピクセル単位で返します。これはフォームプロパティのデフォルトウインドウサイズプロパティの値です。

- デフォルトウインドウサイズプロパティに自動サイズを指定している場合、幅と高さはフォーム内の全オブジェクトおよび全変数が表示可能となるように計算されます。
- デフォルトウインドウサイズプロパティに指定サイズを指定している場合、幅と高さはプロパティ内の幅と高さに入力されたものになります。
- デフォルトウインドウサイズプロパティにオブジェクトを指定している場合、幅と高さはこのオブジェクトの位置を元に計算されます。

ページ数引数は、0ページを除いたフォーム内のページ数を返します。

幅変更と高さ変更引数には、フォームの幅と高さがサイズ変更可であるか（True）それともウインドウ幅とウインドウ高さが固定に設定されているか（False）をそれぞれに返します。

タイトル引数には、フォームに定義されているウインドウタイトルを返します。名前が定義されていないとタイトル引数には空白を返します。

参照

GET FORM OBJECTS、Open form window、SET FORM SIZE

SET FORM SIZE

SET FORM SIZE ({オブジェクト; }水平; 垂直{;* })

引数	タイプ		説明
オブジェクト	文字列	→	フォームの境界を示すオブジェクト名
水平	倍長整数	→	“*”を渡した場合：水平マージン（ピクセル） “*”を省略した場合：幅（ピクセル）
垂直	倍長整数	→	“*”を渡した場合：垂直マージン（ピクセル） “*”を省略した場合：高さ（ピクセル
*	*	→	渡した場合： ・自動サイズ調整、引数<水平>と<垂直>に指定されたマージンを追加、または ・オブジェクトに基づいたサイズ調整（オブジェクトを渡した場合)、引数<水平>と<垂直>に指定されたマージンを追加 省略した場合： ・フォームの幅および高さとして<水平>と<垂直>を使用する

説明

このコマンドを使用すると、プログラムからカレントフォームのサイズを変更することができます。新しいサイズは、カレントプロセスに対して定義され、フォームには保存されません。

「デザイン」モードと同様に、このコマンドを使用して、3通りの方法でフォームサイズを設定することができます。

■ 自動設定（すべてのオブジェクトを必ず表示するという考えに基づいてフォームサイズを決定)。“*”を渡した場合は、水平や垂直マージンを追加します。

■ フォームオブジェクトが配置されていた位置に基づく。“*”を渡した場合は、この位置に水平や垂直マージンを追加します。

■ “固定”サイズを入力（幅と高さ)。

フォームサイズの変更に関する詳細は、4th Dimensionの『デザインリファレンス』マニュアルを参照してください。

自動サイズ調整

フォームサイズを自動的に設定したい場合、必ず次の構文を使用してください。

SET FORM SIZE(水平; 垂直;*)

この場合、フォームの右側と下側に追加したいマージン（ピクセル単位）を＜水平＞と＜垂直＞に代入して渡します。

オブジェクトに基づいたサイズ調整

任意のオブジェクトに基づいてフォームサイズを設定したい場合、必ず次の構文を使用してください。

SET FORM SIZE(オブジェクト; 水平; 垂直;*)

固定サイズ

フォームの固定サイズを設定したい場合、必ず次の構文を使用してください。

SET FORM SIZE(水平; 垂直)

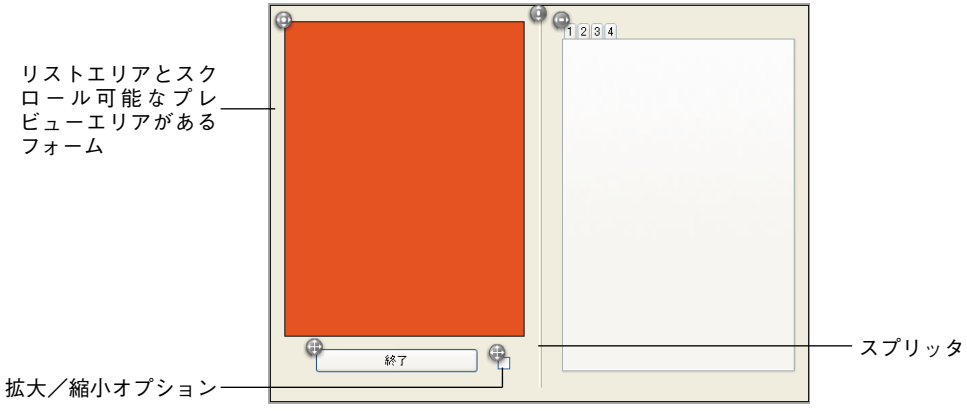
この場合、フォームの幅と高さ（ピクセル単位）を＜水平＞と＜垂直＞に代入して渡します。

SET FORM SIZEコマンドはフォームサイズを変更しますが、サイズ調整プロパティも考慮します。例えば、フォームの最小幅が500ピクセルの場合に、コマンドで幅を400ピクセルに設定すると、新しいフォーム幅は500ピクセルになります。

また、このコマンドはフォームウインドウのサイズは変更しないという点に注意してください。ウインドウサイズを変更しないでフォームサイズを変えることもできます（その逆も同様）。フォームウインドウのサイズを変更する方法については、前述の**RESIZE FORM WINDOW**コマンドを参照してください。

例題

▼ 次の例題は、「エクスプローラ」タイプのウインドウの設定方法を示しています。以下のフォームは「デザイン」モードで作成します。



注：ウインドウサイズは“自動”です。

次のコードを使用して、ウインドウが表示されます。

```
$ref:=Open form window([Table 1];"Form1";Standard form window;
                                    Horizontally centered;Vertically centered;*)
DIALOG([Table 1];"Form1")
CLOSE WINDOW
```

このウインドウの右部分は、拡大／縮小ボックスをクリックすることにより、表示したり、または隠すことができます。



このオプションに関連付けられたオブジェクトメソッドは次の通りです。

```
Case of
    ¥ (Form event=On load)
        C_BOOLEAN(b1;<>collapsed)
        C_LONGINT(margin)
```

```
                margin:=15
                b1:=<>collapsed
                If(<>collapsed)
                        SET FORM HORIZONTAL RESIZING((False)
                        SET FORM SIZE("b1";margin;margin)
                Else
                        SET FORM HORIZONTAL RESIZING(True)
                        SET FORM SIZE("tab";margin;margin)
                End if
        ¥ (Form event=On click)
                <>collapsed:=b1
                If(b1)
                        `縮小
                        GET OBJECT RECT(*;"b1";$l;$t;$r;$b)
                        GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
                        SET WINDOW RECT($lf;$tf;$lf+$r+margin;
                                                $tf+$b+margin;Current form window)
                        SET FORM HORIZONTAL RESIZING(False)
                        SET FORM SIZE("b1";margin;margin)
                Else
                        `展開
                        GET OBJECT RECT(*;"tab";$l;$t;$r;$b)
                        GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
                        SET WINDOW RECT($lf;$tf;$lf+$r+margin;
                        $tf+$b+margin;Current form window)
                        SET FORM HORIZONTAL RESIZING(True)
                        SET FORM SIZE("tab";margin;margin)
                End if
        End case
```

参照

SET FORM HORIZONTAL RESIZING、SET FORM VERTICAL RESIZING

SET FORM HORIZONTAL RESIZING

SET FORM HORIZONTAL RESIZING (リサイズ{; 最小幅{; 最大幅}})

引数	タイプ		説明
リサイズ	ブール	→	True（真）：フォームは水平方向にリサイズ可能 False（偽）：フォームは水平方向にリサイズできない
最小幅	倍長整数	→	許容される最小フォーム幅（ピクセル）
最大幅	倍長整数	→	許容される最大フォーム幅（ピクセル）

説明

SET FORM HORIZONTAL RESIZINGコマンドを使用すると、プログラムからカレントフォームの水平リサイズプロパティを変更することができます。デフォルトとして、これらのプロパティは「デザイン」モードの「フォーム」エディタにおいて設定することができます。新しいプロパティはカレントプロセスに対して設定され、フォームと一緒に保存されません。

引数＜リサイズ＞を用いて、フォームを水平方向にリサイズできるかどうか、つまり、幅の変更が可能かどうかを定義します（ユーザが手動で変更、またはプログラムから変更）。

＜True＞を渡すと、ユーザはフォームの幅を変更することができます。4th Dimensionはマーカとして＜最小幅＞と＜最大幅＞に代入された値を使用します。

＜False＞を渡すと、ユーザはカレントフォームの幅を変更できません。この場合、引数＜最小幅＞と＜最大幅＞に値を渡す必要はありません。

1番目の引数に＜True＞を渡した場合は、任意の引数＜最小幅＞と＜最大幅＞に最小幅と最大幅の新しい値（ピクセル単位）を渡すことができます。これらの引数を省略した場合、「デザイン」モードで設定した値（設定されている場合）が使用されます。

例題

SET FORM SIZEの例題を参照してください。

参照

SET FORM SIZE、SET FORM VERTICAL RESIZING

SET FORM VERTICAL RESIZING

SET FORM VERTICAL RESIZING (リサイズ{; 最小高さ{; 最大高さ}})

引数	タイプ		説明
リサイズ	ブール	→	True（真）：フォームは垂直方向にリサイズ可能 False（偽）：フォームは垂直方向にリサイズできない
最小高さ	倍長整数	→	許容されるフォームの最小の高さ（ピクセル）
最大高さ	倍長整数	→	許容されるフォームの最大の高さ（ピクセル）

説明

SET FORM VERTICAL RESIZINGコマンドを使用すると、プログラムからカレントフォームの垂直リサイズプロパティを変更することができます。デフォルトとして、これらのプロパティは「デザイン」モードの「フォーム」エディタにおいて設定することができます。新しいプロパティはカレントプロセスに対して設定され、フォームと一緒に保存されません。

引数＜リサイズ＞を用いて、フォームを垂直方向にリサイズできるかどうか、つまり、高さの変更が可能かどうかを定義します（ユーザが手動で変更、またはプログラムから変更）。

＜True＞を渡すと、ユーザはカレントフォームの高さを変更することができます。4th Dimensionはマーカとして＜最小高さ＞と＜最大高さ＞に代入された値を使用します。

＜False＞を渡すと、ユーザはカレントフォームの高さを変更できません。この場合、引数＜最小高さ＞と＜最大高さ＞に値を渡す必要はありません。

1番目の引数に＜True＞を渡した場合は、任意の引数＜最小高さ＞と＜最大高さ＞に最小の高さと最大の高さの新しい値（ピクセル単位）を渡すことができます。これらの引数を省略した場合、「デザイン」モードで設定した値（設定されている場合）が使用されます。

例題

SET FORM SIZEの例題を参照してください。

参照

SET FORM HORIZONTAL RESIZING、SET FORM SIZE

GET FORM OBJECTS

GET FORM OBJECTS (オブジェクト配列{; 変数配列{; ページ配列}}{; *})

引数	タイプ		説明
オブジェクト配列	文字列配列	←	フォームオブジェクト名
変数配列	ポインタ配列	←	オブジェクトに関連付けた変数やフィールドへのポインタ
ページ配列	整数配列	←	各オブジェクトのページ番号
*	*	→	渡した場合：カレントページに限定する

説明

GET FORM OBJECTSコマンドは、カレントテーブルのカレントフォームに存在する全オブジェクトのリストを配列形式で返します。このリストは、カレントフォームページのオブジェクトに限定することができます。このコマンドは、入力フォームおよび出力フォームの双方で使用することができます。

引数として渡した配列が事前に定義されていない場合、コマンドはその配列を作成し、サイズを自動的に設定します。しかし、アプリケーションをコンパイルする場合を考慮し、各配列を明示的に宣言することをお勧めします。

オブジェクト名を格納する文字列配列の名前（フォーム内の各オブジェクトはユニークな名前を持つ）を＜オブジェクト配列＞に渡します。配列内でのオブジェクトの出現順序は重要ではありません。

コマンドにより代入される他の任意の配列は、1番目の配列との同期が取られます。

任意の引数＜変数配列＞にはポインタ配列の名前を渡し、この配列にはオブジェクトに関連付けられている変数やフィールドへのポインタが格納されます。オブジェクトに関連付けられた変数が存在しない場合、ポインタ“Nil”が返されます。“サブフォーム”オブジェクトタイプが存在する場合、サブフォームのテーブルへのポインタが返されます。

3番目の配列（任意）＜ページ配列＞には、フォームのページ番号が代入されます。この配列の各要素には、対応するオブジェクトのページ番号が格納されます。

継承フォームによってもたらされるオブジェクトは、カレントページのページ 0に属しているものとみなされます。

任意の引数＜*＞を使用すると、返されるオブジェクトのリストをフォームのカレントページに限定することができます。この引数を渡した場合、コマンドはカレントページ、ページ0、継承ページのオブジェクトだけを返します。このコマンドは、フォームのカレントページ上に存在するあらゆるオブジェクトを処理します。

参照

GET FORM PROPERTIES

GOTO PAGE

GOTO PAGE(ページ番号)

引数	タイプ		説明
ページ番号	数値	→	表示するフォームページ

説明

GOTO PAGEコマンドは、現在表示されているフォームページを＜ページ番号＞で指定したフォームページに変更します。

フォームが表示されていない場合には、**GOTO PAGE**コマンドは何も実行しません。＜ページ番号＞が実際のフォームページの数よりも大きければ、最終ページを表示します。＜ページ番号＞に1以下の数が指定されると、先頭のフォームページを表示します。

フォームページ管理のコマンドについて

「**FIRST PAGE**」、「**LAST PAGE**」、「**NEXT PAGE**」、「**PREVIOUS PAGE**」および「**GOTO PAGE**」コマンドと同様の処理を実行する事が可能な「標準アクション」ボタンは、タブコントロールやポップアップ／ドロップダウンリストなどのオブジェクトに割り当てることができます。適宜、コマンドの代わりに「標準アクション」ボタンを使用してください。

フォームページ管理のコマンドは入力フォーム、もしくはダイアログウインドウに表示されているフォームに使用できます。出力フォームは最初のページのみを使います。フォームには、常に少なくとも最初の1ページが含まれます。フォームのページ数にかかわらず、各フォーム対するフォームメソッドは1つのみという点に留意してください。

表示されているページを確認するときは、**Current form page**関数を使用してください。

注：フォームのデザインは 1 ページからXページについて、もしくは0ページ（0ページに配置されたオブジェクトはすべてのページに表示される）について作業を行うことができます。フォームを使用してpage関数を呼び出した場合、1ページからXページについての作業を行うことになります。0ページ目は自動的に表示されているページと組み合わされます。

例題

▼以下の例はボタンのオブジェクトメソッドです。これは指定したフォームページ3を表示します。

```
GOTO PAGE(3)
```

参照

Current form page、FIRST PAGE、LAST PAGE,、NEXT PAGE、PREVIOUS PAGE

FIRST PAGE

FIRST PAGE

説明

FIRST PAGEコマンドは、現在表示されているフォームページを先頭のフォームページに変更します。フォームが表示されていない場合や最初のフォームページが表示されている場合には、**FIRST PAGE**コマンドは何も実行しません。

例題

▼以下の例は、メニューから呼び出される1行のメソッドです。

これは、先頭のフォームページを表示します。

```
FIRST PAGE
```

参照

Current form page、GOTO PAGE、LAST PAGE、NEXT PAGE、PREVIOUS PAGE

LAST PAGE

LAST PAGE

説明

LAST PAGEコマンドは、現在表示されているフォームページを最終のフォームページに変更します。フォームが表示されていない場合や最終のフォームページが表示されている場合には、**LAST PAGE**コマンドは何も実行しません。

例題

▼以下の例は、メニューから呼び出される1行のメソッドです。これは、最終のフォームページを表示します。

```
LAST PAGE
```

参照

Current form page、FIRST PAGE、GOTO PAGE、NEXT PAGE、PREVIOUS PAGE

NEXT PAGE

NEXT PAGE

説明

NEXT PAGEコマンドは、現在表示されているフォームページを次ページに変更します。フォームが表示されていない場合や最終のフォームページが表示されている場合には、**NEXT PAGE**コマンドは何も実行しません。

例題

▼以下の例は、メニューから呼び出される1行のメソッドです。現在表示されているフォームページの以下のフォームページを表示します。

```
NEXT PAGE
```

参照

Current form page、FIRST PAGE、GOTO PAGE、LAST PAGE、PREVIOUS PAGE

PREVIOUS PAGE

PREVIOUS PAGE

説明

PREVIOUS PAGEコマンドは、現在表示されているフォームページを前のフォームページに変更します。フォームが表示されていない場合や先頭のフォームページが表示されている場合には、**PREVIOUS PAGE**コマンドは何も実行しません。

例題

▼以下の例は、メニューから呼び出される1行のメソッドです。現在表示されているフォームページの前のフォームページを表示します。

```
PREVIOUS PAGE
```

参照

Current form page、FIRST PAGE、GOTO PAGE、LAST PAGE、NEXT PAGE

Current form page

Current form page → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	現在表示されているフォームページ番号

説明

Current form page関数は、現在表示されているフォームページの番号を返します。

例題

▼ フォームにおいて、メニューバーから任意のメニューを選択、またはそのフォームが別プロセスからの呼び出しを受信した場合に、現在表示されているフォームページに応じて異なる動作を実行することができます。この例を以下に示します。

```
` [MyTable]"MyForm"  フォームメソッド
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
                    ¥($vlItemNumber=...)
                        ¥(Current form page=1)
                            `ページ1に合った処理を行う
                        ¥(Current form page=2)
                            `ページ2に合った処理を行う
                            ` ...
                    ¥($vlItemNumber=...)
                            ` ...
                End case
            ¥($vlMenuNumber=...)
                ` ...
        End case
    ¥(Form event=On Outside call)
```

```
        Case of
            ¥(Current form page=1)
                `ページ1に合った処理を行う
            ¥(Current form page=2)
                ` ページ2に合った処理を行う
        End case
        ` ...
    End case
```

参照

FIRST PAGE、GOTO PAGE、LAST PAGE、NEXT PAGE、PREVIOUS PAGE

INPUT FORM

INPUT FORM ({テーブル; }フォーム{; ユーザフォーム}{; *})

引数	タイプ		説明
テーブル	テーブル	→	入力フォームが設定されたテーブル、または、省略した場合はデフォルトテーブル
フォーム	文字列	→	入力フォームとして設定するフォーム名
ユーザフォーム	文字列	→	使用されるユーザフォーム名
*	*	→	自動ウインドウサイズ

説明

INPUT FORMコマンドは、＜フォーム＞または＜ユーザフォーム＞を＜テーブル＞のカレント入力フォームとして設定します。

このコマンドのスコープは、カレントプロセスです。各テーブルは、プロセスごとに個々の入力フォームを持っています。

INPUT FORMコマンドは他のコマンドで表示、あるいは使用するフォームを指定するだけです。フォームを表示することはできません。フォームの作成に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

デフォルト入力フォームは、「デザイン」モードの「エクスプローラ」ウインドウで各テーブルごとに定義します。**INPUT FORM**コマンドで入力フォームを指定しない場合や、指定したフォームが存在しない場合には、ここで指定したデフォルト入力フォームが使用されます。

任意の引数＜ユーザフォーム＞を使用すると、ユーザフォーム（＜フォーム＞を基に作成）をデフォルトの入力フォームとして設定することができます。有効なユーザフォーム名を渡すと、カレントプロセスの入力フォームに代わり、このフォームがデフォルトとして使用されます。これにより、複数の異なるカスタムユーザフォーム（**CREATE USER FORM**コマンドで生成されたフォーム）を同時に使用できるようになり、現在の状況で一番効果を発揮するフォームを使用することができます。

ユーザフォームに関する詳細は、「ユーザフォーム概要」の節を参照してください。

入力フォームは多くのコマンドで表示されますが、一般にデータの入力や修正に使用されます。以下のコマンドは、データ入力や検索用に入力フォームを表示します。

ADD RECORD **MODIFY RECORD**
DISPLAY RECORD **QUERY BY EXAMPLE**

DISPLAY SELECTIONや**MODIFY SELECTION**コマンドは、出力フォームを使用してレコードのリストを表示します。ユーザがリスト上のレコードをダブルクリックすると、入力フォームを表示します。

データ読み込みコマンド**IMPORT TEXT**、**IMPORT SYLK**、**IMPORT DIF**は、レコードの読み込みにカレント入力フォームを使用します。

オプション引数<*>は、「デザイン」モードの「フォームプロパティ」ウインドウおよび**Open window**で設定するフォームプロパティと一緒に使用されます。<*>を指定することにより、次回、（ダイアログボックスや入力フォームとして）フォームを使用する際、フォームプロパティの設定をもとに自動的にウインドウサイズを変更するよう4Dに指示します。詳しくは**Open window**の節を参照してください。

注：オプション引数<*>を使用するしないに関係なく、INPUT FORMコマンドはテーブルの入力フォームを変更します。

例題

1. 以下の例は、**INPUT FORM**コマンドの一般的な使用方法です。

```
INPUT FORM([会社] ; "会社追加") ` 新しい会社を追加するためのフォームを指定する
ADD RECORD([会社])        ` 新しい会社を追加する
```

2. 複数の会社を処理する請求書データベースでは、対応するユーザフォームを用いて請求書を作成しなくてはなりません。

```
Case of
¥(company="4D SA")
        INPUT FORM([請求書];"Input";"4D_SA")
¥(company="4D Inc")
        INPUT FORM(([請求書];"Input";"4D_Inc")
¥(company="Acme")
        INPUT FORM(([請求書];"Input";"ACME")
End case
ADD RECORD([請求書])
```

参照

ADD RECORD、CREATE USER FORM、DISPLAY RECORD、DISPLAY SELECTION、IMPORT DIF、IMPORT SYLK、IMPORT TEXT、MODIFY RECORD、MODIFY SELECTION、Open window、OUTPUT FORM、QUERY BY EXAMPLE

OUTPUT FORM

OUTPUT FORM({テーブル; }フォーム{; ユーザフォーム})

引数	タイプ		説明
テーブル	テーブル	→	出力フォームを設定するテーブル、または省略した場合はデフォルトテーブル
フォーム	文字列	→	フォーム名
ユーザフォーム	文字列	→	使用されるユーザフォーム名

説明

OUTPUT FORMコマンドは、＜フォーム＞または＜ユーザフォーム＞を＜テーブル＞のカレント出力フォームとして設定します。このフォームは、＜テーブル＞に属するものでなければなりません。

このコマンドのスコープは、カレントプロセスです。各テーブルは、個々の出力フォームを持っています。

OUTPUT FORMコマンドは、他のコマンドで印刷、表示、使用されるフォームを指定するだけです。フォームを表示することはできません。フォームの作成に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

デフォルト出力フォームは、「デザイン」モードの「エクスプローラ」で各テーブルごとに定義します。**OUTPUT FORM**コマンドで出力フォームを指定しない場合や、指定したフォームが存在しない場合には、デフォルト出力フォームが使用されます。

任意の引数＜ユーザフォーム＞を使用すると、ユーザフォーム（＜フォーム＞を基に作成）をデフォルトの出力フォームとして設定することができます。有効なユーザフォーム名を渡すと、カレントプロセスの出力フォームに代わり、このフォームがデフォルトとして使用されます。これにより、複数の異なるカスタムユーザフォーム（**CREATE USER FORM**コマンドで生成されたフォーム）を同時に使用できるようになり、現在の状況で一番効果を発揮するフォームを使用することができます。

ユーザフォームに関する詳細は、「ユーザフォーム概要」の節を参照してください。

出力フォームは3つのコマンドグループ（画面上にレコードをリスト表示するグループ、レポートを作成するグループ、データを書き出すグループ）で使用します。**DISPLAY SELECTION**や**MODIFY SELECTION**コマンドは、出力フォームを使用してレコードのリストを表示します。**PRINT LABEL**や**PRINT SELECTION**コマンドを使用してレポートを作成する際には出力フォームを使用します。各データ書き出しコマンド（**EXPORT DIF**、**EXPORT SYLK**、**EXPORT TEXT**）でも出力フォームを使用します。

例題

▼以下の例は、**OUTPUT FORM**コマンドの一般的な使用方法です。この例では、**OUTPUT FORM**コマンドを出力フォームが使用される直前に記述していますが、これは必要はありません。実際、この**OUTPUT FORM**コマンドがこのメソッドの前に実行されていれば、このコマンドを全く別のメソッドで実行しても構いません。

```
INPUT FORM([部品] ; "部品入力")  ` 入力フォームの選択
OUTPUT FORM([部品 ; "部品リスト")  ` 出力フォームの選択
MODIFY SELECTION([部品])  ` このコマンドは両方のフォームを使用する
```

参照

CREATE USER FORM、DISPLAY SELECTION、EXPORT DIF、EXPORT SYLK、EXPORT TEXT、INPUT FORM、MODIFY SELECTION、PRINT LABEL、PRINT SELECTION

グラフコマンド 20

この章では、「ルーチン」エディタの「Graphs」テーマ内にあるグラフコマンドについて説明します。

バージョン6の注意点

バージョン6から、グラフ機能は4th Dimensionの中にあらかじめ組み込まれている4D Chartプラグインでサポートされるようになりました。以前のバージョンの4Dで使用されていたグラフコマンドは4D Chart用に透過的に書き直されます。さらに、フォーム上に配置されたグラフエリアをカスタマイズするために、4D Chartコマンドを使用するには、4D Chart用のプラグイン参照番号として引数＜グラフエリア＞を使用します（このコマンドの説明を参照）。4D Chartコマンドに関する詳細は、『4D Chartリファレンスマニュアル』を参照してください。

GRAPHコマンドに関する注意

GRAPHコマンド、GRAPH SETTINGSコマンドは、4Dフォームのグラフエリアとともに使用されるよう設計されています。このコマンドは、必ずフォームメソッドまたは、フォームオブジェクトのオブジェクトメソッドで使用してください。また、これらのメソッドから呼び出されるメソッド内で使用することもできます。

グラフは、2種類の方法で作成します。レコードのデータからグラフ化（**GRAPH TABLE**コマンド）、またはサブフィールドまたは配列の情報からグラフ化（**GRAPH**コマンド）します。**GRAPH TABLE**コマンドは、グラフを作成するためにレコード上のフィールドのデータを使用します。このコマンドは、自分自身のウインドウにグラフを表示します。**GRAPH**コマンドは、配列またはサブフィールドの情報を使用して、フォーム上のグラフエリアにグラフを作成します。

GRAPH **GRAPH SETTINGS** **GRAPH TABLE**

GRAPH

GRAPH(グラフエリア ; グラフ番号 ; xラベル ; y要素1 { ;...;y要素8})

引数	タイプ		説明
グラフエリア	変数	→	フォーム上のグラフエリア
グラフ番号	数値	→	グラフタイプ番号
xラベル	配列または サブフィールド	→	X軸のラベル
y要素	配列または サブフィールド	→	グラフへのデータ（最大8個）

説明

GRAPHコマンドは、フォームのグラフエリアにグラフを作成します。データは、配列またはサブフィールドから取り出されます。

引数＜グラフエリア＞は、グラフを表示するグラフエリアの名前です。グラフエリアは、「フォーム」エディタでグラフオブジェクトタイプを使用して作成します。グラフ名は、その変数に入力された名前です。グラフエリアの作成に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

引数＜グラフ番号＞は、作成されるグラフタイプを定義します。これは1から8までの数値でなければなりません。グラフタイプは次ページの例題1を参照してください。グラフタイプを変更する場合は、グラフを作成した後で＜グラフ番号＞を変更し、**GRAPH**コマンドをもう一度実行します。

引数＜xラベル＞は、X軸（グラフの一番下）に使用するラベルを定義します。このデータは、文字列、日付、時間、数値タイプのいずれでも構いません。＜xラベル＞は、＜y要素＞のサブレコードや配列要素と同じ数のサブレコードまたは配列要素がなければなりません。

引数＜y要素＞で指定するデータは、グラフにするデータです。このデータは数値でなければなりません。最大8つのデータセットをグラフ化することができます。それぞれをセミコロン（;）で区切って指定します。円グラフは、最初の＜y要素＞のみをグラフ化します。

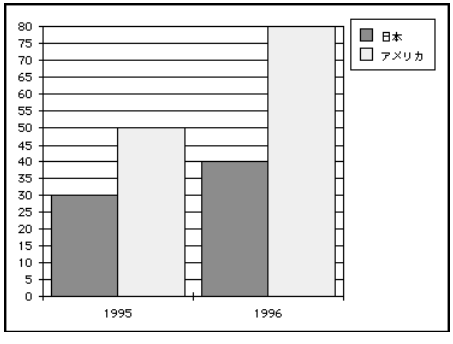
例題

1. 以下の例は、配列を使用してグラフを作成します。このコードは、フォームメソッドやオブジェクトメソッドで使用します。データが定数のため、あまり現実的ではありません。

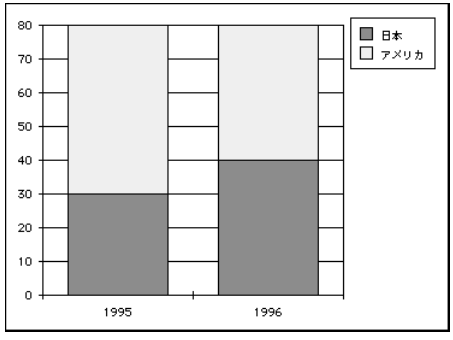
```
ARRAY STRING(4 ; X ; 2)  ` X軸の配列作成
X{1}:="1995"  ` X Label #1
X{2}:="1996"  ` X Label #2
ARRAY REAL(A ;2) ` Y軸の配列作成
A{1}:=30      ` データを挿入
A{2}:=40
ARRAY REAL(B ; 2)` Y軸の配列作成
B{1}:=50      ` データを挿入
B{2}:=80
GRAPH(vグラフ ; vタイプ ; X ; A ; B)      ` グラフ作成
 ` グラフの凡例をセット
GRAPH SETTINGS(vグラフ ; 0 ; 0 ; 0 ; 0 ; False ; False ; True ;"日本" ;
                                                    "アメリカ")
```

下図は、実行結果のグラフを表しています。

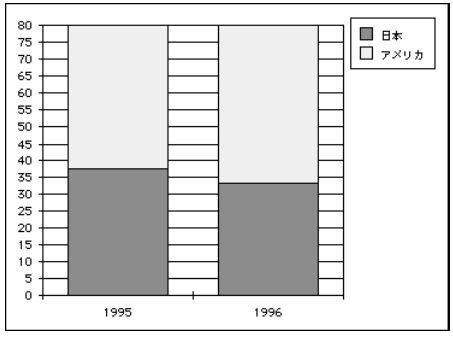
■ 「vタイプ」を1にすると、「棒グラフ」で表示します。



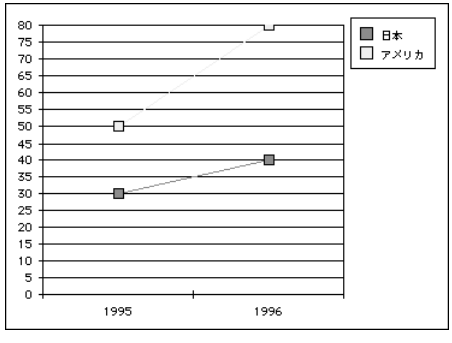
■「vタイプ」を2にすると、「棒・比率グラフ」で表示します。



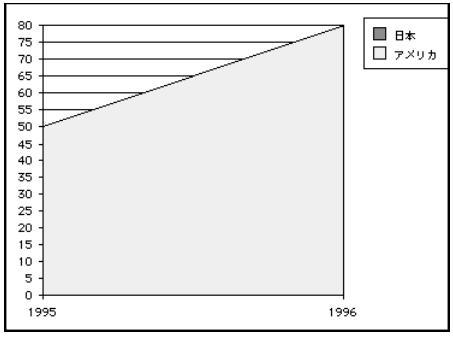
■「vタイプ」を3にすると、「棒・累積グラフ」で表示します。



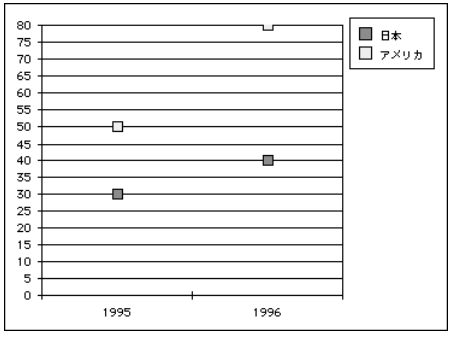
■「vタイプ」を4にすると、「線グラフ」で表示します。



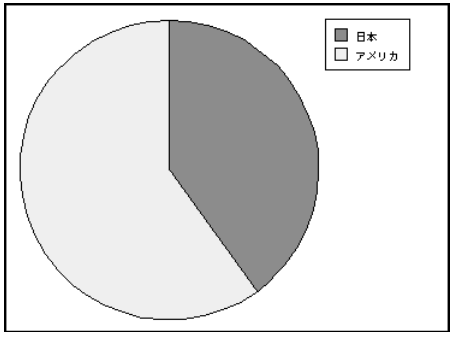
■「vタイプ」を5にすると、「面グラフ」で表示します。



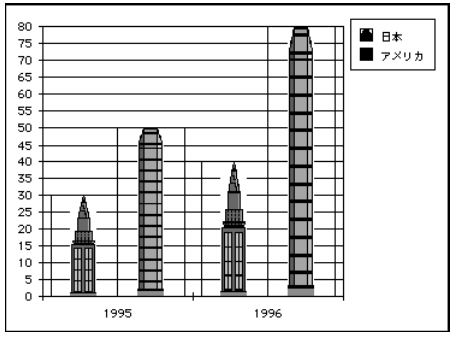
■「vタイプ」を6にすると、「点グラフ」で表示します。



■「vタイプ」を7にすると、「円グラフ」で表示します。



■「vタイプ」を8にすると、「ピクチャグラフ」で表示します。



2. 以下の例は、サブテーブルに入っているセールスマンの売上金額をグラフ化します。サブテーブルには名前、去年の合計売上金額、今年の合計売上金額の3つのサブフィールドがあります。このグラフは最後の2年間のセールスマンのそれぞれの売上を示します。

```
GRAPH(売上グラフ ; 1 ; [従業員]セールス'名前 ; [従業員]セールス'昨年度合計 ;
                                                [従業員]セールス'本年度合計)
```

参照

GRAPH SETTINGS、GRAPH TABLE

GRAPH SETTINGS

GRAPH SETTINGS(グラフ ; x最小値 ; x最大値 ; y最小値 ; y最大値 ; xプロップ; xグリッド ; yグリッド ; タイトル1{; タイトル2 ; ... ; タイトルN})

引数	タイプ		説明
グラフ	変数	→	グラフエリアの名前
x最小値	数値／日付／時間	→	比例グラフのx軸の最小値（線グラフまたは点グラフのプロットのみ）
x最大値	数値／日付／時間	→	比例グラフのx軸の最大値（線グラフまたは点グラフのプロットのみ）
y最小値	数値	→	最小のY軸の値
y最大値	数値	→	最大のY軸の値
xプロップ	ブール	→	比例X軸に対して“True（真）” ノーマルX軸に対して“False（偽）” （線グラフまたは点グラフのプロットのみ）
xグリッド	ブール	→	X軸グリッドに対して“True（真）” 非X軸グリッドに対して“False（偽）” （xプロップが“True（真）”の場合のみ）
yグリッド	ブール	→	Y軸グリッドに対して“True（真）” 非Y軸グリッドに対して“False（偽）”
タイトル	文字列	→	グラフ凡例のタイトル

説明

GRAPH SETTINGSコマンドは、フォームに表示されたグラフの設定値を変更します。グラフは**GRAPH**コマンドを使用して既に表示されている必要があります。**GRAPH SETTINGS**コマンドは、円グラフ（タイプ7）に対しては無効です。

引数＜x最小値＞、＜x最大値＞、＜y最小値＞、＜y最大値＞にグラフのそれぞれの軸の最大値と最小値を指定します。これらの引数の値が空（ヌル）の場合(0、!!00:00:00!!、!00.00.00!等、データ型によって異なります)、デフォルトのグラフ値を使用します。

引数＜xプロップ＞は、線グラフ（タイプ4）と点グラフ（タイプ6）に対する比例プロットを変更します。“True”であれば、点の値に従ってX軸上の各点をプロットします。ただし、値が数値、時間、日付の場合に限ります。

＜xグリッド＞と＜yグリッド＞は、グリッドラインを表示、または非表示にします。X軸のグリッドは、プロットが比例する点グラフまたは線グラフの場合にのみ表示されます。

引数＜タイトル＞は、指定した文字列を凡例のラベルとして表示します。

前ページの**GRAPH**コマンドの例を参照してください。

参照

GRAPH、GRAPH TABLE

GRAPH TABLE

GRAPH TABLE {(テーブル)}

または

GRAPH TABLE({テーブル ;} グラフ番号 ; xフィールド ; yフィールド1{ ;...;
yフィールド8})

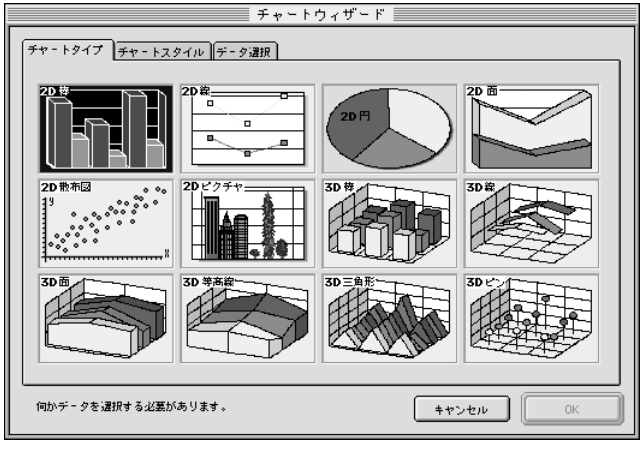
引数	タイプ		説明
テーブル	テーブル	→	グラフ化するテーブル、または 省略した場合、デフォルトテーブル
グラフ番号	数値	→	グラフタイプ番号
xフィールド	フィールド	→	X軸に対するラベル
yフィールド	フィールド	→	グラフ化するフィールド（最大8個）

説明

GRAPH TABLEコマンドには、2つの形式があります。第1の形式では、「チャートウィザード」が表示され、ユーザはグラフにするフィールドを選択します。第2の形式は、グラフ化するフィールドを指定し、「チャートウィザード」を表示しません。

GRAPH TABLEコマンドは、テーブルのフィールドのデータをグラフにします。また、カレントプロセスのカレントセレクションのデータだけをグラフにします。

第1の形式は、「ユーザ」モードの「レポート」メニューから「チャート...」を選択した場合と同じです。以下の図は「チャートウィザード」です。ユーザは、このウインドウでグラフを定義します。



GRAPH TABLEコマンドの第2の形式は、＜テーブル＞に指定したフィールドをグラフにします。

引数＜グラフ番号＞は、作成するグラフのタイプを定義します。番号は1から8までの数値を指定します。グラフタイプに関する詳細は、**GRAPH**コマンドの例題を参照してください。

引数＜xフィールド＞は、X軸（グラフの底部）につけるラベルを定義します。フィールドのタイプは文字、整数、倍長整数、実数、日付のいずれでも構いません。

引数＜yフィールド＞は、グラフのデータです。フィールドのタイプは文字、整数、倍長整数、実数のいずれかでなくてはなりません。＜yフィールド＞は、最大8個まで指定することができ、各引数はセミコロン（;）で区切ります。

いずれの形式でも、**GRAPH TABLE**コマンドは、新しく作成されたグラフで作業できるようにチャートウインドウを開きます。チャートウインドウの使用方法に関する詳細は、『4th Dimensionユーザリファレンス』を参照してください。

注：「クイックレポート」エディタの「出力先」メニューを選択して、フィールドデータからグラフを作成することができます。

例題

1. 以下の例は、**GRAPH TABLE**コマンドの第1の形式の説明です。これは、「チャートウィザード」ウインドウを表示し、グラフ化するフィールドをユーザが選択します。この例では[従業員]テーブルの検索とソートを実行し、「チャートウィザード」を表示します。

```
QUERY([従業員])    ` 従業員テーブルを検索
If(OK=1)
      ORDER BY([従業員])        ` 従業員テーブルをソート
      If(OK=1)
            GRAPH TABLE([従業員])   ` 従業員テーブルをグラフにする
      End if
End if
```

2. 以下の例は、**GRAPH TABLE**コマンドの第2の形式の説明です。最初に[従業員]テーブルを検索して並び替え、その給与をグラフ化します。

```
QUERY([従業員] ; [従業員]役職名="課長") ` 従業員テーブルから課長を検索
ORDER BY([従業員] ; [従業員]給与 ; >)    ` 給与でソート
GRAPH TABLE([従業員] ; 1 ; [従業員]名字 ; [従業員]給与) ` 課長の給与をグラフにする
```

参照

Graph

SET ALLOWED METHODS

SET ALLOWED METHODS (メソッド配列)

引数	タイプ		説明
メソッド配列	文字列配列	→	メソッド名の配列

説明

SET ALLOWED METHODSコマンドを使用して、カレントプロセスの「フォーミュラ」エディタ上に表示されるメソッドを指定することができます。指定したメソッドはコマンドリストの最後に表示され、フォーミュラ上で利用することができます。デフォルトでは（このコマンドを使用しない場合)、「フォーミュラ」エディタ上にメソッドは表示されません。許可されていないメソッド名がフォーミュラで使用されている場合、シンタックスエラーが生成され、フォーミュラの確定はできません。

引数＜メソッド配列＞には、「フォーミュラ」エディタに提示するメソッドリストを格納する配列の名前を渡します。この配列は、事前に定義しておかなければなりません。

メソッド名にワイルドカード記号（@）を使用し、許可されるメソッドグループを1つ以上定義することができます。

注：デフォルトとして許可されていない4Dコマンドやプラグインコマンドをユーザが呼び出せるようにしたい場合、これらのコマンドを取り扱う特定のメソッドを使用しなくてはなりません。

例題

▼次の例題は、名前が“formula”で始まるすべてのメソッドと、“Total_general”メソッドを「フォーミュラ」エディタで許可します。

```
ARRAY STRING(15;methodsArray;2)
methodsArray{1}:="formula@"
methodsArray{2}:="Total_general"
SET ALLOWED METHODS(methodsArray)
```

参照

GET ALLOWED METHODS

GET ALLOWED METHODS

GET ALLOWED METHODS (メソッド配列)

引数	タイプ		説明
メソッド配列	文字列配列	←	メソッド名の配列

説明

GET ALLOWED METHODSコマンドは、フォーミュラの作成に使用できるメソッド名を引数＜メソッド配列＞に返します。これらのメソッドは、エディタ上のコマンドリストの最後に一覧表示されます。

デフォルトでは、「フォーミュラ」エディタ上でメソッドを使用することはできません。各メソッドは、**SET ALLOWED METHODS**コマンドを使用して明示的に許可しなくてはなりません。このコマンドが実行されていない場合、**GET ALLOWED METHODS**コマンドは空の配列を返します。

GET ALLOWED METHODSコマンドは、**SET ALLOWED METHODS**コマンドに渡されたものとまったく同じ値を返します。必要があれば、コマンドは配列の作成やサイズ調整を行います。また、ワイルドカード記号（@）を使用してメソッドグループが設定されている場合、“@”記号を含む文字列が返されます（メソッドグループの名前ではありません）。

このコマンドは、特定のコンテキストのフォーミュラ（例えば、クイックレポートなど）を実行する前に、現在指定されているメソッドの組み合わせの設定を保存しておくのに便利です。

例題

▼ この例題では、レポート作成のため一連の特定メソッドを許可します。

```
GET ALLOWED METHODS(methodsArray)
      ` クイックレポート用のメソッドを定義する
ARRAY TEXT(reports_methods;1)
reports_methods{1}:="Reports_@"
SET ALLOWED METHODS(reports_methods)
QR REPORT([従業員];"MyReport")
      ` カレントパラメータを再設定する
SET ALLOWED METHODS(methodsArray)
```

参照

SET ALLOWED METHODS

EDIT FORMULA

EDIT FORMULA (テーブル; フォーミュラ)

引数	タイプ		説明
テーブル	テーブル	→	デフォルトとしてフォーミュラエディタに表示するテーブル
フォーミュラ	文字列変数	→	フォーミュラエディタに表示するフォーミュラを格納する変数または、エディタだけを表示する場合は“”
		←	ユーザが確定したフォーミュラ

説明

EDIT FORMULAコマンドを使用すると、「フォーミュラ」エディタを開いて次の項目をデフォルトとして表示することができます。

■ 左側のリストには、引数<テーブル>に渡したテーブルのフィールドが表示されます。

■ フォーミュラエリアには、変数<フォーミュラ>に格納されたフォーミュラが表示されます。<フォーミュラ>に空の文字列を渡した場合、フォーミュラは指定せずに「フォーミュラ」エディタが表示されます。

エディタが表示された後は、<フォーミュラ>をそのままの状態で実行可能ですが（構文が正しい場合)、その修正やフォーミュラの作成、新しいフォーミュラのロード、保存を行うこともできます。いずれの場合でも、ユーザがダイアログボックスを確定すると、システム変数OKには1が代入され、変数<フォーミュラ>には実際に実行されたフォーミュラが格納されます。ユーザがフォーミュラをキャンセルすると、システム変数OKには0が代入され、変数<フォーミュラ>は変更されません。

<フォーミュラ>でメソッドが呼び出され、そのメソッドが**SET ALLOWED METHODS**コマンドを用いて「フォーミュラ」エディタ上で事前に“許可”されていない場合は、シンタックスエラーが生成され、フォーミュラを確定することができません。

例題

▼事前に入力されたフォーミュラは指定せず、[従業員]テーブルを使用してフォーミュラエディタを表示します。

```
  $myFormula:=""
  EDIT FORMULA([従業員];$myFormula)
If(OK=1)
APPLY TO SELECTION([従業員];EXECUTE($myFormula))
End if
```

参照

SET ALLOWED METHODS

Load list

Load list(リスト名) → 数値

引数	タイプ		説明
リスト名	文字列	→	「デザイン」モードの「リスト」エディタで作成されたリストの名前
戻り値	数値	←	新しく作成されたリストの参照番号

説明

Load list関数は、内容をリストからコピーし、引数<リスト名>で渡した名前を付けた階層リストを新たに作成します。さらに、新しく作成されたリストにリストの参照番号を返します。

<リスト名>で指定されたリストが存在しない場合、リストは作成されず、**Load list**関数からは0が返されます。

新規リストは「デザイン」モードで定義されたリストのコピーであることに注意してください。したがって、新規リストに変更を加えても、「デザイン」モードで定義されたリストには影響しません。逆に、「デザイン」モードで定義されたリストに変更を加えても、作成したリストに影響しません。

新たに作成したリストを変更し、その変更を恒久的に保存したい場合は、**SAVE LIST**コマンドを呼び出します。

新たに作成したリストを使い終わったら、**CLEAR LIST**コマンドを呼び出してそれを廃棄するのを忘れないでください。廃棄しないと、作業セッションが終了するまで、またはそれを作成したプロセスが終了するかアボートされるまで、そのリストはメモリに留まります。

Tips：「フォームエディタオブジェクトプロパティ」ウインドウの「選択リスト」プロパティを使用して、リストをフォームオブジェクト（階層リスト、タブコントロール、階層ポップアップメニュー）に関連付ける場合、オブジェクトメソッドからLoad list関数またはCLEAR LISTコマンドを呼び出す必要はありません。4th Dimensionはリストを自動的にロードして消去します。

例題

国際市場に対応するデータベースを作成し、そのデータベースの使用中に異なる言語に切り替える必要があるとします。フォームで「hlList」という名前の、標準オプションのリストを示す階層リストを提供します。「デザイン」モードでは、英語版の"Std Options US"、フランス語版の"Std Options FR"、日本語版の"Std Options JP"等、さまざまなリストを準備しました。これに加えて、<>gsCurrentLanguageという名前のインタープロセス変数を持ち、これに、英語版には"US"、フランス語版には"FR"、日本語版には"JP"というように2文字の言語コードを格納します。現在選択されている言語を使用してリストをロードするために、以下のように記述します。

```
`「hlList」階層リストのプロジェクトメソッド
Case of
	¥(Form event = On Load)
		C_LONGINT(hlList)
		hlList:=Load list("Std Options"+<>gsCurrentLanguage)
	¥(Form event = On Unload)
		CLEAR LIST(hlList ; *)
End case
```

参照

CLEAR LIST、SAVE LIST

SAVE LIST

SAVE LIST(リスト ; リスト名)

引数	タイプ		説明
リスト	数値	→	数値
リスト名	文字列	→	「デザイン」モードの「リスト」エディタに表示されるリストの名前

説明

SAVE LISTコマンドは、「デザイン」モードのリストエディタの中にあって、引数<リスト>によって渡された参照番号を持つリストを<リスト名>に渡される名前で保存します。

既にその名前のリストが存在する場合は、その内容が置き換えられます。

参照

Load list

New list

New list → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	数値

説明

New list関数は、新しい空の階層リストをメモリに作成し、ユニークな数値を返します。

警告：階層リストはメモリに保持されます。階層リストを使い終わったら、**CLEAR LIST**コマンドを使用してそれを廃棄し、メモリを解放することが重要です。

階層リストを作成するコマンドは、この他にもあります。

■ **Copy list**関数は、既存のリストからリストを複製します。

■ **Load list**関数は、「デザイン」モードのリストエディタで作成された（手動またはプログラムによって）選択リストをロードすることによりリストを作成します。

■ **BLOB to list**関数は、リストが前回保存されたBLOBのコンテンツから作成されます。

New list関数を使用して階層リストを作成すれば、以下のことが行えます。

■ **APPEND TO LIST**コマンドまたは**INSERT LIST ITEM**コマンドを使用して、項目をそのリストに追加できる

■ **DELETE LIST ITEM**コマンドを使用して、そのリストから項目を削除できる

▼ **APPEND TO LIST**コマンドの例を参照してください。

参照

APPEND TO LIST、BLOB to list、CLEAR LIST、Copy list、DELETE LIST ITEM、INSERT LIST ITEM、Load list.Copy list

Copy list

Copy list(リスト)→ 数値

引数	タイプ		説明
リスト	数値	→	コピー元リストの参照番号
戻り値	数値	←	複製されたリストの参照番号

説明

Copy list関数は、引数<リスト>で渡した参照番号を持つリストを複製し、新しいリストの参照番号を返します。

使い終わった新しいリストをクリアするには**CLEAR LIST**コマンドを使用します。

参照

CLEAR LIST、Load list、New list

CLEAR LIST

CLEAR LIST(リスト {; *})

引数	タイプ		説明
リスト	数値	→	数値
*	*	→	指定した場合、メモリからサブリストをクリア省略した場合、サブリストはクリアしない

説明

CLEAR LISTコマンドは、引数<リスト>に渡された数値を持つ階層リストを廃棄します。

通常は、リストの項目またはサブ項目にサブリストがあれば一緒にクリアするように、オプションの引数<*>を渡します。

プロパティリストでフォームオブジェクトに関連付けられているリストをクリアする必要はありません。そうしたリストのロードとクリアは4Dが自動的に処理します。一方で、BLOBからリストのロード、コピー、抽出を行ったり、プログラムでリストを作成した場合は、リストを使い終わるたびに**CLEAR LIST**コマンドを呼び出します。

フォームに表示されているリストから項目にあるサブリストだけをクリアするには、以下のように行います。

1. 親項目に対して**GET LIST ITEM**コマンドを呼び出して、サブリストのリスト参照を取得する。
2. 親項目で**SET LIST ITEM**コマンドを呼び出して、サブリストを消去する前にサブリストを親項目から切り離す。
3. **CLEAR LIST**コマンドを呼び出して、**GET LIST ITEM**コマンドで取得した参照番号を持つサブリストを消去する。
4. **REDRAW LIST**コマンドを呼び出して、フォームに表示されているリストについてその項目とサブリストを再描画する。

例題

1. もはや必要のないオブジェクトとデータ（例えば、ウインドウが閉じられ、フォームがアンロードされる場合）をすべて消去するクリーンアップルーチンの中で、クリアしようとするリストはすでにユーザによってクリアされたものであるかもしれません。**Is a list**関数を使用して、クリアする必要のあるリストだけをクリアすることができます。

```
`「クリーンアップ」ルーチン
If(Is a list(hlList))
        CLEAR LIST(hlList ; *)
End if
```

2. **Load list**関数の例を参照してください。

3. **BLOB to list**関数の例を参照してください。

参照

BLOB to list、Load list、New list

Count list items

Count list item(リスト{; *}) → 数値

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
*	*	→	省略した場合（デフォルト）：（展開されている）表示リスト項目を返す 指定した場合：すべてのリスト項目を返す
戻り値	倍長整数	←	（展開されている）表示リスト項目の数（“*”を省略した場合）、または リスト項目の合計数（“*”を指定した場合）

説明

Count list items関数は、＜リスト＞に渡した参照番号のリスト上にある現在“表示中”の項目数か、または項目の合計数のいずれかを返します。

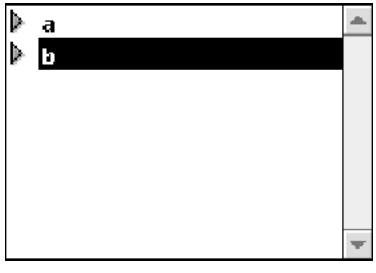
引数＜*＞を使用して、返される情報のタイプを決定します。この引数＜*＞を渡した場合、リストが展開されているか、縮められているかに関わらず、関数は項目の合計数を返します。

この引数を省略すると、関数はリストやそのサブリストの現在の展開／縮小の状態に応じて、表示されている項目数を返します。

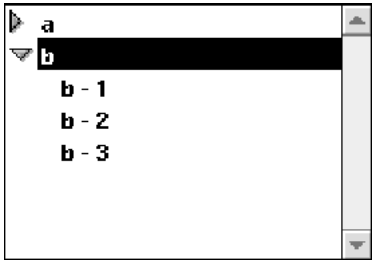
このコマンドは、フォームに表示されているリストに適用します。

例題

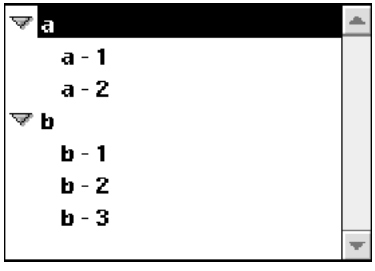
次は、「ユーザ」モードで表示された「hList」という名前のリストです。



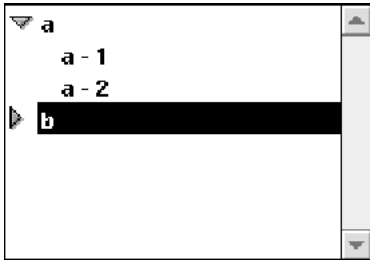

```
$vlNbItems:=Count list items(hList)
`この時点で、「$vlNbItems」変数は2を取得する
```



$vlNbItems:=**Count list items**(hList)
`この時点で、「$vlNbItems」変数は5を取得する



$vlNbItems:=**Count list items**(hList)
`この時点で、「$vlNbItems」変数は7を取得する
$vlNbTItems:=**Count list items**(hList;*) `$vlNbTItemsにも7が代入される



$vlNbItems:=**Count list items**(hList)
`この時点で、「$vlNbItems」変数は4を取得する
$vlNbTItems:=**Count list items**(hList;*) `$vlNbTItemsは7のままである

参照

List item position、Selected list item

Is a list

Is a list(リスト)→ ブール

引数	タイプ		説明
リスト	数値	→	検査される数値の値
戻り値	ブール	←	リストが階層リストの場合はTrue リストが階層リストでない場合はFalse

説明

Is a list関数は、引数<リスト>によって渡された値が階層化リストに対して有効な参照の場合TRUEを返します。その以外の場合は、FALSEを返します。

例題

1. **CLEAR LIST**コマンドの例を参照してください。

2. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照

DRAG AND DROP PROPERTIES

REDRAW LIST

REDRAW LIST(リスト)

引数	タイプ		説明
リスト	変数	→	数値

説明

REDRAW LISTコマンドは、引数<リスト>に渡された参照番号を持つリストの項目やサブリストの位置を再計算します。

フォームにおいて、リストの1つ以上の要素またはそのサブリストを変更した場合、最低1回はこのコマンドを呼び出す必要があります。

警告：式や変数ではなく、リストの実際の変数インスタンスを渡してください。例えば、フォームに「hList」という名前のリストがある場合、

例題

```
`変更が行われた後でリストを再計算する
REDRAW LIST(hList)        ` 良い例
      ` ...

$vlList:=hList
      ` ...
`変更が行われた後でリストを再計算する
REDRAW LIST($vlList)      ` 悪い例
` ...
```

SET LIST PROPERTIES

SET LIST PROPERTIES(リスト;表示形式{;アイコン{;線の高さ{;ダブルクリック{; 複数選択可{; 編集可}}}}})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
表示形式	数値	→	リストの表示形式 1：Macintoshの階層リスト 2：Windowsの階層リスト
アイコン	数値	→	MacOS-ベースのリソース ID 'cicn'、または0=デフォルトプラットフォームのノードアイコン
線の高さ	数値	→	線の高さの最小値（ピクセル単位）
ダブルクリック	倍長整数	→	ダブルクリックでサブリストの展開／縮小：0=Yes（デフォルト）、1=No
複数選択可	倍長整数	→	複数項目の選択： 0=No（デフォルト）、1=Yes
編集可	倍長整数	→	0=ユーザによるリスト編集可（デフォルト）、1=ユーザによるリスト編集不可

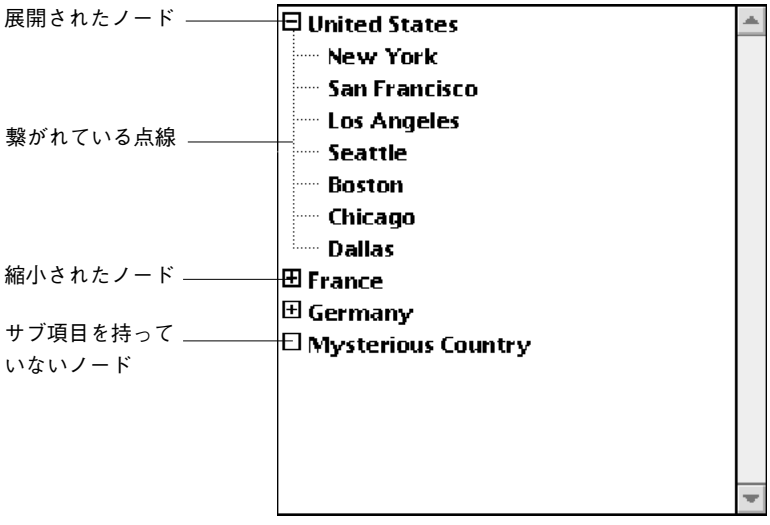
説明

SET LIST PROPERTIESコマンドは、引数＜リスト＞に渡された参照番号を持つ階層リストの表示様式を設定します。

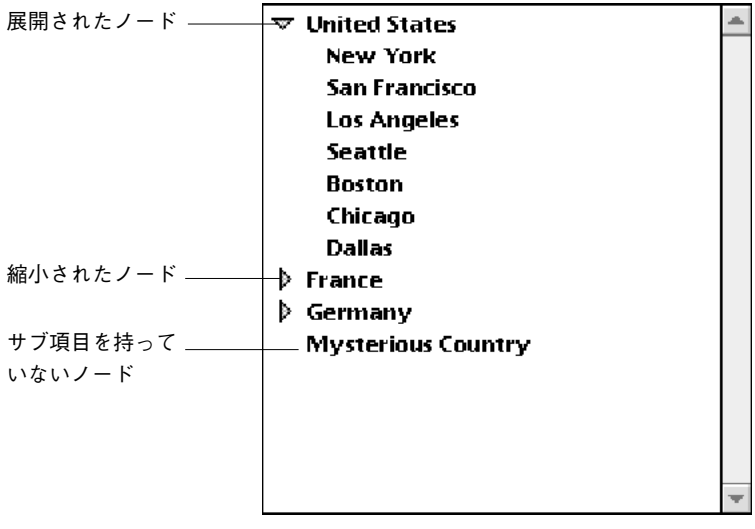
引数＜表示形式＞には、「Hierarchical Lists」テーマにおいて4th Dimensionより提供される次の定義済定数のいずれかを指定します。

定数	タイプ	値
ala Macintosh	倍長整数	1
ala Windows	倍長整数	2

Windows表示では、リストはノードとブランチの間を点線で繋ぎます。1つのアイコン(+)は縮小されたノード、もう1つのアイコンは展開されたノードを示します。子項目がないノードにはアイコンが表示されません。以下は、Windows表示のデフォルトの階層リストです。



Macintosh表示では、リストは点線で繋がれていません。1つのアイコンは縮小されたノード、別のアイコンは展開されたノードを示します。サブ項目を持たないノードにはアイコンがありません。以下はMacintosh表示のデフォルトの階層リストです。



注：SET LIST PROPERTIESコマンドを呼び出さずに階層リストオブジェクトを表示するか、または引数<表示形式>に0を渡すと、リストは「デザイン」モードのフォームエディタでそのオブジェクトに対して選択した「プラットフォームインタフェース」プロパティに応じて、デフォルトのWindowsまたはMacintoshの様式で表示されます。

引数＜アイコン＞は、それぞれのノードを表示するアイコンを示します。＜アイコン＞に渡された値は縮小されたノードのアイコンを設定し、＜アイコン＞+1は展開されたノードのアイコンを設定します。

例えば、15000を渡した場合は、縮小された各ノードに対してカラーアイコン'cicn' ID=15000が表示され、展開された各ノードに対してはカラーアイコン'cicn' ID=15001が表示されます。

したがって、これらの'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに存在することが重要です。カラーアイコンがないと、対応するノードはアイコンなしで表示されます（リストをアイコンなしで表示するには、この方法を利用します）。

警告：cicn'カラーアイコンリソースを作成する場合、15000以上のリソースIDを使用してください。15000未満のリソースIDは4th Dimensionのために予約されています。

デフォルトのMacintoshおよびWindowsノードのリソースIDは、4th Dimensionの以下の定義済定数によって表されます。

定数	タイプ	値
Macintosh node	倍長整数	860
Windows node	倍長整数	138

つまり、4th Dimensionは以下の'cicn'リソースを提供します。

ID番号	説明
860	Macintosh表示の縮小されたノード
861	Macintosh表示の展開されたノード
138	Windows表示の縮小されたノード
139	Windows表示の展開されたノード

引数＜アイコン＞を渡さない場合や0を渡した場合、ノードは選択した表示様式のデフォルトのアイコンで表示されます。

カラーアイコンリソースにはさまざまなサイズがあります。例えば、16*16または32*32のカラーアイコンを作成することができます。

引数＜線の高さ＞を渡さない場合、階層リストの線の高さは、オブジェクトに使用されるフォントおよびフォントサイズによって決定されます。縦または横に大き過ぎるカラーアイコンを使用すると、端が切れて表示されるか、接続のための点線（Windows表示の場合）やその上下のテキストによって消されるか、またはその両方になります。

カラーアイコンのサイズ、フォント、フォントサイズを適切に選択するか、そうでなければ＜線の高さ＞で階層リストの線の高さの最小値を渡します。渡した値が、使用するフォントやフォントサイズから求められる線の高さより大きい場合、階層リストの線の高さは渡された値になります。

注：SET LIST PROPERTIESコマンドは、階層リストの中でのノードの表示方法に影響します。リストの中の個々の項目のアイコンをカスタマイズするには、SET LIST ITEM PROPERTIESコマンドを使用します。

ダブルクリックのオプションパラメータにより、親リスト項目をダブルクリックすることがサブリストの開閉を引き起こさないように定義できます。デフォルトでは親リスト項目をダブルクリックすることにより、子リストが開いたり閉じたりするようになっています。しかし、いくつかのユーザーインタフェイスではこの動作が起こらないようにする必要があります。そのためには、ダブルクリックのパラメータを1に設定します。ダブルクリックの動作だけは発生しなくなりますが、リストのノードをクリックすることによりサブリストの開閉ができます。

パラメータの設定を省略するか、0を設定すると、デフォルトの動作が適用されます。

任意の引数＜複数選択可＞を使用し、＜リスト＞が複数項目の選択を受け入れるかどうかを指示することができます。

デフォルトでは、以前のバージョンの4th Dimensionと同様に、階層リストの項目を複数同時に選択することはできません。リストに対してこの機能を利用可能にしたい場合は、引数＜複数選択可＞に値“1”を渡します。その場合、次の方法で複数選択機能を使用することができます。

■ 手動の場合、連続した項目の選択にはShiftキー+クリック、連続していない項目の選択にはCtrlキー+クリック（Windows）またはCommandキー+クリック（Mac OS）を使用します。

■ プログラムを使用する場合、**SELECT LIST ITEMS BY POSITION**および**SELECT LIST ITEMS BY REFERENCE**コマンドを使用します。

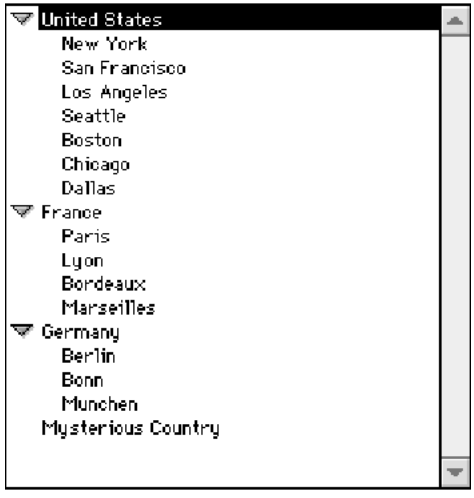
引数＜複数選択＞に0を渡すか、または省略した場合、デフォルトの動作が適用されます。

任意の引数＜編集可＞を使用すると、データ入力中に＜リスト＞がフィールドや変数に関連付けた選択リストとして表示された際に、ユーザによる＜リスト＞の編集を可能にするかどうかを指示することができます。＜リスト＞が編集可である場合、選択リストウインドウに「変更」ボタンが追加され、ユーザは特定のエディタを用いて、値の追加や削除、並び替えを行うことができます。

引数＜編集可＞に値0を渡すか、または省略すると、そのリストは編集可能になります。1を渡すと、リストを編集することはできません。

例題

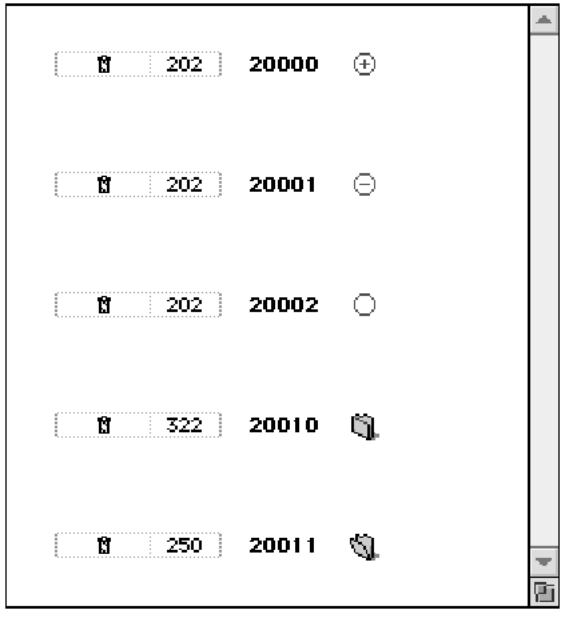
以下の階層リストが、「デザイン」モードのリストエディタで定義されています。



フォームの中で、「hlCities」階層リストオブジェクトがそのリストをこのオブジェクトメソッドで再利用します。

```
Case of
	¥(Form event=On Load)
		hlCities:=Load list("Cities")
		SET LIST PROPERTIES(hlCities ; vlAppearance ; vlIcon)
	¥(Form event=On Unload)
		CLEAR LIST(hlCities ; *)
End case
```

加えて、データベースのストラクチャファイルが以下の'cicn'カラーアイコンリソースを含むように編集されています。



1. 以下の行により、

SET LIST PROPERTIES(hlCities ; ala Macintosh ; Macintosh node)

階層リストは、以下のように表示されます。



2. 以下の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; Windows node)

階層リストは、以下のように表示されます。



3. 以下の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; 20000)

階層リストは、以下のように表示されます。



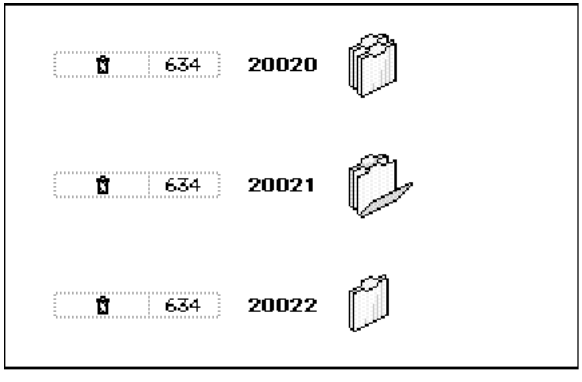
4. 以下の行により、

SET LIST PROPERTIES(hlCities ; ala Macintosh ; 20010)

階層リストは、以下のように表示されます。



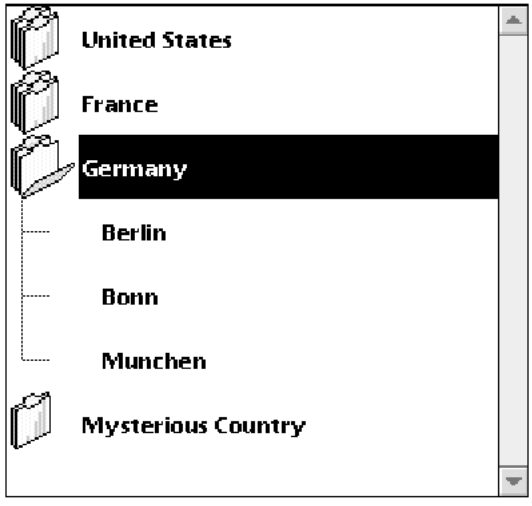
次に、以下に示す'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに追加されます。



5. 以下の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; 20020 ; 32)

階層リストは、以下のように表示されます。



参照

GET LIST ITEM PROPERTIES、GET LIST PROPERTIES、SET LIST ITEM PROPERTIES

GET LIST PROPERTIES

GET LIST PROPERTIES(リスト;表示形式{;アイコン{;線の高さ{;ダブルクリック{; 複数選択可{; 編集可}}}}})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
表示形式	数値	←	リストの表示形式 1：Macintoshの階層リスト 2：Windowsの階層リスト
アイコン	数値	←	MacOS-ベースのリソース ID 'cicn'
線の高さ	数値	←	線の高さの最小値（ピクセル単位）
ダブルクリック	倍長整数	←	ダブルクリックでサブリストの展開／縮小：0=Yes、1=No
複数選択可	倍長整数	←	複数項目の選択：0=No、1=Yes
編集可	倍長整数	←	0=ユーザによるリスト編集可（デフォルト）、1=ユーザによるリスト編集不可

説明

GET LIST PROPERTIESコマンドは、引数＜リスト＞によって渡された参照番号を持つリストについての情報を返します。

引数＜表示形式＞は、リストの表示形式を返します。

引数＜アイコン＞は、リストに表示されるノードアイコンのリソースIDを返します。

オプション引数＜線の高さ＞は、線の高さの最小値を返します。

オプション引数＜ダブルクリック＞が1に設定されている場合、親リスト項目をダブルクリックしても子リストが開いたり閉じたりしません。0に設定されているときは開閉の動作をします（デフォルト値）。

引数＜複数選択可＞に0が代入された場合、その＜リスト＞においてリスト項目を複数選択することはできません（手動でもプログラムからでも）。1が代入された場合は、複数項目を選択することができます。

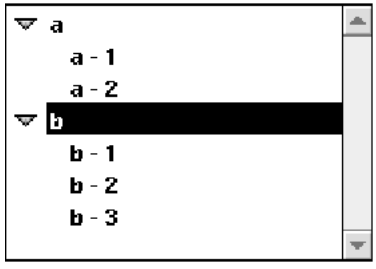
引数＜編集可＞に0が代入された場合、「ユーザ」モードでリストとして表示されると、そのリストは編集可能になります。1が代入された場合、リストを編集することはできません。

これらのプロパティは、**SET LIST PROPERTIES**コマンドおよび、リストが「デザイン」モードのリストエディタで作成された場合、または**SAVE LIST**コマンドを使用して保存された場合は、リストエディタで設定することができます。

リストの表示様式、ノードアイコン、線の高さの最小値、およびダブルクリックの管理についての詳細は、**SET LIST PROPERTIES**コマンドを参照してください。

例題

次に示す「hList」という名前のリストが「ユーザ」モードにあるとします。




```
`「bMacOrWin」ボタンのオブジェクトメソッド
GET LIST PROPERTIES(hList;$vlAppearance;$vlIcon;$vlLH;
                                        $vlClick;$vlSelect;$vlModif)
If($vlAppearance=Ala Macintosh)
        $vlAppearance:=Ala Windows
        $vlIcon:=Windows node
        $vlModif:=1
Else
        $vlAppearance:=Ala Macintosh
        $vlIcon:=Macintosh node
        $vlModif:=1
End if
SET LIST PROPERTIES(hList;$vlAppearance $vlIcon;
                                        $vlLH;$vlClick;$vlSelect;$vlModif)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                                        `そうでなければ、リストが更新されません。
```

このメソッドにより、リストは次のように表示されます。



参照

SET LIST PROPERTIES

SORT LIST

SORT LIST(リスト {; > または <})

引数	タイプ		説明
リスト	数値	→	数値
>または<		→	ソート順 >=昇順ソート <=降順ソート

説明

SORT LISTコマンドは、引数＜リスト＞によって渡された参照番号を持つリストをソートします。

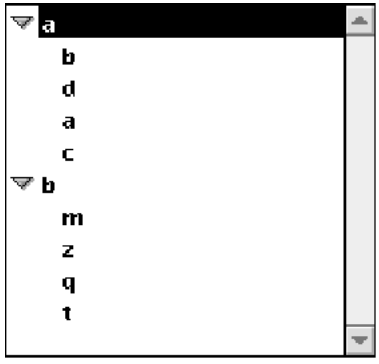
昇順にソートするにはオプション引数「>」を渡し、降順にソートするには「<」を渡します。ソート順パラメータを省略した場合、**SORT LIST**コマンドはデフォルトとして昇順にソートします。

SORT LISTコマンドは、あらゆるレベルのリストをソートします。まずリストの項目をソートし、次にサブリストがあればそれらをソートし、サブリストの中の項目をソートするというようにリストのすべてのレベルに降りていきます。通常、**SORT LIST**コマンドをフォームに表示されているリストに適用するのはこのためです。サブリストのソートは、上位レベルを呼び出したときに順序が変更されるので、ほとんど意味がありません。

SORT LISTコマンドは、選択されたリスト項目またはリストやサブリストの現在の展開または縮小状態は変更しません。ただし、選択した項目が並び替えによって移動されることがあるため、**Selected list items**関数は並び替えの前と後で異なる位置を返す可能性があります。

例題

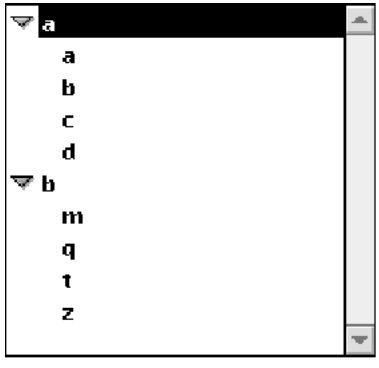
リスト名を「hList」とし、「ユーザ」モードで表示します（Macintosh表示）。



以下のコードを実行すると、

```
` リストとそのサブリストを昇順にソートする
      SORT LIST(hList ; >)
      REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                                    `そうでなければリストは更新されません。
```

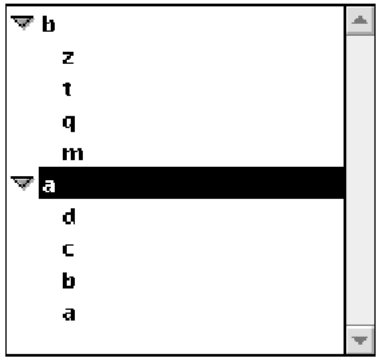
リストは、以下のようになります。



以下のコードを実行すると、

```
`リストとそのサブリストを降順にソートする
      SORT LIST(hList ; <)
      REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                          `そうでなければリストは更新されません。
```

リストは、以下のようになります。



参照

Selected list item

APPEND TO LIST

APPEND TO LIST(リスト; 項目テキスト; 項目参照番号{; サブリスト; 展開})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目テキスト	文字列	→	新規リスト項目のテキスト（最大255文字）
項目参照番号	倍長整数	→	新規リスト項目のユニークな参照番号
サブリスト	倍長整数	→	新規リスト項目に関連付ける任意のサブリスト
展開	ブール	→	任意のサブリストが展開されるか縮小されるかを示す

説明

APPEND TO LISTコマンドは、引数＜リスト＞によって渡された参照番号を持つ階層リストに新規項目を追加します。

項目のテキストは、引数＜項目テキスト＞によって渡します。255文字までの文字列またはテキスト式を渡すことができます。これより長い値を渡した場合は切り捨てられます。

項目のユニークな参照番号は、引数＜項目参照番号＞によって渡します。この項目の参照番号はユニークな番号としましたが、実際にはどのような値でも渡すことができます。詳細については、下記の「項目参照番号」に関する説明を参照してください。

項目にサブ項目を設定する場合は、引数＜サブリスト＞によってサブ階層リストへの有効なリスト参照を渡します。サブリストを展開または縮小するには、オプション引数＜展開＞によってTRUEまたはFALSEを渡します。

＜サブリスト＞によって渡されたリスト参照は既存のリストを参照しなければなりません。既存のリストは、空のリスト、1階層のリスト、サブリストを持つリストのいずれでも構いません。新規項目にサブリストを添付しない場合は、この引数を省略するか0を渡します。＜サブリスト＞を渡して＜展開＞を指定しないと、デフォルトとしてサブリストは展開されません。両引数ともにオプションですが、引数＜サブリスト＞と＜展開＞は組み合わせて渡す必要があります。

Tips

- リストに新規項目を挿入するには、**INSERT LIST ITEM**コマンドを使用します。既存の項目のテキストの変更、そのサブリストや展開または縮小状態の変更を実行するには、**SET LIST ITEM**コマンドを使用します。

- 新たに追加された項目の表示様式を変更するには、**SET LIST ITEM PROPERTIES**コマンドを使用します。

警告：フォームに現在表示されているリスト、またはフォームに現在表示されているリストを持つ項目に添付されているリストに項目を追加する場合は、REDRAW LISTコマンドを必ず呼び出してください。これにより、4Dは変更内容を反映してリストを再計算し、表示します。規則は簡単です。リストのどのレベルを操作しようと、REDRAW LISTコマンドをメインリスト、すなわちフォームでオブジェクトに参照されるリストに適用します。

項目参照番号：これは何に使用するのでしょうか?

階層リストの各項目には、倍長整数の参照番号があります。この値は4th Dimensionの中でしか意味がありません。以下では、この参照番号をどのように使用するかについて説明します。

1. 各項目を一意に識別する必要はありません（初心者レベル）。

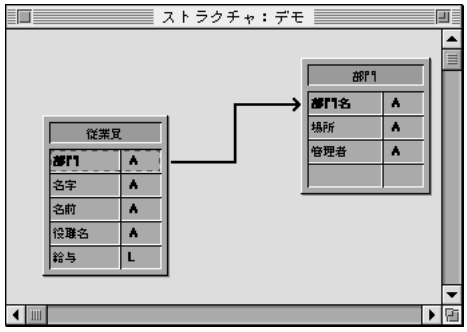
■ 第1の例：プログラムでタブコントロール（例えば、回転式卓上カードファイル）を作成します。タブコントロールは選択されたタブの番号を返すので、おそらくそれ以上の情報は必要ありません。この場合、項目参照番号のことを考慮する必要もないので、引数itemRefに適当な値（ただし0以外）を渡します。回転式卓上カードファイルのタブコントロールについては、「デザイン」モードでA、B、. . . 、Zリストをあらかじめ定義することができます。しかし、レコードのない文字を除外するために（例えば、Qで始まるキーフィールドが1つもない等）、プログラムによって作成したい場合もあるでしょう。

■ 第2の例：データベースで作業しながら、徐々にキーワードのリストを作成します。各セッションの終わりに、**SAVE LIST**コマンドまたは**LIST TO BLOB**コマンドを使用してリストを保存し、各セッションの始めに**Load list**関数または**BLOB to list**関数を使用してそれを再度ロードします。このリストを「パレット」ウインドウに表示し、項目をクリックすると、クリックされたキーワードが最前面にあるプロセスの現在入力可能なエリアに挿入されます。ドラッグアンドドロップも使用できます。いずれにしても、**Selected list items**関数および**DRAG AND DROP PROPERTIES**コマンドにより、取得すべき項目の位置がわかるため、選択した項目（クリックまたはドラッグした項目）を処理することが重要です。この位置を利用し、**GET LIST ITEM**コマンドで項目のテキストを取得することができます。これで終わりです。したがって、各項目を一意に識別する必要はなく、引数itemRefに適当な値（ただし0以外）を渡せばよいのです。

2. リスト項目を部分的に識別する必要があります（中級レベル）。

項目参照番号を使用して、項目を操作する場合に必要な情報を格納します。後述の例がこのような場合にあたります。この例では、項目参照番号を使用してレコード番号を格納します。ただし、[部門]のレコードに対応する項目を[従業員]のレコードに対応する項目と区別する必要があります。これは、このコマンドの2番目の例で行われているので参照してください。

3. リスト項目を一意に識別する必要があります（上級レベル）。



上級プログラミングで階層リストを処理する場合、リストの各レベルで各々の項目を一意に識別することが絶対に必要です。これを実行する最も簡単な方法は、プライベートカウンタを持つことです。ここで、**New list**関数を使用して、「hlList」というリストを作成するとしましょう。この時点では、カウンタvlhCounterを0に初期化します。**APPEND TO LIST**コマンドまたは**INSERT LIST ITEM**コマンドを呼び出すたびに、このカウンタを増分し（vlhCounter:=vlhCounter+1）、このカウンタを項目参照番号として渡します。秘訣は項目を削除してもカウンタを減らさず、したがってカウンタは増加しかしないことです。これにより、事実上、項目参照番号の一意性が保証されます。項目参照番号は倍長整数値なので、再初期化されたリストには、項目を何度も追加、挿入することができます。しかし、数千にものぼる項目を使って作業する場合、リストではなくテーブルを使用することをお勧めします。

注：ビットワイズ演算子を使用する場合、倍長整数値にあたる情報を格納するために項目参照番号も使用できます。すなわち、整数値なら2つ、4バイト値または32の論理値です。

なぜ一意の参照番号が必要なのか?

ほとんどの場合、階層リストをユーザインターフェイスの目的で使用し、選択した項目（クリックまたはドラッグした項目）だけを扱う場合、項目参照番号を使用する必要はまったくありません。**Selected list items**関数と**GET LIST ITEM**コマンドを使用すると、現在の選択項目の処理に必要となるあらゆる情報を得ることができます。加えて、**INSERT LIST ITEM**コマンドや**DELETE LIST ITEM**コマンドのようなコマンドでは、選択されている項目に“相対的に”リストを操作することができます。

基本的に、参照番号を使った処理が必要になるのは、必ずしもリストで現在選択されている項目に限らず、プログラムによってリストの任意の項目に直接アクセスする場合です。

例題

次は、データベースストラクチャの一部です。

[部門]テーブルと[従業員]テーブルには、以下のレコードが入っています。



部門

部門名	場所	管理者
営業	本社3階	太田泰男
技術開発	技術センター	山田幸二
経理	本社7階	井上洋介
活動休眠中	未定	未定



従業員

部門	名字	名前
営業	会田	一郎
経理	佐藤	祐子
技術開発	田中	豊
技術開発	名取	和隆
経理	水野	あやの
経理	高岡	由美
営業	佐藤	幸太郎

ここで、hlListという名前の階層リストを表示します。この階層リストは、部門を表示するとともに、各部門についてその部門で働いている従業員が入ったサブリストを表示します。

```
`「hlList」階層リストのオブジェクトメソッド
Case of
    ¥(Form event=On Load)
        C_LONGINT(hlList ; $hSubList ; $vlDepartment ; $vlEmployee)
        `新たな空の階層リストを作成する
        hlList:=New list
        `[部門]テーブルのすべてのレコードを選択する
        ALL RECORDS([部門])
        ` 各部門に対してループする
        For($vlDepartment ; 1 ; Records in selection([部門]))
            `この部門のすべての従業員を選択する
            RELATE MANY([部門]部門名)
            ` 従業員は何人？
            $vlNbEmployees:=Records in selection([従業員])
            `この部門には最低1人の従業員がいるかどうかを確認する
```

```
        If($vlNbEmployees>0)
            `部門項目のサブリストを作成する
            $hSubList:=New list
            ` 各従業員に対してループする
            For($vlEmployee ; 1 ; Records in selection([従業員]))
                `従業員項目をサブリストに追加する
                `[従業員]レコードのレコード番号が項目参照番号として渡される
                                                    ことに注意してください。
                APPEND TO LIST($hSubList ; [従業員]名字+""+[従業員]名前 ;
                                                Record number([従業員]))
                `以下の[従業員]レコードに進む
                NEXT RECORD([従業員])
            End for
        Else
            `部門項目に従業員もサブリストも存在しない場合
            $hSubList:=0
        End if
        `部門項目をメインリストに追加する
        `[部門]レコードのレコード番号が項目参照番号として渡されることに注意して
        `ください。項目参照番号の31番目のビットが1になり、部門項目と従業員項目
        `が区別できるようになります。このビットを項目についての補足情報として
        `使用できる理由については、後述の注を参照してください。
        APPEND TO LIST(hlList ; [部門]部門名 ;0x80000000 |
                            Record number([部門]) ; $hSublist ; $hSubList # 0)
        `部門項目を太字に設定して、リストの階層を強調する
        SET LIST ITEM PROPERTIES(hlList ; 0 ; False ; Bold ; 0)
        `以下の部門に進む
        NEXT RECORD([部門])
    End for
    `リスト全体を昇順にソートする
    SORT LIST(hlList ; >)
    ` リストをWindowsスタイルで表示し、
                                        線の高さの最小値を14ポイントに設定する
    SET LIST PROPERTIES(hlList ; ala Windows ; Windows node ; 14)
¥(Form event=On Unload)
    `リストはもう必要ありません。忘れずに消去します！
    CLEAR LIST(hlList ; *)
¥(Form event=On Double Clicked)
    `ダブルクリックが生じたら、選択した項目の位置を取得する
    $vlItemPos:=Selected list items(hlList)
    `念のために位置を確認する
```

```
If($vlItemPos # 0)
    `リスト項目の情報を取得する
    GET LIST ITEM(hlList ; $vlItemPos ; $vlItemRef ; $vsItemText
                                        ; $vlItemSubList ; $vbItemSubExpanded)
    `項目が部門項目かを確認する
    If($vlItemRef ?? 31)
        `部門項目の場合は、部門項目をダブルクリックする
        ALERT("部門の項目がダブルクリックされました "+Char(34)
            +$vsItemText+Char(34)+".")
    Else
        `そうでない場合は、従業員項目のダブルクリックである
        ` 親の項目参照番号を使用して、[部門]レコードを検索する
        GOTO RECORD([部門] ; List item parent(hlList ; $vlItemRef) ?- 31)
        ` その従業員の勤務場所と直属上司を取得する
        ALERT("従業員項目がダブルクリックされました"+Char(34)+
                $vsItemText+Char(34)+" 部門内の従業員名："+Char(34)
                    +[部門]部門名+Char(34)+" 管理者名： "+Char(34)+
                        [部門]管理者+Char(34)+".")
    End if
End if
End case
```

注：4th Dimensionは、テーブルあたり最高1600万（正確には16,777,215）のレコードを格納することができます。この値は2^24-1です。レコード番号は、24ビットで表されます。この例では、未使用の上位バイトの31番目のビットを使用して、従業員項目および部門項目を区別します。

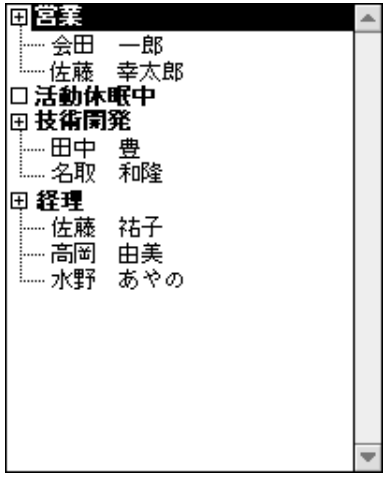
この例では、部門項目と従業員項目を区別する必要がある理由は1つだけです。

1. レコード番号は項目参照番号に格納しますので、おそらく部門項目は従業員項目と同じ項目参照番号を持つことになるでしょう。

2. **List parent item**関数を使用して、選択した項目の親を取得します。対応するレコード番号が10番の従業員項目をクリックし、同じ番号を持つ部門項目も存在する場合に、項目参照番号を渡して項目を検索するためにリストを参照した時、**List parent item**関数によって部門項目がまず最初に見つけられます。この関数は従業員項目の親ではなく、部門項目の親を返します。

したがって、一意の項目参照番号が作成されましたが、これはユニークな項目参照番号が必要なのではなく、[部門]と[従業員]レコードを区別する必要があったからです。

「ユーザ」モードまたは「カスタム」モードでは、リストは以下のように見えます。



注意：上記の例は、比較的少ないレコードを扱う場合には、ユーザインターフェイス目的として役立ちます。リストはメモリに置かれるので、何千という項目を持つ階層リストでユーザインターフェイスを作成すべきではありません。

INSERT LIST ITEM

INSERT LIST ITEM(リスト ; 前項目参照番号 | * ; 項目テキスト ; 項目参照番号
{ ; サブリスト { ; 展開}})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
前項目参照番号	*	倍長整数	*	→	項目の参照番号、または“*”の場合、現在選択されているリスト項目
項目テキスト	文字列	→	新規リスト項目のテキスト（最大255文字）
項目参照番号	倍長整数	→	新規リスト項目のユニークな参照番号
サブリスト	倍長整数	→	新規リスト項目に関連付ける任意のサブリスト
展開	ブール	→	そのサブリストが展開されるか縮小されるかを示す

説明

INSERT LIST ITEMコマンドは、引数<リスト>によって渡される参照番号を持つリストに新規項目を挿入します。

2番目の引数として<*>を渡すと、項目は現在リストで選択されている項目の前に挿入されます。この場合、新たに挿入された項目もまた選択された項目になります。

そうでない場合、特定の項目の前に項目を挿入するには、その項目の項目参照番号を渡します。この場合、新たに挿入された項目が自動的に選択されることはありません。この項目参照番号を持つ項目が存在しない場合、このコマンドは何も行いません。

新規項目のテキストと項目参照番号は、引数<項目テキスト>と<項目参照番号>によって渡します。

例題

以下のコードは、リスト「hList」の現在選択されている項目の直前に項目を挿入します(サブリストは添付されていません)。

```
vlUniqueRef:=vlUniqueRef+1
INSERT LIST ITEM(hList ; * ; "新規項目" ; vlUniqueRef)
REDRAW LIST(hList)
```

参照

APPEND TO LIST

SET LIST ITEM PROPERTIES

SET LIST ITEM PROPERTIES(リスト; 項目参照 | *; 入力可; スタイル; アイコン {; カラー})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照¦*	倍長整数¦*	→	項目参照番号、または最後にリストへ追加された項目の場合は0、または カレントリスト項目の場合は“*”
入力可	ブール	→	True= 入力可、Faule=入力不可
スタイル	数値	→	項目のフォントスタイル
アイコン	数値	→	Mac OSベースのリソースID'cicn'、または 65536+ Mac OSベースのリソースID 'PICT'、または131072+ピクチャ参照番号
カラー	倍長整数	→	RGBカラーの値、または -1=オリジナルのカラーへリセット

説明

SET LIST ITEM PROPERTIESコマンドは、引数＜リスト＞に渡された参照番号のリスト内で、＜項目参照番号＞によって指定された項目を変更します。

＜項目参照番号＞には参照番号を渡すことができます。渡された項目参照番号を持つ項目が存在しない場合、コマンドは何も動作しません。オプションとして＜項目参照番号＞で0を渡すことによって、**APPEND TO LIST**コマンドを使用してリストに追加した最後の項目を変更することができます。

最後に、引数＜項目参照番号＞として“*”を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目となります。選択された項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

注：項目のテキストまたはそのサブリストを変更するには、SET LIST ITEMコマンドを使用します。

項目を入力可能にする場合は、引数＜入力可＞でTrueを渡し、そうでない場合はFalseを渡します。

重要：入力可能にするためには、その項目が入力可能なリストに属する必要があります。SET ENTERABLEコマンドを使用すると、リスト全体を入力可能または入力不可能にすることもできます。SET LIST ITEM PROPERTIESコマンドを使用すると、個々のリスト項目を入力可能または入力不可能にすることができます。「入力可」プロパティをリストレベルで変更しても、項目の「入力可」プロパティは変更されません。しかし、項目が入力可能にできるのは、そのリストが入力可能な場合のみです。

項目のフォントスタイルは、引数＜スタイル＞で指定します。以下の定義済定数の1つ、または複数を組み合わせて渡します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注：Windowsでは、標準（Plain）、または太字（Bold）、斜体（Italic）、アンダーライン（Underline）を組み合わせたスタイルしか使用できません。また、コンデンスおよびシャドウスタイルはサポートされなくなり、互換性上の理由からのみ残されています。

項目にアイコンを関連付ける場合、以下の数値のいずれか1つを渡します。

- N、NはMacintoshベースの'cicn'リソースのリソースIDです。
- Use PICT resource+N、NはMacintoshベースの'PICT'リソースのリソースIDです。
- Use PicRef+N、Nは「デザイン」モードのピクチャライブラリのピクチャの参照番号です。

項目にグラフィックを使用しない場合は、ゼロを渡します。

注：「Use PICT resource」および「Use PicRef」は、「Hierarchical Lists」テーマにある定義済み定数です。

引数＜カラー＞（任意）を使用して、項目テキストの色を変更することができます。この色はRGBフォーマット、つまり0x00RRGGBB形式の4バイトの倍長整数で指定しなくてはなりません。このフォーマットに関する詳細は、**SET RGB COLORS**コマンドの説明を参照してください。項目をオリジナルの色にリセットするには、引数＜カラー＞に1を渡します。

例題

1. **APPEND TO LIST**コマンドの例を参照してください。

2. 次の例題は、＜リスト＞のカレント項目のテキストを太字スタイル、明るい赤に設定します。

```
SET LIST ITEM PROPERTIES(list;*;True;Bold;0;0x00FF0000)
REDRAW LIST(list)
```

参照

GET LIST ITEM PROPERTIES、SET LIST ITEM

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES(リスト; 項目参照 | *; 入力可 {;スタイル
{;アイコン{; カラー}}})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照 ¦ *	倍長整数 ¦ *	→	項目参照番号、または 最後に追加されたリスト項目の場合は 0、またはカレントリスト項目の場合は “*”
入力可	ブール	←	True= 入力可、Faule=入力不可
スタイル	数値	←	項目のフォントスタイル
アイコン	数値	←	Mac OSベースのリソースID 'cicn'、または 65536+ Mac OSベースのリソースID 'PICT'、または131072+ピクチャ参照番号
カラー	倍長整数	←	RGBカラーの値

説明

GET LIST ITEM PROPERTIESコマンドは、引数＜リスト＞に渡されたリスト参照番号のリスト内で、引数＜項目参照＞によって指定された項目のプロパティを返します。

＜項目参照＞には参照番号を渡すか、または最後にリストへ追加された項目を指定するには値“0”を、リストのカレント項目を指定するには“*”を渡すことができます。複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目となります。

“*”を渡したが選択項目が存在しない場合や、渡した項目参照番号を持つ項目が存在しない場合、コマンドは引数を変更しません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

■ その項目が入力可能な場合、引数＜入力可＞がTrueを返します。

■ 引数＜スタイル＞が、項目のフォントスタイルを返します。

■ 引数＜アイコン＞がその項目に割り当てられているアイコンまたはピクチャを返し、何も割り当てられていない場合は0を返します。

■ 引数＜カラー＞は、指定した項目のテキストの色を返します。

これらのプロパティについての詳細は、**SET LIST ITEM PROPERTIES**コマンドの説明を参照してください。

参照

GET LIST ITEM、SET LIST ITEM、SET LIST ITEM PROPERTIES

List item position

List item position(リスト ; 項目参照番号) → 数値

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照番号	倍長整数	→	項目参照番号
戻り値	数値	←	展開されたリストにおける項目の位置

説明

List item position関数は、引数<リスト>によって渡された数値を持つリスト内の、<項目参照番号>によって渡された参照番号を持つ項目の位置を返します。

この位置は、メインリストの最初の項目に相対的に表され、リストとそのサブリストの現在の展開または縮小状態が使用されます。

したがって、結果は1から**Count list items**関数によって返される値の間の数値です。

項目が縮小されているリストにあるために表示されていない場合、**List item position**関数が適切なリストを展開してその項目を表示します。

項目が存在しない場合、**List item position**関数は0を返します。

参照

Count list items、SELECT LIST ITEM BY REFERENCE

List item parent

List item parent(リスト; 項目参照 | *) →倍長整数

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照¦*	倍長整数¦*	→	項目の参照番号、または リストのカレント項目の場合は“*”
戻り値	倍長整数	←	親項目の項目参照番号、または 親項目がない場合は0

説明

List item parent関数は、親項目の項目参照番号を返します。

引数＜リスト＞にはリスト参照番号を渡し、そのリスト項目の項目参照番号または“*”を引数＜項目参照番号＞に渡します。“*”を渡した場合、この関数はリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合は、最後に選択された項目がカレントリスト項目になります。

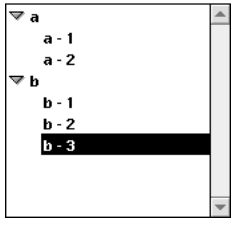
戻り値として、対応する項目がリスト上に存在し、かつその項目がサブリスト上にある場合（つまり親項目を持つ）、親項目の項目参照番号が返されます。

渡した項目参照番号を持つ項目が存在しない場合や、“*”を渡したが選択項目が存在しない場合、または項目が親を持たない場合は、**List item parent**関数から0（ゼロ）が返されます。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

例題

リスト名を「hList」とし、「ユーザ」モードで表示します。



加えて、項目の項目参照番号を以下のように設定します。

項目	項目参照番号
a	100
a - 1	101
a - 2	102
b	200
b - 1	201
b - 2	202
b - 3	203

▼以下のコードを使って、項目“b-3”を選択すると、「$vlParentItemRef」変数は200を取得します。この数値は、項目“b”の項目参照番号です。

```
$vlItemPos:=Selected list items(hList)
GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText)
$vlParentItemRef:=List item parent(hList ; $vlItemRef)
                                        ` $vlParentItemRefは200を取得する
```

▼項目“a-1”が選択された場合、「$vlParentItemRef」変数は100を取得します。この数値は、項目“b”の項目参照番号です。

▼項目“a”または“a”が選択された場合、「$vlParentItemRef」変数は親項目を持っていないので0を取得します。

参照

GET LIST ITEM、List item position、SELECT ITEM BY REFERENCE、SET LIST ITEM

DELETE LIST ITEM

DELETE LIST ITEM(リスト ; 項目参照番号 | * {; *})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照番号¦*	倍長整数¦*	→	項目の参照番号、または“*”を指定した場合、現在選択されているリスト項目
*		→	指定した場合、メモリ上からサブリスト（存在する場合）を消去 省略した場合、サブリスト（存在する場合）は消去されない

説明

DELETE LIST ITEMコマンドは、引数<リスト>によって渡された数値を持つリストから項目を削除します。

2番目の引数として<*>を渡す場合、リストで現在選択されている項目が削除されます。

そうでない場合は、削除する項目の項目参照番号を指定します。指定した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

どの項目を削除しようと、オプション引数の<*>を指定して、その項目にサブリストが添付されている場合、4Dにそれを自動的に削除させることをお勧めします。<*>を指定しない場合、項目に添付されているサブリストの数値をあらかじめ取得しておくとよいでしょう。これは、結局**CLEAR LIST**コマンドを使用してそれを削除する必要があるからです。

例題

▼ 以下の例は、「hList」というリストで現在選択されている項目を削除します。項目にサブリストが添付されている場合、サブリスト（およびサブリストのサブリスト）が消去されます。

```
DELETE LIST ITEM(hList ; * ; *)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出してください。
                   `そうでなければ、リストが更新されません。
```

参照

CLEAR LIST、GET LIST ITEM

GET LIST ITEM

GET LIST ITEM(リスト; 項目位置¦*; 項目参照; 項目テキスト{; サブリスト{; 展開}})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目位置¦*	数値¦*	→	展開されたリストにおける項目の位置、または リストのカレント項目の場合は“*”
項目参照	倍長整数	←	項目参照番号
項目テキスト	文字列	←	リスト項目のテキスト
サブリスト	倍長整数	←	サブリストの参照番号（存在する場合）
展開	ブール	←	サブリストが付属する場合： TRUE=サブリストは現在展開されている FALSE=サブリストは現在縮められている

説明

GET LIST ITEMコマンドは、引数＜リスト＞に渡した参照番号のリストに関し、＜項目位置＞で指定した項目の情報を返します。

位置は、リストとそのサブリストの展開または縮小の現在の状態を使用して相対的に表されます。位置としては、1から**Count list items**関数によって返された値までの数値を渡します。この範囲外の値を渡すと、**GET LIST ITEM**コマンドは空の値（0、""、等）を返します。

このコマンド呼び出すと、以下の情報を取得できます。

■引数＜項目参照番号＞には項目の項目参照番号が返されます。

■引数＜項目テキスト＞には項目のテキストが返されます。

オプション引数の＜サブリスト＞と＜展開＞を渡すと、以下のようになります。

■＜サブリスト＞には、項目に添付されているサブリストの数値が返されます。項目にサブリストがない場合、ゼロが返されます。

■項目にサブリストがある場合、＜展開＞にはサブリストが現在展開されていればTrueが、縮小されていればFalseが返されます。

例題

1. 「hList」はユニークな参照番号の付いた項目を持つリストです。以下のコードは、現在選択されている項目に添付されているサブリストが存在すれば、その展開／縮小の状態をプログラムによって切り換えます。

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
        GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;
                                                $hSublist;$vbExpanded)
        If(Is a list($hSublist))
                SET LIST ITEM(hList ; $vlItemRef ; $vsItemText ; $vlItemRef ;
                                                $hSublist ; Not($vbExpanded))
                REDRAW LIST(hList)
        End if
End if
```

2. **APPEND TO LIST**コマンドの例題を参照してください。

参照

GET LIST ITEM PROPERTIES、List item parent、List item position、Selected list item、SET LIST ITEM、SET LIST ITEM PROPERTIES

SET LIST ITEM

SET LIST ITEM(リスト; 項目参照 | *; 新規項目テキスト; 新規項目参照{; サブリスト; 展開})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照¦*	倍長整数¦*	→	項目の参照番号、または最後にリストへ追加される項目の場合は0、またはそのリストのカレント項目の場合は“*”
新規項目テキスト	文字列	→	新規項目のテキスト
新規項目参照	倍長整数	→	新規項目の参照番号
サブリスト	倍長整数	→	項目に付属する新規サブリスト、またはサブリストがない場合は0（存在すれば、現在のものを取り外す）、または変更しない場合は-1
展開	ブール	→	任意のサブリストを展開するか縮小するかを指示する

説明

SET LIST ITEMコマンドは、引数＜リスト＞に渡した参照番号のリストにおいて、＜項目参照＞で指定した項目を変更します。

＜項目参照＞には参照番号を渡すことができます。渡した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションとして＜項目参照＞に0を渡し、**APPEND TO LIST**コマンドを用いてリストへ最後に追加される項目を指定することができます。

最後に、＜項目参照＞には“*”を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目になります。選択された項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

項目の新しいテキストは、＜新規項目テキスト＞によって渡します。項目参照番号を変更する場合は、＜新規項目参照番号＞で新たな値を渡します。そうでない場合は、＜項目参照番号＞と同じ値を渡します。

項目にリストを添付する場合、数値を＜サブリスト＞で渡します。この場合、新たなサブリストが展開されている場合は＜展開＞にTrueを、そうでない場合はFalseを渡します。

項目に添付されているサブリストを切り離す場合は、＜サブリスト＞に0を渡します。この場合、**GET LIST ITEM**コマンドを使用して、そのリストの参照番号をあらかじめ取得しておくとよいでしょう。そうすれば、リストが必要なくなったときに**CLEAR LIST**コマンドを使用して削除することができます。

項目の＜サブリスト＞プロパティを変更しない場合は、＜サブリスト＞に-1を渡します。

注：引数＜サブリスト＞と＜展開＞はともにオプションですが、指定する場合は組み合わせて指定してください。

例題

1. 「hList」はユニークな参照番号が付いた項目を持つリストです。以下のボタン用オブジェクトメソッドは、現在選択されているリスト項目にサブ項目を追加します。

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
	GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText ; $hSublist ;
	                                                              $vbExpanded)
	$vbNewSubList:=Not(Is a list($hSublist))
	If($vbNewSubList)
		$hSublist:=New list
	End if
	vlUniqueRef:=vlUniqueRef+1
	APPEND TO LIST($hSubList ; "新規項目" ; vlUniqueRef)
	If($vbNewSubList)
		SET LIST ITEM(hList ; $vlItemRef ; $vsItemText ; $vlItemRef ;
		                                                  $hSublist ; True)
	End if
	SELECT LIST ITEMS BY REFERENCE(hList ; vlUniqueRef)
	REDRAW LIST(hList)
End if
```

2. **GET LIST ITEM**コマンドの例を参照してください。

3. **APPEND TO LIST**コマンドの例を参照してください。

参照

GET LIST ITEM、GET LIST ITEM PROPERTIES、SET LIST ITEM PROPERTIES

Selected list items

Selected list items(リスト{; 項目配列}{; *}) → 倍長整数

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目配列	倍長整数配列	←	“*”を省略した場合：配列には展開されたリストにおける選択項目の位置が納められる “*”を渡した場合：配列には選択項目の参照番号が納められる
*	*	→	省略した場合：項目の位置 指定した場合：項目の参照番号
戻り値	倍長整数	←	“*”を省略した場合：展開されたリスト内で現在選択されているリスト項目の位置 “*”を指定した場合：選択項目の参照番号

説明

Selected list item関数は、引数＜リスト＞に渡された参照番号のリストにおいて、選択された項目の位置または参照番号を返します。複数項目が選択されている場合、さらにコマンドは選択された各項目の位置あるいは参照番号を配列＜項目配列＞に返します。ユーザが選択した項目を調べるには、フォームに表示されるリストにこの関数を適用します。

引数＜*＞を使用して、項目の現在位置（この場合、引数＜*＞は省略します）または固定の項目参照（この場合、引数＜*＞を使用してください）のいずれを処理したいのかを指示します。

引数＜項目配列＞には、倍長整数タイプの配列を渡すことができます。必要に応じ、関数は配列の作成やサイズ調整を行います。関数の実行後、＜項目配列＞には次の要素が代入されます。

■ 引数＜*＞を省略した場合、展開されたリスト内の各選択項目の位置。

■ 引数＜*＞を渡した場合、各選択項目の固定参照。

選択された項目が存在しない場合、空の配列が返されます。

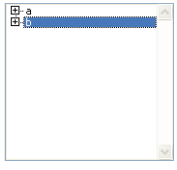
注：選択項目が複数存在する場合、コマンドは＜リスト＞のカレント項目の位置あるいは参照番号を返します。カレント項目は、ユーザが最後にクリックした項目（手動による選択の場合）またはSELECT LIST ITEMS BY POSITIONコマンドかSELECT LIST ITEMS BY REFERENCEコマンドで指定された項目（プログラムによる選択の場合）のいずれかになります。

リストにサブリストがある場合、この関数はメインリスト（フォームで実際に定義されたリスト）のサブリストではなく、メインリストに適用します。各位置は、リストとそのサブリストの現在の展開または縮小状態を用いて、メインリストの最上位の項目と相対的に表わされます。

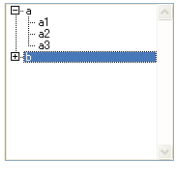
いずれの場合でも、選択された項目が存在しない場合、関数は0を返します。

例題

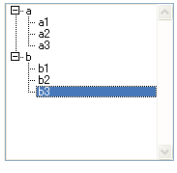
次は、「ユーザ」モードで表示された「hList」という名前のリストです。



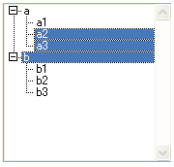
$vlItemPos:=**Selected list items**(hList) ` この時点では$vlItemPosは2を取得する



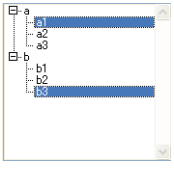
$vlItemPos:=**Selected list items**(hList) ` この時点では$vlItemPosは5を取得する
$vlItemRef:=**Selected list items**(hList;*) ` $vlItemRefには（たとえば）200が代入される



$vlItemPos:=**Selected list items**(hList) `この時点で、$vlItemPosには8が代入される
$vlItemRef:=**Selected list items**(hList;*) `$vlItemRefには（たとえば）203が代入される



$vlItemPos:=**Selected list items**(hList;$arrPos) `この時点で、$vlItemPosには3が代入される
` $arrPos{1} には3、$arrPos{2} には4、$arrPos{3} には5が代入される



$vlItemRef:=**Selected list items**(hList;$arrRefs;*) `この時点で、$vlItemRefには（たとえば）203が代入される)
`$arrRefs{1}には101、$arrRefs{2}には203が代入される（例えば）

参照

SELECT LIST BY REFERENCE、SELECT LIST ITEM

SELECT LIST ITEM BY POSITION

SELECT LIST ITEMS BY POSITION(リスト; 項目位置{; 位置配列})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目位置	数値	→	展開されたリスト内での項目の位置
位置配列	数値配列	→	展開されたリスト内の位置を納めた配列

説明

SELECT LIST ITEMS BY POSITIONコマンドは、＜リスト＞に渡された参照番号のリストにおいて、＜項目位置＞ならびにオプションとして＜位置配列＞に渡された位置にある項目を選択します。

項目の位置は常に、リストとそのサブリストの展開または縮小の現在の状態を使用して表わされます。位置の値として、1から**Count list items**関数によって返される値までの数値を渡します。この範囲外の値を渡すと、項目は選択されません。

引数＜位置配列＞を渡さない場合、引数＜項目位置＞は選択される項目の位置を示します。

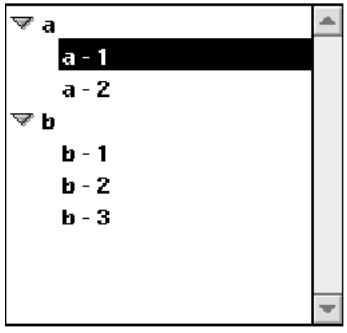
任意の引数である＜位置配列＞を使用すると、＜リスト＞内の複数の項目を同時に選択することができます。＜位置配列＞には必ず配列を渡し、その各要素は選択する項目の位置を示します。

この引数を渡すと、結果として選択された各項目の中で、引数＜項目位置＞に指定した項目がそのリストの新しいカレント項目になります。この項目は、配列によって定義した一連の項目に含まれていない可能性もあります。カレント項目とは、具体的に言うと、**EDIT ITEM**コマンドを使用した場合に編集モードに移行される項目のことです。

注：階層リスト内で複数のリスト項目を同時に選択するには（手動、あるいはプログラムから）、そのリストに対して「複数選択可」プロパティを有効に設定しておかなくてはなりません。このプロパティの設定は、SET LIST PROPERTIESコマンドを使用して行います。

例題

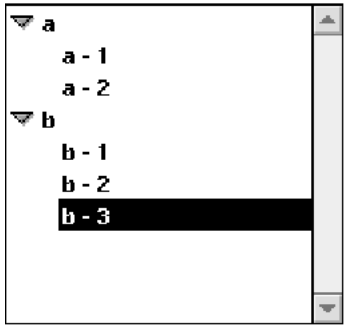
「ユーザ」モードにおいて、次のようなhListという名前の階層リストについて考えてみましょう。



以下のコードを実行すると、

```
SELECT LIST ITEMS BY POSITION(hList;Count list items(hList))
REDRAW LIS T(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                    `そうでなければ、リストが更新されません。
```

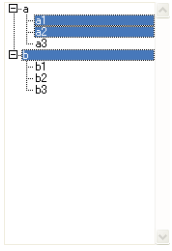
表示されているリスト項目の最後の項目が選択されます。



2. 次のコード行を実行すると、

```
SET LIST PROPERTIES(hList;0;0;18;0;1)
      `複数選択を許可するには、必ず最後の引数として1を渡さなければならない
ARRAY LONGINT($arr;3)
$arr{1}:=2
$arr{2}:=3
$arr{3}:=5
SELECT LIST ITEMS BY POSITION(hList;3;$arr)
      `3番目の項目がカレント項目として指定される
REDRAW LIST(hList)
```

階層リストの2番目、3番目、5番目の項目が選択されます。



参照

SELECT LIST ITEM BY REFERENCE、Selected list item

SELECT LIST ITEM BY REFERENCE

SELECT LIST ITEMS BY REFERENCE(リスト; 項目参照{; 参照配列})

引数	タイプ		説明
リスト	倍長整数	→	リスト参照番号
項目参照	倍長整数	→	項目参照番号
参照配列	倍長整数配列	→	項目参照番号の配列

説明

SELECT LIST ITEMS BY REFERENCEコマンドは、＜リスト＞に渡された参照番号のリストにおいて、＜項目参照＞ならびにオプションとして＜参照配列＞に渡された項目参照番号を持つ項目を選択します。

渡した項目参照番号の項目が存在しない場合、コマンドは何も行いません。

ある項目が現在表示されていない場合（つまり、縮小されているサブリスト内にその項目がある場合)、コマンドは必要なサブリストを展開し、その項目を表示します。

引数＜参照配列＞を渡さない場合、引数＜項目参照＞は選択される項目の参照番号を示します。項目番号がそのリストの項目に対応しない場合、コマンドは何も行いません。

任意の引数である＜参照配列＞を使用すると、＜リスト＞において複数の項目を同時に選択することができます。＜参照配列＞には必ず配列を渡し、その各要素は選択する項目の固定参照番号を示します。

その場合、結果として選択された各項目の中で、引数＜項目参照＞に指定した項目がそのリストの新しいカレント項目になります。この項目は、配列によって定義した一連の項目に含まれていない可能性もあります。カレント項目とは、具体的に言うと、**EDIT ITEM**コマンドを使用した場合に編集モードに移行される項目のことです。

注：階層リスト内で複数のリスト項目を同時に選択するには（手動、あるいはプログラムから)、そのリストに対して「複数選択可」プロパティを有効に設定しておかなくてはなりません。このプロパティの設定は、SET LIST PROPERTIESコマンドを使用して行います。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

例題

「hList」は一意の参照番号が付いた項目を持つリストです。以下のボタン用オブジェクトメソッドは、現在選択されている項目に親項目が存在する場合にはその親項目を選択します。

```
$vlItemPos:=Selected list items(hList) ` 選択された項目の位置を取得する
`選択された項目の項目参照番号を取得
GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText)
`親項目が存在する場合は、親項目の項目参照番号を取得
$vlParentItemRef:=List item parent(hList ; $vlItemRef)
If($vlParentItemRef>0)
    `親項目を選択する
    SELECT LIST ITEM BY REFERENCE(hList ; List item parent(hList ;
                                                    $vlItemRef))
    REDRAW LIST(hList) `REDRAW LISTを忘れずに呼び出します。
                `そうでなければ、リストが更新されません。
End if
```

参照

SELECT LIST ITEM、Selected list item

IMPORT TEXT

IMPORT TEXT({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データが読み込まれるテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データを読み込むドキュメントテーブル

説明

IMPOER TEXTコマンドは、WindowsまたはMacintoshの標準的なテキスト形式ドキュメントテーブルである＜ドキュメント＞から＜テーブル＞に新規レコードを作成してデータを読み込みます。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータの読み込み処理は、入力フォーム上のフィールドや変数をその入力順序に従って読み込みます。このため、フォーム上にあるテキストオブジェクト（フィールドや変数）の前後の順序について注意してください。つまり、最初にデータが読み込まれるオブジェクトはフォームの後ろに位置する必要がある、等です。フォームのフィールドや変数の数が、読み込もうとするフィールドの数と一致しない場合、余分なフィールドは無視されます。読み込み用のフォームにはボタンを配置することはできません。また、サブフォームオブジェクトは無視されます。

注：データが正しいオブジェクトに読み込まれるかどうかを確認する1つの方法は、最初のフィールドが読み込まれるはずのオブジェクトを選択し、それを前面に移動します。次に、フィールドや変数を順次前面に移動し、読み込もうとする各フィールドに対して1つのフィールドまたは変数が対応していることを確認します。

読み込まれるレコードごとに、**On Validate**イベントがフォームメソッドに送られます。データ読み込み用フォームで変数を使用している場合には、このイベントを利用して変数からフィールドへデータをコピーしてください。

引数＜ドキュメント＞には、ボリューム名やディレクトリ（Macintosh版では、フォルダ）名等のパスを指定できます。空の文字列を渡すと、標準の「ファイルを開く」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中にその進捗状況を表わすインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。既に読み込まれたレコードは、ユーザが「中止」ボタンをクリックしても消去されません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの読み込みは、処理の前にASCII入力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII入力テーブルを使用して行います。ASCII入力テーブルは、ASCIIテーブルが異なる他のプラットフォームから読み込むデータを変換するために使用します。

IMPORT TEXTコマンドの、デフォルトのフィールド区切り文字とレコード区切り文字は、それぞれタブ（ASCIIコードの9）とキャリッジリターン（ASCIIコードの13）ですが、2つのシステム変数FldDelimitとRecDelimitに値を代入することによって、これらの区切り文字を変更することができます。また、「ユーザ」モードの「データ読み込み」ダイアログボックスでこれらの指定を修正することもできます。テキストフィールドにはキャリッジリターンが含まれている可能性があるため、テキストフィールドの読み込みを実行する場合、区切り文字にキャリッジリターンを使用する際には注意が必要です。

例題

▼以下の例は、データをドキュメントファイルから読み込みます。まず、メソッドの最初で入力フォームを設定してデータが正しいフォームに読み込まれるようにし、次にシステム変数を使用して区切り文字を修正します。この後で読み込みを実行します。

```
INPUT FORM([従業員] ; "読み込み")        ` データ読み込みのフォームを設定
FldDelimit:=27          ` フィールド区切り文字にエスケープ
RecDelimit:=10          ` レコード区切り文字にラインフィード
IMPORT TEXT([従業員] ; "従業員ドキュメント")
                                ` 従業員ドキュメントからデータを読み込む
```

参照

EXPORT TEXT、IMPORT DIF、IMPORT SYLK、USE ASCII MAP

システム変数とセット

システム変数OKには、読み込み処理が正常に終了すると1が、それ以外の場合には0がセットされます。

EXPORT TEXT

EXPORT TEXT({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データを書き出すテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データが書き出されるドキュメント テーブル

説明

EXPORT TEXTコマンドは、カレントプロセスにおける＜テーブル＞のカレントセレクションのレコードをディスクに書き出します。このデータは＜ドキュメント＞に書き込まれます。＜ドキュメント＞は、WindowsまたはMacintoshの標準的なテキスト形式のドキュメントテーブルです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータの書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に従って書き出します。このため、書き出したい入力可能オブジェクトとフィールドのみを含んだ出力フォームを使用する必要があります。書き出し用のフォームにボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、**On Load**イベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

引数＜ドキュメント＞には、新規または既存のドキュメントファイルを指定することができます。＜ドキュメント＞が既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、＜ドキュメント＞にはボリューム名やフォルダ名等のパスを指定することもできます。空の文字列を渡すと、標準の「ファイルの保存」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中にその進捗状況を表すインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。書き出しが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの書き出しは、処理の前にASCII出力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII出力テーブルを使用して行います。ASCII出力テーブルは、ASCIIテーブルが異なる他のプラットフォーム上での使用に、データを変換するために使用します。

EXPORT TEXTコマンドのデフォルトのフィールド区切り文字とレコード区切り文字は、それぞれタブ（ASCIIコードの9）とキャリッジリターン（ASCIIコードの13）ですが、2つのシステム変数FldDelimitとRecDelimitに値を代入することによって、これらの区切り文字を変更することができます。また「ユーザ」モードの「データ書き出し」ダイアログボックスでこれらの指定を修正することもできます。テキストフィールドには、キャリッジリターンを含めることができますので、テキストフィールドを書き出す場合には、区切り文字としてキャリッジリターンを使用していないかどうか注意してください。

例題

▼以下の例は、データをドキュメントファイルに書き出します。まず、メソッドの最初で出力フォームを設定し、次にシステム変数を使用して区切り文字を修正します。この後、データ書き出しを実行します。

```
OUTPUT FORM([従業員] ; "書き出し")     ` データ書き出しのフォームを設定
FldDelimit:=27          ` フィールド区切り文字にエスケープ
RecDelimit:=10          ` レコード区切り文字にラインフィード
EXPORT TEXT([従業員] ; "従業員ドキュメント")
                  ` 従業員ドキュメントにデータを書き出す
```

参照

EXPORT DIF、EXPORT SYLK、IMPORT TEXT、USE ASCII MAP

システム変数とセット

システム変数OKには、書き出し処理が正常に終了すると1が、それ以外の場合には0がセットされます。

IMPORT SYLK

IMPORT SYLK({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データが読み込まれるテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データを読み込むSYLK ドキュメントテーブル

説明

IMPORT SYLKコマンドは、WindowsまたはMacintoshの標準的なSYLK形式ドキュメントテーブルである＜ドキュメント＞から＜テーブル＞に新規レコードを作成してデータを読み込みます。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータの読み込み処理は、入力フォーム上のフィールドや変数をその入力順序に従って読み込みます。このため、フォーム上にあるテキストオブジェクト（フィールドや変数）の前後の順序について注意してください。つまり、最初にデータが読み込まれるオブジェクトはフォームの後ろに位置する必要がある、等です。フォームのフィールドや変数の数が、読み込もうとするフィールドの数と一致しない場合、余分なフィールドは無視されます。読み込み用のフォームにはボタンを配置することはできません。また、サブフォームオブジェクトは無視されます。

注：データが正しいオブジェクトに読み込まれるかどうかを確認する1つの方法は、最初のフィールドが読み込まれるはずのオブジェクトを選択し、それを前面に移動します。次に、フィールドや変数を順次前面に移動し、読み込もうとする各フィールドに対して1つのフィールドまたは変数が対応していることを確認します。

読み込まれるレコードごとに、**On Validate**イベントがフォームメソッドに送られます。データ読み込み用フォームで変数を使用している場合には、このイベントを利用して変数からフィールドへデータをコピーしてください。

引数＜ドキュメント＞には、ボリューム名やディレクトリ（Macintosh版では、フォルダ）名等のパスを指定できます。空の文字列を渡すと、標準の「ファイルを開く」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中にその進捗状況を表わすインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。既に読み込まれたレコードは、ユーザが「中止」ボタンをクリックしても消去されません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの読み込みは、処理の前にASCII入力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII入力テーブルを使用して行います。ASCII入力テーブルは、ASCIIテーブルが異なるプラットフォームから読み込むデータを変換するために使用します。

例題

▼ 以下の例は、データをSYLKドキュメントファイルから読み込みます。まず、メソッドの最初で入力フォームを設定してデータが正しいフォームに読み込まれるようにし、その後で読み込みを実行します。

```
INPUT FORM([従業員] ; "読み込み")        ` データ読み込みのフォームを設定
IMPORT SYLK([従業員] ; "従業員ドキュメント")
                                        ` 従業員ドキュメントからデータを読み込む
```

参照

EXPORT SYLK、IMPORT DIF、IMPORT TEXT、USE ASCII MAP

システム変数とセット

システム変数OKには、読み込み処理が正常に終了すると1が、それ以外の場合には0がセットされます。

EXPORT SYLK

EXPORT SYLK({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データを書き出すテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データが書き出されるドキュメント テーブル

説明

EXPORT SYLKコマンドは、カレントプロセスにおける＜テーブル＞のカレントセレクションのレコードをディスクに書き出します。このデータは＜ドキュメント＞に書き込まれます。＜ドキュメント＞は、WindowsまたはMacintoshの標準的なDIF形式のドキュメントテーブルです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータの書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に従って書き出します。このため、書き出したい入力可能オブジェクトとフィールドのみを含んだ出力フォームを使用する必要があります。書き出し用のフォームにボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、**On Load**イベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

引数＜ドキュメント＞には、新規または既存のドキュメントファイルを指定することができます。＜ドキュメント＞が既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、＜ドキュメント＞にはボリューム名やフォルダ名等のパスを指定することもできます。空の文字列を渡すと、標準の「ファイルの保存」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中にその進捗状況を表すインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。書き出しが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には 0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの書き出しは、処理の前にASCII出力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII出力テーブルを使用して行います。ASCII出力テーブルは、ASCIIテーブルが異なるプラットフォーム上での使用に、データを変換するために使用します。

例題

▼ 以下の例は、データをSYLKドキュメントファイルに書き出します。まず、メソッドの最初で出力フォームを設定し、次にデータ書き出しを実行します。

```
OUTPUT FORM([従業員] ; "書き出し")     ` データ書き出しのフォームを設定
EXPORT SYLK([従業員] ; "従業員ドキュメント")
                                    ` 従業員ドキュメントにデータを書き出す
```

参照

EXPORT DIF、EXPORT TEXT、IMPORT SYLK、USE ASCII MAP

システム変数とセット

システム変数OKには、書き出し処理が正常に終了すると1が、それ以外の場合には0がセットされます。

IMPORT DIF

IMPORT DIF({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データが読み込まれるテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データを読み込むDIFドキュメント テーブル

説明

IMPORT DIFコマンドは、WindowsまたはMacintoshの標準的なDIF形式ドキュメントテーブルである＜ドキュメント＞から＜テーブル＞に新規レコードを作成してデータを読み込みます。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータの読み込み処理は、入力フォーム上のフィールドや変数をその入力順序に従って読み込みます。このため、フォーム上にあるテキストオブジェクト（フィールドや変数）の前後の順序について注意してください。つまり、最初にデータが読み込まれるオブジェクトはフォームの後ろに位置する必要がある、等です。フォームのフィールドや変数の数が、読み込もうとするフィールドの数と一致しない場合、余分なフィールドは無視されます。読み込み用のフォームにはボタンを配置することはできません。また、サブフォームオブジェクトは無視されます。

注：データが正しいオブジェクトに読み込まれるかどうかを確認する1つの方法は、最初のフィールドが読み込まれるはずのオブジェクトを選択し、それを前面に移動します。次に、フィールドや変数を順次前面に移動し、読み込もうとする各フィールドに対して1つのフィールドまたは変数が対応していることを確認します。

読み込まれるレコードごとに、**On Validate**イベントがフォームメソッドに送られます。データ読み込み用フォームで変数を使用している場合には、このイベントを利用して変数からフィールドへデータをコピーしてください。

引数＜ドキュメント＞には、ボリューム名やディレクトリ（Macintosh版では、フォルダ）名等のパスを指定できます。空の文字列を渡すと、標準の「ファイルを開く」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中にその進捗状況を表わすインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。既に読み込まれたレコードは、ユーザが「中止」ボタンをクリックしても消去されません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの読み込みは、処理の前にASCII入力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII入力テーブルを使用して行います。ASCII入力テーブルは、ASCIIテーブルが異なるプラットフォームから読み込むデータを変換するために使用します。

例題

▼ 以下の例は、データをDIFドキュメントファイルから読み込みます。まず、メソッドの最初で入力フォームを設定してデータが正しいフォームに読み込まれるようにし、その後で読み込みを実行します。

```
INPUT FORM([従業員] ; "読み込み")        ` データ読み込みのフォームを設定
IMPORT DIF([従業員] ; "従業員ドキュメント")
                                        ` 従業員ドキュメントからデータを読み込む
```

参照

EXPORT DIF、IMPORT SYLK、IMPORT TEXT、USE ASCII MAP

システム変数とセット

システム変数OKには、読み込み処理が正常に終了すると1が、それ以外の場合には0がセットされます。

EXPORT DIF

EXPORT DIF({テーブル ;} ドキュメント)

引数	タイプ		説明
テーブル	テーブル名	→	データを書き出すテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	→	データが書き出されるドキュメント テーブル

説明

EXPORT DIFコマンドは、カレントプロセスにおける＜テーブル＞のカレントセレクションのレコードをディスクに書き出します。このデータは＜ドキュメント＞に書き込まれます。＜ドキュメント＞は、WindowsまたはMacintoshの標準的なDIF形式のドキュメントテーブルです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータの書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に従って書き出します。このため、書き出したい入力可能オブジェクトとフィールドのみを含んだ出力フォームを使用する必要があります。書き出し用のフォームにボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、**On Load**イベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

引数＜ドキュメント＞には、新規または既存のドキュメントファイルを指定することができます。＜ドキュメント＞が既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、＜ドキュメント＞にはボリューム名やフォルダ名等のパスを指定することもできます。空の文字列を渡すと、標準の「ファイルの保存」ダイアログボックスが表示されます。このダイアログボックスでユーザが「キャンセル」をクリックすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中にその進捗状況を表すインジケータ（ナンバーやサーモメータ）を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。書き出しが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの書き出しは、処理の前にASCII出力テーブルを変更していない限り（**USE ASCII MAP**コマンドを使用）、処理が実行されるプラットフォームに応じたカレントASCII出力テーブルを使用して行います。ASCII出力テーブルは、ASCIIテーブルが異なるプラットフォーム上での使用に、データを変換するために使用します。

例題

▼ 以下の例は、データをDIFドキュメントファイルに書き出します。まず、メソッドの最初で出力フォームを設定し、次にデータ書き出しを実行します。

```
OUTPUT FORM([従業員] ; "書き出し")      ` データ書き出しのフォームを設定
EXPORT DIF([従業員] ; "従業員ドキュメント")
                                        ` 従業員ドキュメントにデータを書き出す
```

参照

EXPORT SYLK、EXPORT TEXT、IMPORT DIF、USE ASCII MAP

システム変数とセット

システム変数OKには、書き出し処理が正常に終了すると1が、それ以外の場合には0がセットされます。

IMPORT DATA

IMPORT DATA(ファイル名 {; プロジェクト {; *}})

引数	タイプ		説明
ファイル名	文字列	→	読み込みファイルへのパス
プロジェクト	BLOB	→	読み込みプロジェクトの内容
		←	読み込みプロジェクトの新しい内容（* 引数が渡された場合）
*	*	→	データ読み込みダイアログボックスを表示し、プロジェクトを更新します。

説明

このコマンドは、データを＜ファイル名＞ファイルから読み込めるようにします。4Dはテキスト、固定長のテキスト、XML、SYLK、DIF、DBF（dBase）および4th Dimensionフォーマットのデータを読み込むことができます。

空白をファイル名に渡すと、**IMPORT DATA**は標準ファイルセーブダイアログボックスを表示して、ユーザが読み込むファイルの名前、タイプおよび位置を定義することができるようにします。ダイアログボックスが受け入れられると、Documentシステム変数にファイルパスがセットされます。ユーザがキャンセルをクリックすると、コマンドの実行は停止されて、システム変数OKは0になります。

■ オプション引数＜プロジェクト＞を省略した場合、データ読み込みダイアログボックスが表示され、インポートパラメータを定義するかまたは既存の定義ファイルからインポートプロジェクトを読み込むことができます。

注：インポートプロジェクトには、読み込むテーブルやフィールド、区切り符号（デリミタ）のようなインポートに関するすべてのパラメータが含まれています。これらのパラメータはデータ読み込みダイアログボックス内で定義します。プロジェクトはディスクにセーブされ、読み込んで使用する事ができるようになります。

■ 有効なインポートプロジェクトを持つBLOBをプロジェクト引数に渡した場合、ユーザの操作無しに直接データ読み込みが実行されます。プロジェクトはデータ読み込みダイアログボックス内で既に前もって定義し保存しておかなければいけません。これを実行するには、2つの方法があります。

■ プロジェクトをディスクに保存後、**DOCUMENT TO BLOB**コマンドを使用してフィールドまたはBLOB変数にセットします。

■ 空白のプロジェクト引数およびオプション引数 * を指定した**IMPORT DATA**コマンドを実行し、プロジェクト引数のBLOBに保存します（下記参照）。この方法は、ディスク上からプロジェクトを読み込む必要はなく、データと共にプロジェクトを保存することができます。

注：空のプロジェクトの定義に関する例題は、EXPORT DATAコマンドを参照してください。

オプションの引数<*>が指定されていれば、プロジェクト内に定義されたパラメータと共にデータ読み込みダイアログボックスを表示します。これは、パラメータの1つまたはそれ以上を変更できる可能性を持ちながら、前もって定義されたプロジェクトを使用できるようにするものです。さらに、データ読み込みダイアログボックスを閉じた後に、プロジェクト引数は、「新しい」プロジェクトのパラメータを持つことができ、新しいプロジェクトをBLOBフィールドやディスク上等に保存することができます。

データ読み込みが正常に終了すると、システム変数OKは1になります。

参照

EXPORT DATA

システム変数とセット

標準の保存ファイルのダイアログボックスかインポートのダイアログボックスのキャンセルボタンをクリックすると、システム変数OKには0がセットされます。読み込み処理が正常に終了すると1がセットされます。

EXPORT DATA

EXPORT DATA(ファイル名 {; プロジェクト {; *}})

引数	タイプ		説明
ファイル名	文字列	→	書き出しファイルへのパス
プロジェクト	BLOB	→	書き出しプロジェクトの内容
		←	書き出しプロジェクトの新しい内容（*引数が渡された場合）
*	*	→	データ書き出しダイアログボックスを表示し、プロジェクトを更新します。

説明

このコマンドは、データを＜ファイル名＞ファイルから書き出せるようにします。固定長のテキスト、XML、SYLK、DIF、DBF（dBase）および4th Dimensionフォーマットのデータを書き出すことができます。

空白をファイル名に渡すと、**EXPORT DATA**は標準ファイルセーブダイアログボックスを表示して、ユーザが書き出すファイルの名前、タイプおよび位置を定義することができるようにします。ダイアログボックスが受け入れられると、Documentシステム変数にファイルパスがセットされます。ユーザがキャンセルをクリックすると、コマンドの実行は停止されて、システム変数OKは0になります。

■ オプション引数＜プロジェクト＞を省略した場合、データ書き出しダイアログボックスが表示され、エクスポートパラメータを定義するかまたは既存の定義ファイルからエクスポートプロジェクトをロードことができます。

注：エクスポートパラメータには、書き出すテーブルやフィールド、区切り符号（デリミタ）のようなエクスポートに関するすべてのパラメータが含まれています。これらのパラメータはデータ書き出しダイアログボックス内で定義します。プロジェクトはディスクに保存され、読み込みして使用する事ができるようになります。

■ 有効なエクスポートプロジェクトを持つBLOBをプロジェクト引数に渡した場合、ユーザの操作無しに直接データ書き出しが実行されます。プロジェクトはデータ書き出しダイアログボックス内で既に前もって定義し保存しておかなければいけません。これを実行するには、2つの方法があります。

■ プロジェクトをディスクに保存後、**DOCUMENT TO BLOB**コマンドを使用してフィールドまたはBLOB変数にセットします。

■ 空白のプロジェクト引数およびオプション引数＊を指定した**EXPORT DATA**コマンドを実行し、プロジェクト引数のBLOBに保存します（下記参照）。この方法は、ディスク上からプロジェクトを読み込む必要はなく、データと共にプロジェクトを保存することができます。

オプションの引数<*>が指定されていれば、プロジェクト内に定義されたパラメータと共にデータ書き出しダイアログボックスを表示します。これは、パラメータの1つまたはそれ以上を変更できる可能性を持ちながら、前もって定義されたプロジェクトを使用できるようにするものです。さらに、データ書き出しダイアログボックスを閉じた後に、プロジェクト引数は、「新しい」プロジェクトのパラメータを持つことができ、新しいプロジェクトをBLOBフィールドやディスク上等に保存することができます。

データ書き出しが正常に終了すると、システム変数OKは1になります。

例題

この例題は、空のプロジェクトを作成し、ユーザが「書き出し」ダイアログボックスで設定した各パラメータを保存します。

```
C_BLOB($exportParams)
SET BLOB SIZE($exportParams;0)  `BLOBを初期化する
EXPORT DATA("DocExport.txt";$exportParams;*)
                                `「書き出し」ダイアログボックスの表示
```

参照

IMPORT DATA

システム変数とセット

標準のオープンファイルのダイアログボックスかエクスポートのダイアログボックスのキャンセルボタンをクリックすると、システム変数OKには0がセットされます。書き出し処理が正常に終了すると1がセットされます。

ON EVENT CALL

ON EVENT CALL(イベントメソッド {; プロセス})

引数	タイプ		説明
イベントメソッド	文字列	→	インストールするイベントメソッド イベントの中断を中止する場合、空の文字列
プロセス	文字列	→	プロセス名

説明

ON EVENT CALLコマンドは、イベントを検出するメソッドである＜イベントメソッド＞をインストールします。このメソッドは、イベント管理メソッドまたはイベント検出メソッドと呼ばれます。

Tip：このコマンドの使用には、上級のプログラミング知識が必要です。通常、イベントを用いて作業を実行する際に、ON EVENT CALLコマンドを使用する必要はありません。フォームの使用中に、イベントは4th Dimensionによって管理され、適切なフォームやオブジェクトにイベントが送信されます。

Tip：バージョン6では、GET MOUSEやShift down等の新しいコマンドが導入され、イベントに関する情報を取得できるようになりました。これらのコマンドは、オブジェクトメソッドからコールし、オブジェクトに関連するイベントについての必要な情報を取得することができます。これらのコマンドにより、ON EVENT CALLコマンドをもとにしたアルゴリズムを作成する必要がなくなります。

このコマンドのスコープは、現在の作業セッションです。デフォルトでは、別々のローカルプロセス内でメソッドが実行されます。いちどに1つのイベント管理メソッドだけを使用できます。メソッドを用いたイベントの検出を中止するには、再度**ON EVENT CALL**コマンドをコールし、＜イベントメソッド＞に空の文字列を指定します。

イベント管理メソッドは別プロセスとして実行されるため、4th Dimensionのメソッドが1つも実行されなくても、常にアクティブになります。インストール後はイベントが発生するたびに4th Dimensionが＜イベントメソッド＞を呼び出します。管理できるイベントは、マウスボタンのクリックとキーボードからの入力です。

任意の引数＜プロセス＞は、**ON EVENT CALL**コマンドで作成されるプロセスの名前です。＜プロセス＞の先頭にドル記号（$）を付けると、＜プロセス＞はローカルプロセスになります。通常はこのローカルプロセスを使用します。引数＜プロセス＞を省略した場合、デフォルトとして4Dにより$Event Managerという名前のプロセスが作成されます。

警告：イベント管理メソッドで実行する処理については十分注意してください。イベントを発生させるコマンドはコールしないでください。このようなコマンドをコールした場合、イベント管理メソッドの実行から抜けるのが非常に困難になります。「Ctrl+Shift+Alt+Backspace」（Windows）または「command+shift+option+control+delete」（Macintosh）キーにより、イベント管理プロセスから通常のプロセスに切り替えることができます。したがって、それ以降、イベントメソッドは発生するすべてのイベントへ自動的に渡されなくなります。イベント管理が正常な動作ではなくなった場合に、この手法を利用して回復することも可能です。

イベント管理メソッドでは、MouseDown、KeyCode、Modifiers、MouseX、MouseY、MouseProcというシステム変数を読み取ることができます。これらの変数がプロセス変数であるという点に注意してください。したがって、変数のスコープはイベント管理プロセス内です。別のプロセスでこれらの値が必要な場合、インタープロセス変数へコピーしてください。

■ システム変数MouseDownには、イベントがマウスクリックである場合には1が、それ以外の場合には0が代入されます。

■ システム変数KeyCodeには、押されたキーのASCIIコードが代入されます。この変数はASCIIコードまたはファンクションキーコードを返します。付録Bに、ASCIIコード表とファンクションキーコード表が記載されています。4Dでは主要なASCIIコードとファンクションキーコードにたいして定数があらかじめ定義されています。「エクスプローラ」ウインドウでこれらの定数のテーマを参照してください。

■ システム変数Modifiersには、モデファイアキーの値が入ります。システム変数Modifiersはイベントが発生した時点で、次のモディファイアキーが押されていたかどうかを示します。

プラットフォーム	モディファイアキー
Windows	Shiftキー、Caps Lock、Altキー、Ctrlキー、右マウスボタン
Macintosh	shiftキー、daps lock、optionキー、controlキー、commandキー

注：

- WindowsのAltキーは、Macintoshのoptionキーに相当します。
- WindowsのCtrlキーは、Macintoshのcommandキーに相当します。
- Macintoshのcontrolキーは、Windowsには対応するキーがありませんが、Windowsでの右マウスボタンのクリックは、Macintoshのcontrolキー+クリックに相当します。

モディファイアキーでは、イベントは発生しません。必ずマウスボタンをクリック、あるいは他のキーが押されなければなりません。システム変数Modifiersは4バイトの倍長整数タイプの変数で、32ビットの配列としてとらえる必要があります。各モディファイアキーに対応するビットを判定するために、4Dにはビット位置やビットマスクを示す定数があらかじめ定義されています。例えば、イベントに対してshiftキーが押されたかどうかを調べるには、次のようにします。

```
If(Modifiers ?? Shift key bit )` shiftキーが押された場合
```

または

```
If((Modifiers & Shift key mask)#0)  ` shiftキーが押された場合
```

■ システム変数MouseXとMouseYには、マウスがクリックされた時の縦と横の位置が入ります。この位置は、クリックが発生したウインドウのローカル座標システムを用いて表現されます。ウインドウの左上隅の位置が0.0です。これらはマウスクリックした場合にのみ有効です。

■ システム変数MouseProcには、イベントが発生した（マウスクリック）プロセスのプロセス参照番号が入ります。

重要：システム変数MouseDown、KeyCode、Modifiers、MouseX、MouseY、MouseProcには、ON EVENT CALLコマンドでインストールされたイベント管理メソッド内でのみ意味を持つ値が納められます。

例題

次の例は、ユーザが「Ctrl+.（ピリオド）」キー（Windows）または「command+.（ピリオド）」キー（Macintosh）を押したら、印刷を中止します。まず、イベント管理メソッドをインストールします。次にユーザにメッセージを表示して、印刷を中止できることを知らせます。イベント管理メソッド内でインタープロセス変数“<>vbWeStop”に“True”が代入されると、既に印刷されたレコードの数をユーザに知らせます。最後にイベントメソッドをクリアします。

```
PAGE SETUP
If(OK=1)
   <>vbWeStop:=False
   ON EVENT CALL("EVENT HANDLER")
                                  ` イベント管理メソッドのインストール
```

```
        ALL RECORDS([従業員])
        MESSAGE("プリントを中止するには、Ctrl(コマンド)-.(ピリオド)を押します")
        $vlNbRecords:=Records in selection([従業員])
        For($vlRecord;1;$vlNbRecords)
            If(<>vbWeStop)
                ALERT("レコード数："+ String($vlRecord)+" ／ "+String
                (Records in selection([従業員]))+"で印刷が中止されました")
                $vlRecord:=$vlNbRecords+1
            Else
                PRINT FORM([従業員];"レポート")
            End if
        End for
        PAGE BREAK
        ON EVENT CALL("") ` イベント管理メソッドをクリア
    End if
```

「Ctrl＋.（ピリオド）」キーまたは「command+.（ピリオド）」キーが押されると、イベント処理メソッドは“<>vbWeStop”に“True”を代入します。

```
    ` EVENT HANDLER プロジェクトメソッド
    If((Modifiers ?? Command key bit) &(KeyCode = Period))
        CONFIRM("本当に、印刷を中止してもいいですか？")
        If(OK=1)
            <>vbWeStop:=True
            FILTER EVENT ` このコールは忘れないでください。
                                そうでないと、4Dもこのイベントを受け取ります。
        End if
    End if
```

この例題では、**ON EVENT CALL**が使用されている点に注意してください。これは、**For**ループで**PAGE SETUP**、**PRINT FORM**、**PAGE BREAK**コマンドを使い、特別な印刷用レポートを生成しているためです。

PRINT SELECTIONコマンドを使ってレポートを印刷する場合、ユーザに印刷を中断させるようなイベント処理を実行する必要はありません。この処理は**PRINT SELECTION**コマンドにより自動的に行われます。

参照

FILTER EVENT、GET MOUSE、Shift down

Method called on event

Method called on event → 文字列

引数	タイプ		説明
この関数には、引数は必要ありません			
戻り値	文字列	←	イベント時に呼び出されるメソッド名

説明

Method called on event関数は、**ON EVENT CALL**コマンドによって組み込まれたメソッド名を返します。

組み込まれたメソッドが存在しない場合、空の文字列（""）が返されます。

参照

ON EVENT CALL

FILTER EVENT

FILTER EVENT

引数	タイプ	説明

このコマンドには、引数はありません。

説明

FILTER EVENTコマンドは、**ON EVENT CALL**コマンドでインストールされたイベント管理用プロジェクトメソッドから呼び出します。

イベント管理メソッドでこのコマンドを呼び出すと、カレントイベントが4Dに渡されなくなります。

このコマンドを使用すると、イベントキューからカレントイベント（クリック、キー入力）を取り除くことができます。したがって、4Dはイベント管理用プロジェクトメソッド内で発生したイベントに対してそれ以上の処理は行いません。

警告：FILTER EVENTコマンドを呼び出すだけのイベント管理メソッドを作成しないようにしてください。すべてのイベントが4Dから無視されるためです。FILTER EVENTコマンドだけのイベント管理メソッドがある場合には、「Ctrl+Shift+Alt+Backspace」（Macintosh版では、「command+option+shift+control+delete」）キーを押します。これにより、ON EVENT CALLプロセスがイベントをまったく受け取らない通常のプロセスに切り替えます。

特別なケース：DISPLAY SELECTIONや MODIFY SELECTIONコマンドを用いてフォームを表示している場合に、FILTER EVENTコマンドを標準の出力フォームメソッド内で使用することもできます。この特別なケースでは、FILTER EVENTコマンドを使用してレコード上でのダブルクリックをフィルタリングすることができます（また、この方法でページモードでのレコードオープン以外の動作を実行します）。これを行うには、出力フォームメソッドに次の行を追加します。

```
If(Form event=On Double Clicked)
    FILTER EVENT
    ... `ダブルクリックの処理
End if
```

例題

前述の**ON EVENT CALL**コマンドの例を参照してください。

参照

ON EVENT CALL

ON ERR CALL

ON ERR CALL(エラーメソッド)

引数	タイプ		説明
エラーメソッド	文字列	→	インストールするエラー処理メソッド または、エラーの検出を中止する場合には 空の文字列

説明

ON ERR CALLコマンドは、エラー検出用のメソッドとして＜エラーメソッド＞で渡した名前のプロジェクトメソッドをインストールします。このプロジェクトメソッドはエラー管理メソッドまたはエラー検出メソッドと呼ばれます。

このコマンドのスコープは、カレントプロセスです。デフォルトでは、別々のローカルプロセス内でメソッドが実行されます。プロセス単位で一度に1つのエラー管理メソッドだけを使用できます。複数のプロセスに対して異なるエラー管理メソッドを持つことができます。

エラーの検出を中止するには、再度**ON ERR CALL**コマンドをコールし、＜エラーメソッド＞に空の文字列を指定します。

エラー管理プロジェクトのインストール後は、エラーが発生するたびに4th Dimensionが＜エラーメソッド＞を呼び出します。

エラーの種別は、システム変数Errorの値で判断します。このシステム変数にはエラーのコード番号が納められます。このマニュアルの付録Cに、エラーコードが記載されています。詳細は、付録Cの「シンタックスエラー」または「データベースエンジンエラー」を参照してください。システム変数通Errorの値は、エラー管理メソッド内のみで有効です。エラーの原因となったメソッド内でこのエラーコードが必要であれば、システム変数Errorを独自のプロセス変数にコピーしてください。

エラー管理メソッドは適切な方法でエラーを管理、またはユーザに対してエラーメッセージを表示します。エラーは、次のようなものから生成されます。

- 4th Dimensionのデータベースエンジン：例えば、レコードを保存している最中に、重複不可のインデックスを複製しようとした場合。
- 4th Dimension環境：例えば、配列に割り当てるために十分なメモリーがない場合。
- データベースが起動しているオペレーションシステム：例えば、ディスクに空きがなかったり、I／O（入出力）エラーの場合。

実行を終了するには、**ABORT**コマンドを使用します。エラー管理メソッドで**ABORT**コマンドを使用しない場合、4th Dimensionは割り込みをかけたメソッドに制御を戻し、メソッドの実行を続けます。

エラー管理メソッド自体でエラーが発生した場合は、4th Dimensionがエラー管理を行います。したがって、エラー管理メソッドでエラーが発生しないように十分注意してください。また、エラー処理メソッドでは、**ON ERR CALL**コマンドを使用することはできません。

ON ERR CALLエラー処理メソッドをインストールすると、Windows上ではAltキー、Macintosh上ではoptionキーを押しながらマウスボタンをクリックして、メソッドのトレースを実行することができません。これは、“Alt（option）”キーを押しながらマウスボタンをクリックすると、エラーを生成し（エラーコード1006）、**ON ERR CALL**メソッドが即座に起動されてしまうためです。ですが、このエラーコードは**TRACE**をコールして調べることができます。

例題

1. 次のプロジェクトメソッドは、引数で渡された名前のドキュメントテーブルを作成します。ドキュメントが作成できない場合、このプロジェクトメソッドは0またはエラーコードを返します。

```
`「ドキュメント作成」プロジェクトメソッド
    ` ドキュメント作成(文字列; ポインタ) -> 倍長整数
    ` ドキュメント作成(ドキュメント名;->ドキュメント参照番号) -> エラーコード結果
    gError:=0
    ON ERR CALL("IO エラー処理")
    $2->:=Create document($1)
    ON ERR CAL L("")
    $0:=gError
```

“IO エラー処理”プロジェクトメソッドは次の通りです。

```
    `「IO エラー処理」プロジェクトメソッド
    gError:=Error ` エラーコードをプロセス変数gErrorにコピーするだけ
```

現在実行中のメソッド内でエラーコードの結果を取得するために、プロセス変数gErrorを使用している点に注意してください。データベースにこれらのメソッドを作成したら、次のようなコードを使用します。

```
    ` ...
    C_TIME(vhDocRef)
    $vlErrCode:= ドキュメント作成($vsDocumentName;->vhDocRef)
    If($vlErrCode=0)
    `...
        CLOSE DOCUMENT($vlErrCode)
```

```
    Else
        ALERT("ドキュメントが作成できません。 I/O エラー："
                                        +String($vlErrCode))
    End if
```

2. 「配列とメモリ」の節にある例題を参照してください。

3. 複雑な一連の処理を実行中に、各種サブルーチンで異なるエラー管理メソッドが必要となる場合があります。プロセスごとにいちどに1つのエラー管理メソッドしか持つことができないため、次の2通りの方法から対応策を選択することができます。

■ **ON ERR CALL**コマンドを呼び出すたびに現在のエラー管理メソッドを処理します。

■ プロセス配列変数を使用し（この例では、asErrorMethod）、エラー管理メソッドとプロジェクトメソッド（この例では、**ON ERR CALL**）を“積み上げ”て、エラー管理メソッドのインストールとクリアを行います。

プロセスの実行を開始する時点で配列を初期化する必要があります。

```
    ` プロセスメソッド(プロセスを実行するプロジェクトメソッド)の最初で
                                        配列の初期化を忘れないように
    ARRAY STRING(63;asErrorMethod;0)
```

次にカスタムのON ERR CALLメソッドを示します。

```
    ` 「ON ERROR CALL」プロジェクトメソッド
    ` ON ERROR CALL {( 文字列) }
    ` ON ERROR CALL {( メソッド名 ) }
    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
```

この後で、このプロジェクトメソッドを次のように呼び出します。

```
    gError:=0
    ON ERROR CALL("IO ERRORS")
                                ` IO ERRORS エラー管理メソッドをインストールする
    ` ...
    ON ERROR CALL("ALL ERRORS")
                                ` ALL ERRORS エラー管理メソッドをインストールする
    ` ...
    ON ERROR CALL ` ALL ERRORS エラー管理メソッドをクリアして、
                                IO ERRORSエラー管理メソッドを際インストールする
    ` ...
    ON ERROR CALL ` IO ERRORS エラー管理メソッドをクリアする
```

4. 次のエラー管理メソッドはユーザによる割り込みを無視します。

```
    `「SHOW ONLY ERRORS」プロジェクトメソッド
    If(Error#1006)
        ALERT("エラー番号"+String(Error)+" が発生しました。.")
    End if
```

参照

ABORT

Method called on error

Method called on error → 文字列

引数	タイプ		説明
この関数には、引数は必要ありません			
戻り値	文字列	←	エラー時に呼び出されるメソッド名

説明

Method called on error関数は、**ON ERR CALL**コマンドによりカレントプロセスに組み込まれたメソッド名を返します。

組み込まれたメソッドが存在しない場合、空の文字列（""）が返されます。

例題

この関数を使用すると、エラー検出用メソッドを一時的に変更し、後で復元することができるため、コンポーネントの使用時に大変役立ちます。

```
$methCurrent:=Method called on error
ON ERR CALL("NewMethod")
      ` ドキュメントを開けない場合、エラーが生成される
$ref:=Open document("MyDocument")
      ` 以前のメソッドを再度組み込む
ON ERR CALL($methCurrent)
```

参照

ON ERR CALL

ABORT

注：このコマンドはほとんど使用しません。

ABORT

説明

ABORTコマンドは、**ON ERR CALL**コマンドでインストールされたエラー管理プロジェクトメソッド内で使用します。

エラー管理プロジェクトメソッドが存在しない場合、エラーが発生すると（例えば、データベースエンジンのエラー）4Dにより標準のエラーダイアログボックスが表示され、コードの実行が中断されます。実行しているコードにより、次のようになります。

- オブジェクトメソッド、フォームメソッド（あるいはフォームメソッドやオブジェクトメソッドからコールされたプロジェクトメソッド）の場合、現在表示されているフォームに制御が戻ります。
- メニューから呼び出されたメソッドの場合、メニューバーまたは現在表示されているフォームに制御が戻ります。
- プロセスのマスターメソッドの場合、プロセスは終了します。
- データの読み込み、または書き出し処理により直接的あるいは間接的に呼び出されたメソッドの場合、処理は中止されます。順次クエリや並び替え処理に関しても同様です。
- その他

エラー管理メソッドを使用してエラーを検出する場合、4Dは標準のエラーダイアログボックスの表示もコードの実行の中断も行いません。この代りに、4Dはエラー管理プロジェクトメソッドを呼び出し、エラーを発生したメソッドの次のコード行から実行を再開します。

プログラムから処理できるエラーもあります。例えば、データ読み込み処理中にデータベースエンジン複製値エラーを検出した場合、このエラーを“カバー”して、読み込みを続けることができます。しかし、対処できないエラーや“カバー”してはいけないエラーもあります。この場合、エラー管理プロジェクトメソッドから**ABORT**コマンドを呼び出して、実行を中止する必要があります。

以前のバージョンに関する注意

ABORTコマンドはエラー管理プロジェクトメソッド内でのみ使用されるようになっていますが、4Dコミュニティには実行を中断するために他のプロジェクトメソッドで使用しているメンバーも存在します。これが動作するというのは単に副次的な効果に過ぎません。このコマンドをエラー管理メソッド以外のメソッド内で使用することはお勧めできません。

Count parameters

Count parameters → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	実際に渡される引数の数

説明

Count parameters関数は、プロジェクトメソッドに渡された引数の数を返します。

警告：Count parameters関数は、他のメソッド（プロジェクトメソッド、その他）から呼び出されるプロジェクトメソッド内でのみ意味を持ちます。Count parameters関数を呼び出すプロジェクトメソッドがメニューに割り当てられている場合、Count parameters関数は0を返します。

例題

1. 4th Dimensionは右側から始まるオプションの引数を受け付けます。例えば、メソッド“MyMethod(a;b;c;d)”は以下のように呼び出すことができます。

```
MyMethod(a ; b ; c ; d) ` すべての引数が渡される
MyMethod(a ; b ; c) ` 最後の引数は渡されない
MyMethod(a ; b) ` 最後の2つの引数は渡されない
MyMethod(a) ` 最初の引数だけが渡される
MyMethod ` 引数は渡されない
```

“MyMethod”内で**Count parameters**関数を使用し、実際の引数の数を取得してその数に応じて異なる処理を実行することができます。以下の例題では、テストメッセージを表示し、4D Writeエリアにテキストを挿入、またはディスク上のドキュメントにテキストを送信することができます。

```
` 「テキスト追加」プロジェクトメソッド
` テキスト追加(テキスト { ; 倍長整数 { ; 時間 }})
` テキスト追加(テキスト { ; 4D Write エリア { ; ドキュメント参照番号}})
```

```
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
```

このプロジェクトメソッドをアプリケーションに追加すると以下のように記述できます。

```
テキスト追加(vtSomeText) ` テキストメッセージのみ表示
テキスト追加(vtSomeText;$wrArea) ` テキストメッセージを表示し、
        `テキストを$wrAreaに追加
テキスト追加(vtSomeText;0;$vhDocRef) ` テキストメッセージを表示し、
                                        $vhDocRefに書き込む
```

2. 4th Dimensionのプロジェクトメソッドは、右側から始めて、可変数の同タイプの引数を受け付けます。これらの引数を宣言するには、コンパイル命令を使用し、変数として${N}を渡します。Nは最初の引数を示します。**Count parameters**関数を使い、**For**ループと引数の間接参照構文を用いてこれらの引数にアクセスすることができます。この例題は関数で、引数として受け取った最も大きな数値を返します。

```
` 「Max of 」プロジェクトメソッド
` Max of(実数 { ; 実数2... ; 実数N}) -> 実数
` Max of(値 { ; 値2... ; 値N}) -> 最大値
C_REAL($0;${1}) ` すべての関数および引数の結果は実数タイプ
$0:=${1}
For($vlParam;2;Count parameters)
        If(${$vlParam}>$0)
                $0:=${$vlParam}
        End if
End for
```

このプロジェクトメソッドをアプリケーションに追加すると以下のように記述できます。

```
vrResult:=Max of(Records in set("Operation A");Records in set("Operation B"))
```

または

```
vrResult:=Max of(r1 ; r2 ; r3 ; r4 ; r5 ; r6)
```

参照

コンパイラコマンド、C_BLOB、C_BOOLEAN、C_DATE、C_GRAPH、C_INTEGER、C_LONGINT、C_PICTURE、C_POINTER、C_REAL、C_STRING、C_TEXT、C_TIME

Type

Type(引数) → 数値

引数	タイプ		説明
引数	フィールド	→	タイプを求めるデータオブジェクトまたは変数
戻り値	数値	←	データタイプ番号

説明

Type関数は、＜引数＞で渡したフィールドや変数のタイプを示す数値を返します。

4th Dimensionは、以下のようなあらかじめ定義された定数を持っています。

定数	タイプ	値
Is Alpha Field	倍長整数	0
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Real	倍長整数	1
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Date	倍長整数	4
Is Time	倍長整数	11
Is Boolean	倍長整数	6
Is Picture	倍長整数	3
Is Subtable	倍長整数	7
Is BLOB	倍長整数	30
Is Undefined	倍長整数	5
Is Pointer	倍長整数	23
String array	倍長整数	21
Text array	倍長整数	18
Real array	倍長整数	14
Integer array	倍長整数	15
LongInt array	倍長整数	16
Date array	倍長整数	17
Boolean array	倍長整数	22

Picture array	倍長整数	19
Pointer array	倍長整数	20
Array 2D	倍長整数	13

互換性に関する注意点：前バージョンの4Dでは、C_GRAPHコマンドを使ってタイプ宣言した任意のグラフ変数に対し、Type関数は3（Is Picture）を返していました。しかし、バージョン6からは、Type関数はそのグラフ変数に9（Is Longint）を返します。

Type関数をフィールド、インタープロセス変数、プロセス変数、ローカル変数、およびこれらのオブジェクトタイプを参照するポインタで使用することができます。

バージョン6の注意点：バージョン6から、Type関数を引数（$1、$2...,${..}）、プロジェクトメソッド、戻り値（$0）に対して使用することができるようになりました。

例題

1. **APPEND TO CLIPBOARD**コマンドの例を参照してください。

2. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

3. 以下のプロジェクトメソッドは、テーブルのカレントレコードのフィールドの一部あるいはすべてをクリアします。テーブルへのポインタは引数として渡されます。カレントレコードの削除や変更は行いません。

```
` 「EMPTY RECORD」プロジェクトメソッドd
` EMPTY RECORD(ポインタr {; 倍長整数})
` EMPTY RECORD(-> [テーブル] { ; タイプフラグ})
C_POINTER($1)
C_LONGINT($2;$vlTypeFlags)
If(Count parameters>=2)
	$vlTypeFlags:=$2
Else
	$vlTypeFlags:=0xFFFFFFFF
End if
For($vlField;1;Count fields($1))
	$vpField:=Field(Table($1);$vlField)
	$vlFieldType:=Type($vpField->)
	If( $vlTypeFlags ?? $vlFieldType )
		Case of
			¥(($vlFieldType=Is Alpha Field)|($vlFieldType=Is Text))
				$vpField->:=""
			¥(($vlFieldType=Is Real)|($vlFieldType=Is Integer)|
                                                (($vlFieldType=Is LongInt))
				$vpField->:=0
```

```
                ¥($vlFieldType=Is Date)
                        $vpField->:=!00/00/00!
                ¥($vlFieldType=Is Time)
                        $vpField->:=!!00:00:00!!
                ¥($vlFieldType=Is Boolean)
                        $vpField->:=False
                ¥($vlFieldType=Is Picture)
                        C_PICTURE($vgEmptyPicture)
                        $vpField->:=$vgEmptyPicture
                ¥($vlFieldType=Is Subtable)
                        Repeat
                                ALL SUBRECORDS($vpField->)
                                DELETE SUBRECORD($vpField->)
                        Until(Records in subselection($vpField->)=0)
                ¥($vlFieldType=Is BLOB)
                        SET BLOB SIZE($vpField->;0)
        End case
    End if
End for
```

このプロジェクトメソッドをデータベースに作成後、以下のように使用できます。

```
` テーブル [Things To Do]のカレントレコードをすべてクリアする
EMPTY RECORD(->[Things To Do])
` テーブル [Things To Do]のカレントレコードのテキスト、BLOB、
                                                ピクチャフィールドをクリア
EMPTY RECORD(->[Things To Do]; 0 ?+ Is Text ?+ Is BLOB ?+ Is Picture )
` テーブル [Things To Do]のカレントレコードをすべてクリア
                                                （文字フィールド以外）
EMPTY RECORD(->[Things To Do]; -1 ?- Is Alpha Field )
```

参照

Is a variable、Undefined

Self

Self → ポインタ

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ポインタ	←	そのメソッドが現在実行中のフォームオブジェクトへのポインタ（ある場合）またはコンテキスト外の場合にはNil（->[]）

説明

Self関数は、現在実行中のオブジェクトメソッドの属するオブジェクトへのポインタを返します。

Self関数は、オブジェクトメソッド内で変数を参照するために使用します。この関数は、オブジェクトメソッドから呼び出されたときにだけ有効なポインタを返します。プロジェクトメソッドがオブジェクトメソッドから呼び出されていても、この関数をプロジェクトメソッドで使用することはできません。コンテキストの外から**Self**関数が呼ばれた場合、Nilポインタ（->[]）が返されます。

Tip：Self関数は、フォーム上の複数のオブジェクトに同じ処理を実行させる場合に便利です。

例題

RESOLVE POINTERコマンドの例題を参照してください。

参照

RESOLVE POINTER

RESOLVE POINTER

RESOLVE POINTER(ポインタ ; 変数名 ; テーブル番号 ; フィールド番号)

引数	タイプ		説明
ポインタ	ポインタ	→	参照オブジェクトを取得するためのポインタ
変数名	文字列	←	参照される変数名または空の文字列
テーブル番号	数値	←	参照されるテーブルの番号、または配列要素、または0か-1
フィールド番号	数値	←	参照フィールド番号、または0

説明

RESOLVE POINTERコマンドは、＜ポインタ＞式によって参照されるオブジェクトの情報を取得し、引数＜変数名＞、＜テーブル番号＞、＜フィールド番号＞に返します。

参照されるオブジェクトの種類によって、**RESOLVE POINTER**コマンドは、以下のような値を返します。

参照オブジェクト	引数（パラメータ）		
	＜変数名＞	＜テーブル名＞	＜フィールド番号＞
なし(NIL ポインタ)	""(空の文字列)	0	0
変数	変数の名前	-1	0
配列	配列の名前	-1	0
配列要素	配列の名前	要素番号	0
テーブル	""(空の文字列)	テーブル番号	0
フィールド	""(空の文字列)	テーブル番号	フィールド番号

注：＜ポインタ＞に渡す値がポインタ式でない場合には、シンタックスエラーが発生します。

例題

1. フォーム内で、v1～v100までの入力可能な変数100個を作成します。

これを実行するには、以下のような手順を実行します。

a. 入力可能な変数を1つ作成し、“v”と名付ける。

b. オブジェクトのプロパティを設定する。

c. オブジェクトに以下のメソッドを作成する。

```
DoSomething(Self) `DoSomethingがデータベース内でのプロジェクトメソッドになる
```

d. この時点で、必要な回数だけ変数を複製することも、「フォーム」エディタの「グリッドで変数作成」機能を使用することもできる。

e. 「DoSomething」メソッド内で、このメソッドが呼び出される変数のインデックスを知る必要がある場合には、以下のように書き込む。

```
RESOLVE POINTER($1 ; $vsVarName ; $vlTableNum ; $vlFieldNum)
$vlVarNum:=Num(Substring($vsVarName ; 2))
```

この方法でフォームを作成することによって、100個の変数のためのメソッドを一度書くだけで済むことに注目してください。DoSomething(1)、DoSomething(2)からDoSomething(100)までを作成する必要はありません。

2. デバッグのために、メソッドへの2番目の引数（$2）がテーブルへのポインタであることを確認する必要があります。このメソッドの最初で、以下のように記述します。

```
If(<>DebugOn)
    RESOLVE POINTER($2 ; $vsVarName ; $vlTableNum ; $vlFieldNum)
    If(Not(($vlTableNum>0)&($vlFieldNum=0)&($vsVarName="")))
        `警告：ポインタは、テーブルへの参照ではない
        TRACE
    End if
End if
```

3. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照

DRAG AND DROP PROPERTIES、Field、Get pointer、Is a variable、Nil、Table

Nil

Nil(ポインタ) → ブール

引数	タイプ		説明
ポインタ	ポインタ	→	判定されるポインタ
戻り値	ブール	←	True=Nilポインタ False=既存オブジェクトへの有効なポインタ

説明

Nil関数は、＜ポインタ＞がNilポインタ（->[ ]）の場合に“True（真）”を返します。その他の場合（フィールドやテーブル、変数へのポインタ）、この関数は“False（偽）”を返します。

バージョン6より、**Nil**関数を使用する代りに、**RESOLVE POINTER**コマンドを使う方がより便利です。このコマンドはオブジェクトの種類に関わらず（Nilポインタを含む）、参照オブジェクトに関する情報を返します。

参照

Is a variable、RESOLVE POINTER

Is a variable

Is a variable(ポインタ) → ブール

引数	タイプ		説明
ポインタ	ポインタ	→	判定されるポインタ
戻り値	ブール	←	True=変数へのポインタ False=変数以外へのポインタ

説明

Is a variable関数は、＜ポインタ＞が定義済み変数を参照する場合には“True（真）”を返します。その他の場合（フィールドやテーブルへのポインタ、Nilポインタ）、この関数は“False（偽）”を返します。

バージョン6より、**Is a variable**関数を使用する代りに、**RESOLVE POINTER**コマンドを使う方がより便利です。このコマンドはオブジェクトの種類に関わらず（Nilポインタを含む）、参照オブジェクトに関する情報を返します。

参照

Nil、RESOLVE POINTER

Get pointer

Get pointer(変数名) → ポインタ

引数	タイプ		説明
変数名	文字列	→	プロセス変数の名前
戻り値	ポインタ	←	プロセス変数へのポインタ

説明

Get pointer関数は、引数<変数名>に渡した名前を持つ変数へのポインタを返します。

フィールドへのポインタを取得するには**Field**関数を使用します。テーブルへのポインタを取得するには**Table**関数を使用します。

注：例として、Get pointer($ArrName+"{3}"は許されます。しかし、二次元配列のGet pointer($ArrName+"{3}{5}")および配列要素に変数を使用したGet pointer ($ArrName+"{myVar}")は許されません。

例題

フォーム上で、5×10のグリッドの入力可能な変数を作成し、それぞれv1、v2...、v50という名前を付けます。これらの変数をすべて初期化するには次のようにします。

```
For($vlVar ; 1 ; 50)
    $vpVar:=Get pointer("v" + String($vlVar)) ` 変数のポインタを求める
    $vpVar->:=""
End for
```

参照

Field、Table

EXECUTE

注：このコマンドはほとんど使用されません。

EXECUTE(ステートメント)

引数	タイプ		説明
ステートメント	文字列	→	実行するコード

説明

EXECUTEコマンドは、1行の＜ステートメント＞を実行します。ステートメントの文字列は必ず1行だけです。＜ステートメント＞に空の文字列を指定した場合には、**EXECUTE**コマンドは何も行いません。

実行できるステートメントが1行でも、プロジェクトメソッドを指定すれば、適切な処理を実行することができます。

コンパイル済みデータベースでは、このコード行はコンパイルされません。つまり、＜ステートメント＞は実行されますが、コンパイル時にコンパイラによるチェックは行われません。

EXECUTEコマンドを使用すると実行速度が遅くなりますので、なるべく使用を控えるようにしましょう。

使用できるステートメントを次に記します。

■ プロジェクトメソッドの呼び出し

■ 4Dコマンドの呼び出し

■ 代入式

ステートメントにはプロセス変数やインタープロセス変数を含めることができます。1行のコードでなくてはならないため、ステートメントにフロー制御文（If文など）を使用することはできません。

例題

Command Name関数の例題を参照してください。

参照

Command Name

Command name

Command name(コマンド番号)→ 文字列

引数	タイプ		説明
コマンド番号	数値	→	コマンド番号
戻り値	文字列	←	ローカライズされたコマンドの名前

説明

Command name関数は、引数<コマンド番号>に渡したコマンド番号のコマンドの名前を返します。

4th Dimensionは、メソッドで使用するキーワード、定数、およびコマンド名をダイナミックに翻訳して統合します。4Dの日本語版を使用している場合は、次のように記述します。

```
DEFAULT TABLE([MyTable])
ALL RECORDS([MyTable])
```

この同じコードは、4Dのフランス語版でも再オープンされ、次のように読み込まれます。

```
TABLE PAR DEFAUT([MyTable])
TOUT SELECTIONNER([MyTable])
```

ただし、4th Dimensionにはユニークな機能である**EXECUTE**コマンドがあり、これを使用すると、データベースがコンパイルされていてもただちにコードを作成して、このコードを実行することができます。

日本語版（英語版）で**EXECUTE**文を使用して作成されたコードの例は、以下のようになります。

```
EXECUTE("DEFAULT TABLE([MyTable])")
EXECUTE("ALL RECORDS([MyTable])")
```

この同じコードは、4Dのフランス語版でも再オープンされ、以下のようになります。

```
EXECUTER("DEFAULT TABLE([MyTable])")
EXECUTER("ALL RECORDS([MyTable])")
```

4Dは、**EXECUTE**（日本語版／英語版）を**EXECUTER**（フランス語版）に自動的に翻訳しますが、コマンドに渡したテキスト文を翻訳することはできません。

アプリケーションで**EXECUTE**コマンドを使用する場合には、**Command name**関数を使用して異なる言語間のローカライズにおける実行文の問題をこのようにして解決し、文を言語に依存しないようにすることができます。

コードの例は、以下のようになります。

```
EXECUTE(Command name(46)+"([MyTable])")
EXECUTE(Command name(47)+"([MyTable])")
```

4Dのフランス語版を使用すると、このコードは以下のようになります。

```
EXECUTER(Nom commande(46)+"([MyTable])")
EXECUTER(Nom commande(47)+"([MyTable])")
```

注：コマンド番号を知るには、エクスプローラウインドウのその名前の所を選択します。コマンド番号は、ウインドウの右の欄に表示されています。

例題

1. データベースのすべてのテーブルについて、各テーブルの標準データ入力に使用するための“入力フォーム”というフォームがあります。そこで、ポインタまたはテーブル名を渡すテーブルのカレント入力フォームとしてこのフォームを設定する、汎用的なプロジェクトメソッドを追加する場合には、次のように記述します。

```
` 「STANDARD INPUT FORM」プロジェクトメソッド
` STANDARD INPUT FORM(ポインタ {; 文字列 })
` STANDARD INPUT FORM(->テーブル {; テーブル名 })
C_POINTER($1)
C_STRING(31;$2)
If(Count parameters>=2)
   EXECUTE(Command name(55)+"(["+$2+"];"入力フォーム")")
Else
   If(Count parameters>=1)
      INPUT FORM($1-> ; "I入力フォーム")
   End if
End if
```

このプロジェクトメソッドがデータベースに追加された後、以下のように記述します。

```
STANTARD INPUT FORM(->[従業員])
STANTARD INPUT FORM("従業員")
```

注：通常、汎用ルーチンを記述する場合には、ポインタを使用することをお勧めします。その理由としては、まず、データベースがコンパイルされると、コードはコンパイルされるためです。次に、4D Insiderが、ポインタを渡したオブジェクトへの参照を検索するためです。最後に、前述の例でもあるようにテーブルの名前を変更すると、コードは正しく動作しなくなるためです。ただし、EXECUTEコマンドを使用すれば問題が解決する場合もあります。

2. フォームでは、ドロップダウンリストを一般的なサマリーレポートコマンドと共存させるには、そのドロップダウンリストのオブジェクトメソッドに、次のように記述します。

```
Case of
    ¥(Form event =On Before)
        ARRAY TEXT(asCommand ; 4)
        asCommand{1}:=Command name(1) ` Sum関数
        asCommand{2}:=Command name(2) ` Average関数
        asCommand{3}:=Command name(4) ` Min関数
        asCommand{4}:=Command name(3) ` Max関数
        ` ...
End case
```

4Dの日本語版あるいは英語版では、ドロップダウンリストには、Sum、Average、Min、Maxが表示されます。フランス語版では、ドロップダウンリストには、Somme、Moyenne、Min、Maxが表示されます。

参照

EXECUTE

Current method name

Current method name → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	コールされたメソッドの名前

説明

Current method name関数は、メソッドが呼び出されたときに、その名前を返します。この関数は一般的なメソッドをデバッグするときに有効です。

呼ばれるメソッドのタイプにより、返却される文字列は次のようになります。

呼ばれたメソッド	文字列
データベースメソッド	メソッド名
トリガー	トリガーのテーブル名
プロジェクトメソッド	メソッド名
フォームメソッド	テーブル名フォーム名
オブジェクトメソッド	テーブル名フォーム名.オブジェクト名

4Dフォーミュラ内からこの関数を呼ぶことはできません。

TRACE

TRACE

説明

データベースの開発時に、**TRACE**コマンドを使用してメソッドをトレースすることができます。

TRACEコマンドは、カレントプロセス用の4th Dimensionのデバッガを起動します。コードの次の行が実行される前に「デバッグ」ウインドウを表示します。実行するコードを表示しながらメソッドの実行を続けることができます。コードの実行中に、Windows上ではAltキー、Macintosh上ではoptionキーを押しながらマウスボタンをクリックして、デバッガを起動することもできます。

コンパイルされたデータベースでは、**TRACE**コマンドは無視されます。

4D Server：ストアドプロシージャのコンテキスト内で実行されたプロジェクトメソッドからTRACEコマンドをコールすると、「デバッグ」ウインドウはサーバマシン上に表示されます。

Tips

1. **On Activate**および**On Deactivate**イベントが選択されたフォームを使用する場合、**TRACE**コマンドを使用しないでください。「デバッグ」ウインドウが表示されるたびに、これらのイベントが起動されて、この2つのイベントと「デバッグ」ウインドウとの間で永久ループになってしまいます。このような状況になった場合には、「デザイン」モードに移って永久ループから抜けることができます。

これを行うには、イベントウインドウまたは「デバッグ」ウインドウをクリックします。その後で、次の手順を行います。

■ プロジェクトメソッド、またはオブジェクトメソッドから**TRACE**コマンドを呼び出している場合（メソッドは実行時に再ロードされます)、これを削除します。すると、永久ループが中断されます。

■ フォームメソッドから**TRACE**コマンドを呼び出している場合には、フォームから抜けるまで再ロードされません。ループに陥っているため、フォームから抜けることは不可能です。したがって、データベースの再オープンか、プロセスのアボートのいずれかが問題となります。プロセスをアボートできない場合には、データベースを再オープンする必要があります。

2. 画面でフォームの更新中に実行されたフォームメソッドまたはオブジェクトメソッドから**TRACE**コマンドを呼び出す場合にも、フォームの更新と「デバッグ」ウインドウの表示が無限に繰り返されてしまいます。この時点で、「Alt+Shift」キー（Windows）または「option+shift」キー（Macintosh）を押します。すると、カレントウインドウの更新イベントが無効になり、その結果フォームやオブジェクトメソッドからの**TRACE**コマンド呼び出しが中止されます。次に、「デザイン」モードへ移り、**TRACE**コマンド呼び出しを削除します。

上記の2つのTipsは、コードに設定された永続的ブレークポイントによって引き起こされる同様の状況に対しても適用できる点に注意してください。Tip1、ブレークポイントの置かれた位置に関わらず、それを取り除くことができるので、データベースを再オープンしなくても「デバッグ」ウインドウ表示を中止させることができます。

例題

次のコードでは、プロセス変数BUILD_LANGに“US”あるいは“FR”という値を求めています。それ以外の値である場合に、プロジェクトメソッド「DEBUG」を呼び出します。

```
Case of
	¥(BUILD_LANG="US")
		vsBHCmdName:=[Commands]CM US Name
	¥(BUILD_LANG="FR")
		vsBHCmdName:=[Commands]CM FR Name
Else
	DEBUG("BUILD_LANG の値が違います。")
End case
```

「DEBUG」プロジェクトメソッドを次に示します。

```
` 「DEBUG」プロジェクトメソッド
` DEBUG(テキスト)
` DEBUG(任意のデバッグ情報)
C_TEXT($1)
If(<>vbDebugOn) ` On Startup メソッドで設定されたインタープロセス変数
	If(Compiled Application)
		If(Count parameters>=1)
			ALERT($1+Char(13)+"設計者を呼び出してください。
                                                番号：x911")
		End if
	Else
		TRACE
	End if
End if
```

参照

NO TRACE

NO TRACE

NO TRACE

説明

データベースの開発時に、**NO TRACE**コマンドを使用してメソッドをトレースすることができます。

NO TRACEコマンドは、**TRACE**コマンド、エラー、ユーザによって起動されたデバッガを終了します。「デバッグ」ウインドウの「トレースなし」ボタンをクリックしても同じ動作をします。

コンパイルされたデータベースでは、**NO TRACE**コマンドは無視されます。

参照

TRACE

リストボックスの管理

このテーマのコマンドは、新しいフォームオブジェクトタイプであるリストボックスを扱うために設けられました。

リストボックスは、以前のバージョンの4th Dimensionにおける“グループ化したスクロールエリア”に相当します。リストボックスはグループ化したスクロールエリアのすべての機能、特に列や選択可能な行の形式でデータを表示する機能を提供します。しかし、リストボックスには、値の入力、列の並び替え、代替色の定義など、さらに数多くの機能が用意されています。



4th Dimensionの「フォーム」エディタでリストボックスオブジェクトタイプを完全に設定することが可能で、また、プログラムから管理することもできます。リストボックスタイプのオブジェクトの作成および設定に関する詳細は、『デザインリファレンス』マニュアルを参照してください。

“リストボックス”タイプのオブジェクトのプログラミングは、4th Dimensionの他のリストフォームオブジェクトと同じ方法で行われます。ただし、次の節で説明するように、特定のルールに従わなくてはなりません。

作成および値の管理

“リストボックス”オブジェクトには、1つ以上の列を納めることができます。それぞれの列は、一次元の4th Dimension配列に関連付けなくてはなりません。すべての配列タイプが使用可能ですが、ポインタ配列だけは利用できません。各列の表示フォーマットは、「フォーム」エディタや**SET FORMAT**コマンドを使用して指定することができます。

ランゲージを使用する場合、ハイレベル“リストボックス”コマンド（**INSERT LISTBOX ROW**など）ならびに配列操作コマンドを用いて、列の値（データ入力と表示）を管理します。

例えば、列の内容を初期化するには、次の命令を使用することができます。

ARRAY TEXT(カラム名; サイズ)

また、リストを使用することもできます。

LIST TO ARRAY(リスト名; カラム名)

注：リストボックスオブジェクトに複数の列が含まれている場合、関連する各配列のサイズは他の配列と同じでなければなりません（項目数が同じ）。同じサイズではない場合、最も小さな配列の項目数と同じ数だけ表示されます。

オブジェクト、カラム、ヘッダ

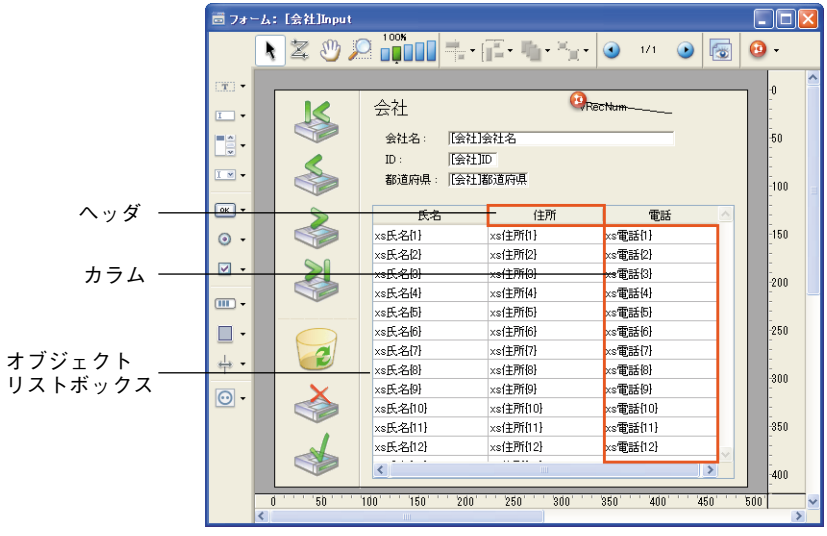
リストボックスオブジェクトは、以下の異なる3つの項目で構成されます。

■ オブジェクト自体

■ 列

■ 列ヘッダ

これらの項目は「フォーム」エディタ上では別々に選択されます。それぞれが独自のオブジェクト名や変数名を持ち、個別に処理されます。



デフォルトでは、リストボックスオブジェクトそのものとは関係なく、フォーム上の列にはカラム1からXまでの名前が付けられ、ヘッダにはヘッダ1からXまでの名前が付けられます。

各項目タイプには、独自の特性ならびに他の項目と共有する特性があります。例えば、文字のフォントはリストボックスオブジェクトに一括して割り当てることも、列やヘッダに対して個別に割り当てることもできます。これとは逆に、入力プロパティは列に対してのみ指定することができます。

このルールは、リストボックスと一緒に使用される「オブジェクトプロパティ」テーマのコマンドに対して適用されます。その機能に応じて、各コマンドをリストボックスや列、列ヘッダに対して使用します。作業を行おうとする項目のタイプを設定するには、その項目に関連付けた名前や変数を渡します。

次の表は、「オブジェクトプロパティ」テーマの各コマンドで使用可能なリストボックスオブジェクトタイプについて詳述したものです。

オブジェクトプロパティコマンド	オブジェクト	列	列ヘッダ
MOVE OBJECT	○		
GET OBJECT RECT	○		
SET FILTER		○	
SET FORMAT		○	
SET ENTERABLE		○	
SET CHOICE LIST		○	
BUTTON TEXT			○
SET COLOR	○	○	

オブジェクトプロパティコマンド	オブジェクト	列	列ヘッダ
SET RGB COLORS	○	○	
FONT	○	○	○
FONT SIZE	○	○	○
FONT STYLE	○	○	○
SET ALIGNMENT	○	○	○
Get alignment	○	○	○
SET VISIBLE	○	○	○
BEST OBJECT SIZE	○	○	○

注：SET LISTBOX COLUMN WIDTH（オブジェクト、列、ヘッダに適用される）とGet listbox column width（列とヘッダにのみ適用される）を除き、新しい「リストボックス」テーマのすべてのコマンドは、リストボックスオブジェクトに対してのみ適用されます。

リストボックスとランゲージ

オブジェクトメソッド

リストボックスオブジェクトやリストボックスの各列に対し、オブジェクトメソッドを付加することができます。オブジェクトメソッドの呼び出しは、次の順で行われます。

1. それぞれの列のオブジェクトメソッド
2. リストボックスのオブジェクトメソッド

列のオブジェクトメソッドにより、そのヘッダ上で発生したイベントを取得できます。

SET VISIBLEおよびヘッダ

ヘッダで**SET VISIBLE**コマンドを使用すると、このコマンドで指定されたヘッダの枠を超えて、コマンドはすべてのリストボックスオブジェクトのヘッダに使用されます。例えば、**SET VISIBLE**(*;"ヘッダ3";False)という命令の場合、指定したヘッダだけではなく、ヘッダ3が属すリストボックスの全ヘッダを非表示にします。

SelfおよびOn Clicked

Self関数（「ランゲージ」テーマ）は、リストボックスやリストボックスの列のオブジェクトメソッド内で使用することができます。双方の場合ともに、クリックされた場所に応じて、列の変数やヘッダの変数へのポインタが返されます。

Focus object

Focus object関数（「ユーザインタフェース」テーマ）は、フォーカスを取得したリストボックスの列（つまり、配列）へのポインタを返します。4th Dimensionのポインタのメカニズムを利用し、修正された配列の項目番号を調べることができます。例えば、ユーザが5行目の列“col2”を変更した場合は、次のようになります。

```
$Column:=Focus object
      ` $Columnには、col2へのポインタが代入される
$Row:= $Column-> `$Rowは 5
```

EDIT ITEM

EDIT ITEMコマンド（「入力制御」テーマ）を使用して、リストボックスオブジェクトのセルを編集モードに移行することができます。

フォームイベント

プログラムによるリストボックス管理、特にドラッグ＆ドロップや並び替え操作を円滑にするため、新しいフォームイベントが作成されました。詳細については、**Form Event**コマンドの節を参照してください。

並び替えの管理

デフォルトとして、ヘッダがクリックされると、リストボックスは自動的に標準的なカラムの並び替えを行います。標準的な並び替えとは、列の値をアルファベット順に並べ替え、続けてクリックされると昇順／降順を交互に切り替えます。常にすべての列は自動的に同期化されます。

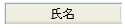
リストボックスの「並び替え可」プロパティの選択を解除すると、ユーザによる標準の並び替えを禁止することができます。

開発者は、**SORT LISTBOX COLUMNS**コマンドを使用するか、または「On header click」と「On after sort」フォームイベント、および4Dの配列管理コマンドを組み合わせることにより、独自の並び替えを設定することができます（**Form Event**コマンドの節を参照）。

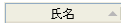
注：列のプロパティ「並び替え可」は、ユーザによる標準の並び替えにのみ影響を与えます。SORT LISTBOX COLUMNSコマンドでは、このプロパティが考慮されません。

列ヘッダに関連付けられた変数の値を使用すると、列の現在の並び替え状況（の読み込み）や並び替え矢印の表示など、追加情報を管理することができます。

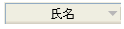
■ 変数を0に設定すると、並び替え矢印は表示されません。



■ 変数を1に設定すると、昇順を表す並び替え矢印が表示されます。



■ 変数を2に設定すると、降順を表す並び替え矢印が表示されます。



選択項目の管理

プログラムからリストボックスの1行または複数の行を選択するために**SELECT LISTBOX ROW**コマンドを使うことができます。

リストボックスオブジェクトに関連付けられた変数を使用して、オブジェクト行の選択項目の取得、設定、保存を行います。

この変数は、4th Dimensionが自動的に作成、保守を行うブールタイプの配列に対応します。この配列のサイズは、リストボックスのサイズにより決定されます。配列には列に関連付けられた最も小さな配列と同じ数の要素が格納されます。

この配列の各要素には、対応する行が選択された場合には“True”が、それ以外の場合は“False”が納められます。4th Dimensionは、ユーザの動作に応じて、この配列の内容を更新します。これとは逆に、配列要素の値を変更して、リストボックス中の選択項目を変えることができます。

その反面、この配列への行の挿入や削除はできず、行のタイプ変更もできません。

注：新しいCount in array関数は、選択された行の数を調べる際に役立ちます。

例題

▼ このメソッドにより、リストボックスの1行めの選択を逆転させます。

```
      ` tBListBoxはリストボックス変数の名前です
If(tBListBox{1} = True)
      tBListBox{1}:= False
Else
      tBListBox{1}:= True
End if
```

INSERT LISTBOX COLUMN

INSERT LISTBOX COLUMN ({*; }オブジェクト; 列位置; 列名; 列変数; ヘッダ名; ヘッダ変数)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
列位置	数値	→	挿入する列の位置
列名	文字列	→	列のオブジェクトの名前
列変数	配列	→	列の変数の名前
ヘッダ名	文字列	→	列ヘッダオブジェクトの名前
ヘッダ変数	整数変数	→	列ヘッダの変数

説明

INSERT LISTBOX COLUMNコマンドは、引数<オブジェクト>および<*>で指定されたリストボックスに列を挿入します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

新しい列は、引数<列位置>を用いて指定された列の直前に挿入されます。引数<列位置>の値が列の合計数よりも大きい場合、最後の列の後ろにカラムが追加されます。

引数<列名>と<列変数>には、挿入される列のオブジェクト名および変数名を渡します。この変数名は、列に表示する内容が格納された配列の名前と一致していなくてはなりません。

引数<ヘッダ名>と<ヘッダ変数>には、挿入される列のヘッダのオブジェクト名および変数を渡します。

注：オブジェクト名は、フォーム内で重複してはいけません。引数<列名>や<ヘッダ名>に渡される名前が既に使用されていないことを必ず確認してください。

重複した名前を指定すると、列は作成されず、エラーが生成されます。

例題

▼ リストボックスの最後に列を追加します。

```
C_LONGINT(HeaderVarName;$Last;$RecNum)
ALL RECORDS([Table1])
$RecNum:=Records in table([Table1])
ARRAY PICTURE(Picture;$RecNum)
$Last:=Get number of listbox columns(*;"listbox1")+1
INSERT LISTBOX COLUMN(*;"listbox1";$Last;"ColumnPicture";Picture;
                                                    "HeaderPicture";HeaderVarName)
```

参照

DELETE LISTBOX COLUMN

DELETE LISTBOX COLUMN

DELETE LISTBOX COLUMN ({*; }オブジェクト; 位置{; 列数})

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
列位置	数値	→	削除する列の番号
列数	数値	→	削除する列の数

説明

DELETE LISTBOX COLUMNコマンドは、引数<オブジェクト>および<*>で指定されたリストボックスから1つ以上の列（表示または非表示の列）を削除します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

任意の引数である<列数>を渡さない場合、コマンドは引数<列位置>で指定された列だけを削除します。

この引数を渡した場合、引数<列数>は、引数<列位置>より（この列を含む）右側にある削除する列の数を示します。

引数<列位置>がリストボックスの列の数よりも大きい場合、コマンドは何も行いません。

参照

Get number of listbox columns、INSERT LISTBOX COLUMN

Get number of listbox columns

Get number of listbox columns ({*; }オブジェクト) → 倍長整数

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>は オブジェクト名（文字列） 省略した場合、<オブジェクト>は 変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、 または変数（“*”を省略した場合）
戻り値	倍長整数	←	列の数

説明

Get number of listbox columns関数は、引数<オブジェクト>および<*>で指定されたリストボックスに存在する列（表示または非表示の列）の合計数を返します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

参照

DELETE LISTBOX COLUMN.

SORT LISTBOX COLUMNS

SORT LISTBOX COLUMNS ({*; }オブジェクト; 列変数; 順序{; 列変数2; 順序2}...{; 列変数X; 順序X)

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
列変数1...X	変数	→	並び替える列の変数
順序1...X	> または <	→	>：昇順に並び替え、または <：降順に並び替え

説明

SORT LISTBOX COLUMNSコマンドは、1つ以上の列の値に基づいて、引数＜オブジェクト＞および＜*＞で指定されたリストボックスの行を並べ替えます。

任意の引数＜*＞を渡すことにより、引数＜オブジェクト＞がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数＜オブジェクト＞が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

＜列変数＞には、列に関連付けられた変数（つまり、配列名）を渡し、この列の値が並び替え条件として使用されます。ピクチャタイプとポインタタイプを除き、任意のタイプの配列データを使用することができます。

＜順序＞には、並び替え順を表わす“>”または“<”記号を渡します。＜順序＞に“～より大きい”を示す記号（>）を指定すると、並び替えは昇順になります。＜順序＞に“～より小さい”を示す記号（<）を指定すると、並び替えは降順になります。

さらに、マルチレベルソートを定義することができます。これを行うには、必要な数のペア（列変数;順序）を渡します。並び替えレベルは、コマンド内の引数の位置によって決まります。

リストボックス操作の原則に従い、それぞれの列は同期化されます。つまり、列の並び順は、自動的にそのオブジェクトの他のすべての列に受け継がれます。

SET LISTBOX COLUMN WIDTH

SET LISTBOX COLUMN WIDTH ({*; }オブジェクト; 幅)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>は オブジェクト名（文字列） 省略した場合、<オブジェクト>は 変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、 または変数（“*”を省略した場合）
幅	整数	→	列の幅（ピクセル単位）

説明

SET LISTBOX COLUMN WIDTHコマンドを使用し、引数<オブジェクト>および<*>で指定されたオブジェクト（リストボックス、列、またはヘッダ）の任意の列の幅、またはすべての列の幅をプログラムから変更することができます。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

引数<幅>には、<オブジェクト>の新しい幅（ピクセル単位）を渡します。

- 指定された<オブジェクト>がリストボックスオブジェクトである場合、リストボックスのすべての列サイズが変更されます。
- 指定された<オブジェクト>が列または列ヘッダである場合、その列のサイズだけが変更されます。

参照

Get listbox column width.

Get listbox column width

Get listbox column width ({*; }オブジェクト) → 整数

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
戻り値	整数	←	列の幅（ピクセル単位）

説明

Get listbox column width関数は、引数<オブジェクト>および<*>で指定された列の幅（ピクセル単位）を返します。引数<オブジェクト>には、リストボックスの列や列ヘッダを自由に渡すことができます。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

参照

SET LISTBOX COLUMN WIDTH

MOVED LISTBOX COLUMN NUMBER

MOVED LISTBOX COLUMN NUMBER ({*; }オブジェクト; 旧位置; 新位置)

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞は オブジェクト名（文字列） 省略した場合、＜オブジェクト＞は 変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、 または変数（“*”を省略した場合）
旧位置	数値	←	移動された列の以前の位置
新位置	数値	←	移動された列の新しい位置

説明

MOVED LISTBOX COLUMN NUMBERコマンドは、＜旧位置＞と＜新位置＞に2つの数値を返し、引数＜オブジェクト＞および＜*＞で指定されたリストボックス内で移動された列の以前の位置と新しい位置をそれぞれ示します。

任意の引数＜*＞を渡すことにより、引数＜オブジェクト＞がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数＜オブジェクト＞が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

このコマンドは、「**On column moved**」フォームイベントと一緒に使用しなくてはなりません（**Form Event**の節を参照）。

注：このコマンドは、非表示の列を考慮します。

参照

Form event、MOVED LISTBOX ROW NUMBER

SELECT LISTBOX ROW

SELECT LISTBOX ROW ({*; }オブジェクト; 位置{; アクション})

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
位置	倍長整数	→	選択する行の番号
アクション	倍長整数	→	選択に関する動作

説明

SELECT LISTBOX ROWコマンドは、引数<オブジェクト>および<*>で指定されたリストボックス内において、<位置>に渡した番号の行を選択します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

任意の引数<アクション>を指定すると、その値を用いて、行のセレクションがリストボックス内に既に存在している場合に実行する選択アクションが決定されます。この引数には、任意の値または次の定数のいずれかを渡すことができます（「リストボックス」テーマ）。

■ Replace listbox selection(0)：選択した行は新しいセレクションとなり、既存のものと置き換えられます。このコマンドは、ユーザが行をクリックした場合と同じ結果になります。

デフォルトでは、この動作が実行されます（引数<アクション>を省略した場合）。

■ Add to listbox selection(1)：選択した行は既存のセレクションに追加されます。指定した行が既存のセレクションに属している場合、コマンドは何も行いません。

■ Remove from listbox selection(2)：選択した行は既存のセレクションから削除されます。指定した行が既存のセレクションに属さない場合、コマンドは何も行いません。

参照

DELETE LISTBOX ROW、INSERT LISTBOX ROW

INSERT LISTBOX ROW

INSERT LISTBOX ROW ({*; }オブジェクト; 位置)

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
位置	倍長整数	→	行を挿入する位置

説明

INSERT LISTBOX ROWコマンドは、引数＜オブジェクト＞および＜*＞で指定されたリストボックスに新しい行を挿入します。

任意の引数＜*＞を渡すことにより、引数＜オブジェクト＞がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数＜オブジェクト＞が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

行は、引数＜位置＞で指定された場所に挿入されます。配列のタイプや表示状態とは関係なく、リストボックスの列に使用されるすべての配列に対し、この場所に新しい行が自動的に追加されます。

＜位置＞の値がリストボックス内の合計行数よりも大きい場合、行は各配列の最後に追加されます。

参照

DELETE LISTBOX ROW

DELETE LISTBOX ROW

DELETE LISTBOX ROW ({*; }オブジェクト; 位置)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
位置	倍長整数	→	削除する行の位置

説明

DELETE LISTBOX ROWコマンドは、引数<オブジェクト>および<*>で指定されたリストボックス（表示または非表示）から、<位置>に指定した番号の行を削除します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

コマンドの実行後、リストボックス内の要素の選択はすべて解除される点に留意してください。

<位置>で指定した行は、リストボックスの列で使用されるすべての配列から自動的に削除されます。

<位置>の値がリストボックス内の合計行数よりも大きい場合、コマンドは何も行いません。

参照

Get number of listbox rows、INSERT LISTBOX ROW

Get number of listbox rows

Get number of listbox rows ({*; }オブジェクト) → 倍長整数

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>は オブジェクト名（文字列） 省略した場合、<オブジェクト>は 変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、 または変数（“*”を省略した場合）
戻り値	倍長整数	←	行数

説明

Get number of listbox rows関数は、引数<オブジェクト>および<*>で指定されたリストボックスの行の数を返します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

注：リストボックスのカラムに連携している配列がすべて同じサイズではない場合、このコマンドは最小の配列に関連づけられている項目の数だけをリストボックスに表示し、返します。

参照

DELETE LISTBOX ROW、INSERT LISTBOX ROW

SET LISTBOX ROWS HEIGHT

SET LISTBOX ROWS HEIGHT({*; }オブジェクト; 高さ)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
高さ	整数	→	行の高さ（ピクセル単位）

説明

SET LISTBOX ROWS HEIGHTコマンドを使用すると、引数<オブジェクト>および<*>で指定されたリストボックス内の行の高さをプログラムにより変更することができます。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

参照

Get listbox rows height

Get listbox rows height

Get listbox rows height ({*; }オブジェクト) → 整数

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
戻り値	整数	←	行の高さ（ピクセル単位）

説明

Get listbox rows height関数は、引数<オブジェクト>および<*>で指定されたリストボックスのカレント行の高さ（ピクセル単位）を返します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

参照

SET LISTBOX ROWS HEIGHT

MOVED LISTBOX ROW NUMBER

MOVED LISTBOX ROW NUMBER ({*; }オブジェクト; 旧位置; 新位置)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
旧位置	数値	←	移動された行の以前の位置
新位置	数値	←	移動された行の新しい位置

説明

MOVED LISTBOX ROW NUMBERコマンドは、<旧位置>と<新位置>に2つの数値を返し、引数<オブジェクト>および<*>で指定されたリストボックス内で移動された行の以前の位置と新しい位置をそれぞれ示します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

このコマンドは、「**On row moved**」フォームイベントと一緒に使用しなくてはなりません（前述の「リストボックスに関連するフォームイベント」の節を参照）。

参照

Form event、MOVED LISTBOX COLUMN NUMBER

GET LISTBOX ARRAYS

GET LISTBOX ARRAYS ({*; }オブジェクト; 列名配列; ヘッダ名配列; 列変数配列; ヘッダ変数配列; 表示配列; スタイル配列)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
列名配列	文字列配列	←	列のオブジェクトの名前
ヘッダ名配列	文字列配列	←	ヘッダオブジェクトの名前
列変数配列	ポインタ配列	←	列の変数へのポインタ
ヘッダ変数配列	ポインタ配列	←	ヘッダの変数へのポインタ
表示配列	ブール配列	←	各列の表示状態
スタイル配列	ポインタ配列	←	スタイルおよびカラー配列へのポインタ

説明

GET LISTBOX ARRAYSコマンドは一連の同期化された配列を返し、引数<オブジェクト>および<*>で指定されたリストボックスの各列（表示または非表示の列）に関する情報を提供します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

コマンドの実行後、各引数には次の値が代入されます。

- 配列<列名配列>には、リストボックス内の各列のオブジェクト名一覧が代入されます。
- 配列<ヘッダ名配列>には、リストボックス内の各列ヘッダのオブジェクト名一覧が代入されます。
- 配列<列変数配列>には、リストボックス内の各列に関連付けられた変数（配列）へのポインタが代入されます。

■ 配列＜ヘッダ変数配列＞には、リストボックス内の各列ヘッダに関連付けられた変数へのポインタが代入されます。

■ 配列＜表示配列＞には各列に関するブール値が代入され、リストボックス内の列が表示（True）または非表示（False）のいずれであるかを示します。

■ 配列＜スタイル配列＞には、3つの配列をそれぞれ指す3つのポインタが代入されます。これら3つの配列は、リストボックス内の各行に対して特定のスタイルやフォントカラー、背景色を適用するために使用されます。これらの配列は、「デザイン」モードの「プロパティリスト」上の「リストボックス」テーマに関連しています。リストボックスに対する配列が指定されていない場合、＜スタイル配列＞内の対応する項目にはNilポインタが代入されます。

参照

Get listbox information

Get listbox information

Get listbox information ({*; }オブジェクト; 情報タイプ) → 倍長整数

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
情報タイプ	倍長整数	→	取得する情報タイプ
戻り値	倍長整数	←	現在の値

説明

Get listbox information関数は、引数<オブジェクト>および<*>で指定されたリストボックスオブジェクトの各種情報を返します。これは、現在の表示状態やヘッダサイズ、スクロールバーに関する情報です。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

引数<情報タイプ>には、取得しようとする情報のタイプを示す値を渡します。この引数には任意の値または「リストボックス」テーマの次の定数のいずれかを使用することができます。

定数	タイプ	値	戻り値
Display listbox header	倍長整数	0	0=非表示、1=表示
Listbox header height	倍長整数	1	高さ（ピクセル単位）
Display listbox hor scrollbar	倍長整数	2	0=非表示、1=表示
Listbox hor scrollbar height	倍長整数	3	高さ（ピクセル単位）
Display listbox ver scrollbar	倍長整数	4	0=非表示、1=表示
Listbox ver scrollbar width	倍長整数	5	幅（ピクセル単位）

このコマンドは、フォーム上のリストボックスエリアの実サイズを計算する際に役立ちます。

参照

SHOW LISTBOX GRID、SHOW LISTBOX SCROLLBAR

SHOW LISTBOX GRID

SHOW LISTBOX GRID ({*; }オブジェクト; 横線; 縦線)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
横線	ブール	→	True=表示、False=非表示
縦線	ブール	→	True=表示、False=非表示

説明

SHOW LISTBOX GRIDコマンドを使用し、水平や垂直グリッドラインを表示、または非表示に設定することができます。これらのグリッドラインは、引数<オブジェクト>および<*>で指定されたリストボックスオブジェクト上のグリッドを構成します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

<横線>と<縦線>には、対応するグリッドラインを表示するか（True）、表示しないか（False）を示すブール値を渡します。デフォルトでは、グリッドが表示されます。

参照

Get listbox information、SET LISTBOX GRID COLOR

SET LISTBOX GRID COLOR

SET LISTBOX GRID COLOR{*; }オブジェクト; カラー; 横線; 縦線)

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
カラー	数値	→	RGBカラーの値
横線	ブール	→	水平グリッドラインに使用されるカラー
縦線	ブール	→	垂直グリッドラインに使用されるカラー

説明

SET LISTBOX GRID COLORコマンドを使用し、引数＜オブジェクト＞および＜*＞で指定されたリストボックスオブジェクト上のグリッドの色を変更することができます。

任意の引数＜*＞を渡すことにより、引数＜オブジェクト＞がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数＜オブジェクト＞が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

＜カラー＞には、RGBカラーの値を渡します。RGBカラーに関する詳細は、**SET RGB COLORS**コマンドの説明を参照してください。

＜横線＞と＜縦線＞を使用し、カラーを適用するグリッドラインを指定することができます。

■ ＜横線＞に“True”を渡した場合、水平グリッドラインにカラーが適用されます。“False”を渡すと、カラーは変更されません。

■ ＜縦線＞に“True”を渡した場合、垂直グリッドラインにカラーが適用されます。“False”を渡すと、カラーは変更されません。

参照

SET RGB COLORS、SHOW LISTBOX GRID

SHOW LISTBOX SCROLLBAR

SHOW LISTBOX SCROLLBAR({*; }オブジェクト; 横; 縦)

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>は変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、または変数（“*”を省略した場合）
横	ブール	→	True=表示、False=非表示
縦	ブール	→	True=表示、False=非表示

説明

SHOW LISTBOX SCROLLBARコマンドを使用すると、引数<オブジェクト>および<*>で指定されたリストボックスオブジェクトにおいて、横スクロールバーや縦スクロールバーを表示、または非表示に設定することができます。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、引数<オブジェクト>が変数であることを示します。この場合、文字列ではなくフィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細は「オブジェクトプロパティコマンド」の章を参照してください。

<横>と<縦>には、対応するスクロールバーを表示するか（True）、表示しないか（False）を示すブール値を渡します。デフォルトでは、スクロールバーが表示されます。

参照

Get listbox information、SHOW LISTBOX GRID

Abs

Abs(数値) → 数値

引数	タイプ		説明
数値	数値	→	絶対値をとる数値
戻り値	数値	←	数値の絶対値

説明

Abs関数は、＜数値＞の絶対値（符号なしの正の値）を返します。＜数値＞が負の場合、正の数を返します。＜数値＞が正の数の場合は、値は変わりません。

例題

▼ 以下の例は、-10.3の正の値である10.3を返します。

```
vベクトル:=Abs(-10.3)      ` 変数“vベクトル”に10.3を代入する
```

Int

Int(数値) → 数値

引数	タイプ		説明
数値	数値	→	整数部を取り出す数値
戻り値	数値	←	数値の整数部

説明

Int関数は、＜数値＞の整数部を取り出して返します。＜数値＞が負の場合は、ゼロから遠い数値に丸めます。

例題

▼ 以下の例は、負の数値と正の数値に対して**Int**関数がどのように機能するかを示しています。数値の少数点以下の部分が取り除かれている点に注意してください。

```
x:=Int(123.4) ` xに123を代入
y:=Int(-123.4) ` yに-124を代入
```

参照

Dec

Dec

Dec(数値) → 数値

引数	タイプ		説明
数値	数値	→	小数点以下の数をとる数値
戻り値	数値	←	小数部

説明

Dec関数は、＜数値＞の小数部（端数）を返します。返す値は、常に正の数またはゼロになります。

例題

▼以下の例は、キログラム単位で実数になっている重量を、整数のキログラムとグラムに変換します。重量が7.311の場合は、7キログラムと311グラムなります。

```
キログラム:=Int(重量)        ` キログラムの部分7を得る
グラム:=Dec(重量) * 1000     ` グラムの部分311を得る
```

参照

Int

Round

Round(数値 ; 桁位置) → 数値

引数	タイプ		説明
数値	数値	→	丸める数値
桁位置	数値	→	丸める少数部の桁
戻り値	数値	←	＜桁位置＞に指定した桁で丸められた数値

説明

Round関数は、＜数値＞を指定された＜桁位置＞で四捨五入します。

＜桁位置＞が正の数の場合は、＜数値＞の小数部を丸め、＜桁位置＞が負の場合には、整数部（小数点より左側）を丸めます。

＜桁位置＞で指定した桁位置の数字を四捨五入します。＜桁位置＞が0の場合は、小数第1位を、-1の場合には、整数部の1の位を四捨五入します。

例題

▼ 下記は、さまざまな引数を使用して**Round**関数の機能を示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される値についての説明です。

```
v結果:=Round(16.857 ; 2)  ` v結果に16.86を代入
v結果:=Round(32345.67 ; -3)  ` v結果に32000を代入
v結果:=Round(29.8725 ; 3) ` v結果に29.873を代入
v結果:=Round(-1.5 ; 0)  ` v結果に-2を代入
```

参照

Trunc

Trunc

Trunc(数値 ; 桁位置) → 数値

引数	タイプ		説明
数値	数値	→	切り捨てる数値
桁位置	数値	→	切り捨てる少数部の桁
戻り値	数値	←	＜桁位置＞で少数部が切り捨てられた数値

説明

Trunc関数は、指定された＜桁位置＞の小数部を切り捨てた＜数値＞を返します。Trunc関数は、常に元の値よりも小さい値を返します。

＜桁位置＞が正の数の場合は、＜数値＞の小数部を切り捨て、＜桁位置＞が負の場合には、整数部（小数点の左側）を切り捨てます。

例題

▼ さまざまな引数を使用した**Trunc**関数の機能を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される値についての説明です。

```
v結果:=Trunc(216.897 ; 1)  ` v結果に216.8を代入
v結果:=Trunc(216.897 ; -1) ` v結果に210を代入
v結果:=Trunc(-216.897 ; 1) ` v結果に-216.9を代入
v結果:=Trunc(-216.897 ; -1)` v結果に-220を代入
```

参照

Round

Random

Random → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	乱数

説明

Random関数は、0から32,767までの範囲の整数（乱数）を返します。

整数の範囲は、以下のように記述して変えることができます。

```
Random %(最大-最小+1) +最小
```

最小は範囲の最小値で、最大は最大値です。

例題

▼ 以下の例は、10から30までの範囲の整数の乱数を変数“v結果”に代入します。

```
v結果:=(Random % 21) +10
```

Mod

Mod(数値1 ; 数値2) → 数値

引数	タイプ		説明
数値1	数値	→	割り算される数値
数値2	数値	→	割り算する数値
戻り値	数値	←	余り

説明

Mod関数は、＜数値2＞で＜数値1＞を割り算し、その余りの整数を返します。

注：

・Mod関数は、整数、倍長整数、実数を受け入れます。しかし、＜数値1＞または＜数値2＞が実数の場合は、Mod関数はまずその値を丸めてから、計算を実行します。

・サイズの大きな実数（2^31を超える）を用いてMod関数を使用する場合は注意が必要です。この場合、処理が標準的なプロセッサの計算能力の限界に達してしまう可能性があります。

また、余りを計算するためにモジューロ演算子（%）を使用することができます（第11章の「数値演算子」の節を参照）。

警告：モジューロ演算子（%）は整数、または倍長整数式を用いて有効な結果を返します。実数値のモジューロを計算するには、Mod関数を使用しなければなりません。

例題

▼ 以下の例は、**Mod**関数が異なる引数でどのように機能するかを紹介しています。各行は変数“v結果”に値を代入し、コメント行にその結果を記述しています。

```
v結果:=Mod(3 ; 2)   ` 変数“v結果”は1を得る
v結果:=Mod(4 ; 2)   ` 変数“v結果”は0を得る
v結果:=Mod(3.5 ; 2) ` 変数“v結果”は0を得る
```

参照

数値演算子

Square root

Square root(数値) → 数値

引数	タイプ		説明
数値	数値	→	平方根を求める数値
戻り値	数値	←	平方根の値

説明

Square root関数は、＜数値＞の平方根を返します。

例題

▼ 以下の行は、値1.414213562373を「$SquareRoot」変数に割り当てます。

```
$SquareRoot:=Square root(2)
```

▼ 以下のメソッドは右側の三角形の斜辺を返します。この三角形の2つの斜辺以外の辺は引数として渡されます。

```
`「Hypotenuse」メソッド
` Hypotenuse(実数 ; 実数) ->実数
` Hypotenuse(辺A ; 辺B) -> 斜辺
C_REAL($0 ; $1 ; $2)
$0 := Square root(($1^2)+($2^2))
```

例えば、斜辺（4;3）は5を返します。

参照

数値演算子

Log

Log(数値) → 数値

引数	タイプ		説明
数値	数値	→	自然対数を取る数値
戻り値	数値	←	自然対数

説明

Log関数は、＜数値＞の自然対数を返します。**Log**関数は、**Exp**関数の逆の演算を実行する関数です。

注：4Dでは定義済み定数e number（2.71828...）が提供されています。

例題

以下の行は1を表示します。

```
ALERT(String(Log(Exp(1))))
```

参照

Exp

Exp

Exp(数値) → 数値

引数	タイプ		説明
数値	数値	→	評価する数値
戻り値	数値	←	＜数値＞乗の自然対数の底

説明

Exp関数は、自然対数の底（e=2.71828182845904524）の＜数値＞乗の値を計算します。**Exp**関数は、**Log**関数の逆の関数です。

注：4Dでは定義済み定数e number（2.71828...）が提供されています。

例題

▼ 以下の例は、eの2乗を変数“v”に代入します。

```
v:=Exp(2)
```

参照

Log

Sin

Sin(数値) → 数値

引数	タイプ		説明
数値	数値	→	サイン（sine）で返す値（単位...ラジアン）
戻り値	数値	←	サイン値

説明

Sin関数は、＜数値＞のサイン（Sine、正弦）を返します。＜数値＞はラジアンで指定します。

注：4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値（3.14159...）を返し、Degreeはラジアン（0.01745...）で表わされた1度（ディグリ）を、Radianは度数（57.29577...）で表わされた1ラジアンを返します。

参照

Arctan、Cos、Tan

Cos

Cos(数値) → 数値

引数	タイプ		説明
数値	数値	→	コサイン（cosine）で返す値（単位...ラジアン）
戻り値	数値	←	コサイン値

説明

Cos関数は、＜数値＞のコサイン（Cosine、余弦）を返します。＜数値＞はラジアンで指定します。

注意：4Dでは Pi、Degree、Radian.という定数があらかじめ定義されています。Piはパイの数値（3.14159...）を返し、Degreeはラジアン（0.01745...）で表わされた1度（ディグリ）を、Radianは度数（57.29577...）で表わされた1ラジアンを返します。

参照

Arctan、Sin、Tan

Tan

Tan(数値) → 数値

引数	タイプ		説明
数値	数値	→	タンジェント（Tangent）で返す値（単位...ラジアン）
戻り値	数値	←	タンジェント値

説明

Tan関数は、＜数値＞のタンジェント（Tangent、正接）を返します。＜数値＞はラジアンで指定します。

注意：4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値（3.14159...）を返し、Degreeはラジアン（0.01745...）で表わされた1度（ディグリ）を、Radianは度数（57.29577...）で表わされた1ラジアンを返します。

参照

Arctan、Cos、Sin

Arctan

Arctan(数値) → 数値

引数	タイプ		説明
数値	数値	→	ラジアンで返すタンジェントの値
戻り値	数値	←	アークタンジェント値

説明

Arctan関数は、＜数値＞のアークタンジェント（Arctangent、逆正接）をラジアンで返します。＜数値＞はタンジェントです。

注意：4Dでは Pi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値（3.14159...）を返し、Degreeはラジアン（0.01745...）で表わされた1度（ディグリ）を、Radianは度数（57.29577...）で表わされた1ラジアンを返します。

例題

▼ 以下の例は、パイの値を表示します。

```
ALERT("パイの値：" + String(Arctan(1)*4))
```

参照

Cos、Sin、Tan

SET REAL COMPARISON LEVEL

SET REAL COMPARISON LEVEL(イプシロン値)

引数	タイプ		説明
イプシロン値	数値	→	実数の同等性の比較に使用するイプシロン値

説明

SET REAL COMPARISON LEVELコマンドは、4th Dimensionが実数と数式の同等性を比較するために使用するイプシロン値を設定します。

コンピュータは常に実数の計算を概算で実行するため、実数の同等性をテストする時には、この概算があることを考慮する必要があります。4th Dimensionは、実数を比較する時に2つの実数の差が一定の値より大きいかどうかをテストすることによって、概算を確認します。この値はイプシロン値と呼ばれ、以下のような機能があります。

2つの実数aとbがある時、Abs（a-b）がイプシロン値より大きい場合には、これら2つの実数は等しくないとみなされます。それ以外の場合には、これらの実数は等しいとみなされます。

デフォルトでは、4th Dimensionはイプシロン値を10の-6乗（10^-6）に設定しています。

イプシロン値は、常に正数を指定してください。

例

■ 0.00001=0.00002は、Falseを戻します。誤差0.00001が10^-6より大きいためです。

■ 0.000001=0.000002は、Trueを戻します。誤差0.000001が10^-6より大きくないためです。

■ 0.000001=0.000003は、Falseを戻します。誤差0.000002が10^-6より大きいためです。

SET REAL COMPARISON LEVELコマンドを使用すると、必要に応じてイプシロン値を増やしたり減らすことができます。

注：10^-6より小さい値が納められた数値タイプのインデックス付きフィールドに対してクエリや並び替えを実行したい場合、インデックスの構築の前に必ずSET REAL COMPARISON LEVELコマンドを実行してください。

警告：通常、デフォルトのイプシロン値を変更するためにこのコマンドを使用する必要はありません。

重要：イプシロン値を変更しても、実数の同等性の比較に影響があるだけで、他の実数計算や実数値の表示には影響はありません。

参照

比較演算子

実数の表示について

注：複数のプラットフォームによる開発に関わっていない場合には、この項は参照しなくても構いません。

コンピュータでは、浮動小数点計算は、数学的知識ではなく技法の1つにすぎません。例えば、3分の1（1／3）は、小数点の後に数字3が無限に続く形で表記できると学習したことでしょう。一方、コンピュータはこのことを認識しないため、式を計算しなければなりません。同様に、3×1／3が1に等しいことは概念的に理解していますが、コンピュータはその解を得るために式を計算します。使用するコンピュータの種類によっては、1／3は小数点の後に数字の3が限られた桁数だけ続く有限小数ととして計算されます。この桁数は、そのコンピュータの精度と呼ばれます。

68KのMacintoshでは、精度数は19で、1／3は有効数字19桁で計算されます。WindowsやPower Macintoshでは、この数は15になり、1／3は有効数字15桁で表示されます。式1／3を4th Dimensionの「デバッグ」ウインドウで表示すると、68KのMacintoshでは0.3333333333333333333と表示され、WindowsやPower Macintoshでは0.3333333333333333148のように表示されます。WindowsおよびPower Macintoshでの精度数は68KのMacintoshでの精度数より小さいため、最後の3桁が異なっているという点に注意が必要です。それでも、（1／3）*3と表示すると、いずれのコンピュータでも解は1になります。

浮動小数点計算で、例えば、裏庭の広さのヘクタールを算出する場合には、小数点以下の桁数等は関係ないため、無視できることでしょう。一方、税務署の申告書を作成している場合には、時にはコンピュータが小数点以下も正確に計算しているかどうかが気になるところでしょう。それでも、小数点以下19桁または15桁まで計算していれば、何十億円もの歳入があったとしても、十分に正確な結果が得られます。

68KのMacintoshとWindowsやPower Macintoshでは、1／3の値が異なるのはなぜでしょうか?

68KのMacintoshでは、オペレーティングシステム（OS）は実数を10バイト（80ビット）で格納しているのに対し、WindowsやPower Macintoshでは、実数を8バイト（64ビット）で格納しています。この結果、68KのMacintoshでは実数の有効数字が19桁で、WindowsやPower Macintoshではそれが15桁になります。

それでも、式（1／3）*3によりいずれのコンピュータでも1が戻るのはなぜでしょうか?

コンピュータは近似的に計算を実行できるだけにすぎません。したがって、数字を比較したり計算する時には、コンピュータは実数を数学的なオブジェクトとしてではなく、近似値として処理します。この例では、0.3333...を3倍すると0.9999...になります。0.9999...と1の差は非常に小さいため、コンピュータはこの結果を1と等しいとみなし、1を戻します。この問題については、**SET REAL COMPARISON LEVEL**コマンドについての説明を参照してください。

実数には2つの側面があるため、以下の2つの点について区別しておかなければなりません。

■ 実数がどのように計算され、比較されるか

■ 実数が画面やプリンタにどのように出力されるか

もともと4th Dimensionは、68KのMacintoshで提供されていた標準の10バイトデータ型を使用して実数を処理していました。その結果、ディスクのデータファイルに格納される実数は、この形式を使用して保存されます。68KのMacintosh、Windows、Power Macintoshのそれぞれのバージョンの4th Dimensionの間で互換性を保つために、4th Dimensionのデータファイルでは引き続き10バイトデータ型を使用して実数を保存しています。浮動小数点計算はWindowsまたはPower Macintoshで8バイト形式を使用して実行されるため、4th Dimensionは値を10バイト形式から8バイト形式に変換したり、8バイト形式から10バイト形式に変換しています。このため、68KのMacintoshで保存された実数が含まれるレコードをWindowsやPower Macintoshにロードする場合には、精度の一部が失われる（有効数字が19桁から15桁になる）可能性があります。一方、WindowsやPower Macintoshで保存された実数が含まれるレコードを68KのMacintoshにロードした場合には、精度が失われることはありません。基本的には、68KのMacintoshのデータベースを使用した場合でも、WindowsやPower Macintoshのデータベースを使用した場合でも、浮動小数点計算では、有効数字19桁ではなく15桁までを信頼するようにします

Customizer Plusユーティリティを使用すると、68KのMacintoshまたはWindowsやPower Macintoshで実数の表示を簡略化する場合には、省略する桁数を設定することができます。デフォルトの設定では、68KのMacintoshでは省略する桁数はゼロ、WindowsやPower Macintoshでは5桁です。

Euro converter

Euro converter(値;元の通貨;変換通貨)　→　実数

引数	タイプ		説明
値	実数	→	値を変換
元の通貨	文字列	→	表示されている値の通貨コード
変換通貨	文字列	→	変換したい値の通貨コード
戻り値	実数	←	変換した値

説明

Euro converter関数は"ユーロ"に所属するユーロ通貨の元と先の異なった通貨の値を変換します。

変換できるものは

・ユーロから国際通貨

・国際通貨からユーロ

・他の国際通貨から国際通貨。この場合、ヨーロッパ組織の特質として、変換はユーロの仲介によって計算されます。

例えば、ベルギーフランをドイツマルクに変換すると、4Dは以下の計算を実行します。

ベルギーフラン－＞ユーロ－＞ドイツマルク
最初の引数を変換する値とします。
2番目の引数は表示されている値の通貨コードを示します。
3番目の引数は変換したい値の通貨コードを示します。

特に通貨コードを、4th Dimensionは以下のあらかじめ定義された定数を提供します。

定数	タイプ	値
Austrian Schilling	文字列	ATS
Belgian Franc	文字列	BEF
Deutsche mark	文字列	DEM
Euro	文字列	EUR
Finnish Markka	文字列	FIM
French Franc	文字列	FRF
Greek drachma	文字列	GRD
Irish Pound	文字列	IEP
Italian Lire	文字列	ITL
Luxembourg Franc	文字列	LUF
Netherlands Guilder	文字列	NLG

Portuguese Escudo	文字列	PTE
Spanish Peseta	文字列	ESP

必要な場合、4th Dimensionは変換した結果を小数点2位をもって自動的に四捨五入します。例外としてイタリアリラ、ベルギーフラン、ルクセンブルグフラン、スペインペセタ等の変換に4Dは小数点0位を四捨五入します（結果は整数の数値です）。

修正されたユーロと12の参加メンバー国の通貨の変換レートです。

通貨	1ユーロの値
Austrian Schilling	13.7603
Belgian Franc	40.3399
Deutsche mark	1.95583
Finnish Markka	5.94573
French Franc	6.55957
Greek drachma	340.750
Irish Pound	0.787564
Italian Lire	1936.27
Luxembourg Franc	40.3399
Netherlands Guilder	2.20371
Portuguese Escudo	200.482
Spanish Peseta	166.386

例題

以下の例題はこの関数を使用して変換したものです。

```
$value:=10000 `フランスフランで値を表示
`ユーロの値に変換
$InEuros:=Euro converter($value;French Franc; Euro)
`イタリアリラの値へ変換
$InLires:=Euro converter($value;French Franc; Italian Lira)
```

メニューの管理

用語について

メニューに関する説明では、メニューの行について述べる部分で「メニュー」と「メニュー項目」という用語が同等の意味で使用されています。

メニューバー

メニューバーは、名前ではなく番号で識別されます。最初のメニューバーは、メニューバー#1です。これは、デフォルトのメニューバーでもあります。アプリケーションを開いた時点でメニューバー#1でないメニューバーを表示したい場合には、On Startupデータベースメソッドで**MENU BAR**コマンドを使用して強制的にメニューバーを切り替る必要があります。

各メニューには、それに付属する1つのプロジェクトメソッドがあります。メニューにメソッドを割り当てていないと、そのメニューを選択したときに4th Dimensionは「カスタム」モードから抜けて、「ユーザ」モードへ戻ります。ユーザが4th Dimensionの「カスタム」モードバージョンを使って作業をしていたり、「ユーザ」モードへのアクセスが行えない場合には、終了してデスクトップへ戻ります。

すべてのメニューバーは、前もってWindows版では「ファイル」メニュー、「編集」メニュー、「ヘルプ」メニューを、Macintosh版では「アップル」メニュー、「ファイル」メニュー、「編集」メニューを持っています。

■ 「ファイル」メニューには、「終了」項目しかありません。「終了」項目にメソッドが割り当てられていないことに注目してください。つまり、これが4th Dimensionに「カスタム」モードを終了させる方法です。「ファイル」メニューは、名前を変えたり、メニューを追加することもできますし、そのままにしておくことも可能です。「ファイル」メニューの名前を変更すると、「編集」メニューの左に表示されなくなります。メニュー項目「終了」は、「ファイル」メニューの一番最後のメニュー項目にすることをお勧めします。

■「編集」メニューには、標準的な編集コマンドがあります。この「編集」メニューは「メニュー」エディタに表示されない上、修正することもできません。

■Windows版では、上記の他に「ヘルプ」メニューがあります。この「ヘルプ」メニューの中に「4th Dimension®について...」メニューとそのアプリケーションで利用できる「Windowsヘルプ」ファイルが含まれています。なお、「ヘルプ」メニューは「メニュー」エディタに表示されない上、修正することもできません。ただし、「4th Dimension®について...」メニューは**SET ABOUT**コマンドを用いて修正することができます。

■Macintosh版では、上記の他に「アップル」メニューがあります。この「アップル」メニューの中に「4th Dimension®について...」メニューと、システムフォルダの「アップルメニュー」フォルダに収められたアプリケーションが含まれています。なお、「アップル」メニューは「メニュー」エディタに表示されない上、修正することもできません。ただし、「4th Dimension®について...」メニューは**SET ABOUT**コマンドを用いて修正することができます。

メニュー番号とメニュー項目番号

メニューバーと同様に、メニュー項目にも番号が付けられています。ただし、「編集」メニュー、「ヘルプ」メニュー、「アップル」メニューは変更できないため番号は付いていません。したがって、「ファイル」メニューを1として左から右に（2、3、4...という具合に）番号が付けられています。メニュー番号は、例えば**Menu selected**関数を使って作業を行っている場合等では重要になってきます。メニューがフォームに関連付けられていると、このメニューの番号付けは変わってきます。最初に追加されたメニューは、2049番で始まります。追加されたメニューを参照するには、通常のメニュー番号に2048を追加します。

各メニュー内のメニュー項目は、項目の1番上から下に向かって連続番号が付けられています。1番上のメニューがアイテム1です。

関連メニューバー

フォームに付随するメニューバーは、「フォーム」エディタを表示しているときに表示される「フォーム」メニューの「メニューバー連結」を使用してフォームに関連付けます。本書において、このようなメニューバーは“フォームメニューバー”と呼ばれます。

フォームメニューバーのメニューは、フォームが表示された時点でカレントメニューに付加されます。入力フォームについては、「ユーザ」モードと「カスタム」モードの両方でこのメニューが追加されます。出力フォームについては、「カスタム」モードでこのメニューが追加されます。

フォームメニューバーは、メニューバー番号で指定します。カレントフォームに付随して表示されたメニューバーの番号が、そのフォームに追加されたメニューバーの番号と同じである場合には、メニューバーは追加されません。

フォームメニューバーに対して負の数を指定した場合には、4th Dimensionはその絶対値を使用します。例えば、メニューバーとして-3を指定した場合には、メニューバー#3が使用されます。ただし、フォームメニューバーを負の数で指定した場合には、メニューバー（スプラッシュスクリーンやフォーム）のすべてのメニューは、割り当てられたメソッドを実行します。

フォームメニューバーに対して正の数を指定した場合には、追加したメニューバーからメニューを選択しても割り当てられたメソッドを実行することはできません。その代わりに、**On Menu Selected**イベントがフォームメソッドに送られるため、**Menu Selected**関数を使い、どのメニューが選択されたかを調べることができます。

プログラムからメニュー項目を修正する

4th Dimensionには、下記のコマンドが用意されており、現在表示されているメニューバー、あるいはプロセスにインストールされたメニューバーのメニューに対して、メニュー項目の追加、削除、挿入、修正を実行することができます。

- **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**

これらのコマンドのスコープは、メニューバーの使用中です。再度**MENU BAR**コマンドを呼び出すと即座に、メニューおよびメニュー項目はすべて「デザイン」モードの「メニューバー」エディタで定義された状態に戻ります。

これらのコマンドにはメニューおよびメニュー項目番号が必要です。

前述したように、メニューは左から右へ順に1からNという番号が付けられています。例えば、「ファイル」メニューは常に一番最初のメニューになります。Windowsでは、「編集」および「ヘルプ」メニューにはこの番号が付けられません。また、Macintoshでは、「アップル」および「編集」メニューに番号付けが行われません。

Count menus関数はこれらのメニューを考慮に入れていない点に注意してください。例えば、「ファイル」、「顧客」、「請求書」というメニューで構成されるメニューバーの場合、**Count menus**関数は3を返し、4Dにより管理されるシステムメニューは無視されます。

メニュー項目は、区切り線も含め、上から下へ順に1からNという番号が付けられています。

フォームに関連付けたメニューバーを用いてメニューバーに追加され、その結果カレントメニューバーに追加されたメニューには、左から右へ2049（2048+1からN）で始まる番号が振られます。

Menu Selected関数はこの規則にしたがい、メニューおよびメニュー項目の番号を返します。

警告：これらのコマンドでは4Dにより管理されるシステムメニューへのアクセスは行えません（つまり、Windowsでは「編集」および「ヘルプ」メニュー。Macintoshでは、「アップル」および「編集」メニュー）。

連結メニュー：メニューは、メニューバーに連結することができます。連結されたメニューが上記のコマンドのいずれかを使って修正されると、そのメニューの他のインスタンスすべてにこれらの変更が反映されます。連結メニューに関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

MENU BAR

MENU BAR(メニューバー番号 {; プロセス} {; *})

引数	タイプ		説明
メニューバー番号	数値	→	メニューバーの番号または名前
プロセス	数値	→	プロセス参照番号
*		←	メニューバーの状態を保存

説明

MENU BARコマンドは、カレントプロセスに対してのみ、カレントメニューバーを＜メニューバー番号＞で指定したメニューバーに変更します。引数＜メニューバー番号＞には、新しいメニューバーの番号または名前を渡します。

注：メニューバー名には、31桁までのユニークな名前を指定することができます。

オプション引数＜プロセス＞は、指定したプロセスのメニューバーを＜メニューバー＞に変更します。オプション引数＜*＞により、メニューバーの現在の状態を保存することができます。この引数が省略された場合、このコマンドが実行されると、**MENU BAR**コマンドはメニューバーを元の状態に戻します。

例えば、**MENU BAR**(1)を実行したとします。次に、**DISABLE MENU ITEM**コマンドを使い、複数のメニューを使用不可にします。

MENU BAR(1)を2度目に実行すると、その実行が同じプロセスからでも別のプロセスからでも、メニューはすべて、元の使用可の状態に戻ります。

MENU BAR(1 ; *)を実行すると、メニューバーは前と同じ状態を保っており、使用不可にしたメニューは使用不可のままです。

注：オプション引数＜プロセス＞を指定しない場合、＜*＞は2番目の引数になります。つまりMENU BAR(1 ; 2 ; *)とMENU BAR(1 ; *)はともに有効な命令文です。

ユーザが「カスタム」モードに移ると、最初の間メニューバー（メニューバー#1）が表示されます。データベースを開く際に「On Startup」データベースメソッド、またはStartupメソッドで目的のメニューバーを指定して、個々のユーザ用にメニューバーを変更することができます。

例題

1. 以下の例は、カレントメニューバーをメニューバー#3に変更し、メニューの状態を元に戻します。

```
MENU BAR(3)
```

2. 以下の例題は、カレントメニューバーを“FormMenuBar1”という名前のメニューバーに変更し、メニューコマンドの状態を保存します。

```
MENU BAR("FormMenuBar1";*)
```

3. 以下の例は、レコードの変更中にフォームのメニューバーをメニューバー#3に変更します。レコードの変更が済むと、メニューの状態を保存してメニューバーをメニューバー#2に戻します。

```
MENU BAR(3)        ` 以下のフォームにメニューバー#3を設定する
ALL RECORDS(顧客)
MODIFY SELECTION(顧客)` フォームを開く
MENU BAR(2 *)      ` 変更後メニューバーを戻す
```

参照

メニューの管理

HIDE MENU BAR

HIDE MENU BAR → 数値

説明

HIDE MENU BARコマンドは、メニューバーを隠します。

既にメニューバーが隠れている場合は、このコマンドは何も行いません。

例題

▼以下のメソッドは、マウスボタンをクリックするまでフルスクリーン表示（Macintosh）でレコードを表示します。

```
HIDE TOOL BAR
HIDE MENU BAR
Open window(-1;-1;1+Screen width ; 1+Screen height ; Alternate dialog box)
INPUT FORM([描画];"フルスクリーン")
DISPLAY RECORD([描画])
Repeat
    GET MOUSE($vlX ; $vlY ; $vlButton)
Until($vlButton#0)
CLOSE WINDOW
SHOW MENU BAR
SHOW TOOL BAR
```

注：Windows上では、ウインドウはアプリケーションウインドウの範囲内に制限されます。

参照

HIDE TOOL BAR、SHOW MENU BAR、SHOW TOOL BAR

SHOW MENU BAR

SHOW MENU BAR → 数値

説明

SHOW MENU BARコマンドは、メニューバーを隠します。

既にメニューバーが表示さている場合は、このコマンドは何も行いません。

例題

▼ **HIDE MENU BAR**コマンドの例を参照してください。

参照

HIDE MENU BAR、HIDE TOOL BAR、SHOW TOOL BAR

SET ABOUT

SET ABOUT(アイテム ; メソッド)

引数	タイプ		説明
アイテム	文字列	→	アバウトメニュー項目の新しいテキスト
メソッド	文字列	→	メニュー項目が選択されたときに呼び出すメソッド

説明

SET ABOUTコマンドは、「ヘルプ」（Macintosh版では、「アップル」）メニューの「4th Dimension®について...」を＜アイテム＞に変更します。

SET ABOUTコマンドを実行した後、ユーザがこのメニューを選択すると、＜メソッド＞が実行されます。一般的に、このメソッドはデータベースに関するバージョン情報を示すダイアログボックスを表示します。

4th Dimensionアイコン、バージョン番号、ならびに著作権に関する注意がダイアログの上部に追加されます。

例題

1. 以下の例は、「4th Dimension®について」メニューを「スケジュールについて...」に置き換えます。メソッド“ABOUT”は、カスタムのアバウトボックスを表示します。

```
SET ABOUT("スケジュールについて..." ; "ABOUT")
```

2. 以下の例は、「4th Dimension®について」メニューを元のアバウトボックスに戻します。

```
SET ABOUT("4th Dimension®について..." ; "")
```

Menu selected

Menu selected → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	選択されたメニュー 上位バイトにメニュー番号 下位バイトにメニュー項目番号

説明

Menu selected関数は、フォームが表示さている状態でのみ使用可能です。このコマンドは、メニューから選択されたメニュー項目を検出します。

Tip：可能な限り、Menu selected関数を使用しないで、関連メニューバー（負のメニューバー番号を用いた）のメニューに割り当てられたメソッドを使用してください。いずれのメニューが選択されたかを調べる必要がないため、関連メニューバーはより簡単に管理できます。しかし、APPEND MENU ITEMやINSERT MENU ITEMコマンドを使用する場合には、Menu selected関数を使う必要があります。これは、これらのコマンドを使って追加されたメニュー項目にはメソッドが割り当てられていないためです。

Menu selected関数は、選択されたメニューの番号を倍長整数型で返します。選択されたメニュー番号を知るためには、返された数値を65536で割り、その結果を整数型に変換します。選択されたメニュー番号を知るには、係数65536を使い、返された数値のモジューロを計算します。

```
メニュー:=Menu selected // 65536
メニュー項目:=Menu selected % 65536
```

バージョン6より、ビットワイズ演算子を使用してこれらの値を取得することができます。

```
メニュー:=(Menu selected & 0xFFFF0000) >> 16
メニュー項目:= Menu selected & 0xFFFF
```

メニューが選択されていない場合には、**Menu selected**関数は0を返します。

例題

▼ 以下の例は、**SET MENU ITEM MARK**コマンドの引数を求めるために**Menu selected**関数を使用しています。

```
Case of
    ¥(Form event=On Menu Selected)
        If(Menu selected # 0)
            SET MENU ITEM MARK(Menu selected // 65536;
        Menu selected %65536;Char(18))
        End if
End case
```

参照

メニューの管理

Count menus

Count menus({プロセス}) → 数値

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号
戻り値	数値	←	カレントメニューバーのメニューの数

説明

Count menus関数は、カレントメニューバー上にあるメニューの数を返します。

オプション引数＜プロセス＞を省略すると、**Count menus**関数はカレントプロセスのメニューバーに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューバーに適用されます。

参照

Count menu item

Count menu items

Count menu items(メニュー {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
プロセス	数値	→	プロセス参照番号
戻り値	数値	←	メニュー内のメニュー項目の数

説明

Count menu items関数は、引数＜メニュー＞に渡されたメニュー番号を持つメニュー内にあるメニューの数を返します。

オプション引数＜プロセス＞を省略すると、**Count menu items**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

参照

Count menus

Get menu title

Get menu title(メニュー {; プロセス}) → 文字列

引数	タイプ		説明
メニュー	数値	→	メニュー番号
プロセス	数値	→	プロセス参照番号
戻り値	文字列	←	メニューのタイトル

説明

Get menu title関数は、引数＜メニュー＞に渡されたメニュー番号を持つメニューのタイトルを返します。

オプション引数＜プロセス＞を省略すると、**Get menu title**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

参照

Count menus

Get menu item

Get menu item(メニュー ; メニュー項目 {; プロセス}) → 文字列

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
プロセス	数値	→	プロセス参照番号
戻り値	文字列	←	メニュー項目のテキスト

説明

Get menu item関数は、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー項目番号を持つメニュー項目のテキストを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

参照

SET MENU ITEM

SET MENU ITEM

SET MENU ITEM(メニュー番号 ; メニュー項目番号 ; アイテムテキスト {; プロセス})

引数	タイプ	説明
メニュー番号	数値	メニュー番号
メニュー項目番号	数値	メニュー項目番号
アイテムテキスト	文字列	メニュー項目の新テキスト
プロセス	数値	プロセス参照番号

説明

SET MENU ITEMコマンドは、＜メニュー番号＞と＜メニュー項目番号＞で渡された番号を持つメニュー項目のテキストを設定します。

オプション引数＜プロセス＞を省略すると、**SET MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

参照

Get menu item、SET MENU ITEM KEY

Get menu item style

Get menu item style(メニュー ; メニュー項目 {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
プロセス	数値	→	プロセス参照番号
戻り値	数値	←	メニュー項目のフォントスタイル

説明

Get menu item style関数は、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー番号を持つメニューのフォントスタイルを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item style**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

Get menu item style関数は、以下のようなあらかじめ定義された定数の組み合わせを加算した数値を返します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注：Windows版では、“Plain”または“Bold”、“Italic”、“Underline”の組み合わせしか利用できません。

例題

▼以下の例は、任意のメニューが太字（ボールド）で表示されるかどうかをテストします。

```
If((Get menu item style($vlMenu ; $vlItem) & Bold)#0)
   `...
End if
```

参照

SET MENU ITEM STYLE

SET MENU ITEM STYLE

SET MENU ITEM STYLE(メニュー ; メニュー項目 ; スタイル{; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
スタイル	数値	→	新しいメニュー項目のフォントスタイル
プロセス	数値	→	プロセス参照番号

説明

SET MENU ITEM STYLEコマンドは、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー番号を持つメニューのフォントスタイルを変更します。

オプション引数＜プロセス＞を省略すると、**SET MENU ITEM STYLE**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

SET MENU ITEM STYLEコマンドは、以下のようなあらかじめ定義された定数の組み合わせを加算した数値を返します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注：Windows版では、“Plain”、または“Bold”、“Italic”、“Underline”の組み合わせしか利用できません。

参照

Get menu item style

Get menu item mark

Get menu item mark(メニュー ; メニュー項目 {; プロセス}) → 文字列

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
プロセス	数値	→	プロセス参照番号
戻り値	文字列	←	カレントメニュー項目のマーク

説明

Get menu item mark関数は、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー項目番号を持つメニュー項目のチェックマークを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

メニューにマークが付いていない場合、**Get menu item mark**関数は空の文字列を返します。

注：SET MENU ITEM MARKコマンドの説明におけるMacintoshおよびWindows上のマークに関する記述を参照してください。

例題

▼ 以下の例は、メニュー項目のチェックマークを切り替えます。

```
SET ITEM MARK($vlMenu ; $vlItem ; Char(18)*Num(Get menu item mark
($vlMenu ; $vlItem)=""))
```

参照

SET MENU ITEM MARK

SET MENU ITEM MARK

SET MENU ITEM MARK(メニュー ; メニュー項目 ; マーク {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
マーク	数値	→	新しいメニュー項目のマーク
プロセス	数値	→	プロセス参照番号

説明

SET MENU ITEM MARKコマンドは、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー番号を持つメニューのチェックマークを変更します。

オプション引数＜プロセス＞を省略すると、**SET MENU ITEM MARK**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

空の文字列を渡すと、メニューからすべてのマークが取り除かれます。空の文字列でない場合は、以下のようになります。

■ Macintosh上では、文字列の最初の文字がメニュー項目のマークになります。通常、Char(18)を受け渡します。これは、Macintoshメニューのチェックマーク文字です。

■ Windows上では、メニュー項目には標準のチェックマークが割り当てられます。

▼ **Get item mark**関数の例を参照してください。

参照

Get menu item mark

Get menu item key

Get menu item key(メニュー ; メニュー項目 {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
プロセス	数値	→	プロセス参照番号
戻り値	数値	←	メニュー項目のショートカットキーのASCIIコード

説明

Get menu item key関数は、引数<メニュー>と<メニュー項目>に渡されたメニュー番号およびメニュー番号を持つメニューのキーボード（Windows上ではCtrl、Macintosh上ではcommand）ショートカットキーのASCIIコードを返します。

オプション引数<プロセス>を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用されます。<プロセス>を指定した場合は、<プロセス>に渡されたプロセス参照番号のプロセスのメニューに適用されます。

メニューにショートカットキーがない場合、**Get menu item key**関数は0を返します。

参照

SET MENU ITEM KEY

SET MENU ITEM KEY

SET MENU ITEM KEY(メニュー ; メニュー項目 ; キー {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
キー	数値	→	新しいメニュー項目キーのASCIIコード
プロセス	数値	→	プロセス参照番号

説明

Get menu item keyコマンドは、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー番号を持つメニューのキーボード（Windows上ではCtrl、Macintosh上ではcommand）ショートカットキーを引数＜キー＞に渡されたASCIIコードに変更します。

オプション引数＜プロセス＞を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

引数＜キー＞に0を渡すと、メニュー項目からすべてのショートカットキーが取り除かれます。

注：一貫性のあるユーザインタフェースでは、Ctrlキー（Windows）またはcommandキー（Macintosh）以外のモデファイアキーを使用せずに、キーボード上で利用できる数字や記号に大文字を使用します。

参照

Get menu item key

DISABLE MENU ITEM

DISABLE MENU ITEM(メニュー番号 ; メニュー項目 {; プロセス})

引数	タイプ		説明
メニュー番号	数値	→	メニューの番号
メニュー項目	数値	→	メニュー項目の番号
プロセス	数値	→	プロセス参照番号

説明

DISABLE MENU ITEMコマンドは、＜メニュー番号＞と＜メニュー項目＞で指定したメニュー項目を選択不可の状態（グレー表示）にします。

オプション引数＜プロセス＞を省略すると、**DISABLE MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

Tip：メニューのアイテムを即座に選択不可／選択可にするには、＜メニュー項目＞に0を指定します。

参照

ENABLE MENU ITEM

ENABLE MENU ITEM

ENABLE MENU ITEM(メニュー番号 ; メニュー項目 {; プロセス})

引数	タイプ		説明
メニュー番号	数値	→	メニューの番号
メニュー項目	数値	→	メニュー項目の番号
プロセス	数値	→	プロセス参照番号

ENABLE MENU ITEM(メニュー番号 ; メニュー項目番号)

説明

ENABLE MENU ITEMコマンドは、＜メニュー番号＞と＜メニュー項目＞で指定したメニュー項目を選択可の状態にします。

オプション引数＜プロセス＞を省略すると、**ENABLE MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

Tip：メニューのアイテムを即座に選択不可／選択可にするには、＜メニュー項目＞に0を指定します。

参照

DISABLE MENU ITEM

APPEND MENU ITEM

APPEND MENU ITEM(メニュー ; テキスト {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
テキスト	文字列	→	新しいメニュー項目のテキスト
プロセス	数値	→	プロセス参照番号

説明

APPEND MENU ITEMコマンドは、引数＜メニュー＞で渡されたメニュー番号を持つメニューに新規メニュー項目を追加します。

オプション引数＜プロセス＞を省略すると、**APPEND MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

APPEND MENU ITEMコマンドは、1回の呼び出しで1つまたは複数のメニュー項目を追加することができます。

追加されるメニュー項目を以下のようにして引数＜テキスト＞で定義します。

■ セミコロン（;）で各メニューを区切る。例えば、

“項目テキスト1 ; 項目テキスト2 ; 項目テキスト3”

■ メニュー項目を使用不可にする場合は、引数＜テキスト＞の中に開き丸カッコ“(”を配置する。

■ メニュー項目の区切り線を指定する場合は、引数＜テキスト＞に“(”を渡す。

■ 行にフォントスタイルを指定する場合は、引数＜テキスト＞の中に小なり記号（<）の後に下記の文字の1つを配置する。

<B　太字（ボールド）

<I　斜体（イタリック）

<U　下線（アンダーライン）

<O　アウトライン（Macintoshのみ）

<S　シャドウ（Macintoshのみ）

■ 任意のメニュー項目にチェックマークを追加するには、引数＜テキスト＞の中にのクエスチョンマーク（?）を指定し、その後にチェックマークとして使用する文字を配置する。Macintosh上では、その配置された文字が表示される。Windows上では、どんな文字が渡されようともチェックマークが表示される。

■ 任意のメニュー項目にアイコンを追加するには、引数＜テキスト＞の中にハット（^）を指定し、ASCIIコードから48を引いた値がMacintoshベースのリソースIDである文字をその後に配置する。

■ 任意のメニュー項目にキーボードショートカットを追加するには、引数＜テキスト＞の中にスラッシュ（/）を指定し、その後にショートカット用の文字を配置する。

注：メニュー項目の数が程よいメニューを使用してください。もし、50以上のメニュー項目を表示したい場合は、メニューの代わりにスクロールエリアの使用を検討してみてください。

注：ITEM MENU ITEMコマンドは255バイトまでですが、APPEND MENU ITEMコマンドは最大32000バイトまで使用することができます。

重要：新しいメニュー項目には、メソッドが割り当てられていません。したがって、Menu selected関数を使ってフォームメソッドからこれを管理しなくてはなりません。

例題

▼ 以下の例は、「フォント」メニューに利用可能なフォントの名前を追加します。この例では、カレントメニューバーの 6番目のメニューです。

```
`「On Startup」データベースメソッド内
`フォントリストがロードされ、メニューが作成される
FONT LIST(<>asAvailableFont)
<>atFontMenuItems:=""
For($vlFont ; 1 ; Size of array(<>asAvailableFont))
      <>atFontMenuItems:=<>atFontMenuItems+";"+<>asAvailableFont{$vlFont}
End for
```

そして、任意のフォームメソッドまたはプロジェクトメソッド内に以下のコードを記述します。

```
APPEND MENU ITEM(6 ; <>atFontMenuItems)
```

参照

DELETE MENU ITEM、INSERT MENU ITEM

INSERT MENU ITEM

INSERT MENU ITEM(メニュー ; メニュー項目 ; テキスト {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
テキスト	文字列	→	挿入されるメニュー項目のテキスト
プロセス	数値	→	プロセス参照番号

説明

INSERT MENU ITEMコマンドは、引数＜メニュー＞に渡されたメニュー番号を持つメニューに対し、引数＜メニュー項目＞に渡された番号を持つメニュー項目の後ろに新しいメニュー項目を挿入します。

オプション引数＜プロセス＞を省略すると、**INSERT MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

INSERT MENU ITEMコマンドは、1回の呼び出しで1つまたは複数のメニュー項目を挿入することができます。

INSERT MENU ITEMコマンドは、**APPEND MENU ITEM**コマンドと同じように機能しますが、以下の点が異なります。

- **INSERT MENU ITEM**コマンドはメニュー内のどんな場所にでもメニュー項目を挿入できるのに対して、**APPEND MENU ITEM**コマンドは常にメニューの最後にしかメニュー項目を追加することができない。

- **INSERT MENU ITEM**コマンドは引数＜テキスト＞には最大255バイトまでしか使用できないのに対して、**APPEND MENU ITEM**コマンドは＜テキスト＞には最大32000バイトまで使用できる。

引数＜テキスト＞に渡すメニュー項目の定義に関する詳細は、**APPEND MENU ITEM**コマンドの節を参照してください。

重要：新しいメニュー項目には、メソッドが割り当てられていません。そのため、Menu selected関数を使ってフォームメソッドからこれを管理する必要があります。

参照

APPEND MENU ITEM

DELETE MENU ITEM

DELETE MENU ITEM(メニュー ; メニュー項目 {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニュー項目	数値	→	メニュー項目番号
プロセス	数値	→	プロセス参照番号

説明

DELETE MENU ITEMコマンドは、引数＜メニュー＞と＜メニュー項目＞に渡されたメニュー番号およびメニュー項目番号を持つメニュー項目を削除します。

オプション引数＜プロセス＞を省略すると、**DELETE MENU ITEM**コマンドはカレントプロセスのメニューに適用されます。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用されます。

注：一貫性のあるユーザインタフェースでは、メニューのないメニューを保持してはいけません。

参照

APPEND MENU ITEM、INSERT MENU ITEM

MESSAGES ON
MESSAGES OFF

MESSAGES ON

MESSAGES OFF

説明

MESSAGES ONコマンドと**MESSAGES OFF**コマンドは、時間のかかる処理を行っている際に4th Dimensionから表示されるインジケータの表示／非表示を切り替えます。デフォルトでは、メッセージは表示されます。

インジケータを表示する「ユーザ」モードのメニューを以下の表に示します。

処理	処理	処理
フォーミュラで更新	クイックレポート	並び替え
データ書き出し	データ読み込み	チャート
フォームによるクエリ	フォーミュラによるクエリ	クエリ

インジケータを表示するコマンドを以下の表に示します。

APPLY TO SELECTION	**IMPORT SYLK**	**QUERY**
DISTINCT VALUES	**IMPORT TEXT**	**QUERY BY FORMULA**
EXPORT DIF	**RELATE MANY SELECTION**	**QUERY BY EXAMPLE**
EXPORT SYLK	**RELATE ONE SELECTION**	**QUERY SELECTION**
EXPORT TEXT	**REDUCE SELECTION**	**ORDER BY FORMULA**
GRAPH TABLE	**REPORT**	**ORDER BY**
IMPORT DIF	**SCAN INDEX**	
QUERY SELECTION BY FORMULA		

例題

▼ 以下の例は、「並び替え...」コマンドを実行する前にインジケータを非表示にし、「並び替え...」コマンドの実行が完了した時点で表示に戻します。

```
MESSAGES OFF
ORDER BY([住所] ; [住所]郵便番号 ; > ; [住所]名前2 ; >)
MESSAGES ON
```

ALERT

ALERT(メッセージ {; OKボタンタイトル})

引数	タイプ		説明
メッセージ	文字列	→	「警告」ダイアログボックスに表示するメッセージ
OKボタンタイトル	文字列	→	「OK」ボタンのタイトル

説明

ALERTコマンドは、注意アイコンとメッセージ、OKボタンで構成される警告ダイアログボックスを表示します。

引数<メッセージ>をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

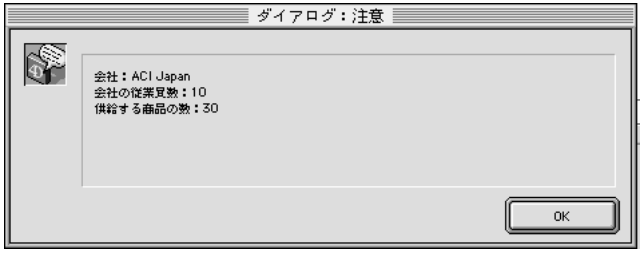
デフォルトでは、OKボタンのタイトル（ラベル）は“OK”です。このOKボタンタイトルを変更するには、オプション引数<OKボタンタイトル>の中に任意のボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、OKボタンの幅を左方向にリサイズすることができます。

例題

1. 以下の例は、会社に関する情報を示すアラートボックスを表示します。表示する文字列中にキャリッジリターンが含まれてることに注目してください。これは、キャリッジリターンから後ろの文字列を次の行に改行するためです。

```
CR:=Char(13)
ALERT("会社：" + [会社]名称 + CR +"会社の従業員数：" +
                String(Records in selection([従業員])) + CR +"供給する商品の数："
                + String(Records in selection([部品])))
```

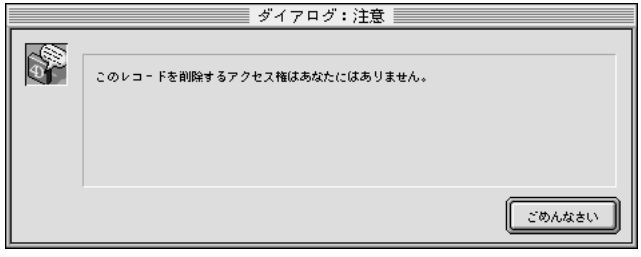
以下の図は上記の**ALERT**コマンドで表示される警告ダイアログボックスを示しています。



注：このALERTコマンドのステートメント行がマニュアルの関係上、3行に渡って記述されていますが、4th Dimensionの「メソッド」エディタでは、ステートメントはすべて1行で記述する必要があります。

2. 以下の例は、OKボタンのタイトルを任意のボタンタイトルに変更したものを示しています。

```
ALERT("このレコードを削除するアクセス権はあなたにはありません。";
                                        "ごめんなさい")
```



参照

CONFIRM、DISPLAY NOTIFICATION、Request

CONFIRM

CONFIRM(メッセージ {; OKボタンタイトル{; キャンセルボタンタイトル}})

引数	タイプ		説明
メッセージ	文字列	→	「確認」ダイアログボックスに表示するメッセージ
OKボタンタイトル	文字列	→	「OK」ボタンのタイトル
キャンセルボタンタイトル	文字列	→	「キャンセル」ボタンのタイトル

説明

CONFIRMココマンドは、確認アイコンとメッセージ、OKボタン、キャンセルボタンで構成される「確認」ダイアログボックスを表示します。

引数＜メッセージ＞をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトルは“OK”、「キャンセル」ボタンのタイトルは“キャンセル”です。この2つのボタンタイトルを変更するには、オプション引数＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞の中に任意のボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、OKボタンの幅を左方向にリサイズすることができます。

「OK」ボタンは、デフォルトボタンです。ユーザが「OK」ボタンをクリックするか、または“enter”キーを押すとシステム変数OKに1が代入されます。ユーザが「キャンセル」ボタンをクリックするとシステム変数OKに0が代入されます。

Tip：On ActivateまたはOn Deactivateフォームイベントを処理するフォームメソッド、またはオブジェクトメソッドのセクションからCONFIRMココマンドをコールしないでください。永久ループになります。

例題

1. 以下の例は、レコード削除処理の確認をユーザに要請するための「確認」ダイアログボックスを表示します。システム変数OKの状態を示すためにIf文とALERTコマンドを使用します。

```
CONFIRM("警告！：この処理は元に戻すことはできません。")
If(OK=1)
    ALL RECORD([元従業員])
    DELETE SELECTION([元従業員)
```

```
Else
        ALERT("処理が取り消されました。")
End if
```

以下の図は、この**CONFIRM**コマンドで表示される「確認」ダイアログボックスを示しています。



2. 以下の例は、引数オプションの＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞に任意のボタンラベルを指定しています。

```
CONFIRM("警告：この処理を続行したら、レコードを元に戻すことができません。"
                + Char(13) + "どうしますか？" ; "続行しない" ; "続行")
```

以下の図は、この**CONFIRM**コマンドで表示される「確認」ダイアログボックスを示しています。



参照

ALERT、 Request

Request

Request(メッセージ {; デフォルト応答 {; OKボタンタイトル{; キャンセルボタンタイトル}}}) → 文字列

引数	タイプ		説明
メッセージ	文字列	→	「リクエスト」ダイアログボックスに表示するメッセージ
デフォルト応答	文字列	→	テキスト入力エリアに表示するデータ
OKボタンタイトル	文字列	→	「OK」ボタンのタイトル
キャンセルボタンタイトル	文字列	→	「キャンセル」ボタンのタイトル
戻り値	文字列	←	ユーザによって入力される値

説明

Request関数は、メッセージ、テキスト入力エリア、OKボタン、キャンセルボタンで構成される「リクエスト」ダイアログボックスを表示します。

引数＜メッセージ＞をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトル（ラベル）は“OK”、「キャンセル」ボタンのタイトル（ラベル）は“キャンセル”です。この2つのボタンタイトルを変更するには、オプション引数＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞の中に任意のボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、OKボタンの幅を左方向にリサイズすることができます。

「OK」ボタンは、デフォルトボタンです。ユーザが「OK」ボタンをクリックするか、または“enter”キーを押すとシステム変数OKに1が代入されます。ユーザが「キャンセル」ボタンをクリックするとシステム変数OKに0が代入されます。

ユーザは、テキスト入力エリアに文字列データを入力することができます。デフォルトの値を指定するには、引数＜デフォルト応答＞にデフォルトのテキストを渡します。ユーザが「OK」ボタンをクリックすると、**Request**関数はその文字列を返します。ユーザが「キャンセル」ボタンをクリックすると、**Request**関数は空の文字列（""）を返します。返される値が数値または日付のいずれかでなければならない場合は、**Request**関数が返した文字列に対して**Num**関数や**Date**関数を使用して正しいデータタイプに変換する必要があります。

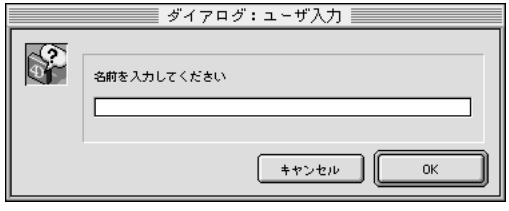
Tip：On ActivateまたはOn Deactivateフォームイベントを処理するフォームメソッド、またはオブジェクトメソッドのセクションからRequest関数をコールしないでください。永久ループになります。

Tip：ユーザから複数の情報を得なければならない場合には、「リクエスト」ダイアログボックスを何度も表示するのではなく、フォームを作成し、DIALOGコマンドを使用してそれを表示してください。

例題

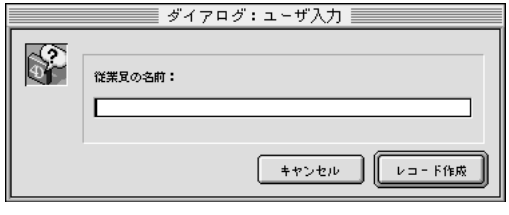
1. 以下の例は、一般的な「リクエスト」ダイアログボックスを表示します。ユーザが入力した情報は、変数“vリターン”に格納されます。

```
vリターン:=Request("名前を入力してください")
```



2. 以下の例は、引数オプションの＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞に任意のボタンラベルを指定して「リクエスト」ダイアログボックスを表示します。

```
vsPrompt:=Request("従業員の名前：" ; "" ; "レコード作成" ; "キャンセル")
If(OK=1)
    ADD RECORD({従業員})
    `変数「vsPrompt」は、フォームメソッド内のOn Loadイベント中に
        [従業員]名字フィールドにコピーされます。
End if
```



3. 以下の例は、デフォルト値として今日の日付を入力し、その日付文字列を日付タイプに変換して「リクエスト」ダイアログボックスを表示します。

```
$vdPrompt := Date(Request("検索日付を入力してください：";
                                String(Current date)))
```



参照

ALERT、CONFIRM

MESSAGE

MESSAGE(メッセージ)

引数	タイプ		説明
メッセージ	文字列	→	表示するメッセージ

説明

MESSAGEコマンドは、通常ユーザに対して何らかの動作を知らせるために使用します。画面上の特殊なメッセージウインドウに＜メッセージ＞を表示します。このメッセージウインドウは、**Open window**関数（後述）を使ってあらかじめ表示されたウインドウを使用していないかぎり、**MESSAGE**コマンドをコールするたびに表示されて、また閉じられます。このメッセージは一時的なもので、レイアウトを表示する、またはメソッドの実行が終了するとすぐに消去されます。別の**MESSAGE**コマンドを実行すると古いメッセージは、消去されます。

Open window関数でウインドウを開いている場合は、**MESSAGE**コマンドに続くコールはすべてそのウインドウにメッセージを表示します。ウインドウはターミナルのような役割をします。

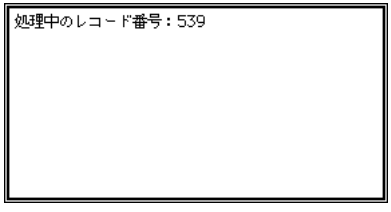
- 一連のメッセージは、このウインドウで表示されると、前のメッセージを消去しません。その代わりに、新しいメッセージは既存のメッセージに続けて表示されます。
- メッセージがウインドウの幅よりも長い場合、4th Dimensionは自動的に改行を行います。
- メッセージの行がウインドウの高さより長い場合、4th Dimensionは自動的にメッセージウインドウをスクロールします。
- 行の制御を行うには、メッセージ中にキャリッジリターン－**Char**(13)－を挿入します。
- ウインドウの特定の位置にメッセージを表示するには、**GOTO XY**コマンドを使用します。
- ウインドウの内容を消去するには、**ERASE WINDOW**コマンドを使用します。
- このウインドウは単なる出力用ウインドウであり、別のウインドウがオーバーラップしても再描画されません。

例題

1. 以下の例は、レコードセレクションを処理し、**MESSAGE**コマンドをコールしてユーザに処理の進捗状況を知らせます。デフォルトのメッセージウインドウ上にメッセージを表示します。

```
For($vlRecord;1;Records in selection([テーブル1]))
    MESSAGE("処理中のレコード番号："+String($vlRecord))
    `レコードを使った処理を行う
    NEXT RECORD([テーブル1])
End for
```

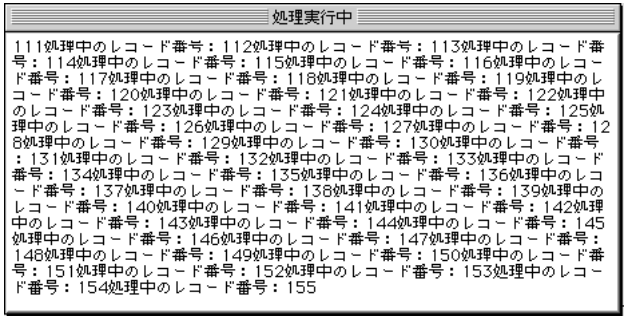
MESSAGEコマンドをコールするたびに、以下のウインドウが表示されては消えます。



2. ウインドウの“ブリンキング（ちらつき）”を避けるため、以下の例題のように**Open window**関数で開いたウインドウにメッセージを表示することができます。

```
Open window(50;50;500;250;5;"処理実行中")
For($vlRecord;1;Records in selection([テーブル1]))
    MESSAGE("処理中のレコード番号："+String($vlRecord))
    ` レコードを使った処理を実行
    NEXT RECORD([テーブル1])
End for
CLOSE WINDOW
```

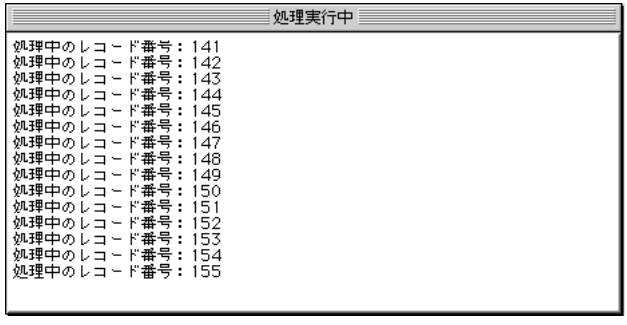
以下のウインドウが表示されます。



3. キャリッジリターンを加えると、より見やすくなります。

```
Open window(50;50;500;250;5;"処理実行中")
For($vlRecord;1;Records in selection([テーブル1]))
    MESSAGE("処理中のレコード番号："+String($vlRecord)+Char(13))
    ` レコードを使った処理を実行
    NEXT RECORD([テーブル1])
End for
CLOSE WINDOW
```

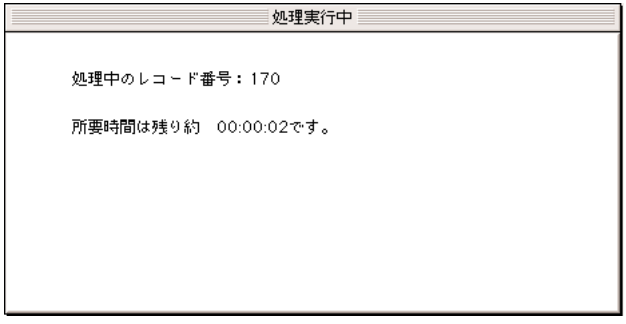
以下のような結果になります。



4. **GOTO XY**コマンドを使用して、何行か追加します。

```
Open window(50;50;500;250;5;"処理実行中")
$vlNbRecords:=Records in selection([テーブル1])
$vhStartTime:=Current time
For($vlRecord;1;$vlNbRecords)
GOTO XY(5;2)
    MESSAGE("処理中のレコード番号："+String($vlRecord)+Char(13))
    ` レコードを使った処理を実行
    NEXT RECORD([テーブル1])
    GOTO XY(5;5)
    $vlRemaining:=(($vlNbRecords/$vlRecord)-1)*(Current time-$vhStartTime)
    MESSAGE("所要時間は残り約　"+Time string($vlRemaining) + "です。")
End for
CLOSE WINDOW
```

以下のような結果になります。



参照

CLOSE WINDOW、ERASE WINDOW、GOTO XY、Open window

GOTO XY

GOTO XY(x ; y)

引数	タイプ		説明
x	数値	→	カーソルのx（水平）位置
y	数値	→	カーソルのy（垂直）位置

説明

GOTO XYコマンドは、**Open window**関数で開いたウインドウにメッセージを表示する際に、**MESSAGE**コマンドとともに使用します。

GOTO XYコマンドは、文字カーソル（見えないカーソル）の位置を指定して、ウインドウに表示される以下のメッセージの位置を設定します。

ウインドウの左上隅の位置が0,0です。カーソルは、ウインドウを開いたときと、**ERASE WINDOW**コマンドを実行した後には、自動的に0.0に置かれます。

GOTO XYコマンドでカーソルの位置を指定してから、**MESSAGE**コマンドでウインドウに文字を表示することができます。

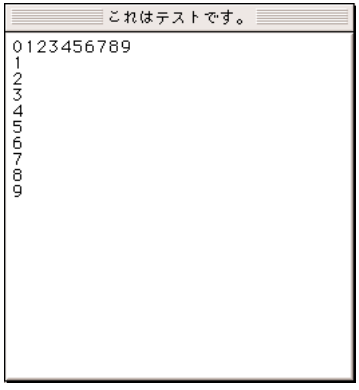
Tip：メッセージには等幅フォントを使用すると、GOTO XYコマンド、MESSAGEコマンドを使って表示した結果が見やすくなります。詳細は、MESSAGEコマンドの説明を参照してください。

例題

1. **MESSAGE**コマンドの例題を参照してください。
2. **Milliseconds**コマンドの例題を参照してください。
3. 以下の例題では30秒間ウインドウを表示します。

```
Open window(50;50;300;300;5;"これはテストです。")
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!!))



参照

MESSAGE

DISPLAY NOTIFICATION

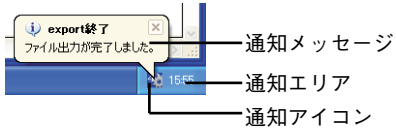
DISPLAY NOTIFICATION (タイトル; テキスト{; 表示時間})

引数	タイプ		説明
タイトル	文字255→		メッセージタイトル
テキスト	文字255→		メッセージテキスト
表示時間	数値	→	表示を継続する時間（秒単位）

説明

注：このコマンドはWindows上でのみ動作します。

DISPLAY NOTIFICATIONコマンドは、Windowsのタスクバーの通知領域にメッセージを表示します。



通常、このタイプのメッセージは、OSやアプリケーションがユーザに外部イベント（ネットワーク切断、アップグレードの提供）を知らせるために使用されます。

＜タイトル＞と＜テキスト＞には、表示するメッセージのタイトルとテキストを渡します（上記の例では、タイトルは“export終了”）。255桁までの文字を入力することができます。

デフォルトでは、ユーザがクローズボックスをクリックするまで、メッセージウインドウは表示されたままです。任意の引数＜表示時間＞を渡すと、ユーザがクローズボックスをクリックしなかった場合、指定した継続時間に達した時点でウインドウが自動的に閉じられます。ユーザがウインドウをクローズした場合でも、＜表示時間＞が終了するまで通知アイコンは表示されたままになります。

参照

ALERT

命名セレクションコマンド 30

命名セレクションを使用することによって、複数のセレクションを同時に扱うことができます。命名セレクションはプロセスのテーブルの順序付けされたリストです。このレコードの順序リストは名前を付けてメモリに残すことができます。命名セレクションによって簡単にセレクションをメモリに置くことができます。セレクションの順序とセレクションのカレントレコードも保存されます。

以下のコマンドを使い、命名セレクション用いた作業を行うことができます。

COPY NAMED SELECTION **CUT NAMED SELECTION**
CLEAR NAMED SELECTION **USE NAMED SELECTION**
CREATE SELECTION FROM ARRAY

命名セレクションは、**COPY NAMED SELECTION**コマンド、**CUT NAMED SELECTION**コマンドまたは**CREATE SELECTION FROM ARRAY**コマンドで作成することができます。一般的に命名セレクションは、1つ以上のセレクションで作業、保存し、後で順序付けしたセレクションを復元するために使用します。プロセス内の各テーブルに対して複数の命名セレクションを持つことができます。カレントセレクションとして命名セレクションを再利用するには、**USE NAMED SELECTION**コマンドを呼び出します。命名セレクションでの作業が終了したら、**CLEAR NAMED SELECTION**コマンドを呼び出します。

命名セレクションには、使用する範囲の違いにより、プロセス命名セレクションとインタープロセス命名セレクションの2種類があります。

先頭が“<>”文字で始まる名前の命名セレクションがインタープロセス命名セレクションで、任意のプロセスから利用することができます。

これに対し、先頭が“<>”文字以外の名前で始まるのがプロセス命名セレクションで、それが作成されたプロセス内でしか利用することはできません。

4D Clientと4D Serverを使用してる場合、インタープロセス命名セレクションはそれを作成したクライアントのプロセスでしか利用することはできません。つまり、別のクライアントマシンでそれを利用することはできません。

警告：命名セレクションを作成するには、テーブルのセレクションへのアクセスが必要となります。セレクションはサーバ上に保持され、またローカルプロセスはサーバデータへアクセスする方法がないため、ローカルプロセス内で命名セレクションを使用することはできません。

命名セレクションとセット

セットと命名セレクションとの違いは、以下の通りです。

■ 命名セレクションがレコードの順序付けされたリストであるのに対して、セットは順序付けされていません。

■ セットは、テーブル内の各レコードに対して1ビットしか必要としないためメモリの面から見ると効率的です。命名セレクションは、セレクション内の各レコードに対して4バイトを必要とします。

■ セットと違って命名セレクションは、ディスクへ保存することができません。

■ セットには交差（INTERSECTION）、結合（UNION）、差異（DIFFERENCE）といった標準演算があるのに対して、命名セレクションは他の命名セレクションとの組み合わせ操作は行えません。

命名セレクションとセットには以下のような類似点があります。

■ セットと同じように命名セレクションもメモリ内に存在します。

■ 命名セレクションもセットもレコードの参照を格納します。レコードを変更または削除すると、命名セレクションやセットは無効になることがあります。

■ セットと同じように命名セレクションは、その命名セレクションが作成された時点のカレントレコードを“記憶”します。

セットに関する詳細は、後述の「セットコマンド」を参照してください。

COPY NAMED SELECTION

COPY NAMED SELECTION({テーブル ;} 名前)

引数	タイプ		説明
テーブル	テーブル	→	セレクションコピーするテーブル 省略した場合、デフォルトテーブル
名前	文字列	→	作成するセレクションの名前

説明

COPY NAMED SELECTIONコマンドは、＜テーブル＞のカレントセレクションを命名セレクション＜名前＞にコピーします。オプション引数＜テーブル＞が指定されていない場合は、そのプロセスのデフォルトテーブルを使用します。＜名前＞にはセレクションのコピーが納められます。そのプロセスにおける＜テーブル＞のカレントセレクションとカレントレコードは変更されません。

命名セレクションは実際にレコードを含むわけではなく、レコードへの順序付けされたポインタを含みます。各ポインタは4バイトです。したがって、**COPY NAMED SELECTION**コマンドを使用してセレクションをコピーすると、セレクション内のレコード数×4バイト分のメモリが必要になります。命名セレクションはメモリに置かれるため、命名セレクションに必要な分とプロセス内のテーブルのカレントセレクションに必要な分のメモリを確保しなければなりません。

4D Server：命名セレクション＜名前＞とカレントセレクションは、両方ともサーバのメモリに置かれるため、サーバには十分なメモリを確保してください。

命名セレクション＜名前＞が使用したメモリを解放するには、**CLEAR NAMED SELECTION**コマンドを使用します。

例題

▼ 以下の例では、[従業員]テーブルに未払いの送り状があるかどうかを調べています。セレクションをソートして保存します。送り状が未払いのレコードをすべて検索します。その後、そのセレクションを再利用してメモリ内の命名セレクションを消去します。

ソートしたセレクションを後で使いたい場合には、命名セレクションを消去しなくても構いません。

```
ALL RECORDS([従業員])
ORDER BY([従業員])          ` セレクションをソートする
` ソートしたセレクションを命名セレクションとして保存する
COPY NAMED SELECTION([従業員] ; "ユーザソート")
QUERY([従業員] ;[ 従業員]未払い=True)    ` 送り状が未払いのレコードを探す
```

```
If(Records in selection([従業員]) > 0)      ` レコードが見つかった場合
      ALERT("未払いの送り状があります")      ` ユーザに知らせる
End if
USE NAMED SELECTION("ユーザソート") ` ソートした命名セレクションを利用
CLEAR NAMED SELECTION("ユーザソート") ` セレクションをメモリから消去
```

参照

CLEAR NAMED SELECTION、CUT NAMED SELECTION、USE NAMED SELECTION

CUT NAMED SELECTION

CUT NAMED SELECTION({テーブル ;} 名前)

引数	タイプ	説明
テーブル	テーブル	セレクションカットするテーブル 省略した場合、デフォルトテーブル
名前	文字列	作成するセレクションの名前

説明

CUT NAMED SELECTIONコマンドは、命名セレクション＜名前＞を作成し、＜テーブル＞のカレントセレクションをそこへ移します。このコマンドは、カレントセレクションをコピーするのではなく、実際に＜テーブル＞のカレントセレクションを移す点が**COPY NAMED SELECTION**コマンドと異なります。

このコマンドを実行した後、カレントプロセス内の＜テーブル＞のカレントセレクションは空になります。そのため、**CUT NAMED SELECTION**コマンドはレコードが修正されている最中は使用しないでください。

CUT NAMED SELECTIONコマンドは、**COPY NAMED SELECTION**コマンドよりも効率的です。**COPY NAMED SELECTION**コマンドでは選択したレコードの数×4バイトをメモリ内で複製します。**CUT NAMED SELECTION**コマンドではリストの参照だけを移動します。

例題

▼以下の例では、簡単に[顧客]テーブルのカレントセレクションを空にします。

```
CUT NAMED SELECTION([顧客] ; "クリア")
CLEAR NAMED SELECTION("クリア")
```

参照

CLEAR NAMED SELECTION、COPY NAMED SELECTION、USE NAMED SELECTION

USE NAMED SELECTION

USE NAMED SELECTION(名前)

引数	タイプ		説明
名前	文字列	→	使用する命名セレクションの名前

説明

USE NAMED SELECTIONコマンドは、命名セレクション＜名前＞をそれが属するテーブルのカレントセレクションとして使用します。

命名セレクションを作成すると、その命名セレクションはカレントレコードを記憶します。**USE NAMED SELECTION**コマンドはこのレコードの位置を取り出し、そのレコードを新しいカレントレコードにします。このコマンドは、カレントレコードをロードします。命名セレクション＜名前＞を作成した後にカレントレコードが変更された場合、変更情報を失わないために**USE NAMED SELECTION**コマンドを実行する前にそのレコードを保存してください。

■ **COPY NAMED SELECTION**コマンドを使用して命名セレクション＜名前＞を作成した場合は、＜名前＞はそれが属するテーブルのカレントセレクションにコピーされます。＜名前＞は消去されるまでメモリに残ります。命名セレクション＜名前＞が使用しているメモリを消去するには、**CLEAR NAMED SELECTION**コマンドを用います。

■ **CUT NAMED SELECTION**コマンドを使用して命名セレクション＜名前＞を作成した場合は、カレントセレクションが＜名前＞に設定され、＜名前＞はメモリから消去されます。

命名セレクションは、それが作成された時点でのレコードのセレクションを表わす点に注意してください。命名セレクションが表わしているレコードが変更されると、命名セレクションは正確ではなくなります。したがって、命名セレクションは通常、あまり変更されないレコードの集まりを表わします。命名セレクションが無効になる原因はいくつかあります。例えば、命名セレクションのレコードの変更、削除、および命名セレクションを決定する条件の変更等があります。

また、トランザクションの際には一時的なレコードアドレスが使用される点にも注意が必要です。トランザクション中に命名セレクションが作成されると、トランザクションが確定または取り消された場合には無効となるアドレスを含むことがあります。これは、トランザクションが受け入れられた後にレコードがその最終的な実アドレスを受け取るためです。

参照

COPY NAMED SELECTION、CUT NAMED SELECTION、USE NAMED SELECTION

CLEAR NAMED SELECTION

CLEAR NAMED SELECTION(名前)

引数	タイプ		説明
名前	文字列	→	消去する命名セレクションの名前

説明

CLEAR NAMED SELECTIONコマンドは、命名セレクション＜名前＞をメモリから消去して、それが使用していたメモリを解放します。**CLEAR NAMED SELECTION**コマンドはテーブル、セレクション、およびレコードには影響しません。命名セレクションはメモリを使用するため、不要になったら消去する習慣をつけることをお勧めします。

CUT NAMED SELECTIONコマンドを使用して命名セレクション＜名前＞を作成し、**USE NAMED SELECTION**コマンドで処理した場合には、命名セレクション＜名前＞は既にメモリ上にはありません。この場合は、**CLEAR NAMED SELECTION**コマンドを使用する必要はありません。

参照

COPY NAMED SELECTION、CUT NAMED SELECTION、USE NAMED SELECTION

CREATE SELECTION FROM ARRAY

CREATE SELECTION FROM ARRAY(テーブル;レコード配列 {;セレクション名})

引数	タイプ		説明
テーブル	テーブル	→	セレクションを作成するテーブル
レコード配列	倍長整数配列	→	レコード番号の配列、またはブール配列（True=レコードはセレクションに含める False=レコードはセレクションに含めない）
セレクション名	文字列	→	作成する命名セレクションの名前 省略した場合は、コマンドをカレントセレクションに適用する。

説明

このコマンドは、下記の方法で命名セレクションセレクション名を作成します。

■ 絶対レコード番号の配列から、またはブールの配列から。

この場合、配列の値はテーブルの各レコードが作成する命名セレクションに含める（True）か含めない（False）かを示します。

セレクション名を省略した場合や空白を渡した場合は、コマンドはカレントセレクションに適用され更新されます。

このコマンドで倍長整配列を使用すると、配列の各要素は作成されるセレクション名内のレコードのレコード番号を表わします。レコード番号が不正確（作成されていないレコード）な場合、エラー-10503が発生します。

このコマンドでブール配列を使用すると、配列のN番目の要素がTrueかFalseにより、レコード番号Nが、作成するセレクション名に含まれるかどうかを表わしています。配列要素数は、テーブル内のレコード数と等しくなければなりません。

配列数がレコード数よりも少ない場合は、配列によって定義されたレコードのみが判断対象となります。

注：ブール配列では、コマンドは配列要素0から配列要素N-1を使用します。

警告：命名セレクションが作成されてメモリ内にロードされます。したがって、このコマンドを実行する前に十分なメモリがあることを確認してください。

参照

CREATE SET FROM ARRAY

オブジェクトプロパティコマンド 31

この章では、「ルーチン」エディタの「Object Properties」テーマ内にあるオブジェクトプロパティコマンドについて説明します。この章のコマンドは、フィールドや変数等の入力フォームエリアに影響を与えます。例えば、これらのコマンドはフォームが表示や印刷される場合にのみ有効なフォームオブジェクトを変更します。このオブジェクトは新しいフォームやレコードが表示されると、「デザイン」モードで割り当てられた属性に戻ります。これらのコマンドはフォームメソッドやオブジェクトメソッドで使用されます。

BUTTON TEXT	**FONT STYLE**	**SET ENTERABLE**
DISABLE BUTTON	**GET OBJECT RECT**	**SET FILTER**
ENABLE BUTTON	**MOVE OBJECT**	**SET FORMAT**
FONT	**SET CHOICE LIST**	**SET VISIBLE**
FONT SIZE	**SET COLOR**	**SET RGB COLORS**

「オブジェクトプロパティ」コマンドは、フォームの中にあるオブジェクトのプロパティに対して作用します。「ユーザ」モードまたは「カスタム」モードでフォームを使用しながら、「オブジェクトプロパティ」コマンドによって、オブジェクトの表示様式や動作を変更することができます。

重要：これらのコマンドの範囲は現在使われているフォームです。フォームを終了すると、変更内容が無効になります。

オブジェクト名またはデータベース名によるオブジェクトへのアクセス

「オブジェクトプロパティ」コマンドは、以下で説明するのと同じシンタックスを共有します。

コマンド名({*;} オブジェクト {; 各コマンド固有のその他引数)

オプション引数<*>を指定する場合は、引数<オブジェクト>によってオブジェクト名（文字列）を指定します。

1回の呼び出しで1つのフォームの複数のオブジェクトを指定するには、そのオブジェクト名の中でワイルドカード記号“@”を使用します。以下の表に、このコマンドを使用して指定できるオブジェクト名の例を示します。

オブジェクト名	呼び出しによって影響を受けるオブジェクト
mainGroupBox	オブジェクト「mainGroupBox」のみ
main@	名前が"main"で始まるオブジェクト
@GroupBox	名前が"GroupBox"で終わるオブジェクト
@Group@	名前が"Group"という文字を含むオブジェクト
main@Btn	名前が"main"で始まり、"Btn"で終わるオブジェクト
@	フォームの中にあるすべてのオブジェクト

オプション引数＜*＞を省略する場合は、引数＜オブジェクト＞によってフィールドや変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドまたは変数オブジェクトのみ）を指定します。

注：この2番目のシンタックスは、4th Dimensionの旧バージョンと互換性があります。

参照

BUTTON TEXT、DISABLE BUTTON、ENABLE BUTTON、FONT、FONT SIZE、FONT STYLE、SET CHOICE LIST、SET ENTERABLE、SET FILTER、SET FORMAT、SET RGB COLOR、SET VISIBLE

FONT

FONT({*;} オブジェクト ; フォント)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場） 変数（*を省略した場合）
フォント	文字列／数値	→	フォント名またはフォント番号

説明

FONTコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォント＞で渡したフォント名またはフォント番号を設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名に関する詳細は、この前の節を参照してください。

例題

1. 以下の例は、“bOK”という名前のボタンに対して“Times”フォントをセットします。

```
FONT(bOK ; "Times")
```

2. 以下の例は、“info”という名前を含んだすべてのフォームオブジェクトに対して“Times”フォントをセットします。

```
FONT(* ; @info@ ; "Times")
```

参照

FONT SIZE、FONT STYLE

FONT SIZE

FONT SIZE({*;} オブジェクト ; フォントサイズ)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
フォントサイズ	数値	→	フォントサイズ（ポイント）

説明

FONT SIZEコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォントサイズ＞で渡したフォントサイズ（ポイント数）を設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

引数＜サイズ＞は、1から255までの整数です。指定したフォントサイズが存在しない場合には、文字をスケーリングします。

オブジェクトの領域は、フォームで指定したように、指定したフォントサイズを表示するのに十分な大きさがなければなりません。十分な大きさがない場合には、テキストが途中までしか表示されなかったり、全く表示されなくなります。

例題

1. 以下の例は、“v情報”と名付けられた変数に対して14ポイントのフォントサイズを設定します。

```
FONT SIZE(v情報 ; 14)
```

2. 以下の例は、“hl”という名前で始まるすべてのフォームオブジェクトに対して14ポイントのフォントサイズを設定します。

```
FONT SIZE(* ; "hl@" ; 14)
```

参照

FONT、FONT STYLE

FONT STYLE

FONT STYLE({*;} オブジェクト ; フォントスタイル)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
フォントスタイル	数値	→	フォントスタイル

説明

FONT STYLEコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォントスタイル＞で渡したフォントスタイルを設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

下記で示したフォントスタイルの定数を加算した数値を引数＜フォントスタイル＞で渡します。フォントスタイルの定数は、以下の表のようにあらかじめ定義されています。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注：Windows版では、“Plain”、“Bold”、“Italic”、“Underline”しか利用できません。

例題

1. 以下の例は、“b追加”という名前のボタンに対して“ボールド（太字）+イタリック（斜体）”のフォントスタイルを設定します。

```
FONT STYLE(b追加 ; Bold + Italic)
```

2. 以下の例は、“vt”という名前で始まるすべてのフォームオブジェクトに対して“Plain（標準）”フォントスタイルを設定します。

```
FONT STYLE(* ; "vt@" ; Plain)
```

参照

FONT、FONT SIZE

ENABLE BUTTON

ENABLE BUTTON({*;} オブジェクト)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）

説明

ENABLE BUTTONコマンドコマンドは、引数<オブジェクト>で指定されたフォームオブジェクトを使用可能にします。

使用可能なボタンまたはオブジェクトは、マウスクリックおよびキーボードショートカットに反応します。

オプション引数<*>を指定すると、<オブジェクト>にオブジェクト名を指定します。オプション引数<*>を省略すると、<オブジェクト>にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドは、以下のようなオブジェクトタイプに適用されます。

- ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ラジオピクチャ
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリスト
- サーモメータ、ルーラ

注：自動動作属性を割り当てられたオブジェクトでこのコマンドを使用するのは現実的ではありません。4th Dimensionが必要な時にその制御のステータスを変更するからです。

例題

1. 以下の例は、「b有効」ボタンを使用可能にします。

```
ENABLE BUTTON(b有効)
```

2. 以下の例は、"btn"を含んだ名前を持つフォームオブジェクトのすべてを使用可能にします。

```
ENABLE BUTTON(* ; "@btn@")
```

3. **BUTTON TEXT**コマンドの例を参照してください。

参照

BUTTON TEXT、DISABLE BUTTON

DISABLE BUTTON

DISABLE BUTTON({*;} オブジェクト)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）

説明

DISABLE BUTTONコマンドコマンドは、引数<オブジェクト>で指定されたフォームオブジェクトを使用不可にします。

使用不可のボタンまたはオブジェクトは、マウスクリックおよびキーボードショートカットに反応せず、グレー表示されます。

オプション引数<*>を指定すると、<オブジェクト>にオブジェクト名を指定します。オプション引数<*>を省略すると、<オブジェクト>にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドは、以下のようなオブジェクトタイプに適用されます。

- ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ラジオピクチャ
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリスト
- サーモメータ、ルーラ

注：自動動作属性を割り当てられたオブジェクトでこのコマンドを使用するのは現実的ではありません。4th Dimensionが必要な時にその制御のステータスを変更するからです。

例題

1. 以下の例は、「b有効」ボタンを使用不可にします。

```
DISABLE BUTTON(b有効)
```

2. 以下の例は、"btn"を含んだ名前を持つフォームオブジェクトのすべてを使用不可にします。

```
DISABLE BUTTON(* ; "@btn@")
```

3. **BUTTON TEXT**コマンドの例を参照してください。

参照

BUTTON TEXT、ENABLE BUTTON

BUTTON TEXT

BUTTON TEXT({*;} オブジェクト ; ボタンテキスト)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
ボタンテキスト	文字列	→	ボタンのタイトル（ラベル）

説明

BUTTON TEXTコマンドは、引数＜オブジェクト＞で指定されたオブジェクトのボタンタイトルを引数＜ボタンテキスト＞で渡した値に変更します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

BUTTON TEXTコマンドは、テキストを表示するボタン（ボタン、チェックボックス、ラジオボタン）にのみ影響を与えます。

一般的に、このコマンドは一度に1つのボタンに適用します。ボタンエリアは、ボタンテキストを表示するだけの十分な大きさが必要です。ボタンエリアが小さすぎると、テキストを途中までしか表示されません。引数＜ボタンテキスト＞にキャリッジリターンは使用しないでください。

例題

▼ 以下の例は、**MODIFY SELECTION**コマンドを使って表示された出力フォームのフッタエリアにある検索ボタンのオブジェクトメソッドです。このメソッドは、[従業員]テーブルを検索し、その検索結果によって“b削除”とラベルの付いたボタンを使用可または使用不可にして、そのボタンタイトルを変更します。

```
QUERY([従業員]; [従業員]名前 = vName)
Case of
    ¥(Records in selection([従業員]) = 0) `削除する従業員がいない場合
        BUTTON TEXT(b削除;" 削除")
        DISABLE BUTTON(b削除)
    ¥(Records in selection([従業員]) = 1) ` 削除する従業員が1人の場合
        BUTTON TEXT(b削除;"単一削除")
```

```
            ENABLE BUTTON(b削除)
        ¥(Records in selection([従業員]) > 1) ` 削除する従業員が複数の場合
            BUTTON TEXT(b削除;"複数削除")
            ENABLE BUTTON(b削除)

    End case
```

参照

DISABLE BUTTON、ENABLE BUTTON

Get format

Get format ({*; } オブジェクト) → 文字列

引数	タイプ		説明
*	*	→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、またはフィールドまたは変数（“*”を省略した場合）
戻り値	文字列	←	オブジェクトの表示フォーマット

説明

Get format関数は、引数<オブジェクト>で指定されたオブジェクトに適用された現在の表示フォーマットを返します。

任意の引数<*>を渡すことにより、引数<オブジェクト>がオブジェクト名であることを示します（この場合<オブジェクト>には文字列を渡す）。引数<*>を渡さない場合、引数<オブジェクト>がフィールドまたは変数であることを示します。その場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を渡してください。

この関数は、オブジェクトの現在の表示フォーマット、つまり、「デザイン」モードや**SET FORMAT**コマンドで定義されたフォーマットを返します。**Get format**関数は、表示フォーマットを受け入れるあらゆるタイプのフォームオブジェクト（フィールドや変数）に対して使用することができます（つまり、ブール、日付、時間、ピクチャ、文字列、数値、ボタングリッド、ダイヤル、サーモメータ、ルーラ、ピクチャポップアップメニュー、ピクチャボタン、3Dボタン）。これらのオブジェクトの表示フォーマットに関する詳細は、**SET FORMAT**コマンドの説明を参照してください。

注：関数を一連のオブジェクトに対して適用した場合、最後に選択されたオブジェクトのフォーマットが返されます。

Get format関数を日付、時間、ピクチャタイプのオブジェクト（定数として定義されたフォーマット）に対して適用すると、定数のASCIIコードに相当する文字列が返されます。定数の値を取得するには、この戻り値に対して**Ascii**関数を適用してください（後述）。

例題

(1) この例題により、“myphoto”という名前のピクチャ変数に対して適用されたフォーマット定数の値を取得することができます。

```
C_STRING(2;$format)
SET FORMAT(*;"myphoto";Char(On background))
        `バックグラウンドフォーマットを適用（値=3）
$format:=Get format(*;"myphoto")
ALERT("フォーマット番号："+String(Ascii($format)))
        `値“3”を表示
```

(2)この例題を使用し、ブールタイプのフィールド“[メンバー]婚姻区分”に対して適用されたフォーマットを取得することができます。

```
C_STRING(30;$format)
$format:=Get format([メンバー]婚姻区分)
ALERT($format)
        `例えば“既婚：独身”のようなフォーマットを表示。
```

参照

SET FORMAT

SET FORMAT

SET FORMAT({*;} オブジェクト ; 表示フォーマット)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
表示フォーマット	文字列	→	オブジェクトに使用する表示フォーマット

説明

SET FORMATコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの表示フォーマットを引数＜表示フォーマット＞で渡したフォーマットに変更します。新たに設定したフォーマットはカレント表示のみに適用されます。フォームと一緒には保存されません。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。オブジェクト名についての詳細はこの章の初めを参照してください。

SET FORMATコマンドは、入力フォームと出力フォーム（表示用または印刷用）の両方で使用することができます。また、フィールドや変数（入力可あるいは入力不可）に適用できます。

通常、オブジェクトに表示するデータタイプに合った表示フォーマットか、オブジェクト自体を使用します。

ブール

ブールフィールドをフォーマットする方法は2つあります。

- 1つの値を＜表示フォーマット＞に渡します。この場合、フィールドはチェックボックスとして表示され、指定された値がそのラベルになります。
- 2つの値をセミコロン(;)で区切り＜表示フォーマット＞に渡します。この場合、フィールドは2つのラジオボタンとして表示されます。

日付

日付フィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に“Char(n)”を渡します。nは以下の表にある4Dにあらかじめ定義されている定数です。

定数	タイプ	値
Short	倍長整数	1
Abbreviated	倍長整数	2
Long	倍長整数	3
MM DD YYYY	倍長整数	4
Month Date Year	倍長整数	5
Abbr Date, year	倍長整数	6
MM DD YYYY Forcedr	倍長整数	7

時間

時間フィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に“Char(n)”を渡します。nは以下の表にある4Dにあらかじめ定義されている定数です。

定数	タイプ	値
HH MM SS	倍長整数	1
HH MM	倍長整数	2
Hour Min Sec	倍長整数	3
Hour Min	倍長整数	4
HH MM AM PM	倍長整数	5

ピクチャ

ピクチャフィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に“Char(n)”を渡します。nは以下の表にある4Dにあらかじめ定義されている定数です。

定数	タイプ	値
Truncated Centered	倍長整数	1
Scaled to Fit	倍長整数	2
On Background	倍長整数	3
Truncated non Centered	倍長整数	4
Scaled to fit proportional	倍長整数	5
Scaled to fit prop centered	倍長整数	6

文字列と数値

文字列および数値のフィールドもしくは変数をフォーマットするには、フォーマットのラベルを直接、引数<表示フォーマット>に渡します。

文字および数値の表示フォーマットに関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

注：引数<表示フォーマット>で、「ツールボックス」->「フィルタ」であらかじめ定義した表示フォーマットを使用するには、フォーマット名の前に縦棒（¦）を挿入します。

ピクチャボタン

ピクチャボタンをフォーマットするには、以下のシンタックスに従い、引数<display Format>に文字列を渡します。

cols;lines;picture;flags{;ticks}

■ cols = ピクチャのカラム番号

■ lines = ピクチャの行番号

■ picture = ピクチャライブラリ、ピクチャ変数、もしくはPICTリソースから使用するピクチャ。

  ■ ピクチャライブラリのピクチャを使う場合、クエスチョンマーク?の後にその番号を入力（例："?250"）

  ■ ピクチャ変数のピクチャを使う場合、変数名を入力。

  ■ PICTリソースのピクチャを使う場合、コロン":"の後にその番号を入力（例：":62500"）。

■ flags = ピクチャボタンのモードおよび動作を表示。このパラメータは、0、1、2、16、32、64および128の値をすべて受け入れます。これらの各値は表示モードもしくは動作モードを表し、合計することができます。例えば、1および64のモードを有効にしたい場合、引数<flag>に65の値を渡します。各値についての詳細は以下のとおりです。

  ■ flags = 0（no option）

    ユーザがピクチャをクリックすると、次のピクチャが表示されます。Shiftキーを押さえたままピクチャをクリックすると、その前のピクチャが表示されます。

  ■ flags = 1（Switch Continuously）

今までのオプションと似ていますが、ユーザはマウスボタンを押さえることによりピクチャを継続して表示することができます（例：動画など）。最後のピクチャに到達しても、オブジェクトは最初のピクチャには戻りません。

■ flags = 2（Loop Back to First Frame）

今までのオプションと似ていますが、ピクチャは継続的にループして表示されます。到達した最後のピクチャで再びクリックすると最初のピクチャが表示されまた同じ動作が繰り返されます。

■ flags = 16（Switch when Roll Over）

ピクチャボタンは、マウスカーソルを受けると変更します。カーソルがピクチャボタンを離れると、再び最初のピクチャが現れます。このモードはマルチメディアのアプリケーションやHTMLドキュメントで頻繁に利用されます。「Use Last Frame as Disabled」オプション（128）が選択されていないときは、サムネールの最後のピクチャがここで表示されます。このオプションが選択された場合、サムネールの最後から 2 番目のピクチャが表示されます。

■ flags = 32（Switch Back when Released）

このモードは2つのピクチャの動作で使われます。最初のピクチャは、ユーザがボタンをクリックしているとき以外、常に表示されます。2つ目のピクチャは、クリックされている間（マウスが押さえられている間）表示されます。マウスボタンが放されると最初のピクチャが再び表示されます。このモードにより（クリックされているかどうかという）状況を表示するためのアクションボタンを生成することができます。また、3D効果の作成や動画の表示にも、このモードを利用することができます。

■ flags = 64（Transparent）

このオプションにより、背景のピクチャを透明にできます。

■ flags = 128（Use Last Frame as Disabled）

このモードにより、ボタンがDisabledになっているときに最後のサムネールを表示するように設定できます。このモードが選択されると、ボタンがDisabledになっている場合、最後のサムネールが表示されます。このモードが0、1および2のモードとともに使われた場合、通常の操作で最後のサムネールは表示されません。表示されるのは、ボタンがDisabledになっている場合のみです。

■ ticks= 「Switch every n Ticks」モードを有効にし、それぞれのピクチャが表示される時間の間隔を設定します。このオプションのパラメータが渡された場合、ピクチャボタンの内容を指定されたスピードで表示することができます。例えば、"2;3;?16807;0;10" が入力された場合、ピクチャボタンは10チック（ticks）ごとに異なるピクチャを表示します。このモードが選ばれているときに利用できるのは「Transparent」モード（64）のみです。

ポップアップメニュー

ポップアップメニューをフォーマットするには、以下のシンタックスに従い、引数<表示フォーマット>に文字列を渡します。

cols;lines;picture;hMargin;vMargin;flags

■ cols = ピクチャのカラム番号

■ lines = ピクチャの行番号

■ picture = ピクチャライブラリ、ピクチャ変数、もしくはPICTリソースから使われたピクチャ。

  ■ ピクチャライブラリのピクチャを使う場合、クエスチョンマーク?の後にその番号を入力（例："?250"）。

  ■ ピクチャ変数のピクチャを使う場合、変数名を入力。

  ■ PICTリソースのピクチャを使う場合、コロン:の後にその番号を入力（例：":62500"）

■ hMargin = ピクセル単位で表されたメニューとピクチャ間の水平方向の余白制限値。

■ vMargin = ピクセル単位で表されたメニューとピクチャ間の垂直方向の余白制限値。

■ flags = 「ピクチャポップアップ」メニューの「Transparent」モード。0と64の値を受け入れます。

  ■ mode = 0：「ピクチャポップアップ」メニューは不透明（not transparent）

  ■ mode = 64：「ピクチャポップアップ」メニューは透明（transparent）

サーモメータおよびルーラ

サーモメータもしくはルーラのオブジェクトをフォーマットするには、以下のシンタックスに従い、引数<表示フォーマット>に文字列を渡します。

min;max;unit;step;flags{;format}

■ min = インジケータの最初の目盛の値

■ max = インジケータの最後の目盛の値

■ unit = インジケータの目盛の間隔

■ step = インジケータでカーソルが動く最小の間隔

■ flags = モードの表示およびインジケータの操作を行う。このパラメータは0、2、3、16および32の値を受け入れます。これらの値は、いくつかのオプションを設定するために組み合わせることができます。各値についての詳細は以下のとおりです。

■ flags = 0：unitを表示しない。

■ flags = 2：インジケータの右もしくは下にunitを表示する。

■ flags = 3：インジケータの左もしくは上にunitを表示する。

■ flags = 16：unitに隣接して目盛を表示する。

■ flags = 32：インジケータの調整中に「On Data Change」イベントが実行される。この値が適用されない場合、インジケータの調整が終了してからのみ「On Data Change」イベントが実行されます。

■ format = インジケータの目盛のフォーマットを表示する。unitと目盛は、インジケータのオブジェクトサイズによって正しく表示されるよう受け入れられなかった場合、自動的に非表示になりますので留意してください。

ダイアル

ダイアルのオブジェクトをフォーマットするには、以下のシンタックスに従い、引数<表示フォーマット>に文字列を渡します。

min;max;unit;step{;flags}

■ min = インジケータの最初の目盛の値

■ max = インジケータの最後の目盛の値

■ unit = インジケータの目盛の間隔

■ step = インジケータでカーソルが動く最小の間隔??

■ flags = ダイアルの動作モード（オプション）。このパラメータは32の値のみを受け入れる。インジケータの調整中に「On Data Change」イベントが実行される。この値が適用されない場合、インジケータの調整が終了してからのみ「On Data Change」イベントが実行される。

ボタングリッド

ボタングリッドをフォーマットするには、以下のシンタックスに従い、引数<display Format>に文字列を渡します。

cols;lines

■ cols = グリッドのカラム番号

■ lines = グリッドの行番号

注：フォームオブジェクトにおけるフォーマットの表示についての詳細は、『4th Dimensionデザインリファレンス』マニュアルを参照してください。

3Dボタン

3Dボタンにフォーマットを適用するには、次のシンタックスに従い、引数<表示フォーマット>に文字列を渡します。

タイトル;ピクチャ;バックグラウンド;タイトル位置;タイトル表示;アイコン表示;スタイル;水平マージン;垂直マージン;アイコンオフセット;ポップアップメニュー

■ タイトル=ボタンタイトル。この値は、テキストまたはリソース番号として表わすことができます（例：“:16800,1”）。

■ ピクチャ=ボタンに関連付けられるピクチャ。ピクチャライブラリ、ピクチャ変数、またはPICTリソースから参照します。

  ■ ピクチャライブラリから参照するピクチャの場合、疑問符記号（?）に続けてその番号を入力します（例：“?250”）。

  ■ ピクチャ変数から参照するピクチャの場合、変数名を入力します。

  ■ PICTリソースから参照するピクチャの場合、コロンに続けてその番号を入力します（例：“:62500”）。

■ バックグラウンド=ボタンに関連付けられる背景ピクチャ（カスタムスタイル）。ピクチャライブラリ、ピクチャ変数、またはPICTリソースから参照します（上記参照）。

■ タイトル位置=ボタンタイトルの位置。5つの値を指定できます。

  ■ タイトル位置=0：中央

  ■ タイトル位置=1：右

  ■ タイトル位置=2：左

  ■ タイトル位置=3：下

  ■ タイトル位置=4：上

■ タイトル表示=タイトルを表示するかどうかを指定します。2つの値を指定できます。

  ■ タイトル表示=0：タイトルを隠す

  ■ タイトル表示=1：タイトルを表示する

■ アイコン表示=アイコンを表示するかどうかを指定します。2つの値を指定できます。

■ アイコン表示=0：アイコンを隠す

■ アイコン表示=1：アイコンを表示する

■ スタイル=ボタンのスタイル。このオプションの値により、他の各種オプションを考慮するかどうかが決定されます（例えば、バックグラウンド）。10種類の値を指定できます。

■ スタイル=0：なし

■ スタイル=1：背景のオフセット

■ スタイル=2：プッシュボタン

■ スタイル=3：ツールバーボタン

■ スタイル=4：カスタム

■ スタイル=5：円

■ スタイル=6：四角（システム小）

■ スタイル=7：Office XP

■ スタイル=8：ベベル

■ スタイル=9：角の丸いベベル

■ 水平マージン=水平マージン。ボタンの左マージンと右マージンの内側境界を定めるピクセル数（アイコンとテキストを配置してはならない領域）。

■ 垂直マージン=垂直マージン。ボタンの上マージンと下マージンの内側境界を定めるピクセル数（アイコンとテキストを配置してはならない領域）。

■ アイコンオフセット=右方向または下方向へのアイコンの移動。この値はピクセル単位で表わされ、ボタンのクリック時に、ボタンアイコンを右方向および下方向へ移動することを指示します（両方向ともに同じ値が使用されます）。

■ ポップアップメニュー=ポップアップメニューとボタンとの組み合わせ。3つの値を指定できます。

■ ポップアップメニュー=0：ポップアップメニューなし

■ ポップアップメニュー=1：リンクしたポップアップメニューあり

■ ポップアップメニュー=2：分離したポップアップメニューあり

これらのオプションのなかには、すべての3Dボタンスタイルで使用できないものもあります。また、状況によっては、一部のオプションだけを変更したい場合もあります。任意のオプションを渡さないようにするには、対応する値を省略してください。例えば、「タイトル表示」と「垂直マージン」オプションを渡さないようにするには、次のように記述します。

```
SET FORMAT(myVar;"NiceButton;?256;:562;1;;1;4;5;;5;0")
```

例題

1. 以下のコードは"[Employee] Date Hired"のフィールドをMM DD YYYY Forcedにフォーマットします。

```
SET FORMAT ([Employee]Date Hired; Char(Month Date Year))
```

2. 以下の例は、郵便番号の長さによってフィールド“[会社]郵便番号”に対するフォーマットを変更します。

```
If(Length([会社]郵便番号)=5)
    SET FORMAT([会社]郵便番号 ; "###-##")
Else
    SET FORMAT([会社]郵便番号 ; "###")
End if
```

3. 以下の例は、ブールフィールドをデフォルトの“Yes（真）”と“No（偽）”ではなく、“既婚”と“未婚”を表示するためにフォーマットをセットします。

```
SET FORMAT([従業員]既婚歴 ; "既婚;未婚")
```

4. 以下の例では"Classified"というチェックボックスを表示するためのプールフィールドのフォーマットを設定します。

```
SET FORMAT([Folder]Classification; "Classified")
```

5. 以下の例ではピクチャボタンPictureButtonに行１行と列４列（"default"、"clicked"、"roll over"および"disabled"）のサムネール（ピクチャライブラリID=15000）と「Switch when roll Over」、「Switch back when Released」および「Use Last Frame as Disabled」のオプションを割り当てています。（16+32+128=176）

```
SET FORMAT(*;"PictureButton"; "4;1;?15000;176")
```

参照

Get format、SET FILTER

SET FILTER

SET FILTER({*;} オブジェクト ; 入力フィルタ)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
入力フィルタ	文字列	→	入力エリアに使用する新しい入力フィルタ

説明

SET FILTERコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの入力フィルタを引数＜入力フィルタ＞に変更します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

SET FILTERコマンドは、入力フォームおよびダイアログ用フォームに対して使用でき、「デザイン」モードで入力フィルタを受け付けるフィールドや入力可変数に適用できます。

引数＜入力フィルタ＞に空の文字列を指定すると、オブジェクトのカレント入力フィルタを消去します。

注：このコマンドは、サブフォームのリストフォームに配置されたフィールドには使用できません。

注：引数＜入力フィルタ＞で「データベースプロパティ」ダイアログボックスであらかじめ定義した入力フィルタを使用するには、入力フィルタ名の前に縦棒（¦）を挿入します。

例題

1. 以下の例は、郵便番号フィールドに対する文字フィルタを設定します。住所が米国の場合は、米国の郵便番号フィルタを設定します。それ以外の場合は、任意の入力ができるように設定します。

```
If([国]国名="US")  ` 郵便番号フォーマットのフィルタを設定
    SET FILTER([会社]郵便番号 ; "&9#####")
Else  ` 文字、数値、大文字を受け入れるためフィルタを設定
    SET FILTER([会社]郵便番号 ; "~@")
End if
```

2. 以下の例は、フィールド“フィールド”内に入力できる2文字を“a”、“b”、“c”、“g”に制限します。

```
$フィルタ:="&" + Char(Double quote) + "a;b;c;g" + Char(Double quote) + "##"
SET FILTER([テーブル]フィールド ; $フィルタ)
```

注：この例では、入力フィルタに“&"a;b;c;g"##”を設定します。

参照

SET FORMAT

SET CHOICE LIST

SET CHOICE LIST({*;} オブジェクト ; リスト)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
リスト	文字列	→	選択リストに使用するリスト名 （「デザイン」モードで定義）

説明

SET CHOICE LISTコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの選択リストを引数＜リスト＞で渡した階層リスト（「デザイン」モードのリストエディタで定義）に変更します。

このコマンドは、入力フォームまたはダイアログ用フォームにおいて、値がテキストとして入力されるフィールドおよび入力可変数に対して適用できます。データの入力中にユーザがテキストエリアを選択すると、リストが表示されます。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

注：このコマンドは、サブフォームのリストフォームに配置されたフィールドには使用できません。

例題

▼以下の例は、“船積み業者”フィールドに対する選択項目リストを設定します。船積みを夜間に行う場合に、夜間に船積みすることができる船積み業者のリストを選択項目リストに設定します。それ以外の場合は通常の船積み業者に設定します。

```
If([船積み]夜間)
    SET CHOICE LIST([船積み]船積み業者 ; "夜間可")
Else
    SET CHOICE LIST([船積み]船積み業者 ; "通常")
End if
```

SET ENTERABLE

SET ENTERABLE({*;} 入力エリア; 入力可)

引数	タイプ		説明
*	*	→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞はフィールドまたは変数
入力エリア	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、またはテーブル、フィールド、変数のいずれか（“*”を省略した場合）
入力可	ブール	→	True：入力可、False：入力不可

説明

SET ENTERABLEコマンドは、引数＜オブジェクト＞を入力可または入力不可に設定します。

任意の引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。任意の引数＜*＞を省略した場合は、＜オブジェクト＞にはテーブル、フィールド、変数のいずれかを指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドを使用することは、「フォーム」エディタの「プロパティリスト」ウインドウでフィールドや変数に対し「入力可」を選択することと同じです。このコマンドは、サブフォームのオブジェクトメソッド内で使用されている場合にのみ、そのサブフォーム内で機能します。

引数＜入力エリア＞が入力可能（True）であれば、ユーザはそのエリアにカーソルを移動、、またはデータを入力することができます。引数＜入力エリア＞が入力不可（False）の場合、ユーザはそのエリアにカーソルを移動、またはデータを入力することはできません。

また、**MODIFY SELECTION**や**DISPLAY SELECTION**コマンドを用いて表示されたサブフォーム、およびリストフォームに対して、プログラムから「リスト更新可」オプションを有効にするために、**SET ENTERABLE**コマンドを使用することもできます。

■ サブフォームの場合、例えば**SET ENTERABLE**(*;"Subform";True)のように、サブフォームテーブル名またはサブフォームオブジェクト自体の名前を引数＜入力エリア＞に渡すことができます。

■ リストフォームの場合、例えば**SET ENTERABLE**([MyTable];True)のように、引数＜入力エリア＞にはフォームのテーブル名を渡さなくてはなりません。

オブジェクトを入力不可にしても、プログラムから値が変更できなくなるわけではありません。

例題

1. 以下の例は、船積みの重量に応じて、船積みフィールドを設定します。船積みが1オンス以下の場合は、船積み業者に米国郵便を設定し、このフィールドを入力不可にします。それ以外の場合には、入力可に設定します。

```
If([船積み]重量<=1)
    [船積み]船積み業者:="米国郵便"
    SET ENTERABLE([船積み]船積み業者 ; False)
Else
    SET ENTERABLE([船積み]船積み業者 ; True)
End if
```

2. 次の例題は、リストのヘッダに配置されたチェックボックスのオブジェクトメソッドであり、リスト更新モードを制御します。

```
C_BOOLEAN(bEnterable)
SET ENTERABLE([Table1];bEnterable)
```

参照

DISABLE BUTTON、ENABLE BUTTON、SET VISIBLE

SET VISIBLE

SET VISIBLE({* ;} オブジェクト ; 表示)

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクト名（文字列） 省略した場合、フィールドまたは変数
オブジェクト	フォーム オブジェクト	→	オブジェクト名（*を指定した場合）、 またはフィールドまたは変数（*を省略した場合）
表示	ブール	→	表示の場合はTrue、非表示の場合はFalse

説明

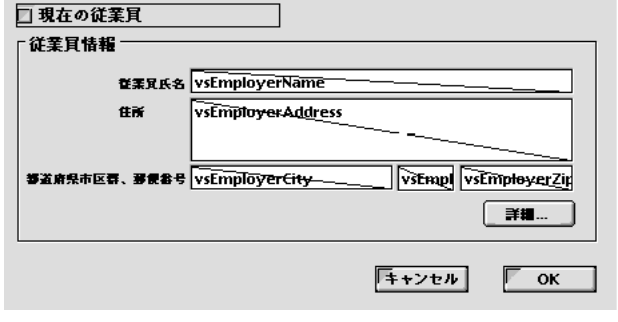
SET VISIBLEコマンドは、引数<オブジェクト>によって指定されるオブジェクトを表示、あるいは隠します。

オプション引数の<*>を指定した場合は、引数<オブジェクト>にオブジェクト名(文字列)を指定します。省略した場合は、引数<オブジェクト>にフィールドまたは変数を指定します。この場合、文字列の代わりに、フィールドまたは変数参照（フィールドまたは変数オブジェクトのみ）を指定してください。

引数<表示>にTrue渡すと、オブジェクトが表示されます。引数<表示>にFalseを渡すと、オブジェクトが隠されます。

例題

以下の図は、「デザイン」モードにおける標準的なフォームです。

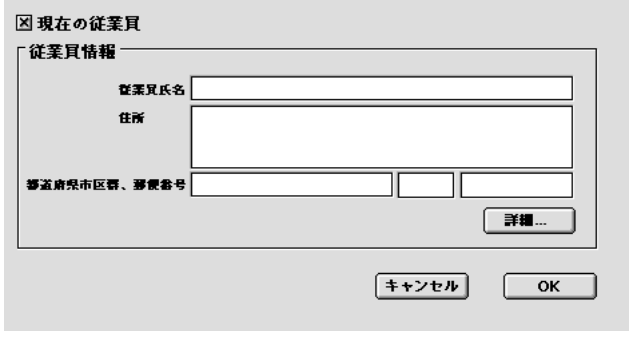


「従業員情報」グループボックスにあるオブジェクトは、(グループボックスを含めて)どれもオブジェクトの名前に“Employer”という文字が入っています。「現在の従業員」チェックボックスをオンにすると、オブジェクトが表示されます。チェックボックスをオフにすると、オブジェクトが表示されなくなります。

以下は、チェックボックスのオブジェクトメソッドです。

```
`「現在の従業員」(cbCurrentlyEmployed)チェックボックスのオブジェクトメソッド
Case of
    ¥(Form event=On Load)
        cbCurrentlyEmployed:=1
    ¥(Form event=On Clicked)
        `オブジェクト名に“Emp”を含んだオブジェクトをすべて表示
                                                または非表示する
        SET VISIBLE(* ; "@emp@" ; cbCurrentlyEmployed # 0)
        `ただし、常にチェックボックス自身は表示されていなければならない
        SET VISIBLE(cbCurrentlyEmployed ; True)
End case
```

したがって、「ユーザ」モードまたは「カスタム」モードでは、フォームは以下のように表示されます。



または、



参照

DISABLE BUTTON、ENABLE BUTTON、SET ENTERABLE

SET COLOR

SET COLOR({*; }オブジェクト; カラー{; 代替色})

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞はフィールドまたは変数
オブジェクト	フィールドまたは変数	→	オブジェクト名（“*”を指定した場合）、またはフィールドまたは変数（“*”を省略した場合）
カラー	数値	→	オブジェクトの新しいカラー
代替色	数値	→	リストボックスの代替色

説明

SET COLORコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトの描画色と背景色を設定します。＜オブジェクト＞がリストボックスである場合は、引数を更に使用して偶数行の描画色と背景色を設定します（代替色）。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指定します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

＜代替色＞を使用し、リストボックスやリストボックスの列の偶数行に代替色を指定することができます。この引数を渡すと、引数＜カラー＞は奇数行にのみ適用されます。代替色を使用すると、リストが読みやすくなります。

＜オブジェクト＞により、リストボックスオブジェクトが指定された場合、代替色はリストボックス全体に対して使用されます。＜オブジェクト＞により、任意の列が指定された場合、設定した色はそのカラムに対してだけ使用されます。

引数＜カラー＞（および＜代替色＞）は、描画色と背景色を設定します。カラーは以下の計算式で求められます。

カラー:= -(描画色 +(256 * 背景色))

描画色と背景色はカラーパレット内のカラー番号（0から255まで）です。引数＜カラー＞は、常に負の数値です。例えば、前景色が20で、背景色が10の場合、＜カラー＞は、-(20 +(256 * 10))または-2580となります。

注：「フォーム」エディタの「プロパティリスト」ウインドウでカラーパレットを見ることができます。

一般的に使用されるカラーの番号は、「Colors」テーマにある次の定義済み定数により提供されています。

定数	タイプ	値
White	倍長整数	0
Yellow	倍長整数	1
Orange	倍長整数	2
Red	倍長整数	3
Purple	倍長整数	4
Dark Blue	倍長整数	5
Blue	倍長整数	6
Light Blue	倍長整数	7
Green	倍長整数	8
Dark Green	倍長整数	9
Dark Brown	倍長整数	10
Dark Grey	倍長整数	11
Light Grey	倍長整数	12
Brown	倍長整数	13
Grey	倍長整数	14
Black	倍長整数	15

注：SET COLORコマンドがデフォルトの4th Dimensionカラーパレットのインデックスカラーを使って作業するのに対して、バージョン6で新しく追加されたSET RGB COLORSコマンドは任意のRGBカラーで作業することができます。

例題

▼ 以下の例は、“MyButton”という名前のButtonに対して色を設定します。この色は“v描画色”と“v背景色”という名前の変数に値を設定しています。

```
SET COLOR(MyButton; -(v描画色 +(256 * v背景色)))  ` MyButtonに色を設定
```

参照

SET RGB COLORS

SET RGB COLORS

SET RGB COLORS({*; }オブジェクト; 描画色; 背景色{; 代替背景色})

引数	タイプ		説明
*		→	指定した場合、＜オブジェクト＞はオブジェクト名（文字列） 省略した場合、＜オブジェクト＞はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、またはフィールドまたは変数（“*”を省略した場合）
描画色	数値	→	描画色のRGBカラーの値
背景色	数値	→	背景色のRGBカラーの値
代替背景色	数値	→	代替背景色のRGBカラーの値

説明

SET RGB COLORSコマンドは、引数＜オブジェクト＞とオプション引数の＜*＞によって指定されるオブジェクトの描画色と背景色を変更します。

オプション引数の＜*＞を指定した場合は、引数＜オブジェクト＞にオブジェクト名（文字列）を指定します。省略した場合は、引数＜オブジェクト＞にフィールドまたは変数を指定します。コマンドがリストボックスに対して適用される場合、引数を更に使用して行の代替色を変更することができます。

任意の引数である＜代替背景色＞により、偶数行の代替背景色を設定することができます。この引数は、指定されたオブジェクトがリストボックスまたはリストボックスの列である場合にのみ使用できます。この引数を使用した場合、引数＜背景色＞は奇数行に対してのみ使用されます。代替色を使用すると、リストがより読みやすくなります。

＜オブジェクト＞により、リストボックスオブジェクトが指定された場合、代替色はリストボックス全体に対して使用されます。＜オブジェクト＞により、リストボックスの任意の列が指定された場合、設定した色はそのカラムに対してだけ使用されます。

引数＜描画色＞、＜背景色＞、＜代替背景色＞にはRGBのカラー値を指定します。RGBの値は、4バイトの倍長整数です。そのフォーマット（0x00RRGGBB）は、以下の表で説明します（バイトには、右から左へ順に0から3までの数字が付けられます）。

バイト	説明
3	絶対RGBカラーの場合は、ゼロでなければなりません。
2	色のうちの赤の成分（0～255）
1	色のうちの緑の成分（0～255）
0	色のうちの青の成分（0～255）

以下の表は、RGBのカラー値の例をいくつか示します。

値	説明
0x00000000	黒
0x00FF0000	赤
0x0000FF00	緑
0x000000FF	青
0x007F7F7F	グレー
0x00FFFF00	黄
0x00FF7F7F	赤（パステル）
0x00FFFFFF	白

この代わりに、4th Dimensionがオブジェクトの描画時に色を自動設定するために使用する“システム”カラーのいずれかを指定することができます。4th Dimensionでは、以下のような定数があらかじめ定義されています。

定数	タイプ	値
Foreground color	倍長整数	-1
Background color	倍長整数	-2
Dark shadow color	倍長整数	-3
Light shadow color	倍長整数	-4
Highlight text background color	倍長整数	-7
Highlight text color	倍長整数	-8
Highlight menu background color	倍長整数	-9
Highlight menu text color	倍長整数	-10
Disable highlight item color	倍長整数	-11

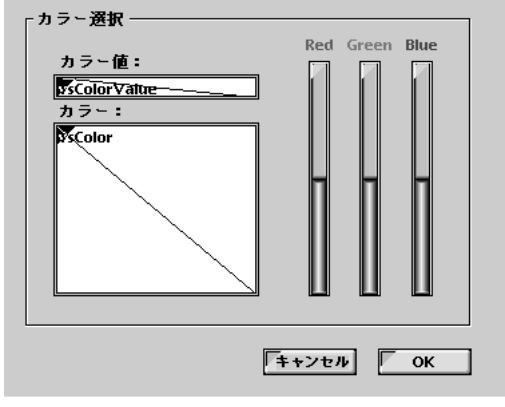
（標準システムにおける）これらの色を次に示します。

Windows		Mac OS
	Foreground color	
	Background color	
	Dark shadow color	
	Light shadow color	
	Highlight text background color	
	Highlight text color	
	Highlight menu background color	
	Highlight menu text color	
	Disable highlight item color	

注：これらの自動色はシステムに依存します。システム色を変更すると、4th Dimensionは、それに従ってその自動色を調整します。自動色の値を使用すると、オブジェクトは上に示すサンプル色ではなく、システム色で設定されます。

例題

以下のフォームには、「vsColorValue」と「vsColor」という2つの入力不可変数と「thRed」、「thGreen」、「thBlue」という 3つのサーモメータが含まれます。



以下は、このオブジェクト用のメソッドです。

```
`「vsColorValue」入力不可変数のオブジェクトメソッド
Case of
	¥(Form event=On Load)
		vsColorValue:="0x00000000"
End case
`「vsColor」入力不可変数のオブジェクトメソッド
Case of
	¥(Form event=On Load)
```

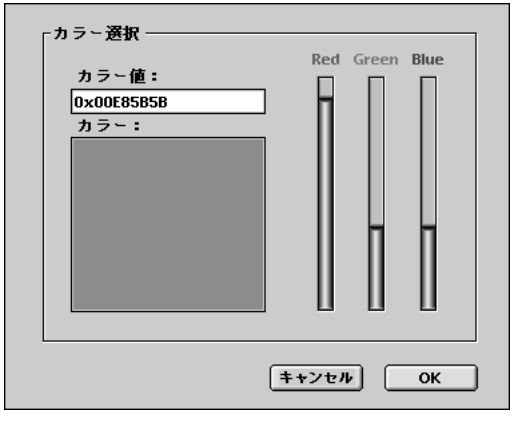
```
            vsColor:=""
            SET RGB COLOR(vsColor ; 0x00FFFFFF ; 0x0000)
    End case
    `「thRed」サーモメータのオブジェクトメソッド
    CLICK IN COLOR THERMOMETER
    `「thGreen」サーモメータのオブジェクトメソッド
    CLICK IN COLOR THERMOMETER
    `「thBlue」サーモメータのオブジェクトメソッド
    CLICK IN COLOR THERMOMETER
```

3つのサーモメータによって呼び出されるプロジェクトメソッドは、以下のようになります。

```
`「CLICK IN COLOR THERMOMETER」プロジェクトメソッド
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
```

サーモメータの値からカラーの値を計算するために、ビットワイズ演算子が使用されている点に注意してください。

「ユーザ」モードまたは「カスタム」モードでは、フォームが以下のようになります。



参照

ビットワイズ演算子、SET COLOR

GET OBJECT RECT

GET OBJECT RECT({*;} オブジェクト ; 左 ; 上 ; 右 ; 下)

引数	タイプ		説明
*	*	→	指定した場合=オブジェクトは オブジェクトの名前（文字列） 省略した場合=オブジェクトは変数
オブジェクト	オブジェクト	→	オブジェクト名（*が指定された場合） またはフィールドまたは変数 （*が省略された場合）
左	倍長整数	←	オブジェクトの左の座標
上	倍長整数	←	オブジェクトの上の座標
右	倍長整数	←	オブジェクトの右の座標
下	倍長整数	←	オブジェクトの下の座標

説明

このコマンドは、引数*およびオブジェクトによって指定された、現在のフォームのオブジェクトの左、上、右、下の座標（ポイント）を返します。

オプション引数*を指定すると、オブジェクト引数がオブジェクト名（文字列）であることを示し、オプション引数*を省略すると、オブジェクト引数がフィールドまたは変数であることを示します。この場合、文字列ではなくフィールドまたは変数の参照（オブジェクトタイプがフィールドまたは変数のみ）を指定します。

オブジェクトにオブジェクト名としてワイルドカード（@）を使用し、複数のオブジェクトを指定すると、返される座標は関連する全オブジェクトで形成される長方形の座標になります。

注：バージョン6.5からは、文字列に含まれるワイルドカード文字（@）の取り扱い方を設定することができます。このオプションは、「オブジェクトプロパティ」コマンドに影響を与えます。

オブジェクトが存在しない場合やコマンドがフォーム内で呼び出されていない場合、座標(0;0;0;0)が返されます。

例題

▼ "button"で始まるすべてのオブジェクトによって形成される長方形の座標を得たい場合を仮定します。

```
GET OBJECT RECT(*;"button@";left;top;right;botton)
```

参照

MOVE OBJECTR

MOVE OBJECT

MOVE OBJECT({*;} オブジェクト ; 水平移動 ; 垂直移動 ;{; 水平リサイズ {; 垂直リサイズ}}{; *})

引数	タイプ		説明
*	*	→	指定された場合=オブジェクトは オブジェクトの名前（文字列） 省略された場合=オブジェクトは変数
オブジェクト	オブジェクト	→	オブジェクト名（*が指定されている場合） またはフィールドまたは変数 （*が省略された場合）
水平移動	倍長整数	←	オブジェクトの水平移動距離 （>0=右へ、<0=左へ）
垂直移動	倍長整数	←	オブジェクトの垂直移動距離 （>0=下へ、<0=上へ）
水平リサイズ	倍長整数	←	オブジェクトの水平方向のサイズ変更値
垂直リサイズ	倍長整数	←	オブジェクトの垂直方向のサイズ変更値
*	*		指定されている場合=絶対座標 省略された場合=相対座標

説明

このコマンドは、*とオブジェクトの引数で定義された、現在のフォーム内のオブジェクトを、水平方向にピクセル、垂直方向にピクセル移動させます。また、（オプションで）オブジェクトを水平方向に水平リサイズピクセル、垂直方向にピクセル、サイズの変更をすることもできます。

移動とサイズ変更の方向は、水平移動および垂直移動引数に渡された値に依ります。

- 値が正であれば、オブジェクトは右および下へそれぞれ移動され、サイズ変更されます。
- 値が負であれば、オブジェクトは左および上へそれぞれ移動され、サイズ変更されます。

最初のオプション引数<*>を指定すると、オブジェクト引数がオブジェクト名（文字列）であることを示し、最初のオプション引数 * を省略すると、オブジェクト引数がフィールドまたは変数であることを示します。この場合、文字列ではなくフィールドまたは変数の参照（オブジェクトタイプがフィールドまたは変数のみ）を指定します。

オブジェクトにオブジェクト名としてワイルドカード（@）を使用し、複数のオブジェクトを指定すると、関連する全オブジェクトが移動またはサイズが変更されます。

注：バージョン6.5からは、文字列に含まれるワイルドカード文字（@）の取り扱い方を設定することができます。このオプションは、「オブジェクトプロパティ」コマンドに影響を与えます。

デフォルトでは、水平移動、垂直移動、水平リサイズ、垂直リサイズの値は、オブジェクトの以前の位置からの相対的な値です。引数が絶対位置を表わすようにしたい場合は、最後のオプションの引数<*>を渡します。

このコマンドは次のコンテクストで働きます。

■ **DIALOG**コマンドを使用して表示されたフォーム

■ インプットフォームのデータ入力

■ **MODIFY SELECTION**もしくは**DISPLAY SELECTION**コマンドで表示される出力フォームのヘッダーとフッター

■ フォーム出力イベント

例題

▼ 下記のコードは、"button_1"を右に10ピクセル、上に10ピクセル移動させ、幅を30ピクセル、高さを40ピクセルにサイズ変更します。

```
MOVE OBJECT(*;"button_1";10;-20;30;40)
```

▼ 下記のコード=は、"button_1"をを以下の座標に移動します：(10;20)(30;40)

```
MOVE OBJECT(*;"button_1";10;20;30;40;*)
```

参照

GET OBJECT RECT

BEST OBJECT SIZE

BEST OBJECT SIZE({*;}オブジェクト; 最適幅; 最適高さ{; 最大幅})

引数	タイプ		説明
*	*	→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、オブジェクトは変数
オブジェクト	オブジェクト	→	オブジェクト名（* を指定した場合） または、フィールドまたは変数 （* を省略した場合）
最適幅	倍長整数	←	オブジェクトの最適な幅
最適高さ	倍長整数	←	オブジェクトの最適な高さ
最大幅	倍長整数	→	オブジェクトの最大幅

説明

BEST OBJECT SIZEコマンドは、引数＜*＞と＜オブジェクト＞で指定されたフォームオブジェクトの“最適な”幅と高さを、引数＜最適幅＞と＜最適高さ＞に納めて返します。これらの値は、ピクセルで表わされます。

現在のコンテンツが制限内にすべて含まれるように、返される最適値はオブジェクトの最小サイズを表わします。もちろん、これらの値はテキストを含むオブジェクトに関してのみ意味を持ちます。この計算には、フォント、フォントサイズ、フォントスタイルおよびオブジェクト内容が考慮されます。さらに、ハイフンや改行も考慮されます。指定された＜オブジェクト＞が空の場合、＜最適幅＞には0が返されます。

返されるサイズは、オブジェクトの周囲に貼り付けられたグラフィックフレームやスクロールバーを計算に入れていません。画面上のオブジェクトの実際のサイズを取得するには、これらの要素の幅を加算する必要があります。

任意の引数＜最大幅＞により、オブジェクトに最大幅を割り当てることができます。オブジェクトの最適な幅がこの値よりも大きい場合、**BEST OBJECT SIZE**コマンドは＜最大幅＞を＜最適幅＞に代入して返し、この結果として最適な高さを大きくします。

任意の引数＜*＞を渡すと、引数＜オブジェクト＞がオブジェクト名（文字列）であることを表わします。引数＜*＞を渡さない場合、＜オブジェクト＞はフィールドまたは変数となります。この場合には、文字列ではなくフィールドまたは変数への参照（オブジェクトタイプのみ）を渡してください。

このコマンドは、次のオブジェクトを処理することができます。

■ スタティックテキストエリア

■ 参照するフォームに挿入されたテキスト

■ 次のタイプのフィールドや変数：文字、テキスト、実数、整数、倍長整数、日付、時間、ブール（チェックボックスとラジオボタン）

■ ボタン

この他のオブジェクトタイプ（グループエリア、タブ、矩形、直線、円／楕円、プラグインエリア等）に対して、**BEST OBJECT SIZE**コマンドは現在のオブジェクトサイズ（「フォーム」エディタや**MOVE OBJECT**コマンドで指定）を返します。

例題

▼ 後述の**SET PRINT MARKER**コマンドの例題を参照してください。

Get alignment

Get alignment({*; }オブジェクト) → 数値

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合）、または フィールドまたは変数（*を省略した場合）
戻り値	数値	←	整列コード

説明

Get alignment関数は、引数<オブジェクト>および<*>で指定されたオブジェクトに適用された整列タイプを示すコードを返します。

オプション引数である<*>を指定した場合、引数<オブジェクト>にはオブジェクト名（文字列）を指定します。引数<*>を省略した場合は、引数<オブジェクト>にフィールドまたは変数を指定します。その場合、文字列ではなくフィールドまたは変数の参照（フィールドまたは変数オブジェクトのみ）を指定します。

注：この関数を一連のオブジェクトに対して適用すると、最後のオブジェクトの整列値だけが返されます。

返されるコードは、「Object alignment」テーマ内にある以下の定数のいずれかとなります。

定数	タイプ	値
Align default	倍長整数	1
Align left	倍長整数	2
Center	倍長整数	3
Align right	倍長整数	4

整列を適用できるフォームオブジェクトは次の通りです。

■ スクロールエリア

■ コンボボックス

■ スタティック・テキスト

■ ポップアップメニュー／ドロップダウンリスト

■ フィールド

■ 変数

参照

SET ALIGNMENT

SET ALIGNMENT

SET ALIGNMENT({*; }オブジェクト) → 数値

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合）、または フィールドまたは変数（*を省略した場合）
戻り値	数値	←	整列コード

説明

SET ALIGNMENTコマンドは、引数<オブジェクト>および“*”で指定されたオブジェクトに適用するアラインメントを設定します。

オプション引数である<*>を指定した場合、引数<オブジェクト>にはオブジェクト名（文字列）を指定します。引数<*>を省略した場合は、引数<オブジェクト>にフィールドまたは変数を指定します。その場合、文字列ではなくフィールドまたは変数の参照（フィールドまたは変数オブジェクトのみ）を指定します。

「Object alignment」テーマ内にある以下の定数の1つを渡します。

定数	タイプ	値
Align default	倍長整数	1
Align left	倍長整数	2
Center	倍長整数	3
Align right	倍長整数	4

整列を適用できるフォームオブジェクトは次の通りです。

- スクロールエリア
- コンボボックス
- スタティック・テキスト
- ポップアップメニュー／ドロップダウンリスト
- フィールド
- 変数

参照

Get alignment

リスト（統計）上の関数 32

この章では、統計関数について説明します。

Average	**Std deviation**	**Sum squares**
Max	**Sum**	**Variance**
Min		

統計関数は、数値フィールドに関する計算を行います。

Average、**Max**、**Min**、**Sum**、**Sum squares**、**Std deviation**、そして**Variance**の各関数はフィールドやサブフィールドに使用されます。フィールドに用いる場合、レコードセレクションに使用されます。サブフィールドに用いる場合には、カレントレコードのサブレコードのセレクションに使用されます。**Sum squares**、**Std deviation**、**Variance**の各関数は印刷時のみにフィールドで使用されます。

これらの関数は、数値データに対してだけ機能します。これらの関数は、すべて数値を返します。

フィールドを使用する

プリント処理以外で**Average**、**Max**、**Min**、**Sum**の各関数をフィールドに対して使用すると、結果を算出するためにカレントセレクションのレコードをロードする必要があります。レコード数が多いと、処理に時間がかかります。これを避けるには、フィールドにインデックスを付けます。

レポート作成時にこれらの関数を使用すると、他の場合とは異なる動作をします。印刷時には、レコードを毎回ロードしなければならないためです。**PRINT SELECTION**コマンドによって印刷する、あるいは「ユーザ」モードで「ファイル」メニューの「プリント...」を選択して印刷する場合に、フォームメソッドやオブジェクトメソッドの中でこれらの関数を使用します。

レポートに統計関数を使用する際に、戻り値が意味を持つのは、ブレーク処理がオンの場合でフブレークレベルが0の時だけです。つまり、これらの関数はレポートの最後、すべてのレコードを処理し終えた時にのみ有効です。

通常、これらの関数は、B0ブレークエリアにある入力不可エリアのオブジェクトメソッド内でのみ使用します。

統計関数に引数として渡されるフィールドは、数値フィールドでなければならないことを覚えておいてください。

参照

Average、Max、Min、Std deviation、Sum、Sum Squares、Variance

Sum

Sum(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	合計を得るためのデータ
戻り値	数値	←	数値フィールドの合計

説明

Sum関数は、＜数値フィールド＞の合計値を返します。＜数値フィールド＞がインデックスフィールドの場合は、合計値を求めるためにインデックスが使用されます。

例題

▼以下の例は、フォーム上の変数“v合計”のオブジェクトメソッドです。変数に従業員全員の給与合計を代入します。

```
v合計:=Sum([従業員]給与)
```

▼以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([従業員])
ORDER BY([従業員] ; [従業員]名字 ; >)
BREAK LEVEL(1)
ACCUMULATE([従業員]給与)
OUTPUT FORM([従業員] ; "印刷")
PRINT SELECTION([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

ACCUMULATE、Average、BREAK LEVEL、Max、Min、ORDER BY、PRINT SELECTION、Subtotal

Average

Average(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	平均を計算するためのデータ
戻り値	数値	←	数値フィールドの平均値

説明

Average関数は、＜数値フィールド＞の平均値を返します。＜数値フィールド＞がインデックスフィールドの場合に、平均値を求めるためにインデックスが使用されます。

例題

▼ 以下の例は、出力フォームのB0ブレークエリアにある変数に値を代入します。このコードは変数“v平均売上”のオブジェクトメソッドです。オブジェクトメソッドは、レベル0のブレークが発生したときに実行されます。

```
v平均売上:=Average([従業員]給与)
```

▼ 以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([従業員])
ORDER BY([従業員] ; [従業員]名字 ; >)
BREAK LEVEL(1)
ACCUMULATE([従業員]給与)
OUTPUT FORM([従業員] ; "印刷")
PRINT SELECTION([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

ACCUMULATE、BREAK LEVEL、Max、Min、ORDER BY、PRINT SELECTION、Subtotal、Sum

Min

Min(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	最小値を得るためのデータ
戻り値	数値	←	数値フィールドの最小値

説明

Min関数は、＜数値フィールド＞の最小値を返します。＜数値フィールド＞がインデックスフィールドの場合、最小値を求めるためにインデックスが使用されます。

例題

1. 以下の例は、変数“v最小”のオブジェクトメソッドです。この変数は、フォームのブレーク0のエリアに配置されています。変数は、レポートの最後に印刷されます。オブジェクトメソッドは、フィールドの値の最小値を変数に代入します。この変数は、レポートの最後のブレークが発生したときに印刷されます。

```
v最小:=Min([従業員]給与)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([従業員])
ORDER BY([従業員] ; [従業員]名字 ; >)
BREAK LEVEL(1)
ACCUMULATE([従業員]給与)
OUTPUT FORM([従業員] ; "印刷")
PRINT SELECTION([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

2. 以下の例は、従業員売上の最小値を求め、結果をアラートボックスに表示します。売上高は、サブフィールド[社員]売上'金額に格納されています。

```
ALERT("売上高の最低は" + String(Min([社員]売上'金額)) + "円です。")
```

参照

Execute on server、Execute on server、GET PROCESS VARIABLE、Max、 Processes、SET PROCESS VARIABLE

Max

Max(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	最大値を得るためのデータ
戻り値	数値	←	数値フィールドの最大値

説明

Max関数は、＜数値フィールド＞の最大値を返します。＜数値フィールド＞がインデックスフィールドの場合に、最大値を求めるためにインデックスが使用されます。

例題

▼ 以下の例は、変数“v最大”のオブジェクトメソッドです。この変数は、フォームのブレーク0のエリアに配置されています。変数は、レポートの最後に印刷されます。オブジェクトメソッドは、フィールドの値の最大値を変数に代入します。この変数は、レポートの最後のブレークが発生したときに印刷されます。

```
v最大:=Max([従業員]給与)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([従業員])
ORDER BY([従業員] ; [従業員]名字 ; >)
BREAK LEVEL(1)
ACCUMULATE([従業員]給与)
OUTPUT FORM([従業員] ; "印刷")
PRINT SELECTION([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

Min

Std deviation

Std deviation(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	標準偏差を得るためのデータ
戻り値	数値	←	数値フィールドの標準偏差

説明

Std deviation関数は、＜数値フィールド＞の標準偏差を返します。＜数値フィールド＞がインデックスフィールドの場合に、標準偏差を求めるためにインデックスが使用されます。この関数は、レポートの印刷時に、フィールドに対してのみ使用することができます。

例題

▼以下の例は、変数“v偏差”のオブジェクトメソッドです。一連のデータに関する標準偏差を変数“v偏差”に代入します。

```
v偏差:=Std deviation([テーブル1]一連データ)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([テーブル1])
ORDER BY([テーブル1] ; [テーブル1]一連データ ; >)
BREAK LEVEL(1)
ACCUMULATE([テーブル1]一連データ)
OUTPUT FORM([テーブル1] ; "印刷")
PRINT SELECTION([テーブル1])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

Average、Sum、Sum Squares、Variance

Variance

Variance(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	分散を得るためのデータ
戻り値	数値	←	数値フィールドの分散

説明

Variance関数は、＜数値フィールド＞の分散を返します。＜数値フィールド＞がインデックスフィールドの場合に、分散を求めるためにインデックスが使用されます。この関数は、レポートの印刷時に、フィールドに対してのみ使用することができます。

例題

▼ 以下の例は、変数“v分散”のオブジェクトメソッドです。一連のデータに関する分散を変数“v分散”に代入します。

```
v分散:=Variance([学生]成績)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([学生])
ORDER BY([学生] ; [学生]クラス ; >)
BREAK LEVEL(1)
ACCUMULATE([学生]成績)
OUTPUT FORM([学生] ; "印刷")
PRINT SELECTION([学生])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

Average、Std deviation、Sum、Sum squares

Sum squares

Sum squares(数値フィールド) → 数値

引数	タイプ		説明
数値フィールド	フィールドまたはサブフィールド	→	平方和を得るためのデータ
戻り値	数値	←	数値フィールドの平方和

説明

Sum squares関数は、＜数値フィールド＞の平方和を返します。＜数値フィールド＞がインデックスフィールドの場合、平方和を求めるためにインデックスが使用されます。この関数は、レポート印刷時に、フィールドに対してのみ使用することができます。

例題

▼ 以下の例は、変数“v平方和”のオブジェクトメソッドです。一連のデータに対する平方和を変数“v平方和”に代入します。変数“v平方和”はレポートの最後のブレークが発生したときに印刷されます。

```
v平方和:=Sum squares([ファイル]一連データ)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます。

```
ALL RECORDS([テーブル1])
ORDER BY([テーブル1] ; [テーブル1]一連データ ; >)
BREAK LEVEL(1)
ACCUMULATE([テーブル1]一連データ)
OUTPUT FORM([テーブル1] ; "印刷")
PRINT SELECTION([テーブル1])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

参照

Average、Std deviation、Sum、Variance

演算子

演算子とは、式の間で行われる操作を指定するために使用する記号です。演算子の機能は、以下のとおりです。

■ 数値、日付、時間の計算を行います。

■ 文字列操作、論理式における論理演算、図形上での特殊演算を行います。

■ 単純な式を組み合わせて新たな式を生成します。

優先順位

式を評価する順番を“優先順位”と呼びます。4th Dimensionの優先順位は厳密に左から右で、代数的順序は守られていません。次の例を見てみましょう。

```
3+4*5
```

上記の式は、35を返します。最初に式3+4の結果7を求め、それに5を乗じるので、結果は35になります。

左から右の優先順位を無視するためには、必ずカッコを使用します。

```
3+(4*5)
```

この式は、23を返します。カッコがあるため、最初に式（4*5）の結果20を求め、それに3を加えて、結果は23になります。

カッコは、他のカッコの組の内側にネストすることができます。式の評価が正しく行われるように、必ず各左カッコに対応する右カッコを指定してください。カッコが足りなかったり、誤って使用した場合は、予測できない結果、または式が無効になります。また、4D Compilerでアプリケーションをコンパイルする場合は、左カッコと右カッコは同じ数でなければなりません。コンパイラは、組になっていないカッコをシンタックスエラーとして検出します。

代入演算子

代入演算子（:=）は必ず他の演算子と区別してください。代入演算子は、右側の式の値を演算子の左側の変数やフィールドにコピーします。例えば、次のステートメントは、数値3（"Aci"の文字数）を変数“MyVar”に代入します。“MyVar”のデータタイプは数値です。

```
MyVar:=Length("Aci")
```

重要：代入演算子（:=）と等号比較演算子（=）とを混同しないでください。

文字列演算子

文字列演算子を使用する式は、文字列を返します。次の表に、文字列演算子を示します。

演算子	シンタックス	結果	例
連結 +	文字列 + 文字列	文字列	"abc" + "def" → "abcdef"
反復 *	文字列 * 数値	文字列	"ab" * 3 → "ababab"

数値演算子

数値演算子を使用する式は数値を返します。次の表に、数値演算子を示します。

演算子	シンタックス	結果	例
加算 +	数値 + 数値	数値	2 + 3 → 5
減算 -	数値 - 数値	数値	3 - 2 → 1
乗算 *	数値 * 数値	数値	5 * 2 → 10
除算 /	数値 / 数値	数値	5 / 2 → 2.5
整数値を返す除算 //	数値 // 数値	数値	5 // 2 → 2
モジューロ %	数値 % 数値	数値	5 % 2 → 1
指数 ^	数値 ^ 数値	数値	2 ^ 3 → 8

モジューロ演算子

モジューロ演算子（%）は最初の数値を2番目の数値で除算し、その余りの整数を返します。次に例を示します。

10%2は、0を返します。10は2で割り切れるからです。
10%3は、1を返します。余りが1だからです。
10.5%2は、0を返します。余りが整数ではない（0.25）からです。

警告：

・倍長整数の整除演算子//は、整数値に対して操作した場合にのみ、有効値を返します。

*・モジューロ演算子（%）は倍長整数の範囲内（-2^31から(2^31)-1まで）の数値に対して有効値を返します。この範囲外の数値のモジューロ演算を実行するには、**Mod**コマンドを使用します。*

日付演算子

日付演算子を使用する式は、その処理に応じて日付または数値を返します。日付演算はすべて、閏年も考慮に入れて正確な結果を返します。次の表に、日付演算子を示します。

演算子	シンタックス	結果	例
日付の差 -	日付 - 日付	数値	!96.01.20! - !96.01.01! → 19
日付の加算 +	日付 + 数値	日付	!96.01.20! + 9 → !96.01.29!
日付の減算 -	日付 - 数値	日付	!96.01.20! - 9 → !96.01.11!

時間演算子

時間演算子を使用する式は、その処理に応じて時間または数値を返します。次の表に、時間演算子を示します。

演算子	シンタックス	結果	例
加算 +	時間 + 時間	時間	!!02:03:04!! + !!01:02:03!! → !!03:05:07!!
減算 -	時間 - 時間	時間	!!02:03:04!! - !!01:02:03!! → !!01:01:01!!
加算 +	時間 + 数値	数値	!!02:03:04!! + 65 → 7449
減算 -	時間 - 数値	数値	!!02:03:04!! - 65 → 7319
乗算 *	時間 * 数値	数値	!!02:03:04!! * 2 → 14768
除算 /	時間 / 数値	数値	!!02:03:04!! / 2 → 3692
整数値を返す除算 //	時間 // 数値	数値	!!02:03:04!! // 2 → 3692
モジューロ %	時間 % 数値	数値	!!02:03:04!! % 2 → 0

Tips：

(1)時間式を数値と組み合わせた式から時間式を取得するには、TimeコマンドとTime stringコマンドを使用します。

次の行は、$vlSecondsに夜中の12時から今から1時間後までの秒数を割り当てます。

```
$vlSeconds:=Current Time+3600
```

次の行は、$vHSoonに今から1時間後の時間を割り当てます。

```
$vhSoon:=Time(Time string(Current time+3600))
```

2行目はより簡単な方法で書けます。

次の行は、$vHSoonに今から1時間後の時間を割り当てます。

```
$vhSoon:=Current time+!!01:00:00!!
```

ただし、アプリケーションを開発する際に、秒数で表された遅延時間を時間値に加えるといった必要があった場合、数値表現を用いることでのみ実現可能です。

この場合、次のヒントを参考にしてください。

(2)状況によっては、時間表現を数値表現に変換する必要があります。

例えば、**Open Document**関数を使用してドキュメントをオープンすると、ドキュメント参照（DocRef）が返されますが、これは元々時間式です。後で、ドキュメント参照として数値を受け取る4D ExtensionルーチンにDocRefを渡すとします。この場合、ゼロに加えることにより、時間値からその値を変更することなく数値を取得できます。

```
` ドキュメントを選択して、オープンする
$vhDocRef:=Open document("")
If(OK=1)
`数値表現としてのDocRef時刻表現を4D Extensionルーチンに渡します。
      DO SOMETHING SPECIAL(0+$vhDocRef)
End if
```

比較演算子

この節の表は文字列、数値、日付、時間、ポインタに適用する比較演算子を示しています。比較演算子を使用する式は、“True（真）”または“False（偽）”のブール値を返します。

文字列比較演算子	シンタックス	結果	例
等号(=)	文字列 = 文字列	ブール	"abc"="abc" → “True(真)” "abc"="abd" → “False(偽)”
不等号(#)	文字列 # 文字列	ブール	"abc"#"abd" → “True(真)” "abc"#"abc" → “False(偽)”
より大きい(>)	文字列 > 文字列	ブール	"abd">"abc" → “True(真)” "abc">"abc" → “False(偽)”
より小さい(<)	文字列 < 文字列	ブール	"abc"<"abd" → “True(真)” "abc"<"abc" → “False(偽)”
より大きいまたは等しい(>=)	文字列 >= 文字列	ブール	"abd">="abc" → “True(真)” "abc">="abd" → “False(偽)”
より小さいまたは等しい(<=)	文字列 <= 文字列	ブール	"abc"<="abd" → “True(真)” "abd"<="abc" → “False(偽)”

数値比較演算子	シンタックス	結果	例
等号(=)	数値 = 数値	ブール	10 = 10 → “True(真)” 10 = 11 → “False(偽)”
不等号(#)	数値 # 数値	ブール	10 # 11 → “True(真)” 10 # 10 → “False(偽)”
より大きい(>)	数値 > 数値	ブール	11 > 10 → “True(真)” 10 > 11 → “False(偽)”
より小さい(<)	数値 < 数値	ブール	10 < 11 → “True(真)” 11 < 10 → “False(偽)”
より大きいまたは等しい(>=)	数値 >= 数値	ブール	11 >= 10 → “True(真)” 10 >= 11 → “False(偽)”
より小さいまたは等しい(<=)	数値 <= 数値	ブール	10 <= 11 → “True(真)” 11 <= 10 → “False(偽)”

日付比較演算子	シンタックス	結果	例
等号(=)	日付 = 日付	ブール	!96.01.01!=!96.01.01!→ “True(真)” !96.01.20!=!96.01.01!→ “False(偽)”
不等号(#)	日付 # 日付	ブール	!96.01.20!#!96.01.01!→ “True(真)” !96.01.01!#!96.01.01!→ “False(偽)”
より大きい(>)	日付 > 日付	ブール	!96.01.20!>!96.01.01!→ “True(真)” !96.01.01!>!96.01.01!→ “False(偽)”
より小さい(<)	日付 < 日付	ブール	!96.01.01!<!96.01.20!→ “True(真)” !96.01.01!<!96.01.01!→ “False(偽)”
より大きいまたは等しい(>=)	日付 >= 日付	ブール	!96.01.20!>=!96.01.01!→ “True(真)” !96.01.01!>=!96.01.20!→ “False(偽)”
より小さいまたは等しい(<=)	日付 <= 日付	ブール	!96.01.20!<=!96.01.20!→ “True(真)” !96.01.20!<=!96.01.01!→ “False(偽)”

時間比較演算子	シンタックス	結果	例
等号(=)	時間 = 時間	ブール	!!01:02:03!!=!!01:02:03!!→ “True(真)” !!01:02:03!!=!!01:02:04!!→ “False(偽)”
不等号(#)	時間 # 時間	ブール	!!01:02:03!!#!!01:02:04!!→ “True(真)” !!01:02:04!!#!!01:02:03!!→ “False(偽)”
より大きい(>)	時間 > 時間	ブール	!!01:02:04!!>!!01:02:03!!→ “True(真)” !!01:02:03!!>!!01:02:03!!→ “False(偽)”
より小さい(<)	時間 < 時間	ブール	!!01:02:03!!<!!01:02:04!!→ “True(真)” !!01:02:03!!<!!01:02:03!!→ “False(偽)”
より大きいまたは等しい(>=)	時間 >= 時間	ブール	!!01:02:04!!>=!!01:02:03!!→ “True(真)” !!01:02:03!!>=!!01:02:04!!→ “False(偽)”
より小さいまたは等しい(<=)	時間 <= 時間	ブール	!!01:02:04!!<=!!01:02:04!!→ “True(真)” !!01:02:04!!<=!!01:02:03!!→ “False(偽)”

次のポインタの場合

```
` vPtrA およびvPtrB は同じオブジェクトを参照する
vPtrA:=->anObject
vPtrB:=->anObject
` vPtrCは別のオブジェクトを参照する
vPtrC:=->anotherObject
```

演算子	シンタックス	結果	例
等号(=)	ポインタ = ポインタ	ブール	vPtrA =vPtrB → “True(真)” vPrtA =vPrtC → “False(偽)”
不等号(#)	ポインタ # ポインタ	ブール	vPrtA #vPrtC → “True(真)” vPrtA #vPrtB → “False(偽)”

文字列比較詳細

■ 文字列は文字ごとに比較されます。

■ 文字列が比較されるとき、文字の大小文字は無視されます。したがって、"a"="A"は“True（真）”を返します。大文字と小文字を意識して比較する場合には、ASCIIコードで比較してください。例えば、次の式は“False（偽）”です。

```
Ascii("A")=Ascii("a")`65と97とは等しくないため
```

■ 文字列が比較される場合、使用するコンピュータのシステム文字比較テーブルを使用して比較されます。例えば、以下の式は日本語システムではTRUEを返します。

```
"あ" = "ｱ"
"ｱ" = "ｱﾞ"
" 1 "="1"
```

■ ワイルドカード記号（@）は、すべての文字列の比較に使用することができます。これは、任意の数の文字を比較します。したがって、次の式は“True（真）”になります。

```
"abcefghij"="abc@"
```

任意の数の文字を比較するために、ワイルドカード記号は、2番目のオペランド（比較演算子の右側の式）で使用しなければなりません。最初のオペランドでは“@”は1桁だけであると解釈されるため、次の式は“False（偽）”です。

```
"abc@"="abcefghij"
```

ワイルドカードは、"1つ以上の文字または無"という意味です。次の式は“True（真）”になります。

```
"abcdefghij" = "abcdefghij@"
"abcdefghij" = "@abcdefghij"
"abcdefghij" = "abcd@efghij"
"abcdefghij" = "@abcdefghij@"
"abcdefghij" = "@abcde@fghij@"
```

一方、どのような場合でも、ワイルドカードを2つ連続して使用した文字列比較は常にFalse（偽）を返します。次の式はFalse（偽）になります。

```
"abcdefghij" = "abc@@"
```

Tips：

文字列をデータ入力から取得する場合、文字列に@マークが含まれている場合があります。このワイルドカードを他の文字と同じように扱うことはできません。次の例を考えてみましょう。

```
$vsValue:=Request("検索したい値を入力してください。")
If(OK=1)
    QUERY([顧客];[顧客]名前=$vsValue+"@")
End if
```

値は、Requestコマンドを使用して要求します。次に、この値は“先頭一致”クエリで使用されます。前述したように、@マークを2つ続けると比較結果がFALSEになるので、値の後に@を付けることができるのは、最後の文字が@でない場合のみです。これは、次のように変更した例で実行することができます。

```
$vsValue:=Request("検索したい値を入力してください。")
If(OK=1)
    If(Ascii($vsValue[[Length($vsValue)]]) # 64)
        $vsValue:=$vsValue+"@"
    End if
    QUERY([顧客];[顧客]名前=$vsValue)
End if
```

次の式は（$vsValueが空でなければ）常にTRUEを返すので、ここでは**Ascii**関数を使用する必要があります。

```
$vsValue[[Length($vsValue)]])="@"
```

さらにこの例について述べると、「リクエスト」ダイアログボックスに複数の@マークを使った“@@D@OE@@@”のような文字列を入れることもできます。以下のコードは、文字列に存在する@文字をすべて削除します。

```
`「No at signs」プロジェクトメソッド
` No at signs(文字列) → 文字列
` No at signs(任意の文字列 ) → @なしの文字列
$0:=""
For($vlChar ; 1 ; Length($1))
    If(Ascii($1[[$vlChar]]) # 64)
        $0:=$0+$1[[$vlChar]]
    End if
End for
```

すなわち、この短いプロジェクトメソッドは**Replace string**関数と同じ操作を行います。ただし、次の**Replace string**式は常に空の文字列を返すので、このプロジェクトメソッドが必要になります（また、ASCIIコードを使用します）。

```
Replace string($vsValue ; "@" ; "") ` すべての文字が削除される
```

最終的に、この例は以下のようになります。

```
$vsValue:=Request("検索したい値を入力してください。")
If(OK=1)
    QUERY([顧客];[顧客]名前=Not at signs($vsValue)+"@")
End if
```

このコードでは、「リクエスト」ダイアログボックスにどのような文字を入力しようと、クエリは、常に“先頭一致”クエリになります。

論理演算子

4th Dimensionは、論理積（&）と論理和（¦）をサポートしています。演算子は、両方ともブール式に関して機能します。論理積“&”は両方の式が“True（真）”である場合に“True（真）”を返します。論理和“¦”は少なくとも一方の式が“True（真）”の時に“True（真）”を返します。

また4th Dimensionには、**True**、**False**、**Not**というブール関数もあります。この関数の詳細は、第16章を参照してください。

次の表に、論理演算子を示します。

演算子	シンタックス	結果	例
論理積（AND）	ブール & ブール	ブール	("A"="A")&(15 #3)→ “True(真)” ("A"="B")&(15 #3)→ “False(偽)” ("A"="B")&(15 =3)→ “False(偽)”
論理和（OR）	ブール ¦ ブール	ブール	("A"="A")¦(15 #3)→ “True(真)” ("A"="B")¦(15 #3)→ “True(真)” ("A"="B")¦(15 =3)→ “False(偽)”

次の表に、論理演算子（&）の真偽表を示します。

式 1	式 2	式 1 & 式 2
True(真)	True(真)	True(真)
True(真)	False(偽)	False(偽)
False(偽)	True(真)	False(偽)
False(偽)	False(偽)	False(偽)

次の図に、論理演算子（¦）の真偽表を示します。

式 1	式 2	式 1 ¦ 式 2
True(真)	True(真)	True(真)
True(真)	False(偽)	True(真)
False(偽)	True(真)	True(真)
False(偽)	False(偽)	False(偽)

Tips：式1と式2の排他的結合子演算を実行する必要がある場合、次の評価式を使用します。

(式1 | 式2) & **Not**(式1 & 式2)

ピクチャ演算子

ピクチャ演算子を使用する式はピクチャを返します。次の表は、4th Dimensionのピクチャ演算子を示ししたものです。

演算子	シンタックス	動作
水平連結(+)	ピクチャ1＋ピクチャ2	ピクチャ2をピクチャ1の右側に連結する
垂直連結(/)	ピクチャ1/ピクチャ2	ピクチャ2をピクチャ1の下に連結する
XOR 重ね(&)	ピクチャ1＆ピクチャ2	ピクチャ1とピクチャ2の排他的ORを実行する
OR 重ね(¦)	ピクチャ1¦ピクチャ2	ピクチャ2の上にピクチャ1を重ねる
水平移動(+)	ピクチャ＋数値	ピクチャを指定ピクセル分、横に移動する
垂直移動(/)	ピクチャ/数値	ピクチャを指定ピクセル分、縦に移動する
サイズ変更(*)	ピクチャ*数値	割合によってピクチャのサイズを変更する
水平スケーリング(*+)	ピクチャ*+数値	割合によって水平にピクチャサイズを変更する
垂直スケーリング(*/)	ピクチャ*/数値	割合によって垂直にピクチャサイズを変更する

元のピクチャの種類に関わらず、2つの演算子“&”と“¦”は常にビットマップピクチャを返します。この理由は、4th Dimensionはまずメモリビットマップにピクチャを描画し、次にビットマップのピクセルに関してグラフィックの排他的または非排他的OR演算を実行して結果のピクチャを計算するためです。

他のピクチャ演算子は、2つの元のピクチャがベクトルの場合に、ベクトルピクチャを返します。しかし、表示形式On Backgroundでプリントされたピクチャはビットマップとしてプリントされる点に留意してください。

比較演算子はピクチャでは使用することはできません。ピクチャフィールドやピクチャ変数は、0を掛けることにより空にすることができます。

例えば、このように記述します。

```
vピクチャ:=vピクチャ*0
```

ビットワイズ演算子

ビットワイズ演算子は、倍長整数式また倍長整数値の演算を行います。

注：ビットワイズ演算子に整数値または実数値を渡すと、4th Dimensionは値を倍長整数値として評価してから、ビットワイズ演算子を使用した式を計算します。

ビットワイズ演算子を使用する場合、倍長整数値を32ビットの配列と考える必要があります。これらのビットには、右から左に0～31の番号が付けられます。

それぞれのビットは0か1なので、倍長整数値は32の論理値を格納できる値と考えることもできます。1に等しいビットはTrue、0に等しいビットはFalseを意味します。

ビットワイズ演算子を使用する式は倍長整数値を返しますが、「Bitテスト」演算子は例外的に式がブール値を返します。次の表にビットワイズ演算子とそのシンタックスを示します。

演算	演算子	シンタックス	結果
Bitwise AND	&	倍長整数 & 倍長整数	倍長整数
Bitwise OR(包合)	¦	倍長整数 ¦ 倍長整数	倍長整数
Bitwise OR(排他)	^¦	倍長整数 ^¦ 倍長整数	倍長整数
Left Bit Shift	<<	倍長整数 << 倍長整数	倍長整数(備考1)
Right Bit Shift	>>	倍長整数 >> 倍長整数	倍長整数(備考1)
Bitセット	?+	倍長整数 ?+ 倍長整数	倍長整数(備考2)
Bit消去	?-	倍長整数 ?- 倍長整数	倍長整数(備考2)
Bitテスト	??	倍長整数 ?? 倍長整数	ブール(備考2)

備考：

1. 「Left Bit Shift」および「Right Bit Shift」演算では、2番目のオペランドは、結果値において1番目のオペランドのビットがシフトされるビット数を示します。したがって、この2番目のオペランドは、0～32の間でなければなりません。0ビットシフトするとその値がそのまま返されます。また、31ビットより多くシフトするとすべてのビットがなくなるので、0x00000000が返されます。もう1つの値を2番目のオペランドとして渡すと、結果は符号無しになります。
2. 「Bitセット」、「Bit消去」、「Bitテスト」演算では、2番目のオペランドは、作用の対象となるビット番号を示します。したがって、この2番目のオペランドは0～31の間です。そうでない場合、「Bitセット」と「Bit消去」に対しては1番目のオペランドがそのまま返され、「Bitテスト」に対してはFalseが返されます。

次の表は、ビットワイズ演算子とその効果を示します。

演算子	説明
Bitwise AND	それぞれの結果ビットは2つのオペランドのビットの論理ANDです。 下記は、論理ANDの真偽表です。 1 & 1 → 1 0 & 1 → 0 1 & 0 → 0 0 & 0 → 0 すなわち、両方のオペランドのビットが1の場合、結果ビット が1になり、その他の場合は結果ビットが0になります。
Bitwise OR(包含)	それぞれの結果ビットは2つのオペランドのビットの論理ORです。 下記は、論理ORの真偽表です。 1 ¦ 1 → 1 0 ¦ 1 → 1 1 ¦ 0 → 1 0 ¦ 0 → 0 すなわち、いずれかのオペランドのビットが1の場合、 結果ビットが1になり、その他の場合は結果ビットが0になります。
Bitwise OR(排他)	それぞれの結果ビットは2つのオペランドのビットの論理XORです。 下記は、論理XORの真偽表です。 1 ^¦ 1 → 0 0 ^¦ 1 → 1 1 ^¦ 0 → 1 0 ^¦ 0 → 0 すなわち、オペランドのビットのいずれか一方だけが1の場合 結果ビットが1になり、その他の場合は結果ビットが0になります。
Left Bit Shift	最初のオペランド値が結果値に設定され、次に結果ビットが2番目 のオペランドで示されたビット数だけ左にシフトします。 左側のビットがなくなり、右側の新しいビットは0に設定されます。 注：正の数だけを考えると、Nビット左にシフトすることは、 2^Nを掛けることと同じです。
Right Bit Shift	最初のオペランド値が結果値に設定され、次に結果ビットが2番目 のオペランドで示されたビット数だけ右にシフトします。 右側のビットがなくなり、左側の新しいビットは0に設定されます。 注：正の数だけを考えると、Nビット左にシフトすることは、 2^Nで割ることと同じです。

Bitセット	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが1に設定されます。他のビットはそのままです。
Bit消去	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが0に設定されます。他のビットはそのままです。
Bitテスト	最初のオペランドのうち、2番目のビットで示されたビットが1の場合、Trueが返されます。最初のオペランドのうち、2番目のビットで示されたビットが0の場合、Falseが返されます。

例：

(1)次の表にそれぞれのビット演算子の例を示します。

演算子	例	結果
Bitwise AND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitwise OR(包合)	0x0000FFFF ¦ 0xFF00FF00	0xFF00FFFF
Bitwise OR(排他)	0x0000FFFF & 0xFF00FF00	0xFF0000FF
Left Bit Shift	0x0000FFFF << 8	0x00FFFF00
Right Bit Shift	0x0000FFFF >> 8	0x000000FF
Bitセット	0x00000000 ?+16	0x00010000
Bit消去	0x00010000 ?-16	0x00000000
Bitテスト	0x00010000 ?? 16	True

(2)4th Dimensionのバージョン6以降では、多くの定義済み定数が提供されています。これらの定数の中には、そのリテラルが"bit"または"mask"で終わるものがあります。例えば、「Resources properties」テーマで提供される定数の場合を以下に示します。

定数	タイプ	値
System heap resource mask	倍長整数	64
System heap resource bit	倍長整数	6
Purgeable resource mask	倍長整数	32
Purgeable resource bit	倍長整数	5
Locked resource mask	倍長整数	16
Locked resource bit	倍長整数	4
Protected resource mask	倍長整数	8
Protected resource bit	倍長整数	3
Preloaded resource mask	倍長整数	4
Preloaded resource bit	倍長整数	2
Changed resource mask	倍長整数	2
Changed resource bit	倍長整数	1

これらの定数により、**Get resource properties**関数が返した値をテストする、または**SET RESOURCE PROPERTIES**コマンドに渡す値を作成することができます。リテラルが"bit"で終わる定数は、テスト、消去またはセットするビットの位置を指定します。リテラルが"mask"で終わる定数は、テスト、消去またはセットするビットが1の場合のみ、倍長整数値を指定します。

例えば、（プロパティが変数$vlResAttrで取得された）リソースがパージ可能かどうかをテストするには、以下のように記述します。

```
If($vlResAttr ?? Purgeable resource bit) `リソースはパージ可能か?
```

または

```
If(($vlResAttr & Purgeable resource mask) # 0) リソースはパージ可能か?
```

逆に、これらの定数を使用して同じビットを設定することもできます。以下のように記述します。

```
$vlResAttr:=$vlResAttr ?+ Purgeable resource bit
```

または

```
$vlResAttr:=$vlResAttr | Purgeable resource mask
```

(3)この例では、2つの整数値を倍長整数値に格納します。以下のように記述します。

```
$vlLong:=($viIntA<<16) | $viIntB ` 2つの整数を倍長整数に格納します。
$vlIntA:=$vlLong>>16 ` ハイワードに格納されている整数を抽出し直します。
$viIntB:=$vlLong & 0xFFFF ` ローワードに格納されている整数を抽出し直します。
```

Tips：倍長整数または整数の値を数値とビットワイズ演算子を組み合わせた式で操作する場合は注意が必要です。ハイビット（倍長整数の場合はビット31、整数の場合はビット15）は、その値が消去された場合には正の数で、値が設定された場合には負の数です。数値演算子はこのビットを使用して、値の符号を検出します。ビットワイズ演算子はこのビットの意味を無視します。

PRINT LABEL

PRINT LABEL({テーブル} { ; ラベルドキュメント}{; *})

引数	タイプ		説明
テーブル	テーブル	→	印刷するテーブル 省略した場合、デフォルトテーブル
ラベルドキュメント	文字列	→	ディスクラベルドキュメントの名前
*¦>	*¦>		* 印刷ダイアログを省略 > 印刷設定の再初期化禁止

説明

PRINT LABELコマンドは、＜テーブル＞のセレクションのデータを使用してラベルを印刷します。

引数＜ラベルドキュメント＞を指定しないと、カレント出力フォームを使用して、ラベルとして＜テーブル＞のカレントセレクションを印刷します。サブフォームの印刷にこのコマンドを使用することはできません。ラベルのフォーム作成に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

引数＜ラベルドキュメント＞を指定すると、**PRINT LABEL**コマンドは、「ラベル」ウィザード（下図）を表示するか、またはディスクに保存された既存のラベルドキュメントを印刷します。以下の説明を参照してください。



両方の場合ともに、オプション引数のアスタリスク（*）を使用して、ダイアログボックスの表示を取り消すことができます。「ラベル」ウィザードを使用する際にはこの引数は無効なので注意してください。

「ラベル」ウィザードを使用しない場合、ラベルがすべてプリントされるとシステム変数OKに1が代入されます。それ以外の場合には0が代入されます（プリンタダイアログボックスでキャンセルがクリックされた場合）。

引数＜ラベルドキュメント＞を指定すると、ラベルは＜ラベルドキュメント＞に定義されたラベル設定情報に従って印刷されます。＜ラベルドキュメント＞に空のストリング（""）を指定すると、「ファイルを開く」ダイアログボックスが表示され、ユーザはラベル設定として使用するファイルを指定することができます。＜ラベルドキュメント＞に存在しないドキュメント名を指定すると（たとえば、char(1)を渡す）、ラベルウィザードが表示され、ユーザはラベル設定を定義することができます。

PRINT LABELコマンドは印刷の前にはデフォルトで印刷ダイアログを表示します。ダイアログをユーザーがキャンセルした場合は、**PRINT LABEL**コマンドの実行は中止され、印刷は行われません。このダイアログの表示を省略するために、オプション引数＜*＞＜＞を使います。

＜*＞引数を指定した時は、デフォルトの印刷設定、もしくはコマンド**PAGE SETUP**で定義した設定で印刷されます。

「>」引数を指定した時は、印刷設定の再初期化を行うことなく、カレント印刷設定のまま、印刷を行います。この引数の利用は、あらかじめカスタム化しておいた印刷設定を維持しながら繰り返し印刷する際に便利です。**PRINT RECORD**コマンドの使用例題を参照してください。

4D Server：このコマンドは、ストアドプロシージャの枠内において4D Server上で実行することができます。この状況では、次の制約があります。

・サーバマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。これを実現するには、コマンドを呼び出す際に、引数<*>または<>>を指定する必要があります。

・4D Serverでは、ラベルエディタを表示させる構文は動作しません。これを行った場合、システム変数OKは0に設定されます。

・プリンタ関連の問題が発生しても（用紙切れ、プリンタ接続切断、等）、エラーメッセージは生成されません。

例題

▼以下の例は、テーブルの出力フォームを使用してラベルを印刷します。この例では2つのメソッドを使用します。最初のプロジェクトメソッドは、、正しい出力フォームを設定し、ラベルを印刷します。

```
vCR:=Char(13)
ALL RECORDS([住所])      ` すべてのレコードを選択
OUTPUT FORM([住所] ; "ラベル出力")    ` 出力フォームの設定
PRINT LABEL([住所])      ` ラベルの印刷
OUTPUT FORM([住所] ; "出力1")   ` デフォルト出力フォームの設定
```

▼以下の例は、フォーム“ラベル出力”のフォームメソッドです。このフォームは、各フィールドの内容を連結した結果を格納するための1つの変数“vラベル”を含みます。変数“vラベル”は、フィールド“住所2”が空白の場合、メソッドで読み飛ばします。「ラベル」ウィザードを使用しても、この処理が自動的に実行される点に注意してください。フォームメソッドは、各レコードに対して1つずつラベルを作成します。

```
` フォーム“ラベル出力”のフォームメソッド
` “vラベル”はフォーム上の変数
`郵便番号、都道府県、市町村、住所、名前を連結する
Case of
    ¥(Form event=On load)
        vラベル:=[住所]名前1+"      "+[住所]名前2+Char(13)+[住所]住所1
                                                    +Char(13)
        If([住所]住所2 # "")   ` 住所2が空白でない場合
            vラベル:=vラベル+"  "+住所2+Char(13)
```

```
                                                    `vラベルに住所2を連結する
                End if
                vラベル:=郵便番号+vCR+都道府県+"　"+市町村+vCR+vラベル+vCR+
                                                                名前1+"　"+名前2
        End Case
```

▼以下の例は、ユーザが[住所]テーブルを検索し、ドキュメントファイル“住所ラベル”に格納されている「ラベル」ウィザードの設定情報を使用して印刷します。

```
QUERY([住所 ])
If(OK=1)
        PRINT LABEL([住所 ]; "住所ラベル")
End if
```

▼以下の例は、ユーザが[住所]テーブルを検索し、印刷するラベルを選択します。

```
QUERY([住所 ])
If(OK=1)
        PRINT LABEL([住所 ];"")
End if
```

▼以下の例は、ユーザが[住所]テーブルを検索し、「ラベル」ウィザードを表示して任意のラベルの設計、保存、ロード、印刷を行います。

```
QUERY([住所 ])
If(OK=1)
        PRINT LABLE([住所 ];Char(1))
End if
```

参照

PRINT SELECTION、QR REPORT

PRINT SELECTION

PRINT SELECTION({テーブル} {;} {*})

引数	タイプ		説明
テーブル	テーブル	→	印刷するテーブル 省略した場合、デフォルトテーブル
*¦>	*¦>		*印刷ダイアログを省略 >印刷設定の再初期化禁止

説明

PRINT SELECTIONコマンドは、テーブルのカレントセレクションを印刷します。レコードは、カレントプロセスのテーブルのカレント出力フォームを使用して印刷されます。**PRINT SELECTION**コマンドは、「ユーザ」モードの「プリント...」メニューと同じ動作を実行します。セレクションが空の場合、**PRINT SELECTION**コマンドは何も行いません。

デフォルトでは、**PRINT SELECTION**コマンドは、印刷前にプリンタダイアログボックスを表示します。オプション引数のアスタリスク（*）を使用して、ダイアログボックスの表示を取り消すことができます。ユーザがプリンタダイアログボックスの「キャンセル」ボタンをクリックすると、このコマンドが取り消され、レポートは印刷されません。<*>を設定した場合は、フォームを作成した時点のページ設定情報でレポートが印刷されます。あるいは、**PAGE SETUP**コマンドを使用してページ設定を行うことができます。

印刷中に、「デザイン」モードのフォームおよびオブジェクトプロパティウインドウで有効にしたイベントと、実際に発生しているイベントに応じて、出力フォームのフォームメソッドとオブジェクトメソッドが実行されます。

■ ヘッダを印刷する直前に**On Header**イベントが発生する。

■ レコードを印刷する直前に**On Printing Detail**イベントが発生する。

■ ブレークエリアを印刷する直前に**On Printing Break**イベントが発生する。

■ フッタを印刷する直前に**On Printing Footer**イベントが発生する。

PRINT SELECTIONコマンドが最初のヘッダを印刷しているかどうかは、**On Header**イベントで**Before selection**関数を判定することによって調べることができます。また**On Printing Footer**イベントで**End selection**関数を判定することによって、最後のフッタかどうかをチェックすることができます。これら関数の詳細は、それぞれの関数の説明および第24章「フォームイベント関数」を参照してください。

PRINT SELECTIONコマンドを使用し、小計やブレーク付きでソートしたセレクションを印刷するには、まずそのセレクションをソートしなければなりません。次に、レポートの各ブレークエリアに、小計を変数に代入するオブジェクトメソッドを持つ変数を設定します。変数に値を代入する際に、**Sum**関数や**Average**関数のような統計関数と算術関数を使用することもできます。詳細は、**Subtotal**、**BREAL LEVEL**、**ACCUMULATE**コマンドの説明を参照してください。

警告：PRINT SELECTIONコマンドと一緒にPAGE BREAKコマンドを使用してはいけません。PAGE BREAKコマンドはPrint form関数と一緒に使用します。

PRINT SELECTIONコマンドの呼び出し後、プリントが正常に終了するとシステム変数OKに1がセットされます。プリントが中断された場合には、システム変数OKには0がセットされます（つまり、ユーザがプリントダイアログボックスでキャンセルをクリックした場合）。

4D Server：このコマンドは、ストアドプロシージャの枠内において4D Server上で実行することができます。この状況では、次の制約があります。

・サーバマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。これを実現するには、コマンドを呼び出す際に、引数<*>または<>>を指定する必要があります。

・4D Serverでは、ラベルエディタを表示させる構文は動作しません。これを行った場合、システム変数OKは0に設定されます。

・プリンタ関連の問題が発生しても（用紙切れ、プリンタ接続切断など）、エラーメッセージは生成されません。

例題

▼以下の例は、最初に[従業員]テーブルのすべてのレコードを選択します。次に**DISPLAY SELECTION**コマンドを使用して、すべてのレコードを表示し、ユーザがプリントするレコードを選択します。最後に**USE SET**コマンドにより、選択されたレコードを**PRINT SELECTION**コマンドで印刷します。

```
ALL RECORDS([従業員])  `全レコードを選択
DISPLAY SELECTION([従業員];*)  `レコードを表示
USE SET("UserSet")  `ユーザが選択したレコードだけを使用
PRINT SELECTION([従業員])  `ユーザが選択したレコードを印刷
```

参照

ACCUMULATE、BREAK LEVEL、Level、PAGE SETUP、Subtotal

Print form

Print form({table;} form{; area1{; area2}})→数値

引数	タイプ		説明
Table	テーブル	→	印刷するテーブル、または、省略する場合は、デフォルトテーブル
form	文字列	→	印刷するフォーム
area1	数値	→	印刷マーカー、または開始エリア（area2が指定される場合）
area2	数値	→	終了エリア（area1が指定される場合）
戻り値	数値	←	印刷されるセクションの高さ

説明

Print form関数は、フィールドや変数の現在の値を＜フォーム＞に印刷します。通常は、印刷のプロセスをメソッドで完全に制御する必要のある非常に複雑なレポートを印刷するために使用します。**Print form**関数はレコード処理、ブレーク処理、ページブレーク（改ページ）処理、ヘッダ処理、フッタ処理を全く行いません。これらの処理は、すべてデザイナが行います。**Print form**関数は固定された大きさの枠のなかにフィールドや変数を印刷します。

Print form関数は、フォームの印刷の後にページブレーク（改ページ）を行わないため、同じページに異なるフォームを容易に配置することができます。したがって、**Print form**関数は、異なるテーブルや異なるフォームを含む複雑な印刷処理には最適です。ページブレーク（改ページ）を強制的に行うには**PAGE BREAK**コマンドを使用してください。印刷可能領域を超える高さのフォームの印刷を次のページに持ち越すには、**PAGE BREAK**コマンドを使用する前に**CANCEL**コマンドを呼び出してください。

以下の3つの構文を使用できます。

■ 詳細エリア印刷

```
height:=Print form(myTable;myForm)
```

この場合、Print formはフォームの詳細エリア（ヘッダ行と詳細行の間のエリア）だけを印刷します。

■ フォームエリア印刷

```
height:=Print form(myTable;myForm;marker)
```

この場合コマンドはmarkerで示されるセクションを印刷します。以下の“Form area”テーマの定数のうちの1つをmarker引数に渡します。

定数	タイプ	値
Form Header	倍長整数	200
Form Header1 ... 10	倍長整数	201...210
Form Detail	倍長整数	0（デフォルト値）
Form Break0 ... 9	倍長整数	300...309
Form Footer	倍長整数	100

■ セクション印刷

height:=Print form(myTable;myForm;areaStart;areaEnd)

この場合、コマンドはareaStartおよびareaEnd引数間のセクションを印刷します。値はピクセルで単位で入力しなければいけません。

Print form関数によって返される値は印刷可能範囲の高さを表します。この値は、**Get printed height**コマンドに自動的に考慮されます。

Print form関数を使用する場合、プリンタダイアログボックスは表示されません。レポートでは「デザイン」モードでフォームに割り当てられた用紙設定が使用されません。**Print form**関数を実行する前に用紙設定を指定する方法は2通りあります。

■ **PRINT SETTINGS**コマンドを使用する。この場合、ユーザが設定を行う。

■ **PAGE SETUP**コマンドを使用する。この場合、用紙設定はプログラムで指定する。

Print form関数は、メモリ中にそれぞれ印刷するページを作成します。各ページはメモリのページがいっぱいになるか、**PAGE BREAK**コマンドを実行すると印刷されます。**Print form**関数の使用後の最後のページの印刷を確実に行うためには、**PAGE BREAK**コマンドで終了しなければなりません。そうでないと、最後のページはメモリ中に残り印刷されません。

警告：サブフォームや外部オブジェクトは、Print form関数では印刷できません。このようなオブジェクトを持つフォームを1つだけ印刷するには、代りにPRINT RECORDコマンドを使用します。

Print form関数により、フォームメソッドの**On Printing Detail**イベントが発生します。

4D Server：このコマンドは、ストアドプロシージャの枠内において4D Server上で実行することができます。この状況では、次の制約があります。

・サーバマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。

・プリンタ関連の問題が発生しても（用紙切れ、プリンタ接続切断、等）、エラーメッセージは生成されません。

例題

(1)以下の例は、単純な**PRINT SELECTION**コマンドをシミュレートします。レコードが小切手用か預金用であるかによって2種類のフォーム内の1つを使用します。

```
QUERY([レジスタ])          ` レコードの選択
If(OK=1)
        ORDER BY([レジスタ])          ` レコードのソート
        If(OK=1)
                PRINT SETTINGS          ` 用紙の設定
                If(OK=1)
                        For($i ; 1 ; Records in selection([レジスタ]))
                                                ` 選択した全レコードのループ
                                If([レジスタ]タイプ="小切手") ` “タイプ” フィールド
                                                が"小切手"の場合
                                        Print form([レジスタ] ; "小切手印刷")
                                                ` 小切手の印刷
                                Else ` “タイプ” フィールドが"小切手"でない場合
                                        Print form([レジスタ] ; "預金印刷")
                                                ` 預金の印刷
                                End if
                                NEXT RECORD([レジスタ]) ` 以下のレコードに移動
                        End for
                        PAGE BREAK ` 最後のページを印刷
                End if
        End if
End if
```

(2)**SET PRINT MARKER**コマンドの例を参照してください。

参照

CANCEL、PAGE BREAK、PAGE SETUP、PRINT SETTINGS

PAGE BREAK

PAGE BREAK {(* | >)}

引数	タイプ		説明
*¦>		→	**Print form**で開始した印刷ジョブをキャンセルする、または >：1つのプリントジョブを強制する

説明

PAGE BREAKコマンドは**Print form**関数とともに使用すると（「On Printing Detai」フォームイベント中に）、強制的にページブレークを行い、メモリ上に作成された最終ページを印刷します。**PAGE BREAK**コマンドは、**PRINT SELECTION**コマンドとともに使用しないでください。この代りに、**Subtotal**関数または**BREAK LEVEL**コマンドにオプション引数を使用してページブレークを行ってください。

<*>と<>>引数は両方とも省略できます。

<*>引数により、**Print form**関数によって開始したプリントジョブをキャンセルすることができます。このコマンドを実行すると、進行中のプリントジョブが直ちに中止されます。

<>>引数は、**PAGE BREAK**コマンドの振る舞いを変更します。この形式は2種類の効果を持ちます。

■ **PAGE BREAK**コマンドが引数なしで再度実行されるまで、プリントジョブの開始を止めます。

■ プリントジョブに優先権を与えます。プリントジョブが終了するまで、他のプリントは行われません。

2番目のオプションは、スプールされるプリントジョブとともに使用すると、特に有効です。<>>引数により、プリントジョブは1つのファイルにスプールされます。これは、プリント時間を減少させます。

例題

1. **Print form**関数の例を参照してください。

2. **SET PRINT MARKER**コマンドの例題を参照してください。

参照

CANSEL、Print form

PRINT RECORD

PRINT RECORD({テーブル} {;*})

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードを印刷するテーブル、または省略した場合デフォルトテーブル
*¦>	*¦>		*印刷ダイアログボックスを省略、または印刷設定の再初期化を行わない

説明

このコマンドは＜テーブル＞のカレントレコードを、カレントセレクションを変更せずに印刷します。カレント出力フォームが印刷に用いられます。＜テーブル＞にカレントレコードが存在しない場合、**PRINT RECORD**コマンドは何も行いません。

PRINT RECORDコマンドを使ってサブフォームや外部オブジェクトを印刷することができます。この操作は、**Print form**関数では実行できません。

注：レコードに対して行われた修正が保存されていない場合、ディスク上の修正前のフィールド値ではなく、修正後の値が印刷されます。

PRINT RECORDコマンドは印刷の前にはデフォルトで印刷ダイアログを表示します。

ダイアログをユーザーがキャンセルした場合は、**PRINT RECORD**コマンドの実行は中止され、印刷は行われません。このダイアログの表示を省略するために、オプション引数＜*＞＜>＞を使います。

＜*＞引数を指定した時は、デフォルトの印刷設定、もしくはコマンド**PAGE SETUP**で定義した設定で印刷されます。

＜>＞引数を指定した時は、印刷設定の再初期化を行うことなく、カレント印刷設定のまま、印刷を行います。この引数の利用は、あらかじめカスタム化しておいた印刷設定を維持しながら繰り返し印刷する際に便利です。**PRINT RECORD**コマンドの使用例題を参照してください。

4D Server：このコマンドは、ストアドプロシージャの枠内において4D Server上で実行することができます。この状況では、次の制約があります。

・サーバマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。これを実現するには、コマンドを呼び出す際に、引数＜*＞または＜>＞を指定する必要があります。

・プリンタ関連の問題が発生しても（用紙切れ、プリンタ接続切断など）、エラーメッセージは生成されません。

▼以下の例では、カレントレコードを印刷します。このコードは入力フォームのボタン内に記述されています。ユーザがそのボタンをクリックすると、レコードは指定した出力フォームで印刷されます。

```
OUTPUT FORM([テーブル1] ; "レコード印刷")
PRINT RECORD([テーブル1] ; *)
OUTPUT FORM([テーブル1] ; "出力")
```

例題

```
PRINT SETTINGS  `印刷設定を定義
If(OK=1)
        OUTPUT FORM([従業員];"ディテイル")  `最初の印刷帳票を使う
⇒       PRINT RECORD([従業員];>)  `ユーザーが定義した印刷設定で印刷
        OUTPUT FORM([従業員];"Simple")  `二つ目の印刷帳票を使う
⇒       PRINT RECORD([従業員];>)  `ユーザーが定義した印刷設定で印刷
        OUTPUT FORM([従業員];"Output")  `デフォルト出力フォームへ戻す
End if
```

参照

Print form

Printing page

Printing page → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	現在プリント中のページのページ番号

説明

Printing page関数は、印刷中のページ番号を返します。この関数は、**PRINT SELECTION**コマンドまたは「ユーザ」モードの「プリント...」メニューの選択によって印刷する場合にのみ使用することができます。

▼以下の例は、両面印刷フォーマットのレポートにページ番号を設定します。ページ番号の位置を変更するために、フォームはページ番号を表示する変数を2つ持っています。変数“v左”は、偶数のページ番号を印刷します。変数“v右”は、奇数のページ番号を印刷します。この例は、偶数ページを判定し適切な変数に値を代入します。

```
Case of
    ¥(Form event=On Printing Footer)
        If((Printing page % 2) = 0)
                                ` 印刷中のページを2で割った時の余りが0の場合
            v左:=String(Printing page) ` 左ページの番号を設定
            v右:=""          ` 右ページの番号をクリア
        Else
            v右:=String(Printing page) ` 右ページの番号を設定
            v左:=""          ` 左ページの番号をクリア
        End if
End case
```

参照

PRINT SELECTION

PRINTERS LIST

PRINTERS LIST(名前配列{; 場所配列{; モデル配列}})

引数	タイプ		説明
名前配列	テキスト配列	←	プリンタ名
場所配列	テキスト配列	←	プリンタの場所
モデル配列	テキスト配列	←	プリンタのモデル

説明

PRINTERS LISTコマンドは、引数として渡された各配列にそのマシンで使用できるプリンタの名前、ならびにオプションとしてプリンタの場所とモデルを代入します。

注：プリントサーバ（スプーラ）を使用して、プリンタを管理している場合、フルアクセスパス（Windows）またはスプーラの名前（Mac OS）が返されます。

引数＜名前配列＞には、テキスト配列の名前を渡します。コマンドの実行後、この配列には使用できるプリンタの名前が代入されます。任意の引数である＜場所配列＞と＜モデル配列＞を渡した場合、各プリンタのネットワーク上の位置（またはローカルポート）とモデルが返されます。

注：引数＜場所配列＞と＜モデル配列＞は、Windows上でのみ使用できます。

4Dで選択されたプリンタの変更や取得を行うには、**SET CURRENT PRINTER**コマンドと**Get current printer**関数を使用してください。

Windows上では、プリンタ名はOSレベルで手動にて変更することができます。一方、プリンタの場所とモデルタイプは、その物理的特性に関連しています。したがって、任意の配列の値を使用して、選択したプリンタの特性を調べることができます。通常は、クライアントマシンがすべて同じプリンタを使用していることをチェックします。

Mac OS上では、プリンタ名（プリンタサーバの名前）を使用して、このチェックを行います。このプリンタ名は、接続している各マシンに対して同じ名前になります。

コマンドが正常に終了すると、システム変数OKには1が代入されます。それ以外の場合には0が代入され、配列には何も返されません。

参照

SET CURRENT PRINTER、Get current printer

SET CURRENT PRINTER

SET CURRENT PRINTER(プリンタ名)

引数	タイプ		説明
プリンタ名	文字列	→	使用するプリンタの名前

説明

注：Mac OS 9上では、このコマンドは動作しません。Windowsでは、Windows 2000以降が必要です。

SET CURRENT PRINTERコマンドを使用し、現行の4Dアプリケーションでの印刷に使用するプリンタを指定することができます。

引数<プリンタ名>には、選択するプリンタの名前を渡します。使用できるプリンタ一覧を取得するには、このコマンドの前に新規コマンドである**PRINTERS LIST**コマンドを使用します。

<プリンタ名>に空の文字列を渡した場合、システムで定義された現行のプリンタが使用されます。

SET CURRENT PRINTERコマンドは、利用可能なオプションが選択したプリンタに対応するように、必ず**SET PRINT OPTION**コマンドの前に呼び出してください。それに対し、**SET CURRENT PRINTER**コマンドは、**PAGE SETUP**コマンドの後で呼び出さなければなりません。これを行わないと、印刷設定が失われます。

このコマンドは、**PRINT SELECTION**、**PRINT LABEL**、**PRINT RECORD**、**Print form**、および**QR REPORT**コマンドと一緒に使用することができ、「デザイン」モードを含め、4th Dimensionにおけるすべての印刷に対して適用されます。

指定した設定が失われないように、印刷コマンドは必ず引数<>>を用いて呼び出さなければなりません（適用できる場合）。

参照

Get current printer、PRINTERS LIST

Get current printer

Get current printer→ 文字列

引数	タイプ		説明
このコマンドに引数はありません。			
戻り値	文字列	←	現在のプリンタの名前

説明

注：Mac OS 9上では、このコマンドは動作しません。Windowsでは、Windows 2000以降が必要です。

Get current printer関数は、4Dアプリケーションで定義された現在のプリンタ名を返します。デフォルトとして、4Dの起動時には、システムで定義されたプリンタが現行プリンタとなります。

プリントサーバ（スプーラ）を使用して、現行プリンタを管理している場合、フルアクセスパス（Windows）またはスプーラの名前（Mac OS）が返されます。

使用できるプリンタ一覧および追加情報を取得するには、**PRINTERS LIST**コマンドを使用します。現在のプリンタを変更するには、**SET CURRENT PRINTER**を使用してください。

プリンタがインストールされていない場合、システム変数OKには0が代入されます。それ以外の場合には1が代入されます。

参照

SET CURRENT PRINTER、PRINTERS LIST

BREAK LEVEL

BREAK LEVEL(レベル {; ページブレーク})

引数	タイプ		説明
レベル	数値	→	ブレークレベルの数
ページブレーク	数値	→	ページブレークを行うブレークレベル

説明

BREAK LEVELコマンドは、**PRINT SELECTION**コマンドを使ってプリントするレポートのブレークの数を指定します。

警告：コンパイルモードでは、ブレーク処理を行うレポートを印刷する前に、BREAK LEVELコマンドとACCUMULATEコマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を生成します。Subtotalコマンドの説明を参照してください。

＜レベル＞は、ブレーク処理を実行するレベルのことです。少なくとも同数のレベルでレコードをソートしなければなりません。ブレークレベルよりも多いレベルでソートすると、これらのレベルはソートされたものとして印刷されますが、ブレークに対しての意味は持ちません。

生成される各ブレークレベルは、フォーム中の対応するブレークエリアやヘッダエリアを印刷します。フォーム中のブレークエリアは、＜レベル＞の数だけ存在しなければなりません。フォーム中により多くのブレークエリアがある場合、それらは無視され、印刷されません。

2番目の（オプションの）引数＜ページブレーク＞は、印刷中にページブレークを発生させるために使用します。

▼ 以下の例は、2つのブレークレベルを持つレポートを印刷します。このセレクションは 4つのレベルに対してソートされますが、**BREAK LEVEL**コマンドは2つのレベルだけにブレークを指定します。[従業員]給与フィールドは**ACCUMULATE**コマンドで累計されます。

```
` 4つのレベルでソート
ORDER BY([従業員] ; [従業員]部門 ;> ; [従業員]役職名 ; > ; [ 従業員]名字 ; > ;
 [従業員]名前 ; >)
      ` 部門と役職名のフィールドでブレーク処理を行う
BREAK LEVEL(2)
ACCUMULATE([従業員]給与)      ` 給与の累計
OUTPUT FORM([従業員] ; "部門別給与") ` 出力フォームの設定
PRINT SELECTION([従業員])      ` レポートの印刷
```

参照

ACCUMULATE、ORDER BY、PRINT SELECTION、Subtotal

SET PRINT OPTION

SET PRINT OPTION(オプション; 値1{; 値2})

引数	タイプ		説明
オプション	倍長整数	→	オプション番号
値1	倍長整数¦文字列	→	オプションの値1
値2	倍長整数	→	オプションの値2

説明

SET PRINT OPTIONコマンドを使用し、プログラムからプリントオプションの値を変更することができます。プリントパラメータを変更する他のコマンド（**PRINT SETTINGS**、引数“>”を使用しない**PRINT SELECTION**）が呼び出されない限り、このコマンドを使用して定義された各オプションは、4Dセッションの間データベース全体に対して適用されます。

引数＜オプション＞を使用し、変更するオプションを指定することができます。値を渡すか、または「Print options」テーマ内にある次の定義済定数のいずれかを渡すことができます。

定数	タイプ	値
Paper option	倍長整数	1
Orientation option	倍長整数	2
Scale option	倍長整数	3
Number of copies option	倍長整数	4
Paper source option	倍長整数	5
Color option	倍長整数	8
Destination option	倍長整数	9
Double sided option	倍長整数	11
Spooler document name option	倍長整数	12

指定した＜オプション＞の新しい値は、引数＜値1＞と（任意の）＜値2＞に渡します。渡す値の数値と種類は、指定したオプションのタイプによって異なります。オプションと考えられる値に関しての詳細は、次の表を参照してください

オプション	定数	値1	値2
1	Paper option	名前	-
		高さ	幅
2	Orientation option	1=縦方向、2=横方向	-
3	Scale option	数値（%）	-
4	Number of copies option	数値	-
5	Paper source option	Windowsのみ	
		インデックス（数値）	-
8	Color option	Windowsのみ	
		1=N/B、2=カラー	-
9	Destination option	1=プリンタ	-
		2=ファイル（PC）／PS（Mac）、	アクセスパス
		3=PDF（Mac）	アクセスパス
		4=EPS（Mac）	アクセスパス
		5=スクリーン（Mac）	
11	Double sided option	Windowsのみ	
		0=片面（標準）	-
		1=両面	とじしろ位置：0=左（デフォルト）、1=上
12	Spooler document name option	印刷する文書名	-
13	Mac spool file format option	0=PDFモード、1=PostScriptモード	-

■ Paper option(1)：**PRINT OPTION VALUES**コマンドを使用して取得した、使用可能なすべての用紙タイプの名前一覧。
＜値1＞（この場合、＜値2＞は省略）に用紙の名前を渡すか、または＜値1＞に用紙の高さ、＜値2＞に用紙の幅を渡します。幅と高さは、画面のピクセル単位で表わさなくてはなりません。

■ Orientation option(2)：1（縦方向）または2（横方向）のいずれかを＜値1＞に渡します。

■ Scale option(3)：倍率を＜値1＞に渡します。プリンタのなかには倍率を変更できないものがあるので注意が必要です。無効な値を渡すと、印刷時にこのプロパティが100%にリセットされます。

■ Number of copies option(4)：印刷する部数を＜値1＞に渡します。

■ Paper source option(5)：使用する用紙トレイのインデックスに対応する数値を、**PRINT OPTION VALUES**コマンドから返されるトレイの配列に渡します。

注：このオプションは、Windowsでのみ使用可能です。

■ Color option(8)：＜値1＞に、カラー処理用のモードを示すコードを渡します。1=黒白（モノクロ）、2=カラー。

注：このオプションは、Windowsでのみ使用可能です。

■ Destination option(9)：＜値1＞に、印刷先のタイプを示すコードを渡します：1=プリンタ、2=ファイル（PC）/PS（Mac）、3=PDFファイル（Mac OSのみ）、4=EPSファイル（Mac OSのみ）、5=スクリーン（Mac OS Xドライバオプション）。
＜値1＞が1または5以外の場合、出力結果となる文書へのアクセスパスを＜値2＞に渡します。別のパスが指定されるまで、セッションの間このパスが使用されます。出力先に同じ名前のファイルが既に存在する場合は、新しいファイルと置き換えられます。＜値2＞に空の文字列を渡した場合や、この引数を省略した場合（Windows）、印刷時にファイル保存ダイアログボックスが表示されます。

■ Double sided option(11)：0（片面または標準）か1（両面）のいずれかを＜値1＞に渡します。
＜値1＞が1の場合、適用するとじしろを定義することができます。＜値2＞を使用して、0= 左とじ（デフォルト）または1= 上とじを設定します。

注：このオプションは、Windowsでのみ使用可能です。

■ Spooler document name option(12)：＜値1＞に、プリントサーバ文書の一覧に表示しなければならない印刷文書の名前を渡します。標準操作（メソッドの場合はメソッド名、レコードの場合はテーブル名を使用等）を使用、または復帰するには、＜値1＞に空の文字列を渡します。

警告：新しい名前か空の文字列が渡されるまで、この命令文で指定した名前がセッションの印刷文書全体に対して使用されます。

■ Mac spool file format option(13)：値1に対し、印刷ジョブをPDFモード（デフォルト値）に設定する場合は0を渡し、印刷ジョブを“強制的に”PostScriptモードに設定する場合は1を渡します。また、このオプションにより、4D Write、4D View、4D Drawプラグインの印刷モードを変更することができます。一方、Windowsにおいて、このコマンドは無効です。

注：Mac OS X上では、デフォルトとして印刷はPDFとして行われます。しかし、PDFのプリントドライバでは、カプセル化PostScript情報（EPS）を含むPICTピクチャがサポートされません。これらのピクチャは、具体的に言うと、ベクトル描画ソフトにより生成されます。

この問題を回避するため、このオプションを使用して、カレントセッション中にMac OS X上で用いる印刷モードを変更することができます。PostScriptモードで印刷を行うと、好ましくない副次的な結果をもたらす可能性があることを覚えておいてください。

このコマンドを使用して設定を行うと、4Dアプリケーション全体のセッションの間中、そのプリントオプションが保持されます。**PRINT SELECTION**、**PRINT LABEL**、**PRINT RECORD**、**Print form**、**QR REPORT**コマンドおよび「デザイン」モードを含めた4th Dimensionの印刷全般に対して、この設定が使用されます。

注：SET PRINT OPTIONコマンドを用いて設定したプリントオプションがリセットされないように、PRINT SELECTION、PRINT LABEL、PRINT RECORDおよび PAGE BREAKコマンドでは、任意の引数“>”を必ず使用してください。

通常、複数の印刷設定用コマンドを使用する場合の優先順位は次のようになります。

1. SET CURRENT PRINTER（リセット）

2. 印刷用のパラメータをすべて変更するPAGE SETUPやその他のコマンド

3. SET PRINT OPTION（印刷オプションの個別変更）

＜オプション＞に渡した値が無効であるか、そのプリンタで＜オプション＞が利用できない場合、コマンドはエラーを返し（**ON ERR CALL**コマンドでインストールされたエラー管理メソッドを用いて、このエラーを阻止できます）、オプションの現在の値がそのまま保持されます。コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合には0が代入されます。

参照

GET PRINT OPTION、PRINT OPTION VALUES

GET PRINT OPTION

GET PRINT OPTION(オプション; 値1{; 値2})

引数	タイプ		説明
オプション	倍長整数	→	オプション番号
値1	倍長整数¦文字列	←	オプションの値1
値2	倍長整数	←	オプションの値2

説明

GET PRINT OPTIONコマンドは、プリント＜オプション＞の現在の値を返します。

引数＜オプション＞により、取得するオプションを指定することができます。値を渡すか、または「Print options」テーマ内にある次の定義済定数のいずれかを渡すことができます。

定数	タイプ	値
Paper option	倍長整数	1
Orientation option	倍長整数	2
Scale option	倍長整数	3
Number of copies option	倍長整数	4
Paper source option	倍長整数	5
Color option	倍長整数	8
Destination option	倍長整数	9
Double sided option	倍長整数	11
Spooler document name option	倍長整数	12
Mac spool file format option	倍長整数	13

このコマンドは、指定した＜オプション＞の現在の値を引数＜値1＞と（任意の）＜値2＞に返します。オプションおよび考えられる値に関する詳細は、**SET PRINT OPTION**コマンドの説明を参照してください。次に示す**GET PRINT OPTION**コマンド特定の機能に注意してください。

■ オプション=1（paper option）：＜値2＞を省略した場合、＜値1＞には現在の用紙名が返されます。＜値2＞を渡した場合、このコマンドは用紙の高さと幅をそれぞれ＜値1＞と＜値2＞に返します。プリンタが提供するすべての用紙フォーマットの名前、高さ、幅を取得するには、**PRINT OPTION VALUES**コマンドを使用してください。

■ オプション=2（orientation option）：1（縦方向）または2（横方向）が返されます。異なる方向オプションが使用されている場合、＜値1＞には0が代入されます。

■ オプション=5（paper source option）：＜値1＞には使用する用紙トレイの（**PRINT OPTION VALUES**コマンドから返されたトレイ配列内の）インデックスが返されます（＜値2＞は必ず省略してください）。

注：このオプションはWindows上でのみ使用できます。

■ オプション=8（color option）：＜値1＞にはカラー処理用のモードを表わすコードが返されます（1= 白黒（モノクロ）、2=カラー）。

注：このオプションは、Windows上でのみ使用できます。

■ オプション=9（destination option）：定義済リスト上に現在の値が存在しない場合、＜値1＞には-1が代入され、システム変数OKには1がセットされます。エラーが発生した場合、＜値1＞とシステム変数OKには0が代入されます。＜値1＞に1または5以外の定義済の値が代入されている場合、＜値2＞には印刷されたファイルのアクセスパスが納められます。

■ オプション=11（double sided option）：＜値1＞に、0（デフォルト値である標準か片面印刷）または1（両面）が返されます。
＜値1＞1の場合、＜値2＞には0=左とじ（デフォルト）または1=上とじのいずれかの値が返されます。

注：このオプションは、Windows上でのみ使用できます。

■ オプション=12（spooler document name option）：事前に定義されている場合、＜値1＞には現在の印刷文書の名前が返されます。事前定義されていない場合には、空の文字列が返されます。

コマンドが正常に終了すると、システム変数OKには1が代入され、それ以外の場合には0が代入されます。

参照

PRINT OPTION VALUES、SET PRINT OPTION

PRINT OPTION VALUES

PRINT OPTION VALUES(オプション; 名前配列{; 情報1配列; 情報2配列})

引数	タイプ		説明
オプション	倍長整数	→	オプション番号
名前配列	テキスト配列	←	値の名称
情報1配列	倍長整数配列	←	オプションの値（1）
情報2配列	倍長整数配列	←	オプションの値（2）

説明

PRINT OPTION VALUESコマンドは、指定したプリント＜オプション＞に対して使用できる値の名称リストを引数＜名前配列＞に返します。オプションとして、＜情報1配列＞と＜情報2配列＞内に各値に関する情報を取得することができます。

引数＜オプション＞を使用して、取得するオプションを指定することができます。必ず、「Print options」テーマ内にある次の定数のいずれかを渡してください（値名の一覧を返すことができるオプション）。

定数	タイプ	値
Paper option	倍長整数	1
Paper source option	倍長整数	5

コマンドの実行後、配列＜名前配列＞ならびに配列＜情報1配列＞と＜情報2配列＞（適用可能な場合）には、使用できる値の名称と情報が代入されます。

引数＜オプション＞に値1（paper option）を渡した場合、コマンドから次の情報が返されます。

■ ＜名前配列＞には、使用できる用紙フォーマットの名前。

■ ＜情報1配列＞には、各用紙フォーマットの高さ。

■ ＜情報2配列＞には、各用紙フォーマットの幅。

注：この情報を取得するには、プリンタドライバがそのプリンタの有効なPPD（PostScript Printer Description）へアクセスできなくてはなりません。

SET PRINT OPTIONコマンドを使用して特定の用紙フォーマットを適用するために、＜名前配列＞のいずれかの値を渡すか、または＜情報1配列＞と＜情報2配列＞の対応する値を渡すことができます。

引数＜オプション＞に値5（paper source option）を渡した場合、このコマンドからは利用可能な別のトレイ名が＜名前配列＞に返され、＜情報1配列＞には内部的なWindowsの ID番号が返されます（＜情報2配列＞は空のままです）。

配列内の値の順序は、プリンタドライバにより決まります。**SET PRINT OPTION**コマンドを使用してトレイを指定するには、目的の要素のインデックスを＜名前配列＞または＜情報1配列＞に設定された通りに渡さなくてはなりません。

注：このオプションは、Windows上でのみ使用できます。

各種プリントオプションに関する詳細は、**SET PRINT OPTION**コマンドと**GET PRINT OPTION**コマンドの説明を参照してください。

これらのコマンドから返される情報はすべて、OSにより提供されます。特定のオプションについての詳細は、お使いのシステムのドキュメントを参照してください。

ACCUMULATE

ACCUMULATE(データ1 {;...; データN})

引数	タイプ		説明
データ	フィールド または変数	→	累計する数値フィールドまたは変数

説明

ACCUMULATEコマンドは、**PRINT SELECTION**コマンドを使ってプリントするフォームレポート中で累計するフィールドまたは変数を指定します。

警告：コンパイルモードでは、ブレーク処理を行うレポートを印刷する前に、BREAK LEVELコマンドとACCUMULATEコマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を生成します。Subtotalコマンドの説明を参照してください。

フォームレポート内の数値フィールドまたは変数に対する小計を求める場合に、**ACCUMULATE**コマンドを使用します。**ACCUMULATE**コマンドは、4th Dimensionに対して、引数＜データ＞に対する小計を記憶するように指示します。小計は**BREAK LEVEL**コマンドで指定された各ブレークレベルに対して累計されます。

PRINT SELECTIONコマンドを使ってレポートを印刷する前に、**ACCUMULATE**コマンドを実行します。

フォームメソッドかオブジェクトメソッドで**Subtotal**関数を使用して、引数＜データ＞の1つの小計を求めます。

ACCUMULATEコマンドの使用例は、前ページ**BREAK LEVEL**コマンドの例を参照してください。

参照

BREAK LEVEL、ORDER BY、PRINT SELECTION、Subtotal

Subtotal

Subtotal(データ {; ページブレーク}) → 数値

引数	タイプ		説明
データ	フィールド	→	小計を求める数値フィールドまたは変数
ページブレーク	数値	→	ページブレークを発生させるブレークレベル
戻り値	数値	←	データの小計

説明

Subtotal関数は、現在または最後のブレークレベルに対するデータの小計を返します。**Subtotal**関数は、**PRINT SELECTION**コマンドでソートされたセレクションを印刷する場合と、「ユーザ」モードで「プリント...」メニューを使用して印刷を行う場合にのみ機能します。＜データ＞のタイプは実数、整数、倍長整数のいずれかでなければなりません。印刷するフォームのブレークエリアの変数に、**Subtotal**関数の結果を代入します。

警告：コンパイルモードでは、ブレーク処理を行ない、小計を計算するレポートを印刷する前に、BREAK LEVELコマンドとACCUMULATEコマンドを実行しなければなりません。このコマンドの最後の説明を参照してください。

Subtotal関数は、フォームメソッドまたはフォームのオブジェクトメソッドに記述しなければなりません。4th Dimensionは印刷前にフォームメソッドとオブジェクトメソッドをチェックし、**Subtotal**関数が指定されていればブレーク処理を開始します（インタープリタモードのみ）。

Subtotal関数の2番目の引数（オプション）で印刷中にページブレークを行います。＜ページブレーク＞が0の場合は、**Subtotal**関数はページブレークを行いません。＜ページブレーク＞が1の場合には**Subtotal**関数は各レベル1のブレークに対してページブレークを行います。＜ページブレーク＞が2の場合は、**Subtotal**関数はレベル1と2の各ブレークレベルに対してページブレークを行います。

ソートレベル n に対してブレークを行いたい場合、 n +1のレベルでカレントセレクションをソートしなければなりません（**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用する場合を除く）。これによって、最終レベルのフィールドまでソートすることにより、予期しないブレークが発生することを防ぎます。最終レベルのソートフィールドでブレークを行う場合は、そのフィールドを2回ソートします。

Tip：画面に表示された出力フォームからSubtotal関数を実行すると、エラーが発生し、フォームとエラーウインドウの間で更新作業の無限ループを引き起こします。このループから抜けるには、エラーウインドウの「アボート」ボタンをクリックする際に「Alt+Shift」キー（Windows）または「option+shift」キー（Macintosh）を押します（何回も繰り返さなければならないかもしれません）。これによりフォームのウインドウの更新が一時的に中断されます。エラーが再度発生するように別のフォームを出力フォームとして選択してください。フォームを表示と印刷の両方で使用したい場合、「デザイン」モードに移り、Subtotal関数を“Form event=On Printing Break”という判定式の中に配置します。

▼ 以下の例は、フォームのブレークエリア（B0、B0マーカの上のエリア）のオブジェクトメソッドです。変数“v給与”は、ブレークエリアにあります。このブレークレベルが発生すると、変数に“給与”フィールドの小計が代入されます。

```
Case of
    ¥(Form event=On Printing Break)
        v給与:=Subtotal([従業員]給与)
End case
```

参照

ACCUMULATE、BREAK LEVEL、Level、PRINT SELECTION

Level

Level → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	現在のブレークまたはヘッダのレベル

説明

Level関数は、カレントブレークまたはヘッダのレベルを調べるために使用します。この関数は、**On Header**および**On Printing Break**イベントのレベル数を返します。

レベル0は、印刷する最後のレベルです。総合計を印刷するのに適しています。**Level**関数は、最初のソートフィールドのブレークを印刷するときに1を返し、2番目ソートフィールドでブレークを印刷するときに2を返します。

▼ 以下の例は、フォームメソッドのテンプレートです。集計レポートでフォームが出力フォームとして使用される際に、発生する可能性のあるイベントをすべて示しています。ヘッダやブレークがプリントされるときに**Level**関数が呼び出されます。

```
` 集計レポートの出力フォームとして使用されるフォームのメソッドt
$vpFormTable:=Current form table
Case of
    ` ...
    ¥(Form event=On Header)
        ` ヘッダエリア印刷開始
        Case of
            ¥(Before selection($vpFormTable->))
                ` 最初のブレークヘッダ用のコード
            ¥(Level = 1)
                ` レベル1のブレークヘッダ用のコード
            ¥(Level = 2)
                ` レベル1のブレークヘッダ用のコード
                ` ...
        End case
    ¥(Form event=On Printing Details)
        ` レコードの印刷開始
        ` 各レコードのコード
    ¥(Form event=On Printing Break)
        ` ブレークエリア印刷開始
        Case of
            ¥(Level = 0)
```

```
                    ` ブレークレベル0のコード
                ¥(Level = 1)
                    ` ブレークレベル1のコード
                    ` ...
            End case
        ¥(Form event=On Printing Footer)
            If(End selection($vpFormTable->))
                ` 最後のフッタのコード
            Else
                ` フッタのコード
            End if
    End case
```

参照

ACCUMULATE、BREAK LEVEL、Form event、PRINT SELECTION

PAGE SETUP

PAGE SETUP({テーブル ;} フォーム)

引数	タイプ		説明
テーブル	テーブル	→	フォームを含むテーブル 省略した場合、デフォルトテーブル
フォーム	文字列	→	ページセットアップに使用される フォーム

説明

PAGE SETUPコマンドは、＜フォーム＞に格納される用紙設定情報を設定します。用紙設定情報は、「デザイン」モードでフォームが保存される時に、フォームとともに保存されます。

以下の条件のもとでは、「用紙設定」ダイアログボックスが表示されず、デフォルトの用紙設定でプリントが行われます。

■ オプション引数*を指定して**PRINT SELECTION**コマンドを呼び出した場合

■ オプション引数*を指定して**PRINT RECORD**コマンドを呼び出した場合

■ **PRINT SETTINGS**コマンドを呼び出さずに、一連の**Print form**関数を呼び出した場合

PAGE SETUPコマンドを使用すると、「用紙設定」ダイアログボックスを表示せずにデフォルト以外の用紙設定情報を使用することができます。

▼ [デザイン項目].テーブルには複数のフォーム（空）が作成されています。フォーム“PS100”には縮尺率が100%の用紙設定情報が割り当てられています。また、フォーム“PS90”には縮尺率が90%の用紙設定情報が割り当てられています。以下のプロジェクトメソッドにより、プリントの度に「用紙設定」ダイアログボックス（表示はされません）に縮尺率を指定しなくても、さまざまな縮尺率でテーブルのセレクションをプリントすることができます。

```
` 自動縮尺プリントプロジェクトメソッド
` 自動縮尺プリント(ポインタ ; 文字列 {; 倍長整数})
` 自動縮尺プリント(->[テーブル]; "出力フォーム" {; 縮尺率})
If(Count parameters>=3)
   PAGE SETUP([デザイン項目];"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
```

このプロジェクトメソッドを作成した後は、以下のように使用します。

```
` 現在の請求書の検索
QUERY([請求];[請求]支払=False)
` 集計表を90%の縮尺でプリント
自動縮尺プリント(->[請求];"集計表";90)
` 明細一覧を50%の縮尺でプリント
自動縮尺プリント(->[請求];"明細一覧";50)
```

参照

Print form、PRINT RECORD、PRINT SELECTION

Get print marker

Get print marker(マーカー番号) → 数値

引数	タイプ		説明
マーカー番号	数値	→	マーカー番号
戻り値	数値	←	マーカーの位置

説明

Get print marker関数を使用し、印刷中にマーカーの現在位置を取得することができます。

この関数は次の2つの状況で使用することができます。

■ 「On header」フォームイベント中に、**PRINT SELECTION**および**PRINT RECORD**コマンドを使用した場合。

■ 「On Printing Detail」フォームイベント中に、**Print form**関数を使用した場合。

座標はピクセル単位で返されます（1ピクセル＝1/72インチ）。

引数<マーカー番号>には、「Form area」テーマ内の定数のうちのいずれかを渡します。

定数	タイプ	値
Form Header	倍長整数	200
Form Header1...10	倍長整数	201...210
Form Detail	倍長整数	0
Form Break0...9	倍長整数	300...309
Form Footer	倍長整数	100

例題

SET PRINT MARKERコマンドの例題を参照してください。

参照

MOVE OBJECT、SET PRINT MARKER

PRINT SETTINGS

PRINT SETTINGS

説明

PRINT SETTINGSコマンドはプリンタダイアログボックスを表示します。最初に「用紙設定」ダイアログボックスを表示し、その次に「印刷設定」ダイアログボックスを表示します。

Print form関数の前に**PRINT SETTINGS**コマンドを実行してください。**PRINT SETTINGS**コマンドは、他のコマンドで印刷した場合には全く効果がありません。

「印刷設定」ダイアログボックスには、「プレビュー（Macintosh版では、スクリーンへプリント）」チェックボックスがあります。このチェックボックスでレポートを画面に印刷するように指定することができます。**PRINT SETTINGS**コマンドを実行する前に、**SET PRINT PREVIEW**コマンドを使用してこのチェックボックスをあらかじめセットする、またはセットし直すことができます。

ユーザが両方のダイアログボックスで「OK」ボタンをクリックすると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに0が代入されます。

前述の**PRINT FORM**コマンドの例を参照してください。

参照

PAGE BREAK、Print form、SET PRINT PREVIEW

SET PRINT PREVIEW

SET PRINT PREVIEW(プレビュー)

引数	タイプ		説明
プレビュー	ブール	→	True=スクリーン上でプレビューする False=プレビューしない

説明

このコマンドは、「プリント」ダイアログボックスの「プレビュー（Macintosh版では、スクリーンへプリント）」チェックボックスのオン／オフをメソッドで切り替えるためのものです。＜プレビュー＞に「True」を渡すと、「プレビュー（スクリーンへプリント）」チェックボックスはチェックされ、「False」を渡すと、チェックが外されます。この設定はプロセスに対してローカルであり、他のプロセスやユーザの印刷には影響を与えません。

▼以下の例は、検索結果を表示するために、まず「プレビュー（スクリーンへプリント）」チェックボックスをオンにし、それからオフに切り替えます。

```
QUERY([顧客])
If(OK =1)
    SET PRINT PREVIEW(True)
    PRINT SELECTION([顧客] ; *)
    SET PRINT PREVIEW(False)
End if
```

参照

PRINT RECORD、PRINT SELECTION、PRINT SETTINGS

SET PRINT MARKER

SET PRINT MARKER(マーカー番号; 位置{;*})

引数	タイプ		説明
マーカー番号	数値	→	マーカー番号
位置	数値	→	マーカーの新しい位置
*	*	→	渡した場合、後続のマーカーを移動する 省略した場合、後続のマーカーを移動しない

説明

SET PRINT MARKERコマンドを使用し、印刷の際のマーカー位置を指定することができます。このコマンドを**Get print marker**関数、**MOVE OBJECT**コマンド、**Print form**関数と組み合わせて使用することにより、印刷エリアのサイズを調節することができます。

SET PRINT MARKERコマンドは、次の2つの状況において使用可能です。

■「On header」フォームイベント中に、**PRINT SELECTION**および**PRINT RECORD**コマンドを使用した場合。

■「On Printing Detail」フォームイベント中に**Print form**関数を使用した場合。
この操作により、カスタマイズしたレポートの印刷をスムーズに行えます（例題を参照）。

コマンドの有効範囲は印刷に限定され、画面上には変更が表示されません。フォームに対して行った変更は保存されません。

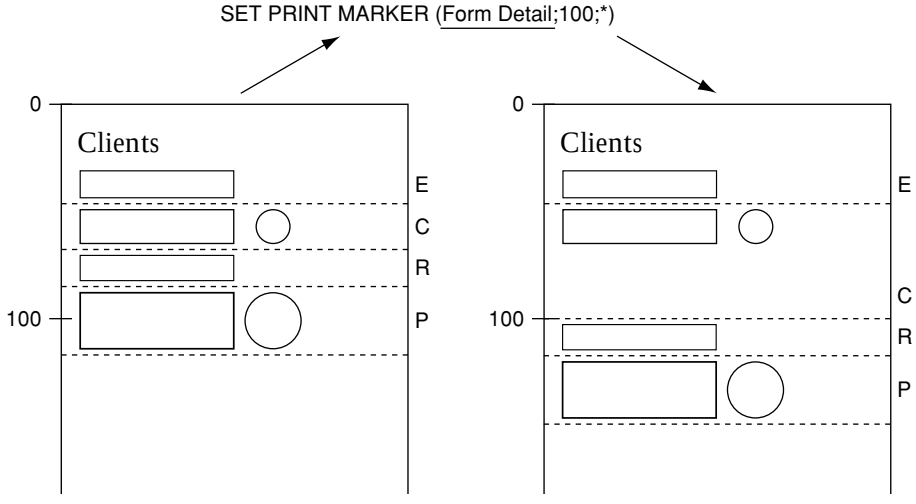
引数<マーカー番号>には、「Form area」テーマ内の定数のいずれかを渡します。

定数	タイプ	値
Form Header	倍長整数	200
Form Header1...10	倍長整数	201...210
Form Detail	倍長整数	0
Form Break0...9	倍長整数	300...309
Form Footer	倍長整数	100

<位置>には、目的とする新しい位置をピクセル単位で渡します。

オプション引数<*>を渡すと、このコマンドの実行時に、<マーカー番号>で指定したマーカーより下側に位置するすべてのマーカーが、指定したマーカーと同じピクセル数だけ、同じ方向へ移動します。このマーカーより下側にあるエリア内のオブジェクトもすべて移動します。

引数<*>を使用すると、後続の各マーカーの最初の位置より下側に<マーカー番号>で指定したマーカーを位置付けることができます。これら後続のマーカーも同時に移動します。

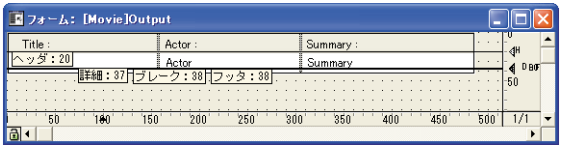


注：

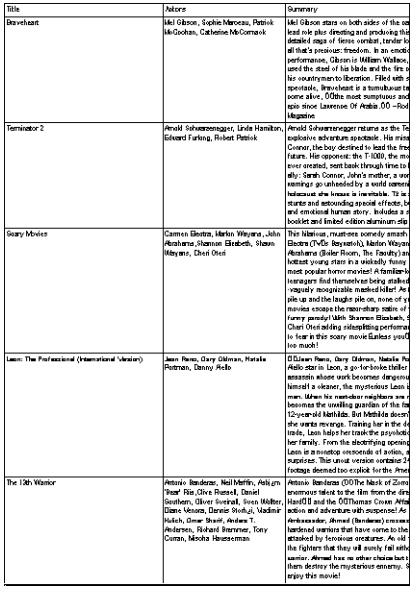
- このコマンドで変更できるのは、既存するマーカーの位置だけです。マーカーを追加することはできません。フォームに存在しないマーカーを指定した場合、コマンドは何も行いません。
- ストラクチャモードにおける印刷用マーカーの仕組みは変わりません。つまり、あるマーカーは、それより上にあるマーカーを飛び越えて移動したり、後続のマーカーよりも下側に移動することはできません（引数<*>を使用しない場合）。

例題

この例題を使用して、3つのカラムがあるレポートを作成することができます。各行の高さは、フィールド内容に応じて実行中に計算されます。印刷に使用する出力フォームは次の通りです。



このフォームに対して、「On Printing Detail」フォームイベントが選択されています（印刷するエリアに関係なく、**Print form**関数はこのタイプのフォームイベントだけを生成する点に留意してください）。レコードごとに、“Actors”または“Summary”カラムの内容に応じて行の高さを調整しなくてはなりません。目的とする結果を次に示します。



Title	Actors	Summary
Braveheart	Mel Gibson, Sophie Marceau, Patrick McGoohan, Catherine McCormack	Mel Gibson stars on both sides of the ca lead role plus directing and producing thi detailed saga of fierce combat, tender lo all that's precious: freedom. In an emoti performance, Gibson is William Wallace, used the steel of his blade and the fire o his countrymen to liberation. Filled with a spectacle, Braveheart is a tumultuous tr some alive, 00the most sumptuous and epic since Lawrence Of Arabia.00 --Rod Magazine
Terminator 2	Arnold Schwarzenegger, Linda Hamilton, Edward Furlong, Robert Patrick	Arnold Schwarzenegger returns as the Te explosive adventure spectacle. His mis Connor, the boy destined to lead the fre future. His opponent: the T-1000, the mo ever created, sent back through time to ally: Sarah Connor, John's mother, a wo warnings go unheeded by a world cameri holocaust she knows is inevitable. Tt is stunts and astounding special effects, b and emotional human story. Includes a booklet and limited edition aluminum slip
Scary Movies	Carmen Electra, Marlon Wayans, John Abraham, Shannon Elizabeth, Shawn Wayans, Cheri Oteri	This hilarious, must-see comedy smash Electra (TV0s Baywatch), Marlon Wayan Abraham (Killer Room, The Faculty) an hottest young stars in a wickedly funny most popular horror movies! A familiar-k teenagers find themselves being stalked -vaguely recognizable masked killer! As pile up and the laughs pile on, none of y movies escape the mercenary satire of funny parody! With Shannon Elizabeth, S Cheri Oteri adding sidesplitting performa to fear in this scary movie Eunless you0 too much!
Leon: The Professional (International Version)	Jean Reno, Gary Oldman, Natalie Portman, Danny Aiello	0 Eileen Reno, Gary Oldman, Natalie Po Aiello star in Leon, a go-for-broke thriller assassin whose work becomes dangerou himself a cleaner, the mysterious Leon i man. When his merciless neighbors are becomes the unwilling guardian of the fa 12-year-old Mathilda. But Mathilda doesn' she wants revenge. Training her in the d trade, Leon helps her track the psychoti her family. From the electrifying opening Leon is a nonstop crescendo of action, a surprises. This uncut version contains 2 footage deemed too explicit for the Amer
The 13th Warrior	Antonio Banderas, Neil Martin, Ashlyn 'Bear' Fiss, Clive Russell, Daniel Southern, Oliver Sweinall, Sean Walter, Diane Venora, Dennis Storhoi, Vladimir Kulich, Omar Sharif, Andrew T. Andersen, Richard Brammer, Tony Curran, Nischa Haussermann	Antonio Banderas (00The Mask of Zorro enormous talent to the film from the dire Hand00 and the 00Thomas Crown Affai action and adventure with suspense! As Ambassador, Ahmed (Banderas) crosses hardened warriors that have come to the attacked by ferocious creatures. As old the fighters that they will surely fail with warrior, Ahmed has no other choice but t them destroy the mysterious enemy. S enjoy this movie!

印刷用のプロジェクトメソッドは次の通りです。

```
C_LONGINT(vLprint_height;$vLheight;vLprinted_height)
C_STRING(31;vSprint_area)
PAGE SETUP([Film];"Print_List3")
GET PRINTABLE AREA(vLprint_height)
vLprinted_height:=0
ALL RECORDS([Film])

vSprint_area:="Header"        `ヘッダエリアの印刷
$vLheight:=Print form([Film];"Print_List3";Form Header)
$vLheight:=21          `Fixed height
vLprinted_height:=vLprinted_height+$vLheight

While(Not(End selection([Film])))
        vSprint_area:="Detail"          `詳細エリアの印刷
        $vLheight:=Print form([Film];"Print_List3";Form Detail)
                `フォームメソッドにおいて詳細行の計算が実行される
        vLprinted_height:=vLprinted_height+$vLheight
        If(OK=0)        `フォームメソッドでCANCELが実行された
                PAGE BREAK
                vLprinted_height:=0
                vSprint_area:="Header"        `ヘッダエリアの再印刷
                $vLheight:=Print form([Film];"Print_List3";Form Header)
```

```
            $vLheight:=21
            vLprinted_height:=vLprinted_height+$vLheight
            vSprint_area:="Detail"
            $vLheight:=Print form([Film];"Print_List3";Form Detail)
            vLprinted_height:=vLprinted_height+$vLheight
        End if
        NEXT RECORD([Film])
    End while
    PAGE BREAK      `最後のページが印刷されたことを確認
```

フォームメソッドPrint_List3は次の通りです。

```
    C_LONGINT($l;$t;$r;$b;$fixed_wdth;$exact_hght;$l1;$t1;$r1;$b1)
    C_LONGINT($final_pos;$i)
    C_LONGINT($detail_pos;$header_pos;$hght_to_print;$hght_remaining)

    Case of
        ¥(vSprint_area="Detail") `詳細行の印刷が進行中
        GET OBJECT RECT([Film]Actors;$l;$t;$r;$b)
        $fixed_wdth:=$r-$l`Actorsテキストフィールドサイズの計算
        $exact_hght:=$b-$t
        BEST OBJECT SIZE([Film]Actors;$wdth;$hght;$fixed_wdth)
            `内容に応じたフィールドの最適サイズ
        $movement:=$hght-$exact_hght
        GET OBJECT RECT([Film]Summary;$l1;$t1;$r1;$b1)
        $fixed_wdth1:=$r1-$l1`Summaryテキストフィールドサイズの計算
        $exact_hght1:=$b1-$t1
        BEST OBJECT SIZE([Film]Summary;$wdth1;$hght1;$fixed_wdth1)
            `内容に応じたフィールドの最適サイズ
        $movement1:=$hght1-$exact_hght1
        If($movement1>$movement)
            `最も高さがあるフィールドを決定する
            $movement:=$movement1
        End if

        If($movement>0)
            $position:=Get print marker(Form Detail)
            $final_pos:=$position+$movement
                `Detailマーカーと、これに続くマーカーを移動
            SET PRINT MARKER(Form Detail;$final_pos;*)
                `テキストエリアのサイズ変更
            MOVE OBJECT([Film]Actors;$l;$t;$r;$hght+$t;*)
```

```
            MOVE OBJECT([Film]Summary;$l1;$t1;$r1;$hght1+$t1;*)
                `分割ラインのサイズ変更
            GET OBJECT RECT(*;"H1Line";$l;$t;$r;$b)
            MOVE OBJECT(*;"H1Line";$l;$final_pos-1;$r;$final_pos;*)
            For($i;1;4;1)
                GET OBJECT RECT(*;"VLine"+String($i);$l;$t;$r;$b)
                MOVE OBJECT(*;"VLine"+String($i);$l;$t;$r;$final_pos;*)
            End for
        End if

            `利用できるスペースの計算
        $detail_pos:=Get print marker(Form Detail)
        $header_pos:=Get print marker(Form Header)
        $hght_to_print:=$detail_pos-$header_pos
        $hght_remaining:=printing_height-vLprinted_height
        If($hght_remaining<$hght_to_print) `高さが不十分
            CANCEL      `フォームを次ページに移動
        End if
    End case
```

参照

BEST OBJECT SIZE、GET OBJECT RECT、Get print marker、MOVE OBJECT、PAGE BREAK、Print form、PRINT RECORD、PRINT SELECTION

GET PRINTABLE MARGIN

GET PRINTABLE MARGIN(左; 上; 右; 下)

引数	タイプ		説明
左	数値	←	左マージン
上	数値	←	上マージン
右	数値	←	右マージン
下	数値	←	下マージン

説明

GET PRINTABLE MARGINコマンドは、**Print form**関数を使用して定義された各種マージンの現在値を返します。

値は、用紙の端からの長さがピクセル単位で返されます。

GET PRINTABLE AREAコマンドを使用すると、用紙サイズの取得、および印刷可能領域の計算を行うことができます。

印刷可能マージンの管理

デフォルトとして、4th Dimensionにおける印刷に関する計算は“印刷可能マージン”に基づいています。このシステムの利点は、フォームが自動的に新しいプリンタに適応するというところです（フォームは印刷可能領域に配置されているため）。その一方で、事前印刷用のフォームの場合は、プリンタを変更すると印刷可能マージンが変わる可能性があるため、正確に印刷されるように各項目を配置することができませんでした。

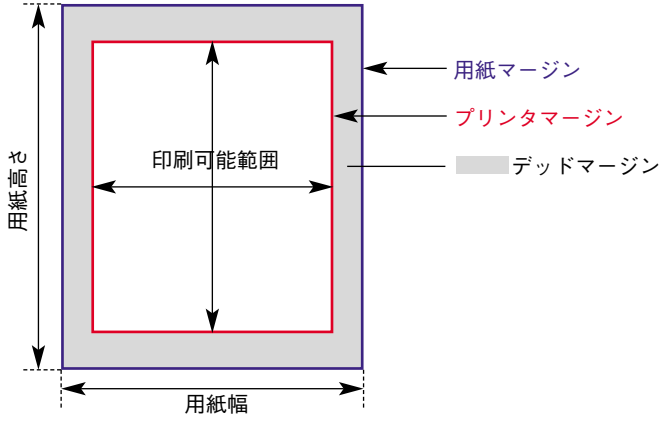
4th Dimensionバージョン6.8.1以降、**Print form**関数、**PRINT RECORD**および**PRINT SELECTION**コマンドを使用して行うフォームの印刷を、各プリンタ共通の固定マージン（用紙マージン、つまり用紙の物理的な許容限界寸法）に基づいて行うことができるようになりました。これを行うには、**GET PRINTABLE MARGIN**、**SET PRINTABLE MARGIN**および**GET PRINTABLE AREA**コマンドを使用します。

印刷用語について

■ **用紙マージン：**用紙マージンは、用紙の物理的な許容限界寸法に相当します。

■ **プリンタマージン：**プリンタマージンは、その位置を越えてプリンタが印刷できないことを示すマージンです（プリントローラー、プリンタヘッドの行程終端等の物質的な理由により）。この値は、プリンタやフォーマットごとに異なります。

■ **デッドマージン：**用紙マージンとプリンタマージンとの間の領域のことです。



参照

GET PRINTABLE AREA、Print form、SET PRINTABLE MARGIN

SET PRINTABLE MARGIN

SET PRINTABLE MARGIN(左; 上; 右; 下)

引数	タイプ		説明
左	数値	→	左マージン
上	数値	→	上マージン
右	数値	→	右マージン
下	数値	→	下マージン

説明

SET PRINTABLE MARGINコマンドを使用すると、**Print form**関数の使用時に、各種印刷マージンの値を設定することができます。

引数<左>、<上>、<右>、<下>には、以下の値のいずれかを渡すことができます。

■ 0= 用紙マージンを使用

■ -1= プリンタマージンを使用

■ 値>0= ピクセル単位のマージン（72dpiの1ピクセルは、約0.4 mm）

引数<右>および<下>の値はそれぞれ、用紙の右端および下端からの長さを表します。

注：印刷管理ならびに4D用語に関する詳細は、**GET PRINTABLE MARGIN**コマンドの説明を参照してください。

デフォルトとして、4th Dimensionはプリンタマージンに基づいて印刷を行います。**SET PRINTABLE MARGIN**コマンドの実行後は、変更後のパラメータがセッション全体の同一プロセス内で維持されます。

例題

(1)次の例題により、デッドマージンのサイズを取得することができます。

```
SET PRINTABLE MARGIN(-1;-1;-1;-1)    `プリンタマージンを設定
GET PRINTABLE MARGIN($l;$t;$r;$b)
      `$l、$t、$rおよび$bは、用紙のデッドマージンに相当する
```

(2)次の例題により、用紙サイズを取得することができます。

```
SET PRINTABLE MARGIN(0;0;0;0)        `用紙マージンを設定
GET PRINTABLE AREA($height;$width)
      `A4サイズの場合：$height=842；$width=595（ピクセル）
```

参照

GET PRINTABLE MARGIN、Get printed height、Print form

GET PRINTABLE AREA

GET PRINTABLE AREA(高さ{; 幅})

引数	タイプ		説明
高さ	数値	←	印刷可能領域の高さ
幅	数値	←	印刷可能領域の幅

説明

GET PRINTABLE AREAコマンドは、印刷可能領域のサイズを引数<高さ>および<幅>へピクセル単位で返します。このサイズは、現在の印刷設定、用紙方向等によって異なります。

返されるサイズは、各ページごとに変わることはありません（例えば、ページブレークの後）。

このコマンドを**Get printed height**関数と一緒に使用すると、印刷可能なピクセル数の認識やページ上のオブジェクトのセンタリングを行うことができるので便利です。

注：印刷管理ならびに4D用語に関する詳細は、GET PRINTABLE MARGINコマンドの説明を参照してください。

ページ全体のサイズを確認するには、次のいずれかの方法を行います。

■ このコマンドで返された値に**GET PRINTABLE MARGIN**コマンドで取得したマージンを加算します。

■ または、次の構文を使用します。

```
SET PRINTABLE MARGIN(-1;-1;-1;-1)  `印刷可能マージンを設定
GET PRINTABLE AREA(hPaper;wPaper)`用紙サイズ
```

参照

GET PRINTABLE MARGIN、Print form

Get printed height

Get printed height → 数値

引数	タイプ		説明
このコマンドには、引数は必要ありません			
戻り値	数値	←	マーカーの位置

説明

Get printed height関数は、**Print form**関数を使って印刷されたセクション全体の高さ（ピクセル単位）を返します。

返される値は、0（ページ上端）から**GET PRINTABLE AREA**コマンドによって返される全体の高さ（印刷可能領域の最大サイズ）までの間の値です。

Print form関数を使って新しいセクションを印刷する場合は、新しいセクションの高さがこの値に加えられます。使用できる印刷可能領域がこのセクションの印刷に不十分である場合は、新しいページが作成され、値0が返されます。

右および左の印刷可能マージンは、上および下のマージン（**SET PRINTABLE MARGIN**コマンドを使って指定可能）とは異なり、返される値に影響を与えません。

注：印刷管理ならびに4D用語に関する詳細は、GET PRINTABLE MARGINコマンドの説明を参照してください。

参照

GET PRINTABLE AREA、Print form、SET PRINTABLE MARGIN

ピクチャ

サポートされるピクチャのフォーマット

以下の表に、MacintoshとWindowsプラットフォームにおいてサポートされるピクチャのフォーマットを要約します。

切り取りと貼り付け：サポートされるフォーマット

	PICT	EMF	WMF	BITMAP
Macintosh	○	-	-	-
Windows	○	○ PicCommentに埋め込み	○ PicCommentに埋め込み	○ Macintosh PICTに変換

表示：サポートされるフォーマット

	PICT	QuickTime	embedded WMF	embedded EMF
Macintosh	○	○	×	×
Windows	○	○ NT & WIN 9x + QT 4	○	○

■ WMFファイル（Windows Metafile）について

これらのファイルは、必ず“ポジショナブル”ファイルであり、ピクチャのサイズと解像度を表わすヘッダが含まれていなくてはなりません。ヘッダが見当たらない場合、4Dはそのピクチャファイルを読み込むことができません。WindowsプラットフォームにおけるWMFファイルは、Mac OSプラットフォームのPICTに相当します。つまり、各システムに対応するベクトルデータとビットマップデータ（ドローとペイント）の両方を納めることができます。WindowsのWMFファイルに関する主な利点は2つあり、表示速度がより高速である点と（変換が不要）、万国共通に利用できる点です。Windowsアプリケーションなら、すべてこのフォーマットで書き出すことができます。

しかし、このフォーマットを使用すると、Macintoshプラットフォーム上ではそのピクチャを表示できなくなるという点に注意が必要です。

■ EMFファイル（Windows Enhanced Metafile）について

このフォーマットは、WMFを改良したものです。将来的に、数々のWindowsアプリケーションでサポートされるようになるでしょう。このフォーマットの主な利点は、ベジェや変形などの基本的要素が機能強化された点です。このフォーマットを使用すると、Macintoshプラットフォーム上ではそのピクチャを表示できなくなります。

4DでApple QuickTimeを使用する

4Dは、Apple社のQuickTimeルーチンを使用して、データベースにおける画像の変換や圧縮を実現しています。

Apple QuickTime圧縮

AppleはQuickTimeを使用して、JPEGのような新たな圧縮技術をインプリメントします。Appleは、オリジナルのPICT仕様に新たなopcodeを追加して、MacintoshアプリケーションがQuickTime画像を変更することなく取り扱えるようにしました。アプリケーションが埋め込まれたQuickTimeデータを含む画像を描画するようシステムに指示すると、QuickTimeが存在する場合にはビットマップが拡張されて表示されます。QuickTimeがインストールされていなければ、QuickTimeのopcodeは無視されます。この技術はユーザに対して透過的です。また1メガバイトの画像が40キロバイトのPICTに格納でき、表示する前に拡張する必要がないため、メモリが最小限で済みます。

Windowsにおいて、4Dはバージョン4 以上のQuickTime for Windowsのインストールを必要とします。インストールされていないと、画像圧縮を行えません。

注：QuickTimeを必要とするがデスクトップファイルを使用する2つの4Dコマンド（LOAD COMPRESS PICTURE FROM FILE、COMPRESS PICTURE FILE）はWindowsでは動作しません。

変換

WRITE PICTURE FILEなどの4Dコマンドを使用して、異なるフォーマットで画像の変換や保存が可能です。通常、これらのコマンドが動作するにはQuickTimeが必要となります。Windowsにおいて、4Dはバージョン4 以上のQuickTime for Windowsのインストールを必要とします。インストールされていない場合、画像の変換を行えません。

4Dの変数やフィールドに画像を保存する際の内部フォーマットは、QuickTimeより供給されます。したがって、QuickTimeのインストールされていない環境では、4Dの変数やフィールドに納められた画像を見ることはできません。

QuickTime 4の変換コード

QuickTime 4で提供される標準の変換コードの一覧を以下に示します。コードはそれぞれ4桁です。すべてのマシンで同じコードが提供されるわけではないという点に注意してください。また、QuickTime 4を使用することにより、カスタマイズされた変換ルーチンを追加できます。**PICTURE TYPE LIST**コマンドを使用すると、コマンドが実行されたマシン上で利用可能なQuickTimeコードを知ることができます。

QuickTime 4 Codes	Names
PICT	QuickDraw PICT
PICS	PICS
GIFf	GIF
PNGf	PNG
TIFF	TIFF
8BPS	Photoshop(2.5 & 3.0)
SGI	Silicon Graphics
BMPf	BMP
JPEG	JPEG
JPEG	JFIF
PNTG	MacPaint
TPIC	TGA(Targa)
qdgx	QuickDraw GX Picture (QuickDraw GX がインストールされている場合)
qtif	QuickTime picture
FPix	FlashPix

画像圧縮エラー

画像圧縮コマンドを使用するが、QuickTimeがシステムにインストールされていない場合、4th Dimensionはエラーコード-9955を返します。QuickTimeが生成する他のエラーも返されます。これらのエラーは、**ON ERR CALL**でインストールされるエラー処理メソッドを使用して受け取ることができます。

参照

BLOB TO PICTURE、COMPRESS PICTURE、COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE、PICTURE PROPERTIES、Picture size、PICTURE TO BLOB、PICTURE TYPE LIST、READ PICTURE FILE、SAVE PICTURE TO FILE、WRITE PICTURE FILE

COMPRESS PICTURE

COMPRESS PICTURE(ピクチャ ; 方法 ; 品質)

引数	タイプ		説明
ピクチャ	ピクチャ	→	圧縮するピクチャ
		←	圧縮されたピクチャ
方法	文字列	→	圧縮方法（4バイト）
品質	整数	→	圧縮の品質

説明

このコマンドは、引数＜ピクチャ＞で渡されたフィールドまたは変数に入っているピクチャを圧縮します。

引数＜方法＞は、圧縮タイプを4バイトの文字列で指定します。

引数＜品質＞は、圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増加します。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

参照

COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE

LOAD COMPRESS PICTURE FROM FILE

LOAD COMPRESS PICTURE FROM FILE(ドキュメント参照番号 ; 方法 ; 品質 ; ピクチャ)

引数	タイプ		説明
ドキュメント参照番号	Docref	→	ドキュメントファイル参照番号
方法	文字列	→	圧縮方法（4バイト）
品質	整数	→	圧縮の品質（1～1000）
ピクチャ	ピクチャ	←	圧縮するピクチャ

説明

このコマンドは、ディスク上のドキュメントからロードされたピクチャを圧縮します。

Open document関数を使って、PICTドキュメントを開くことができます。次に、この関数から返されるドキュメントファイル参照番号を使い、ドキュメント中のPICTデータをロードして圧縮します。このコマンドはピクチャをメモリにロードし、指定した方法と品質に従って圧縮し、それを引数＜ピクチャ＞に返します。

ピクチャは圧縮される前にメモリへロードされます。ピクチャをロードするためのメモリが足りない場合は、**LOAD COMPRESS PICTURE FROM FILE**コマンドを呼び出す前に**COMPRESS PICTURE FILE**コマンドを使ってください。

引数＜方法＞は、圧縮タイプを4バイトの文字列で指定します。引数＜品質＞は、圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増加します。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

例題

以下の例は、「ファイルを開く」ダイアログボックスを表示し、ユーザはそこでPICTファイルを選択します。PICTファイルのピクチャがメモリにロードされて圧縮され、ピクチャ変数に格納されます。この後、ファイルが閉じられます。

```
vRef:=Open document("";"PICT")
If(OK=1)
        LOAD COMPRESS PICTURE FROM FILE(vRef;"jpeg";500;Picture)
CLOSE DOCUMENT(vRef)
End if
```

参照

COMPRESS PICTURE、COMPRESS PICTURE FILE、Pictures、SAVE PICTURE TO FILE

COMPRESS PICTURE FILE

COMPRESS PICTURE FILE(ドキュメント参照番号 ; 方法 ; 品質)

引数	タイプ		説明
ドキュメント参照番号	Docref	→	ドキュメントファイル参照番号
方法	文字列	→	圧縮方法（4バイト）
品質	整数	→	圧縮の品質（1～1000）

説明

このコマンドは、ディスク上のピクチャドキュメントファイルを圧縮します。現在のメモリではロードできないことがわかっているピクチャを圧縮する際に、このコマンドを使用します。圧縮した後は、**LOAD COMPRESS PICTURE FROM FILE**コマンドを使い、メモリにロードすることができます。

引数<方法>は、圧縮するタイプを 4バイトの文字列で指定します。

引数<品質>は、圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増えます。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

例題

以下の例は、「ファイルを開く」ダイアログボックスを表示し、ユーザはそこでPICTファイルを選択します。PICTファイルだけが表示されます。ピクチャはメモリにロードされて圧縮され、ピクチャ変数に格納されます。この後、ファイルが閉じられます。

```
vRef:=Open document("";"PICT")
If(OK=1)
    COMPRESS PICTURE FILE(vRef;"jpeg";500)
    LOAD COMPRESS PICTURE FROM FILE(vRef;"";500;vPict)
    CLOSE DOCUMENT(vRef)
End if
```

参照

COMPRESS PICTURE、LOAD COMPRESS PICTURE FROM FILE、SAVE PICTURE TO FILE

SAVE PICTURE TO FILE

SAVE PICTURE TO FILE(ドキュメント参照番号 ; ピクチャ)

引数	タイプ		説明
ドキュメント参照番号	Docref	→	ドキュメントファイル参照番号
ピクチャ	ピクチャ	→	保存するピクチャ

説明

SAVE PICTURE TO FILEコマンドは、**Create document**関数を使って作成されたドキュメント中に＜ピクチャ＞を保存します。

例題

以下の例は、ドキュメントファイルを作成し、ピクチャを保存します。

```
vRef:=Create document("";"PICT")
If(OK=1)
    SAVE PICTURE TO FILE(vRef;vPict)
    CLOSE DOCUMENT(vRef)
End if
```

参照

COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE.

PICTURE TO GIF

PICTURE TO GIF(ピクチャ; GIF用BLOB)

引数	タイプ		説明
ピクチャ	ピクチャ	→	ピクチャタイプのフィールドまたは変数
GIF用BLOB	BLOB	←	GIFタイプの画像をセットするBLOB

説明

このコマンドは、変数またはフィールドに保存されている（PICTタイプの）ピクチャから、GIFフォーマットの画像を作成します。

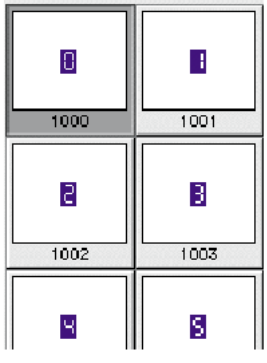
ピクチャタイプの変数またはフィールドをピクチャに、BLOBタイプの変数またはフィールドをblobGIFに渡します。コマンドの実行後、GIF用BLOBの内容はGIFフォーマットの画像になります。

注：GIFフォーマットは256色以上はサポートされていません。元のPICTピクチャがそれ以上の色数の場合は、失われる色があります。このコマンドで作られるGIF画像は色に応じて最適化され、白色部分は透過しますが、interlacedではありません。形式はGIF89aタイプのものです。

blobGIFの画像を**BLOB TO DOCUMENT**コマンドを使ってファイルに保存でき、またそれをWeb上に公開することもできます。

変換が成功したらシステム変数OKが1になります。そうでない場合は0になります。

接続カウンターを表示するGIF画像を作成する場合を想定してみましょう。データベースのピクチャライブラリ内に、すべての番号をピクチャとして登録しておきます。



On Web Connectionデータベースメソッドで、以下のコードを記述します。

```
If(Web Context)
	:
Else
	C_BLOB($blob)
	Case of
	:
	￥($1="/4dcgi/counter") 、GIF counterの生成
		`スタティックホームページを送っている間に、4DがこのURLを
				検出するとGIFカウンターを発生させる
	$blob:=gifcounter(<>nbHits) `GIF pictureを得る
		`<>nbHits変数は接続数
	SEND HTML BLOB($blob;"image/gif")
		`ピクチャを挿入してブラウザへ送る
	:
	End case
End if
```

giftcounterメソッドを示します。

```
C_LONGINT($1)
C_IMAGE($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
PICTURE TO GIF($img;$0)
```

ページをWebブラウザに送る時、4Dは下記のタイプのGIFピクチャを表示します。



PICTURE TO BLOB

PICTURE TO BLOB(ピクチャ;ピクチャBlob;フォーマット)

引数	タイプ		説明
ピクチャ	ピクチャ	→	ピクチャフィールドまたは ピクチャ変数
ピクチャBlob	BLOB	←	変換後のピクチャを受け取るBLOB
フォーマット	文字列(4)	→	ピクチャフォーマット（4桁）

説明

PICTURE TO BLOBコマンドは、4D変数やフィールドに保存されたピクチャを他のフォーマットに変換し、変換後のピクチャをBLOB内に納めます。

ピクチャタイプの4Dフィールドや変数は、引数＜ピクチャ＞に渡します。引数＜ピクチャBlob＞には、変換後のピクチャが入ったBLOB変数やフィールドが渡されます。

引数＜フォーマット＞には、変換フォーマットを設定する4桁の文字列を渡します。

このフォーマットとして指定できるのは、次のうちいずれかです。

■ QuickTimeフォーマット（**PICTURE TYPE LIST**コマンドの説明を参照）。この場合、マシン上にバージョン4以上のQuickTimeがインストールされていなければなりません。

■ GIFf（GIFフォーマット）、またはWBMP（Wireless Bitmap）。この2つにはQuickTime 4は不要です。

このコマンドを実行すると、＜ピクチャBlob＞には指定したフォーマットでピクチャが納められます。

変換が正常に終了すると、システム変数OKには1が代入されます。変換が失敗に終わった場合（QuickTimeがインストールされていない、またはコンバータが使用できない）、システム変数OKに0が代入され、生成されたBLOBは空のままです（0バイト）。

参照

BLOB TO PICTURE、PICTURE TO GIF、PICTURE TYPE LIST、WRITE PICTURE FILE

BLOB TO PICTURE

BLOB TO PICTURE(ピクチャBlob;ピクチャ)

引数	タイプ		説明
ピクチャBlob	BLOB	→	ピクチャの入ったBLOB
ピクチャ	ピクチャ	←	BLOBから取り出したピクチャ

説明

BLOB TO PICTUREコマンドは、BLOBに保存されたピクチャを4Dのピクチャ変数やピクチャフィールドに保存します。このコマンドではオリジナルのフォーマットは問いませんが、QuickTime 4と互換性をもつフォーマットでなくてはなりません。

警告：このコマンドには、Mac OSおよびWindows対応でバージョン4以上のQuickTimeが必要です。QuickTime 4がインストールされていない場合、このコマンドは何も行いません。

このコマンドは、**READ PICTURE FILE**コマンドとよく似ていますが、ファイルではなくBLOBに対してコマンドが適用されます。このコマンドを使用すると、本来のフォーマットでBLOBに保存されているピクチャを表示することができます。ピクチャのBLOBへのロードは、例えば、**DOCUMENT TO BLOB**コマンドまたは**PICTURE TO BLOB**コマンドを使用して行うことができます。

引数＜ピクチャBlob＞には、ピクチャを納めたBLOBタイプの変数やフィールドを渡します。このピクチャのフォーマットは、QuickTime 4でサポートされるものであればいずれの形式でも構いません。**PICTURE TYPE LIST**コマンドを使用すると、使用可能なフォーマットのリストを取得できます。QuickTime 4の標準フォーマットコードに関する説明は、**PICTURE TYPE LIST**コマンドを参照してください。

引数＜ピクチャ＞には、ピクチャを表示する4Dのピクチャフィールドまたはピクチャ変数を渡します。

注：4Dの変数やフィールドに画像を保存する際の内部フォーマットは、QuickTimeより供給されます。したがって、QuickTimeのインストールされていない環境では、4Dの変数やフィールドに納められた画像を見ることはできません。

コマンドが正常に終了すると、引数＜ピクチャ＞には表示するピクチャが納められます。

変換が成功するとシステム変数OKには1が代入されます。変換が失敗に終わった場合（QuickTimeがインストールされていない、またはBLOBに読み込み可能なピクチャが入っていない場合）システム変数OKに0が代入され、ピクチャ変数またはフィールドは空のままです。

参照

PICTURE TO BLOB、PICTURE TYPE LIST、READ PICTURE FILE

WRITE PICTURE FILE

WRITE PICTURE FILE(ファイル名; ピクチャ{; フォーマット})

引数	タイプ		説明
ファイル名	文字列	→	書き出すファイルの名前、または フルパス名、または空の文字列
ピクチャ	ピクチャ	→	書き出す4Dピクチャフィールド またはピクチャ変数
フォーマット	文字列(4)	→	ピクチャ書き出しフォーマットの Quicktimeコード（4桁） デフォルトでは“PICT”

説明

WRITE PICTURE FILEコマンドを使用し、定義した＜フォーマット＞で引数＜ピクチャ＞に渡されたピクチャをディスクに保存することができます。

警告：このコマンドでは、Mac OSおよびWindows対応のQuickTime変換ルーチン（バージョン4以上を推奨）を使用します。QuickTimeがインストールされていない場合、このコマンドはデフォルトとしてPICTファイルを作成します。

引数＜ファイル名＞には、作成するファイルのフルパス名、あるいはファイル名のみを渡すことができます。ファイル名だけを渡した場合、書き出したファイルはデータベースのストラクチャファイルの隣に置かれます。Windowsでは、ファイルの拡張子を指定する必要があります。

＜ファイル名＞に空の文字列（""）を渡すと、標準の「ファイル保存」ダイアログボックスが表示され、作成するファイルの名前、場所、フォーマットを指定できます。

引数＜ピクチャ＞には、ディスクに保存するピクチャが入ったピクチャ変数、またはピクチャフィールドを渡します。

オプションの引数＜フォーマット＞は、ピクチャの保存形式を定義します。この引数には、4桁のQuickTimeコードを指定してください。**PICTURE TYPE LIST**コマンドを使用すると、利用できるフォーマットのリストを取得できます。QuickTime 4の標準フォーマットコードに関する説明は、**PICTURE TYPE LIST**コマンドを参照してください。

引数＜フォーマット＞を省略した場合、またはQuickTimeがインストールされていない場合には、PICT形式でピクチャファイルが作成されます。

コマンドが正常に終了すると、システム変数Documentには作成されたファイルのフルパス名が納められ、システム変数OKには1が代入されます。それ以外の場合、システム変数OKに0が代入されます。

参照

PICTURE TO BLOB、PICTURE TYPE LIST、Pictures、READ PICTURE FILE

READ PICTURE FILE

READ PICTURE FILE(ファイル名; ピクチャ)

引数	タイプ		説明
ファイル名	文字列	→	読み込むファイルの名前または フルパス名、または空の文字列
ピクチャ	ピクチャ	←	ファイルから取得したピクチャ

説明

READ PICTURE FILEコマンドを使用して、ディスクファイル＜ファイル名＞に保存されたピクチャを開き、これを引数＜ピクチャ＞に指定した4Dフィールドまたは変数へロードすることができます。

警告：このコマンドでは、Mac OSおよびWindows対応のQuickTime変換ルーチン（バージョン4以上推奨）を使用します。QuickTimeがインストールされていない場合、このコマンドはPICTファイル形式しか開くことができません。

引数＜ファイル名＞には、読み込むファイルのフルパス名、またはファイル名だけを渡します。ファイル名だけを渡した場合、そのファイルはデータベースのストラクチャファイルの隣に置かれていなくてはなりません。Windowsでは、ファイルの拡張子を必ず指定してください。

＜ファイル名＞に空の文字列（""）を渡すと、標準の「ファイルを開く」ダイアログボックスが表示され、読み込むファイルとともに（QuickTime 4で提供される）利用可能なフォーマットを選択できます。

PICTURE TYPE LISTコマンドを使用すると、利用できるフォーマットのリストを取得できます。QuickTime 4の標準フォーマットコードに関する説明は、**PICTURE TYPE LIST**コマンドを参照してください。

引数＜ピクチャ＞には、読み込まれた画像を取得するピクチャ変数、またはピクチャフィールドを渡します。

注：4Dの変数やフィールドに画像を保存する際の内部フォーマットは、QuickTimeより供給されます。したがって、QuickTimeのインストールされていない環境では、4Dの変数やフィールドに納められた画像を見ることはできません。

コマンドが正常に終了すると、システム変数Documentには開いたファイルのフルパス名が納められ、システム変数OKには1が代入されます。それ以外の場合、システム変数OKに0が代入されます。

参照

BLOB TO PICTURE、PICTURE TYPE LIST、ピクチャ、WRITE PICTURE FILE

PICTURE TYPE LIST

PICTURE TYPE LIST(フォーマット配列{; フォーマット名配列})

引数	タイプ		説明
フォーマット配列	文字列配列(4)	←	使用可能な読み込み／書き出しフォーマットのQuickTimeコード
フォーマット名配列	文字列配列	←	フォーマット名

説明

PICTURE TYPE LISTコマンドは、コマンドが実行されたマシン上で使用できる、ピクチャ読み込み／書き出し用のQuickTimeコードを、配列＜フォーマット配列＞に代入します。

オプションの引数＜フォーマット名配列＞には、それぞれのピクチャフォーマット名が代入されます。フォーマット名の方がコードよりもわかりやすいでしょう。

このコマンドを実行するマシンには、QuickTime（最低でもバージョン4）がインストールされている必要があります。インストールされていない場合、＜フォーマット配列＞にはPICT形式だけが納められます。

PICTURE TYPE LISTコマンドを使い、あるデータベースにいくつかのピクチャフォーマットを使用できるかどうかを調べることができます。デフォルトとしてインストールされない特定のフォーマットが必要になる場合（QuickTime 4の機能）、このコマンドが役立ちます。

＜フォーマット名配列＞に集められた情報を利用して、使用可能なピクチャ書き出しフォーマットを納めたポップアップメニューの作成や表示を行えます。

QuickTime 4の変換コード

QuickTime 4で提供される標準の変換コードの一覧を以下に示します。コードはそれぞれ4桁です。すべてのマシンで同じコードが提供されるわけではないという点に注意してください。また、QuickTime 4を使用することにより、カスタマイズされた変換ルーチンを追加できるため、すべてのマシンで同じコードが提供されるとは限らない点に注意してください。

Picture size

Picture size(ピクチャ) → 整数

引数	タイプ		説明
ピクチャ	ピクチャ	→	サイズを求めるピクチャ
戻り値	数値	←	ピクチャのサイズ（バイト）

説明

この関数は、バイト単位で指定したピクチャのサイズを返します。

参照

PICTURE PROPERTIES

PICTURE PROPERTIES

PICTURE PROPERTIES(ピクチャ; 幅 ; 高さ {; 水平オフセット {; 垂直オフセット {; モード}}})

引数	タイプ		説明
ピクチャ	ピクチャ	→	情報を取得するピクチャ
幅	数値	←	ピクセルで表されたピクチャの幅
高さ	数値	←	ピクセルで表されたピクチャの高さ
水平オフセット	数値	←	背景に表示された際の水平オフセット
垂直オフセット	数値	←	背景に表示された際の垂直オフセット
モード	数値	←	背景に表示された際の転送モード

説明

PICTURE PROPERTIESコマンドは、引数＜ピクチャ＞に渡すピクチャに関する情報を返します。

引数＜幅＞と＜高さ＞は、ピクチャの幅と高さを返します。

オプション引数＜水平オフセット＞、＜垂直オフセット＞、＜モード＞は、フォームのバックグラウンド（背景）に表示された際の、ピクチャの水平、垂直位置と転送モードを返します。

参照

Picture size

CREATE THUMBNAIL

CREATE THUMBNAIL(ピクチャ; サムネール{; 幅{; 高さ{; モード{; 深度}}}})

引数	タイプ		説明
ピクチャ	ピクチャ	→	サムネールに変換する4Dピクチャフィールドまたはピクチャ変数
サムネール	ピクチャ	←	結果のサムネール
幅	整数	→	サムネールの幅（ピクセル単位） デフォルト値 = 48
高さ	整数	→	サムネールの高さ（ピクセル単位） デフォルト値 = 48
モード	整数	→	サムネール作成モード デフォルト値 = scaled to fit prop centered(6)
深度	整数	→	サムネールのカラー（ビット/ピクセル） デフォルト値 = 現在の画面深度(0)

説明

CREATE THUMBNAILコマンドは、指定した元のピクチャのサムネールを返します。通常、サムネールはマルチメディアソフトウェアやWebサイトにおいてピクチャプレビューのために使用されます。

注：このコマンドには、QuickTimeがインストールされていなくても構いません。

引数＜ピクチャ＞には、サムネールに縮小するピクチャが入った4Dの変数またはフィールドを渡します。引数＜サムネール＞には4Dのピクチャフィールドまたはピクチャ変数を渡し、ここに作成されたサムネールが返されます。

オプションの引数＜幅＞および＜高さ＞を使用し、必要とするサムネールのサイズ（ピクセル単位）を定義します。この2つの引数を省略すると、サムネールのデフォルトサイズは 48×48ピクセルになります。

オプションの引数＜モード＞には、サムネールの作成モード、すなわち、縮小モードを指定します。3種類のモードが使用できます。以下の定義済み定数が、定数テーマ「ピクチャ表示フォーマット」として4th Dimensionより提供されています。

定数	タイプ	値
Scaled to fit	倍長整数	2
Scaled to fit proportional	倍長整数	5
Scaled to fit prop centered	倍長整数	6（デフォルト値）

注：CREATE THUMBNAILで使用できるのは、これらの定数だけです。「ピクチャ表示フォーマット」テーマ内のその他の定数をこのコマンドで使用することはできません。

いずれの値も指定しなければ、デフォルトとして“Scaled to fit prop centered”モード(6)が適用されます。

各種モードの画像を以下に示します。

元のピクチャ

作成されるサムネール(48 X 48)

・Scaled to fit = 2

・Scaled to fit proportional = 5

・Scaled to fit prop centered = 6（デフォルトモード）

注：“Scaled to fit proportional”および“Scaled to fit prop centered”を使用すると、空いたスペースが白く表示されます。しかし、これらのモードが4Dフォームのピクチャフィールドまたはピクチャ変数に適用されると、この空きスペースは透明になります。

オプションの引数<深度>を使用し、Mac OS上で作成されるサムネールに対して維持するカラー数（つまり、画面深度）を定義します。この引数は整数で、ピクセル毎のビット数（1、2、4、8、16、32のいずれか）になります。現在の画面深度（デフォルト値）を使用するには、0を指定してください。

注：Windows上では、引数<深度>は無視されます。コマンドは常に現在の画面深度を使用します。

PICTURE LIBRARY LIST

PICTURE LIBRARY LIST(ピクチャ参照番号 ; ピクチャ名)

引数	タイプ		説明
ピクチャ参照番号	数値配列	←	ピクチャライブラリのグラフィックのピクチャ参照番号
ピクチャ名	文字列配列	←	ピクチャライブラリ内のピクチャの名前

説明

PICTURE LIBRARY LISTコマンドは、データベースのピクチャライブラリの中に現在格納されているピクチャの参照番号と名前を返します。

このコマンドを呼び出すと、引数＜ピクチャ参照番号＞配列の中に参照番号、引数＜ピクチャ名＞配列の中にピクチャ名が返されます。この2つの配列は、同期します。つまり、＜ピクチャ参照番号＞配列のn番目の要素は、＜ピクチャ名＞配列のn番目の要素内に返されるピクチャライブラリ内にあるピクチャ名が持つ参照番号になります。

＜ピクチャ参照番号＞配列は、実数、倍長整数または整数の配列です。インタプリタモードでは、その配列が**PICTURE LIBRARY LIST**コマンドへの呼び出しの前に宣言されていない場合、デフォルトとして実数型の配列が作成されます。

＜ピクチャ名＞配列は、文字列またはテキストの配列です。インタプリタモードでは、その配列が**PICTURE LIBRARY LIST**コマンドへの呼び出しの前に宣言されていない場合、デフォルトとしてテキスト型の配列が作成されます。

ピクチャライブラリのピクチャの名前は、最大31バイトです。＜ピクチャ名＞配列に文字列配列を使用する場合、返されるピクチャ名が途中で切り捨てられないように、十分な長さの配列を宣言します。

ピクチャライブラリの中にピクチャがない場合、両方の配列は空で返されます。

ピクチャライブラリの中に現在格納されているピクチャの参照番号を取得するには、**Size of array**関数を使って、2つの配列の1つのサイズを取得します。

例題

1. 以下のコードは、配列“alPicRef”と“alPicName”の中にピクチャライブラリのカタログを返します。

```
PICTURE LIBRARY LIST(alPicRef ; asPicName)
```

2. 以下の例は、ピクチャライブラリが空であるかどうかを検査します。

```
PICTURE LIBRARY LIST(alPicRef ; asPicName)
If(Size of array(alPicRef)=0)
        ALERT("ピクチャライブラリは空です")
Else
        ALERT("ピクチャライブラリは、"+String(Size of array(alPicRef))+
                                                "ピクチャを含んでいます。")
End if
```

3. 以下の例は、ピクチャライブリをディスク上の任意のドキュメントに書き出します。

```
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($vlNbPictures)
                Error:=0
                For($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("ピクチャライブラリは書き出せませんでした。
                                                メモリを増やしてください。")
                        DELETE DOCUMENT(Document)
                End if
        End if
Else
```

```
        ALERT("ピクチャライブラリは空です。")
    End if
```

参照

GET PICTURE FROM LIBRARY、REMOVE PICTURE FROM LIBRARY、SET PICTURE TO LIBRARY

GET PICTURE FROM LIBRARY

GET PICTURE FROM LIBRARY(ピクチャ参照番号 | ピクチャ名;ピクチャ)

引数	タイプ		説明
ピクチャ参照番号またはピクチャ名	文字列	→	ピクチャライブラリのグラフィックのピクチャ参照番号またはピクチャ名
ピクチャ	ピクチャ変数	←	ピクチャライブラリ内のピクチャ

説明

GET PICTURE FROM LIBRARYコマンドは、引数<ピクチャ参照番号>で渡された参照番号を持つか、引数<ピクチャ名>の名前を持つピクチャライブラリ内のピクチャを引数<ピクチャ>に返します。

コンポーネント開発者のための注意：4Dのコンポーネントをピクチャライブラリに画像を保存する目的で使用する場合、ピクチャ名を1番目の引数として渡す必要があります。実際、コンポーネントが所持している必要なピクチャが4D Insiderでインストールされているとき、いくつかのデータベースピクチャが同じ参照番号をもっているとアプリケーションは自動的に新しいピクチャに参照番号を割り当てます。

参照番号または名前を持つピクチャがない場合は、**GET PICTURE FROM LIBRARY**コマンドは<ピクチャ>を変更しません。

例題

1. 以下の例は、参照番号がローカル変数“$vlPicRef”変数に格納されたピクチャを変数“vgMyPicture”変数に返します。

```
GET PICTURE FROM LIBRARY($vlPicRef ; vgMyPicture)
```

2 次の例では、$DDcom_Prot_MyPictureに、ピクチャライブラリ中に保存されているDDcom_Prot_Button1という名前の画像を返します。

```
GET PICTURE FROM LIBRARY("DDcom_Prot_Button"; $DDcom_Prot_MyPictue)
```

3. **PICTURE LIBRARY LIST**コマンドの例を参照してください

参照

PICTURE LIBRARY LIST、REMOVE PICTURE FROM LIBRARY、SET PICTURE TO LIBRARY

システム変数とシステムセット

ピクチャライブラリが存在する場合、OKシステム変数に1が設定されます。そうでない場合は、0が設定されます。

エラー処理

ピクチャに返すための十分なメモリがない場合、エラーコード-108が生成されます。エラー処理メソッドを使って、このエラーを受け取ることができます。

SET PICTURE TO LIBRARY

SET PICTURE TO LIBRARY(ピクチャ ; ピクチャ参照番号 ;ピクチャ名)

引数	タイプ		説明
ピクチャ	ピクチャ	→	新規ピクチャ
ピクチャ参照番号	数値	→	ピクチャライブラリのグラフィックのピクチャ参照番号
ピクチャ名	文字列	→	ピクチャの新しい名前

説明

SET PICTURE TO LIBRARYコマンドは、新規ピクチャを作成するか、またはピクチャライブラリにあるピクチャを置き換えます。

このコマンドを呼び出す前に、下記の引数を渡します。

■ 引数<ピクチャ参照番号>にピクチャ参照番号（1～32767の範囲）

■ 引数<ピクチャ>にピクチャ自身

■ 引数<ピクチャ名>ピクチャの名前（最大31バイト）

同じ参照番号を持つ既存のピクチャライブラリのピクチャがある場合、そのピクチャの内容は置き換えられ、引数<ピクチャ>と<ピクチャ名>に渡された値でピクチャ名が変更されます。

引数<ピクチャ参照番号>に渡された参照番号を持つピクチャライブラリのピクチャがない場合、新規ピクチャがピクチャライブラリに追加されます。

4D Server：SET PICTURE TO LIBRARYコマンドは（ストアードプロシージャまたはトリガを含む）サーバマシン上で実行されるメソッドの中から使用することはできません。SET PICTURE TO LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告：デザインオブジェクト（階層リスト項目、メニュー項目等）は、ピクチャライブラリのピクチャを参照することができます。プログラムによってピクチャライブラリのピクチャを修正する際は、注意して使用する必要があります。

注：引数＜ピクチャ＞に空のピクチャを渡すか、または引数＜ピクチャ参照番号＞に負数またはヌル値を渡すと、SET PICTURE TO LIBRARYコマンドは何も行いません。

例題

1. 以下の例は、ピクチャライブラリの現在の内容が何であろうと、最初にユニークなピクチャ参照番号を探すことによってピクチャライブラリに新規ピクチャを追加します。

```
PICTURE LIBRARY LIST($alPicRef ; $asPicNames)
Repeat
    $vlPicRef:=1+Abs(Random)
Until(Find in array($alPicRef ; $vlPicRef)<0)
SET PICTURE TO LIBRARY(vgPicture ; $vlPicRef ; "新規ピクチャ")
```

2. 以下の例は、**PICTUTE LIBRARY LIST**コマンドの3番目の例で作成した（ディスク上のドキュメントに格納された）ピクチャをピクチャライブラリの中に読み込みます。

```
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("このファイルは、ダメージを受けて
                                        いるようです。")
                End if
            End for
        Else
            ALERT("このファイルは、ダメージを受けているようです。")
        End if
    Else
```

```
            ALERT(Document+"ファイルは、ピクチャライブラリの書き出し
                                    ファイルではありません。")
        End if
        SET CHANNEL(11)
    End
```

参照

GET PICTURE FROM LIBRARY、PICTURE LIBRARY LIST、REMOVE PICTURE FROM LIBRARY

システム変数とシステムセット

何も影響を与えません。

エラー処理

ピクチャライブラリにピクチャを追加するための十分なメモリがない場合、エラーコード-108が生成されます。また、I／Oエラーが返される（例えば、ストラクチャファイルがロックされている等）点にも注意してください。エラー処理メソッドを使って、このエラーを受け取ることができます。

REMOVE PICTURE FROM LIBRARY

REMOVE PICTURE FROM LIBRARY(ピクチャ参照番号)

引数	タイプ		説明
ピクチャ参照番号またはピクチャ名	数値	→	ピクチャライブラリのピクチャの参照番号またはピクチャ名

説明

REMOVE PICTURE FROM LIBRARYコマンドは、引数<ピクチャ参照番号>に渡した参照番号を持つ、または引数<ピクチャ名>の名前を持つピクチャをピクチャライブラリから消去します。

参照番号または名前を持つピクチャがない場合は、このコマンドは何も行いません。

4D Server：REMOVE PICTURE FROM LIBRARYコマンドは（ストアードプロシージャまたはトリガを含む）サーバマシン上で実行されるメソッドの中から使用することはできません。REMOVE PICTURE FROM LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告：デザインオブジェクト（階層リスト項目、メニュー項目等）は、ピクチャライブラリのピクチャを参照することができます。プログラムによってピクチャライブラリのピクチャを修正する際は、注意して使用する必要があります。

例題

1. 以下の例は、ピクチャライブラリから参照番号4444のピクチャを削除します。

```
REMOVE PICTURE FROM LIBRARY(4444)
```

2. 以下の例は、ドル記号（$）で始まる名前を持つピクチャをピクチャライブラリから削除します。

```
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
```

参照

GET PICTURE FROM LIBRARY、PICTURE LIBRARY LIST、SET PICTURE TO LIBRARY

Semaphore

Semaphore(セマフォ{;待ち時間}) → ブール

引数	タイプ		説明
セマフォ	文字列	→	判定や設定を行うセマフォ
待ち時間	整数	→	最大待ち時間
戻り値	ブール	←	セマフォが正常に設定された（False） セマフォは既に設定済み（True）

説明

セマフォは、ワークステーション（各ユーザのコンピュータ）間、または同一ワークステーション上のプロセス間で共有するフラグです。セマフォは、単に存在したり存在しなかったりするだけです。各ユーザが実行しているメソッドでセマフォの存在を調べることができます。セマフォを作成する、またはその存在の有無を調べることにより、ワークステーション間でのメソッドの通信が可能になります。

Semaphore関数は、＜セマフォ＞が存在する場合に“True”を返します。**Semaphore**関数は＜セマフォ＞が存在しない場合にセマフォを作成し、“False”を返します。同時には、1人のユーザしかセマフォを作成することはできません。セマフォが“False”を返すということは、セマフォが存在しなかったことを意味すると同時に、呼び出したプロセスに対して新たにそれが設定されたことを意味します。

Semaphore関数は、セマフォが設定されていなければ“False”を返します。また、呼び出したプロセスが既にセマフォを設定している場合も、“False”を返します。セマフォは先頭に（<>, $）を含めて30文字以内に制限されています。これより長い文字列を指定すると、切り捨てられた文字列を使ってセマフォがテストされます。オプションパラメータ＜待ち時間＞はセマフォが既にセットされている時の待ち時間（tickで）を設定します。この時に、機能はセマフォを解放するか、またはTrueを返す前に待ち時間が終了するのを待ちます。

4th Dimensionバージョン6以降には、ローカルセマフォとグローバルセマフォの2種類があります。

ローカルセマフォは、同じワークステーション上のすべてのプロセスでアクセスすることができます（同一ワークステーション上に限られます）。ローカルセマフォは、セマフォ名の先頭にドル記号（$）を付けて作成します。また、ローカルセマフォは、同一ワークステーション上で実行しているプロセス間で処理を監視する際に使用します。例えば、ローカルセマフォを使用して、シングルユーザデータベース、またはワークステーション上のすべてのプロセスで共用するインタープロセス配列のアクセスを監視します。

グローバルセマフォは、すべてのユーザとそのプロセスからアクセスすることができます。グローバルセマフォはマルチユーザデータベースのユーザ間で処理を監視するために用います。

グローバルセマフォとローカルセマフォは理論的には同じものです。違いは、その有効範囲にあります。4D Serverでは、グローバルセマフォはすべてのクライアントで実行しているすべてのプロセス間で共用されます。ローカルセマフォは、それが作成されたクライアント上で実行しているプロセス間でのみ共用されます。4th Dimensionでは、ユーザは一人だけなため、グローバルセマフォもローカルセマフォもその有効範囲は同じです。ただし、シングルとマルチの両方の形でデータベースを使用する場合は、用途によってグローバルセマフォとローカルセマフォを使い分けてください。

セマフォはレコードのアクセスの保護目的には使用しません。これは4th Dimensionと4D Serverが自動的に行います。セマフォは、複数のユーザが同じ処理を同時に実行するのを防ぐために用います。

例題

1. 以下の例では、2人のユーザが“製品”テーブルの価格を一括更新するのを防ぎます。以下のメソッドではセマフォを用いて、これを実現しています。

```
If(Semaphore("価格更新"))  ` セマフォの作成を試みる
    ALERT("他のユーザが既に価格を更新しています。再度トライして
                                                ください。")
Else
    DoUpdatePrices      ` ここで価格を更新する
    CLEAR SEMAPHORE("価格更新") ` セマフォを消去する
End if
```

2. 以下の例は、ローカルセマフォの使い方です。複数のプロセスを持つデータベースでは、“To Do”リストを管理する必要があります。このリストはファイルではなく、インタープロセス配列で管理します。セマフォを使って同時にアクセスされるのを防ぎます。このような場合には、“To Do”リストは自分だけのものなため、ローカルセマフォで十分です。

インタープロセス配列は、「On Startup」データベースメソッドで初期化します。

```
ARRAY TEXT(<>ToDoリスト ; 0)          ` 最初To Doリストは空
```

“To Do”リストに項目を追加するメソッドを次に示します。

```
    `ADD TO DO LIST project method
    `ADD TO DO LIST(Text )
    `ADD TO DO LIST(To do list item )
C_TEXT($1) ` コマンドに渡されるパラメータ
    ` 既にセマフォが設定されていたら5秒待つ
If(Not(Semaphore("$AccessToDoList";300)))
    $vlElem:=Size of array(<>ToDoList)+1
    INSERT ELEMENT(<>ToDoList ; $vlElem)
    <>ToDoList{$vlElem}:=$1
    CLEAR SEMAPHORE("$AccessToDoList") ` セマフォをクリア
End if
```

このメソッドは、任意のプロセスから呼び出すことができます。

参照

CLEAR SEMAPHORE、Test semaphore

CLEAR SEMAPHORE

CLEAR SEMAPHORE(セマフォ)

引数	タイプ		説明
セマフォ	文字列	→	消去するセマフォ

説明

CLEAR SEMAPHOREコマンドは、**Semaphore**関数で設定された＜セマフォ＞を消去します。

ルールとしては、作成されたすべてのセマフォは消去するべきです。セマフォが消去されない場合、セマフォを作成したプロセスが終了するまで、作成されたセマフォはメモリ上に残ります。プロセスは作成したセマフォしか消去することはできません。セマフォを作成していないプロセス内からセマフォを消去しようとしても、何も行いません。

前出の**Semaphore**関数の例を参照してください。

参照

Semaphore

Test semaphore

Test semaphore(セマフォ) → ブール

引数	タイプ		説明
セマフォ	文字列	→	テストするセマフォの名前
戻り値	ブール	←	True=セマフォが存在する False=セマフォが存在しない

説明

この関数は、セマフォの存在をテストします。

Semaphore関数と**Test semaphore**関数の違いは、**Test semaphore**はセマフォが存在しない場合にはそのセマフォを作成しないということです。セマフォが存在している場合、関数はTrueを返します。そうでない場合はFalseを返します。

例題

▼ 下記の例は、セマフォを変更せずにプロセスの状態（この場合は、コードを変更している間）を知ることを可能にするものです。

```
Open window(x1;x2;y1;y2;-Palette window)
Repeat
    If(Test semaphore("Encrypting code"))
        GOTO XY($x3;$y3)
        MESSAGE("暗号コードが変更されている")
    Else
        GOTO XY($x3;$y3)
        MESSAGE("暗号コードの変更が認証された")
    End if
Until(StopInfo)
CLOSE WINDOW
```

参照

Semaphore

CALL PROCESS

CALL PROCESS(プロセス)

引数	タイプ		説明
プロセス	数値	→	プロセス番号

説明

CALL PROCESSコマンドは、＜プロセス＞の最前面に表示されたフォームを呼び出します。

重要：CALL PROCESSコマンドは、同一マシン上で実行されたプロセス間でのみ有効です。

存在しないプロセスを呼び出した場合には、何も行いません。

＜プロセス＞（目的のプロセス）で現在フォームが表示されていない場合には何も行いません。目的のプロセスで表示されたフォームが**On Outside call**イベントを受け取ります。「デザイン」モードの「フォームプロパティ」ウインドウにおいて、このフォームに対して**On Outside call**イベントを必ず有効にし、フォームメソッドでこのイベントを管理する必要があります。このイベントが無効であったり、またはフォームメソッドでイベントの管理を行わない場合、何も行われません。

注：On Outside callイベントは、受け取り側である入力フォームの入力状況を変更します。特に、フィールドが編集中である場合には、On Data changeイベントが生成されます。

呼び出し元プロセス（**CALL PROCESS**コマンドが実行されたプロセス）は“待機”しません。つまり、**CALL PROCESS**コマンドは即座に効力を持ちます。必要であれば、2つのプロセス間で読み書きを行えるように（**GET PROCESS VARIABLE**および**SET PROCESS VARIABLE**コマンドにより）インタープロセス変数やプロセス変数（この目的のために用意）を用いて、呼び出し元プロセスから応答のための待機用ループを作成する必要があります。

フォームを表示しないプロセスの間で通信を行うには、**GET PROCESS VARIABLE**および**SET PROCESS VARIABLE**コマンドを使用してください。

CALL PROCESSコマンドには**CALL PROCESS**(-1)というもう一つの構文があります。

メソッドの実行速度が遅くならないように、4th Dimensionはインタープロセス変数が変更されるたびに再描画することはしません。もし、プロセス参照番号の代わりに“-1”を**CALL PROCESS**コマンドの引数<プロセス>に渡すと、4th Dimensionはプロセスを1つも呼び出さず、その代わりに、同一マシン上で実行されているプロセス内のすべてのウインドウに表示されるインタプロセス変数をすべて更新します。4D Serverでは、クライアントから実行される**CALL PROCESS**(-1)は、そのクライアント上で実行されているプロセスのインタープロセス変数のみを更新します。

「On Exit」データベースメソッドの例題を参照してください。

参照

Form event、GET PROCESS VARIABLE、SET PROCESS VARIABLE

GET PROCESS VARIABLE

GET PROCESS VARIABLE(プロセス ; 送信元変数 ; 送信先変数 { ; 送信元変数2; 送信先変数2 ; ...; 送信元変数N ; 送信先変数N})

引数	タイプ		説明
プロセス	数値	→	送信元のプロセス番号
送信元変数	変数	→	送信元変数
送信先変数	変数	←	送信先変数

説明

GET PROCESS VARIABLEコマンドは、引数＜プロセス＞で渡す番号の送信元プロセスから＜送信元変数＞（送信元変数2等）プロセス変数を読み込み、その現在の値をカレントプロセスの＜送信先変数＞（送信先変数2等）変数に返します。

それぞれの送信元変数は、変数、配列、配列要素のいずれかを指定できます。ただし、この節で後述する制限事項を参照してください。

＜送信元変数＞と＜送信先変数＞変数の組み合わせにおいて、この2つの変数はみな互換性のあるタイプである必要があり、互換性がない場合には、値を取得しても意味がなくなります。

カレントプロセスは送信元プロセスの変数を「のぞき見」しています。送信元プロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

4D Server：4D Clientを使用し、サーバマシン上で実行される目的のプロセス（ストアドプロシージャ）の変数を読み込むことができます。このためには、引数＜プロセス＞に渡すプロセス番号の前にマイナス記号“-”を付けてください。

GET PROCESS VARIABLE、**SET PROCESS VARIABLE**、**VARIABLE TO VARIABLE**コマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きは常にクライアントプロセスです。

Tip：サーバのプロセス番号がわからない場合でも、サーバのインタープロセス変数を使用することができます。このためには、引数＜プロセス＞に任意の負の値を指定します。つまり、プロセス番号がわからなくてもサーバのインタープロセス変数を用いてGET PROCESS VARIABLEコマンドを使用することができるということです。このことは、「On server startup」データベースメソッドを使用して、ストアドプロシージャが起動されている場合には便利です。クライアントマシンではそのプロセスの番号が自動的にわからないため、引数＜プロセス＞に任意の負の値を渡すことができます。

制限事項

GET PROCESS VARIABLEコマンドは、送信元変数としてローカル変数を受け付けません。

一方で、送信先変数として、インタープロセス変数、プロセス変数、ローカル変数を使用できます。この値は変数にのみ「受け取る」ことができ、フィールドで受け取ることはできません。

GET PROCESS VARIABLEコマンドは、任意のタイプの送信元プロセス変数またはインタープロセス変数を受け付けますが、以下のタイプを除きます。

- ポインタ
- ポインタの配列
- 2次元配列

送信元プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送信元プロセスにはなれません。送信元プロセスが存在しない場合には、このコマンドは何も行いません。

注：インタープリタモードでは、送信元変数が存在しない場合には未定義値が返されます。これをType関数を使って検出し、対応する送信先変数をテストすることができます。

例題

1. 以下のコードは、プロセス番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」の値を読み込み、その値をカレントプロセスのプロセス変数「vtInfo」に返します。

```
GET PROCESS VARIABLE($vlProcess ; vtCurStatus ; vtInfo)
```

2. 以下のコードは上記の例と同じことをしますが、カレントプロセスで実行しているメソッドのローカル変数「$vtInfo」に値を返します。

```
GET PROCESS VARIABLE($vlProcess ; vtCurStatus ; $vtInfo)
```

3. 以下のコードは上記の例と同じことをしますが、カレントプロセスの「vtCurStatus」変数に値を返します。

```
GET PROCESS VARIABLE($vlProcess ; vtCurStatus ; vtCurStatus)
```

注：最初のvtCurStatusは、送信元プロセスにある変数のインスタンスを示しています。2番目のvtCurStatusはカレントプロセスにある変数のインスタンスを示しています。

4. 以下の例は、$vlProcessで示されるプロセスからプロセス配列の要素を順次読み込みます。

```
GET PROCESS VARIABLE($vlProcess ; vl_IPCom_Array ; $vlSize)
For($vlElem ; 1 ; $vlSize)
        GET PROCESS VARIABLE($vlProcess ; at_IPCom_Array{$vlElem} ;
                                                        $vtElem)
        `$vtElemを使った何らかの処理を行う
End for
```

注：この例では、プロセス変数「vl_IPCom_Array」には配列「at_IPCom_Array」のサイズが格納され、送信元プロセスによって管理されている必要があります。

5. 以下の例は上記の例と同じことをしますが、配列の要素を順番に読み込む代わりに配列を全体として読み込みます。

```
GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
For($vlElem ; 1 ; Size of array($anArray))
        `$anArray{$vlElem}を使った何らかの処理を行う
End for
```

6. 以下の例は、変数v1、v2、v3の送信元プロセスインスタンスを読み込み、それらの値をカレントプロセスの同じ変数のインスタンスに返します。

```
GET PROCESS VARIABLE($vlProcess ; v1 ; v1 ; v2 ; v2 ; v3 ; v3)
```

7. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照

CALL PROCESS、DRAG AND DROP PROPERTIES、SET PROCESS VARIABLE、VARIABLE TO BARIABLE

SET PROCESS VARIABLE

SET PROCESS VARIABLE(プロセス ; 送信先変数 ; ソース式 { ; 送信先変数2 ; ソース式2 ; ...; 送信先変数N ; ソース式N})

引数	タイプ		説明
プロセス	数値	→	送信先プロセス番号
送信先変数	変数	→	送信先変数
送信元変数	変数	→	送信元変数

説明

SET PROCESS VARIABLEコマンドは、引数＜送信元変数＞（＜送信元変数2＞等）で渡す値を使用して、＜プロセス＞に渡す番号を持つ送信先プロセスの＜送信先変数＞（＜送信先変数2＞等）プロセス変数を書き込みます。

それぞれのソース変数は、変数、または配列要素のいずれかを指定できます。ただし、この節で後述する制限事項を参照してください。

＜送信先変数＞と＜送信元変数＞変数の組み合わせにおいて、この式は送信先変数と互換性のあるタイプで構成されている必要があり、互換性がない場合には、変数に意味のない値が入ってしまうことになります。インタープリタモードでは、送信先変数が存在しない場合には、この式で送信先変数が作成され、割り当てられます。

カレントプロセスは送信先プロセスの変数を「のぞき見」しています。送信先プロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

4D Server：4D Clientを使用し、サーバマシン上で実行される送信先プロセス（ストアドプロシージャ）の変数を書き込むことができます。このためには、引数＜プロセス＞に渡すプロセス番号の前にマイナス記号“-”を付けてください。

GET PROCESS VARIABLE、**SET PROCESS VARIABLE**、**VARIABLE TO VARIABLE**コマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きは常にクライアントプロセスです。

Tip：サーバのプロセス番号がわからない場合でも、サーバのインタープロセス変数を使用することができます。このためには、引数＜プロセス＞に任意の負の値を指定します。つまり、プロセス番号がわからなくてもサーバのインタープロセス変数を用いてSet process variableコマンドを使用することができるということです。このことは、「On server startup」データベースメソッドを使用して、ストアドプロシージャが起動されている場合には便利です。クライアントマシンではそのプロセスの番号が自動的にわからないため、引数＜プロセス＞に任意の負の値を渡すことができます。

制限事項

SET PROCESS VARIABLEコマンドは、送信先変数としてローカル変数を受け付けません。

SET PROCESS VARIABLEコマンドは、任意のタイプの送信先プロセス変数を受け付けますが、以下のタイプは除きます。

■ ポインタ

■ 任意のタイプの配列：あるプロセスから別のプロセスに配列を全体として書き込むには、**VARIABLE TO VARIABLE**コマンドを使用します。ただし、**SET PROCESS VARIABLE**コマンドは配列の要素を書き込むことはできます。

■ ポインタ配列の要素または2次元配列の要素を書き込むことはできません。

送信先プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送信先プロセスにはなれません。送信先プロセスが存在しない場合には、エラーが生成されます。**ON ERR CALL**コマンドでインストールされたエラー処理メソッドを使用すると、このエラーを受け取ることができます。

例題

1. 以下のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」を(空の文字列に)設定します。

```
SET PROCESS VARIABLE($vlProcess ; vtCurStatus ; "")
```

2. 以下のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」をカレントプロセスで実行中のメソッドの変数「$vtInfo」の値に設定します。

```
SET PROCESS VARIABLE($vlProcess ; vtCurStatus ; $vtInfo)
```

3. 以下のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」をカレントプロセスの同じ変数の値に設定します。

```
SET PROCESS VARIABLE($vlProcess ; vtCurStatus ; vtCurStatus)
```

注：最初のvtCurStatusは、送信先プロセスにある変数のインスタンスを示しています。2番目のvtCurStatusはカレントプロセスにある変数のインスタンスを示しています。

4. 以下の例は、$vlProcessで示されるプロセスのプロセス配列の要素を順次大文字に設定します。

```
GET PROCESS VARIABLE($vlProcess ; vl_IPCom_Array ; $vlSize)
For($vlElem ; 1 ; $vlSize)
    GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};
                                                $vtElem)
```

```
            SET PROCESS VARIABLE($vlProcess ; at_IPCom_Array{$vlElem};
                                                Uppercase($vtElem))
        End for
```

注：この例では、プロセス変数「vl_IPCom_Array」には配列「at_IPCom_Array」のサイズが格納され、ソース／送信先プロセスによって管理されている必要があります。

5. 以下の例は、現在のプロセスからの同じ変数のインスタンスを使用して、変数v1、v2、v3の送信先プロセスインスタンスを書き込みます。

```
SET PROCESS VARIABLE($vlProcess ; v1 ; v1 ; v2 ; v2 ; v3 ; v3)
```

参照

CALL PROCESS、GET PROCESS VARIABLE、VARIABLE TO BARIABLE

VARIABLE TO VARIABLE

VARIABLE TO VARIABLE(プロセス ; 送信先変数 ; 送信元変数 {; 送信先変数2 ; 送信元変数2 ; ...; 送信先変数N ; 送信元変数N})

引数	タイプ		説明
プロセス	数値	→	送信先プロセス番号
送信先変数	変数	→	送信先変数
送信元変数	変数	→	送信元変数

説明

VARIABLE TO VARIABLEコマンドは、引数＜送信元変数＞（＜送信元変数2＞等）で渡す値を使用して、＜プロセス＞に渡す番号を持つ送信先プロセスの＜送信先変数＞（＜送信先変数2＞等）プロセス変数を書き込みます。

VARIABLE TO VARIABLEコマンドは、**SET PROCESS VARIABLE**コマンドと同じ動作をしますが、以下の点が異なります。

■ **SET PROCESS VARIABLE**コマンドは引数に＜送信元変数＞を渡すため、配列全体を渡すことができない。これに対して、**VARIABLE TO VARIABLE**コマンドは明示的に引数として＜送信元変数＞を渡すため、配列を全体として渡すことができる。

■ **SET PROCESS VARIABLE**コマンドの各送信先変数は、任意の変数または任意の配列要素を指定することができるが、配列全体は指定できない。これに対して、**VARIABLE TO VARIABLE**コマンドの各送信先変数は、任意の変数、任意の配列または任意の配列要素を指定することができる。

GET PROCESS VARIABLE、**SET PROCESS VARIABLE**、**VARIABLE TO VARIABLE**コマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きは常にクライアントプロセスです。

＜送信先変数＞と＜送信元変数＞変数の組み合わせにおいて、この送信元変数は送信先変数と互換性のあるタイプである必要があり、互換性がない場合には、変数に意味のない値が入ってしまうことになります。インタープリタモードでは、送信先変数が存在しない場合には、送信元変数のタイプと値を使って、送信先変数が作成され、割り当てられます。

カレントプロセスは、送信先プロセスの変数を「のぞき見」しています。送信先プロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

制限事項

VARIABLE TO VARIABLEコマンドは、送信先変数としてローカル変数を受け付けません。

VARIABLE TO VARIABLEコマンドは、任意の送信先プロセス変数またはインタフェースプロセス変数を受け付けますが、以下のタイプを除きます。

- ポインタ
- ポインタの配列
- 2次元配列

送信先プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送信先プロセスにはなれません。送信先プロセスが存在しない場合には、エラーが生成されます。**ON ERR CALL**コマンドでインストールされたエラー処理メソッドを使用することにより、このエラーを受け取ることができます。

例題

以下の例は、ローカル変数“$vlProcess”で示されたプロセスから任意のプロセスを読み込み、配列要素を順番に大文字に変換して、配列を全体として書き込みます。

```
GET PROCESS VARIABLE($vlProcess ; 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)
```

参照

GET PROCESS VARIABLE、SET PROCESS VARIABLE

HIDE PROCESS

HIDE PROCESS(プロセス)

引数	タイプ		説明
プロセス	数値	→	非表示にするプロセスのプロセス参照番号

説明

HIDE PROCESSコマンドは＜プロセス＞に属するすべてのウインドウを非表示にします。＜プロセス＞のすべてのインタフェース要素は、次に**SHOW PROCESS**コマンドを実行するまで非表示となります。そのプロセスのメニューバーも非表示になります。したがって、プロセスが非表示になっているときにウインドウを開いても画面が再描画されたり表示されません。プロセスが既に非表示になっている場合、このコマンドは何も実行しません。

ただし、デバッグウインドウは例外です。＜プロセス＞を非表示にしてもデバッグウインドウが表示されると、＜プロセス＞は表示され最前面のプロセスとなります。

＜プロセス＞を作成した時点でそれを全く表示したくなければ、**HIDE PROCESS**コマンドをプロセスメソッドの最初のコマンドにします。このコマンドは、「ユーザ／カスタムメニュー」プロセスおよび「キャッシュマネージャ」プロセスを非表示にすることはできません。

プロセスを非表示にした場合でも、そのプロセスは実行し続けます。

例題

▼次の例は、カレントプロセスのすべてのウインドウを非表示にします。

```
HIDE PROCESS(Current process)
```

参照

Process state、SHOW PROCESS

SHOW PROCESS

SHOW PROCESS(プロセス)

引数	タイプ		説明
プロセス	数値	→	表示するプロセスのプロセス参照番号

説明

SHOW PROCESSコマンドは＜プロセス＞に属する全ウインドウを表示します。このコマンドは、＜プロセス＞のウインドウを最前面ウインドウにするわけではありません。これには**BRING TO FRONT**コマンドを使用します。

プロセスが既に表示されている場合は、このコマンドは何も実行しません。

例題

▼次の例は、以前に非表示になっていれば「デザイン」プロセスを表示します。「デザイン」プロセスのプロセス参照はインタープロセス変数“<>DesignProc”に格納されます。

```
SHOW PROCESS(<>DesignProc)
```

参照

BRING TO FRONT、HIDE PROCESS、Process state

BRING TO FRONT

BRING TO FRONT(プロセス)

引数	タイプ		説明
プロセス	数値	→	最前レベルに持って行くプロセスのプロセス参照番号

説明

BRING TO FRONTコマンドは＜プロセス＞に属するウインドウをすべて最前に配置します。ウインドウの順序は変わりません。このプロセスが既に最前のプロセスの場合は、このコマンドは何も行いません。プロセスが非表示の場合に、**SHOW PROCESS**コマンドでプロセスを表示しないと**BRING TO FRONT**コマンドは効果がありません。

「ユーザ／カスタムメニュー」プロセスおよび「デザイン」プロセスは、このコマンドで前面に配置することができます。

例題

▼次の例は、メニューから実行できるプロジェクトメソッドです。これは、最前のプロセスが“<>Customers”プロセスかどうかを調べています。そうでなければ、それを前面に配置します。

```
If(Frontmost process # <>Customers)      ` 顧客リストが表示されていない場合
BRING TO FRONT(<>Customers)  ` リストを前面に持ってくる
End if
```

参照

HIDE PROCESS、Process state、SHOW PROCESS

Frontmost process

Frontmost process {(*)} → 整数

引数	タイプ		説明
*	*	→	最初の非フローティングウインドウのプロセス番号
戻り値	整数	←	前面に配置されたウインドウの属すプロセスの番号

説明

Frontmost process関数は、ウインドウが最前面にあるプロセスの番号を返します。

1つまたは複数のフローティングウインドウがある場合は、次の2種類のウインドウレイヤがあります。

■ 通常のウインドウ

■ フローティングウインドウ

フローティングウインドウのフォームメソッドやオブジェクトメソッドから**Frontmost process**関数を使用すると、この関数はフローティングウインドウレイヤ内の最前面のフローティングウインドウのプロセス参照番号を返します。引数<*>を指定すると、この関数は、通常ウインドウレイヤ内の最前面のアクティブウインドウのプロセス参照番号を返します。

前述の**BRING TO FRONT**コマンドの例を参照してください。

参照

BRING TO FRONT、WINDOW LIST

プロセス

4th Dimensionにおけるマルチタスクにより、個々のデータベース処理を同時に行います。これらの個々のデータベース処理を“プロセス”と呼びます。

マルチプロセスは、同じコンピュータに複数のユーザがいて独自の処理を行っているようなものです。これは、各メソッドが別個のデータベースタスクとして実行できることを意味します。

この章では、以下の項目について説明します。

■ プロセスの作成と消去

■ プロセスの要素

■ ユーザプロセス

■ 4th Dimensionが作成するプロセス

■ ローカルプロセスとグローバルプロセス

■ プロセス間のレコードロック

注：この節ではストアードプロシージャについては触れていません。詳細は『4D Server リファレンス』マニュアルのストアードプロシージャに関する説明を参照してください。

プロセスの作成と消去

新規プロセスを作成するには、以下の3つの方法があります。

■「ユーザ」モードにおいて、「メソッド実行」ダイアログボックスで「新規プロセス」チェックボックスをチェックした後、メソッドを実行する方法。「メソッド実行」ダイアログボックスで選択したメソッドが“プロセスメソッド”です。

■ メニューを選択してプロセスを開始する方法。「デザイン」モードの「メニューエディタ」において、任意のメニューを選択して「新規プロセス開始」チェックボックスをクリックする。メニューに対応するメソッドが“プロセスメソッド”です。

■ **New process**関数を使用する方法。**New process**関数の引数として渡されたメソッドが“プロセスメソッド”です。

プロセスは以下の方法で消去します。最初の2つは自動的に行われます。

■ プロセスメソッドの実行が完了した場合。

■ ユーザがデータベースを終了した場合。

■ メソッドからプロセスを中止するか、またはデバッガから**ABORT**コマンドを使用した場合。

■「デザイン」モードの「プロセス」メニューから「アボート」メニューを選択した場合。

プロセスは別のプロセスを作成することができます。プロセスは階層構造にはなっていません。どのプロセスから作成されようと、すべてのプロセスは同等です。いったん、“親”プロセスが“子”プロセスを作成すると、親プロセスの実行状況に関係なく、子プロセスは処理を続行します。

プロセスの要素

各プロセスには個々の要素があります。プロセスには以下の3種類の要素があります。

■**「インタフェース要素」：**この要素は、プロセスを表示するのに必要な要素で構成されます。

■**「データ要素」：**この要素は、データベース内のデータに関連する情報で構成されます。

■**「ランゲージ（言語）要素」：**この要素は、メソッドの中で使用する要素またはユーザがアプリケーションを開発する際に重要な要素で構成されます。

インタフェース要素

インタフェース要素には以下のものがあります。

■「メニューバー」：各プロセスは、独自のカレントメニューバーを持つことができます。最前面のプロセスのメニューバーがデータベースのカレントメニューバーになります。

■「1つまたは複数のウインドウ」：各プロセスは、1つまたは複数のウインドウを同時に開くことができます。ウインドウをまったく持たないプロセスもあります。

■「1つのアクティブ（最前面）ウインドウ」：プロセスは、複数のウインドウを同時に開くことができますが、アクティブウインドウは各プロセスに1つしかありません。複数のアクティブウインドウを持つには、アクティブウインドウの数だけのプロセスを起動しなければなりません。

データ要素

データ要素は、データベースが使用するデータです。以下のようなデータ要素があります。

■「テーブルごとのカレントセレクション」：各プロセスは、個々にカレントセレクションを持っています。同じテーブルに対して別々のプロセスにそれぞれのカレントセレクションを持つことができます。

■「テーブルごとのカレントレコード」：各テーブルは、プロセスごとに異なるカレントレコードを持つことができます。

注：データ要素に関するこの記述は、プロセスがグローバルの場合に有効です。デフォルトでは、すべてのプロセスはグローバルです。グローバルプロセスとローカルプロセスの違いに関する詳細は、後述の「グローバルプロセスとローカルプロセス」の節を参照してください。

ランゲージ要素

プロセスのランゲージ要素は、4th Dimensionのプログラミングに関連した要素で構成されます。

■「変数」：各変数は、独自のプロセス変数を持っています。詳細は、第2章の「プロセス変数」の節を参照してください。プロセス変数はその本来のプロセスの範囲内でのみ認識されます。

■「デフォルトテーブル」：プロセスごとに独自のデフォルトテーブルを持っています。ただし、**DEFAULT TABLE**コマンドは単なるプログラミング上の規則です。

■「入力フォームと出力フォーム」：各プロセスの各テーブルに対して、デフォルトの入力フォームと出力フォームをメソッドから設定することができます。

■「UserSet、LockedSetおよびプロセスセット」：各プロセスは、独自のプロセスセットを持っています。UserSetとLockedSetはプロセスセットです。プロセスセットはプロセスメソッドが終了すると直ちに消去されます。

■「プロセスごとのOn Error Call」：各プロセスは、独自のエラー処理メソッドを持っています。

■「デバッグウインドウ」：各プロセスは、独自のデバッグウインドウを持つことができます。

ユーザプロセス

ユーザプロセスは、あるタスクを実行するためにユーザによって作成されるプロセスです。ユーザプロセスはカーネルプロセスと処理時間を分け合います。例えば、Web接続プロセスはユーザプロセスです。

4th Dimensionによって作成されるプロセス

以下のプロセスは、4th Dimensionによって作成、管理されます。

■「ユーザ／カスタムメニュープロセス」：ユーザ／カスタムメニュープロセスは、「カスタム」モードと「ユーザ」モードで構成されます。「カスタム」モードのデフォルトの初期画面（スプラッシュ画面）はユーザ／カスタムメニュープロセスの一部でもあります。このプロセスは、4th Dimensionの起動直後に作成されます。

■「デザインプロセス」：デザインプロセスは、別プロセスとして実行される「デザイン」モードで構成されます。デザインプロセスは、「デザイン」モードで「ファイル」メニューの「デザインモード終了」メニューを使用して閉じることができます。コンパイルしたデータベースにはデザインプロセスはありません。デザインプロセスは、ユーザが最初に「デザイン」モードに入った場合にのみ作成されます。デフォルトでアプリケーションが「ユーザ」モードまたは「カスタム」モードで表示される場合には、このプロセスは作成されません。

■「Webサーバプロセス」：Webサーバプロセスは、Web上にデータベースが公開された時に起動します。これに関する詳細は、第61章「Webサーバコマンド」を参照してください。

■「キャッシュマネージャプロセス」：キャッシュマネージャプロセスは、データベースのディスク入出力を管理します。このプロセスは、4th Dimensionまたは4D Serverの起動直後に作成されます。

■「インデックス作成プロセス」：インデックス作成プロセスは、別プロセスとしてデータベースのフィールドのインデックス作成を管理します。フィールドのインデックスを作成、または削除する時にこのプロセスが作成されます。

■「On Event Managerプロセス」：このプロセスは、**ON EVENT CALL**コマンドでイベント処理メソッドがインストールされる際に作成されます。このプロセスは、イベントが発生すると**ON EVENT CALL**コマンドがインストールしたイベントメソッドを実行します。このイベントメソッドがこのプロセスの“プロセスメソッド”です。これはメソッドが実行されていなくても実行し続けます。イベント処理は、「デザイン」モードでも行われます。

グローバルプロセスとローカルプロセス

プロセスは、ローカルとグローバルの両方を作成することができます。デフォルトでは、すべてのプロセスはグローバルです。

グローバルプロセスはデータのアクセスや操作を含め、あらゆる処理を実行できます。ほとんどの場合、グローバルプロセスを使用します。

ローカルプロセスは、データアクセスを必要としない操作の場合にだけ使用することをお勧めします。例えば、イベント処理メソッドを実行、またはフローティングウインドウ等のインタフェース要素を制御するためにローカルプロセスを使用します。

その名前によってプロセスがローカルであることを指定します。ローカルプロセス名は、先頭にドル記号（$）を付ける必要があります。

警告：ローカルプロセスでデータアクセスを行った場合、「ユーザ／カスタム」プロセスでアクセスすることになり、そのプロセス内で実行される処理との間でコンフリクトが起きるおそれがあります。

4D Server：データアクセスを行わない処理に対し、クライアント側でローカルプロセスを使用すると、サーバ集中処理のために、より多くの処理時間を与えることができます。

プロセス間のレコードのロック

他のプロセスが修正のためにレコードを正常にロードすると、そのレコードはロックされます。ロックされているレコードは、別のプロセスでロードすることはできますが、修正することはできません。レコードは、それを修正しているプロセス内でのみロックが解除されます。レコードをロック解除の状態でロードするには、テーブルが“リードライト（書き込み可能）”モードになっていなければなりません。

参照

メソッド、プロジェクトメソッド、変数

New process

New process(メソッド ; スタック {; プロセス{; パラメータ{; パラメータ2; ...; パラメータN}{; *}}}) → 数値

引数	タイプ		説明
メソッド	文字列	→	プロセス内で実行するメソッド
スタック	数値	→	バイト単位のスタックサイズ
プロセス	文字列	→	作成したプロセスの名前
パラメータ	式	→	メソッドに渡すパラメータ
*	*	→	重複しないプロセス
戻り値	数値	←	新しく作成されたプロセス、または既存のプロセスのプロセス参照番号

説明

New process関数は、＜プロセス＞という名前で新しいプロセスを開始し（同じマシン上で)、そのプロセスの参照番号を返します。

プロセスが作成できない場合（例えば、メモリ不足)、**New process**関数は0を返し、エラーが発生します。このエラーは**ON ERR CALL**でインストールされたエラー処理メソッドを使用して検出することができます。

プロセスメソッド：＜メソッド＞には、新しいプロセスのプロセスメソッド名を指定します。4Dが新しいプロセスのコンテキストを設定した後、このメソッドの実行が開始されます。したがってこれはプロセスメソッドになります。

プロセススタック：＜スタック＞には、プロセスのスタックに割り当てらるメモリの量を指定します。このメモリ領域にメソッド呼び出し、ローカル変数、サブルーチンの引数、スタックレコード等のプロセスオブジェクトが“積み上げ”られていきます。この大きさはバイトで表され、通常は少なくとも32K（約32,000バイト）を渡します。しかし、プロセスでサイズの大きいコールを続けて実行する場合等は（サブルーチンからサブルーチンを呼び出す等)、これ以上の値を渡してもかまいません。例えば、必要なら200K（約200,000バイト）を渡すこともできます。

注：スタックはプロセスの合計メモリではありません。各プロセスはレコードやインタープロセス変数等のためにメモリを共有します。また、1つのプロセスはプロセス変数の保存にメモリを余計に使用します。スタックは、ローカル変数やメソッドのコール、サブルーチンの引数、スタックされたレコードを維持するだけです。

プロセス名：オプション引数＜プロセス＞には、新しいプロセスの名前を指定します。ここで指定した名前は「デザイン」モードの「プロセスリスト」ウインドウに表示されます。また、この新しいプロセスに対して**PROCESS PROPERTIES**コマンドを実行するとこの名前が返されます。プロセス名は最大31バイトまで指定することができます。この引数は省略することができます。省略した場合、プロセス名は空の文字列になります。プロセス名の先頭に“$”記号を付けることにより、プロセスをローカルにすることができます。

重要：このローカルプロセスは、クライアント／サーバのもとではデータにアクセスすることができないため注意が必要です。ローカルプロセスに関する詳細は、第10章の「グローバルプロセスとローカルプロセス」の節を参照してください。

プロセスメソッドのパラメータ：バージョン6より、プロセスメソッドにパラメータを渡せるようになりました。サブルーチンにパラメータを渡すのと同じ要領でプロセスメソッドにパラメータを渡します。しかし、制約があります。ポインタ表現は渡すことができません。また、メソッドに対して配列をパラメータとして受け渡すことができない点にも留意してください。プロセスメソッドは、新規プロセスのコンテキスト内で実行を開始する際に、$1や$2等に引数の値を受け取ります。

注：プロセスメソッドにパラメータを渡す場合、必ず引数＜プロセス＞を指定しなければなりません。この場合、この引数は省略できません。

任意の引数＜*＞：この最後の引数を指定した場合、4Dははじめに＜プロセス＞に指定した名前を持つプロセスが既に実行されているかどうかを調べます。同一名のプロセスが存在する場合、4Dは新規プロセスを開始せずにその名前を持つプロセスのプロセス参照番号を返します。

例題

以下のようなプロジェクトメソッドについて考えてみましょう。

```
` 顧客追加
MENU BAR(1)
Repeat
      ADD RECORD([顧客];*)
Until(OK=0)
```

「デザイン」モードの「メニューバー」エディタウインドウで、カスタムメニュー項目にこのプロジェクトメソッドを指定し、「新規プロセス開始」チェックボックスをチェックしている場合、4Dはそのメソッドを実行する新規プロセスを自動的に開始します。**MENU BAR**(1)コマンドを実行すると、この新規プロセスに対してメニューバーが追加されます。ウインドウ（**Open window**コマンドでオープンするウインドウ）が何も存在しない場合、**ADD RECORD**コマンドを実行することにより、自動的にウインドウがオープンされます。

独自のコントロールパネルのボタンをクリックすると、この“顧客追加”プロセスが開始されるようにするには、以下のようにします。

```
` “b顧客追加”ボタンのオブジェクトメソッド
    $vlProcessID:=New process("顧客追加";32*1024;"顧客の追加")
```

このボタンはカスタムメニュー項目と同じ作業を行います。

メニュー項目を選択したり、このボタンをクリックすると、プロセスを開始（そのプロセスが存在しない場合）またはプロセスを前面に配置（そのプロセスが既に実行中の場合）したい場合、以下のように“顧客追加開始”メソッドを作成することができます。

```
` 顧客追加開始
$vlProcessID:=New process("顧客追加";32*1024;"顧客の追加";*)
If($vlProcessID#0)
    BRING TO FRONT($vlProcessID)
End if
```

“b顧客追加”のオブジェクトメソッドは以下のようになります。

```
` “b顧客追加”ボタンのオブジェクトメソッド
顧客追加開始
```

「メニューバー」エディタで“顧客追加”メソッドを“顧客追加開始”メソッドと置き換え、メニュー項目の「新規プロセス開始」チェックボックスを選択解除します。

参照

Execute on server、メソッド、プロセス、プロジェクトメソッド、変数

Execute on server

Execute on server(メソッド ; スタック {; 名前 {; パラメータ {; パラメータ2 ;...; パラメータN}{; *}}}) → 数値

引数	タイプ		説明
メソッド	文字列	→	プロセスの中で実行されるメソッド
スタック	数値	→	スタックサイズ（バイト単位）
名前	文字列	→	作成されるプロセスの名前
パラメータ	式	→	作成されるプロセスの名前
*	*	→	メソッドのパラメータ（引数） 重複しないプロセス
戻り値 番号	数値	←	新しく作成されるプロセスのプロセス または既に実行しているプロセス

説明

Execute on serverコマンドはサーバマシン上（クライアント／サーバで実行された場合）、または同じマシン上（シングルユーザで実行された場合）で新しいプロセスを開始し、そのプロセスのプロセス参照番号を返します。

Execute on serverコマンドを使用してストアドプロシージャを開始します。ストアドプロシージャについての詳細は、『4D Serverリファレンス』マニュアルの「ストアードプロシージャ」の節を参照してください。

クライアントマシンで**Execute on server**コマンドを実行した場合、負のプロセス参照番号が返されます。サーバマシンで**Execute on server**コマンドを実行すると、正のプロセス参照番号が返されます。サーバマシン上で**New process**コマンドを実行すると、**Execute on server**コマンドを実行することと同じになる点に注意してください。

プロセスが作成できない場合（例えば、メモリ不足）、**Execute on server**コマンドは0を返し、エラーが発生します。このエラーは**ON ERR CALL**でインストールしたエラー処理メソッドを使用して検出することができます。

プロセスメソッド：＜メソッド＞には、新しいプロセスのプロセスメソッド名を指定します。4Dは新規プロセスのコンテキストを設定した後、このメソッドの実行を開始します。したがって、これがプロセスメソッドになります。

プロセススタック：＜スタック＞には、プロセスのスタックに割り当てらるメモリの量を指定します。このメモリ領域にメソッド呼び出し、ローカル変数、サブルーチンの引数、スタックレコード等のプロセスオブジェクトが“積み上げ”られてゆきます。この大きさはバイトで表され、通常は少なくとも32K（約32,000バイト）を渡します。しかし、プロセスでサイズの大きいコールを続けて実行する場合等は（サブルーチンからサブルーチンを呼び出す等）、これ以上の値を渡してもかまいません。例えば、必要なら200K（約200,000バイト）を渡すこともできます。

注：スタックはプロセスの合計メモリではありません。各プロセスはレコードやインタープロセス変数等のためにメモリを共有します。また、1つのプロセスはプロセス変数の保存にメモリを余計に使用します。スタックは、ローカル変数やメソッドのコール、サブルーチンの引数、スタックされたレコードを維持するだけです。

プロセス名：＜名前＞には、新しいプロセスの名前を指定します。シングルユーザモードでは、ここで指定した名前が「デザイン」モードの「プロセスリスト」ウインドウに表示されます。また、この新しいプロセスに対して**PROCESS PROPERTIES**コマンドを実行するとこの名前が返されます。クライアント／サーバモードでは、4D Serverのメインウインドウの「ストアドプロシージャリスト」に青字で表示されます。

プロセス名は最大31バイトまで指定することができます。この引数は省略することができます。省略した場合、プロセス名は空の文字列になります。

警告：New Processとは異なり、Execute on serverコマンドの実行時に、プロセス名の先頭に“$”記号を付けて、プロセスをローカルにしないでください。シングルユーザモードではExecute on serverコマンドはNew Processコマンドと同じ処理を実行するため、プロセスをローカルにしても正常に動作します。しかし、クライアント／サーバモードではエラーが発生します。

プロセスメソッドのパラメータ：バージョン6より、プロセスメソッドにパラメータを渡せるようになりました。サブルーチンにパラメータを渡すのと同じ要領でプロセスメソッドにパラメータを渡します。しかし、制約があります。ポインタ表現は渡すことができません。また、メソッドに対して配列をパラメータとして受け渡すことができない点にも留意してください。プロセスメソッドは、新規プロセスのコンテキスト内で実行を開始する際に、$1や$2等に引数の値を受け取ります。

注：プロセスメソッドにパラメータを渡す場合、必ず引数＜名前＞を指定しなければなりません。この場合、この引数は省略できません。

任意の引数＜*＞：この最後の引数を指定した場合、4Dははじめに＜名前＞に指定した名前を持つプロセスが既に実行されているかどうかを調べます。同一名のプロセスが存在する場合、4Dは新規プロセスを開始せずにその名前を持つプロセスのプロセス参照番号を返します。

例題

1. 以下の例は、クライアント／サーバにおいてデータ読み込みの処理速度を飛躍的に向上する方法を示しています。以下のRegular Importメソッドにより、クライアント側で**IMPORT TEXT**コマンドを使用したレコード読み込みに要する時間を調べることができます。

```
` Regular Import プロジェクトメソッド
$vhDocRef:=Open document("")
If(OK=1)
	CLOSE DOCUMENT($vhDocRef)
	INPUT FORM([テーブル1];"Import")
	$vhStartTime:=Current time
	IMPORT TEXT([テーブル1];Document)
	$vhEndTime:=Current time
	ALERT("所要時間："+String(0+($vhEndTime-$vhStartTime))+" 秒")
End if
```

通常のデータ読み込みでは、4D Clientはテキストファイルの解析を行った後、各レコードごとに新規レコードを作成し、読み込んだデータをフィールドに格納して、データベースに追加するためにレコードをサーバマシンに送信します。この結果、ネットワーク上には大量のリクエストが行きかいます。この処理を最適化するには、ストアドプロシージャを利用し、サーバマシン上でローカルにジョブを実行します。クライアントマシンではドキュメントファイルをBLOBにロードし、パラメータにこのBLOBを渡してストアドプロシージャを開始します。この後で、このドキュメントファイルをローカルに読み込みます。したがって、ネットワークリクエストの大部分が必要なくなるため、データ読み込みはシングルユーザ版並の速度でローカルに実行されます。次に"*CLIENT IMPORT*"プロジェクトメソッドを示します。このメソッドはクライアントマシン上で実行され、後の"*SERVER IMPORT*"ストアドプロシジャを開始します。

```
` CLIENT IMPORT プロジェクトメソッド
` CLIENT IMPORT( ポインタ; 文字列)
` CLIENT IMPORT( -> [テーブル] ; 入力フォーム )
C_POINTER($1)
C_STRING(31;$2)
C_TIME($vhDocRef)
C_BLOB($vxData)
C_LONGINT(spErrCode)
` 読み込むドキュメントファイルを選択
$vhDocRef:=Open document("")
If(OK=1)
	` ドキュメントファイルが選択された場合、開いたままにしておかない
	CLOSE DOCUMENT($vhDocRef)
```

```
$vhStartTime:=Current time
` メモリへのロード
DOCUMENT TO BLOB(Document;$vxData)
If(OK=1)
    ` ドキュメントファイルをBLOBにロードできた場合
    ` サーバマシン上でデータ読み込みを実行するストアドプロシージャを開始
    $spProcessID:=Execute on server("SERVER IMPORT";32*1024;
        "Server Import Services";Table($1);$2;$vxData)
    ` この時点で、このプロセスではBLOB は不要になる
    CLEAR VARIABLE($vxData)
    ` ストアドプロシージャにより実行される処理の終了を待機
    Repeat
        DELAY PROCESS(Current process;300)
        GET PROCESS VARIABLE
                ($spProcessID;spErrCode;spErrCode)
        If(Undefined(spErrCode))
            ` 注：ストアドプロシージャが変数spErrCodeの独自
                のインスタンスを初期化しなかった場合、
                未定義の変数が返される可能性がある
            spErrCode:=1
        End if
    Until(spErrCode<=0)
    ` 認識しているストアドプロシージャに通知する
    spErrCode:=1
    SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
    $vhEndTime:=Current time
    ALERT("所要時間："+String(0+($vhEndTime-$vhStartTime))+" 秒")
Else
    ALERT("ドキュメントファイルをロードするための十分なメモリがありません")
End if
End if
```

次に、ストアドプロシージャとして実行されるSERVER IMPORTプロジェクトメソッドを示します。

```
` SERVER IMPORT プロジェクトメソッド
` SERVER IMPORT(倍長整数; 文字列; BLOB)
` SERVER IMPORT(テーブル番号 ; 入力フォーム ; 読み込みデータ)
C_LONGINT($1)
C_STRING(31;$2)
```

```
C_BLOB($3)
C_LONGINT(spErrCode)
` 処理はまだ終了していない、spErrCodeに 1をセット
spErrCode:=1
$vpTable:=Table($1)
INPUT FORM($vpTable->;$2)
$vsDocName:="Import File "+String(1+Random)
DELETE DOCUMENT($vsDocName)
BLOB TO DOCUMENT($vsDocName;$3)
IMPORT TEXT($vpTable->;$vsDocName)
DELETE DOCUMENT($vsDocName)
` 処理は終了、spErrCodeに 0をセット
spErrCode:=0
` 要求元のクライアントが結果を取得するまで待機
Repeat
    DELAY PROCESS(Current process;1)
Until(spErrCode>0)
```

データベースにこの2つのプロジェクトメソッドを作成したら、例えば以下のように“ストアドプロシージャベース”のデータ読み込みを実行できます。

```
CLIENT IMPORT(->[テーブル1];"入力")
```

ベンチマークテストによると、このメソッドを使用してレコード読み込みを実行すると通常の読み込みよりも60倍高速に処理されることがわかっています。

参照

New process

DELAY PROCESS

DELAY PROCESS(プロセス ; 遅延時間)

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号
遅延時間	数値	→	遅延時間(tick数)

説明

DELAY PROCESSコマンドはプロセスの実行を指定したtick数（1tick＝1/60秒）だけ遅らせます。この間、そのプロセスは4th Dimensionおよびユーザのマシンの処理時間を使用しません。プロセスの実行を遅延しても、そのプロセスはメモリ内に残ります。

プロセスが既に遅延状態の場合、このコマンドはそれを再度遅延します。この場合、＜遅延時間＞は残時間に加算されるのではなく、それに置き換わります。したがって、これ以上遅延させたくなければゼロ（0）を渡します。

プロセスが存在しなければ、このコマンドは何も行いません。

警告：DELAY PROCESSコマンドは、「ユーザ／カスタムメニュー」プロセスを含め、カーネルプロセスには何も影響を与えません。

Tip：「ユーザ／カスタムメニュー」プロセスを“遅延”するには、短い“待機”サブルーチンを作成します。このサブルーチンはCurrent timeやTickcount、Millisecondsを使用して経過時間を計測するループを行います。例えば、与えられた時間中にこの目的のためにオープン／クローズしたウインドウにメッセージを表示したい場合等です。

Tip：あるマシン上で、あるプロセスが別のプロセスを待機しており、この別のプロセスがユーザ動作を待つのではなくコードを実行している場合、“待機中”プロセスから繰り返しDELAY PROCESSを呼び出すと（遅延時間に1を指定して）、実行スピードの観点からは最良の結果が得られます。この理由は、スケジューラが大部分の時間を“待機中”プロセスではなく実行中のプロセスに“与える”ためです。

参照

HIDE PROCESS、PAUSE PROCESS

PAUSE PROCESS

PAUSE PROCESS(プロセス)

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号

説明

PAUSE PROCESSコマンドは、**RESUME PROCESS**コマンドで再開されるまで<プロセス>の実行を停止します。この間<プロセス>は4D Serverあるいはユーザマシンの処理時間を使用しません。プロセスは停止されてもメモリ内に残ります。

<プロセス>が既に停止していた場合は、**PAUSE PROCESS**コマンドは何も行いません。プロセスが**DELAY PROCESS**コマンドで遅延されている場合もそのプロセスは停止されます。**RESUME PROCESS**コマンドは、即座にプロセスを再開します。

プロセスの実行を停止している間はそのプロセスのウインドウに入力することはできません。この場合には、ユーザの混乱を避けるためにそのプロセスを非表示にするとよいでしょう。<プロセス>が存在しなければ、このコマンドは何も行いません。

警告：PAUSE PROCESSコマンドは、ユーザが開始したプロセス内だけで使用してください。PAUSE PROCESSコマンドは「ユーザ／カスタムメニュー」プロセスに何も影響を与えません。

参照

DELAY PROCESS、HIDE PROCESS、RESUME PROCESS

RESUME PROCESS

RESUME PROCESS(プロセス)

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号

説明

RESUME PROCESSコマンドは、実行が停止しているプロセスを再開します。プロセスが停止していない場合、このコマンドは何も行いません。

プロセスが事前に遅延されてる場合については、**PAUSE PROCESS**コマンドまたは**DELAY PROCESS**コマンドを参照してください。プロセスが存在しない場合、このコマンドは何も行いません。

参照

DELAY PROCESS、PAUSE PROCESS

Process aborted

Process aborted → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	True=プロセスがアボートされようとしている False=プロセスはアボートされようとしていない

説明

このコマンドは、このコマンドを呼び出したプロセスが不意に中断されようとしている場合、Trueを返します。これは、コマンドの実行が正常に完了しなかったことを意味します。例えば、これは**QUIT 4D**を呼び出した後に発生します。

例題

▼ このコマンドは、コンパイルモードでのみ、Webサーバ上のプログラミングの特別なケースとして使用できます。**While**...**End while**（例を参照）のようなループを使ってWebページを送るメソッドを使用する場合、Webサーバの構造として、Webブラウザのタイムアウト（指定された時間の終了）が発生しても、ループを停止できません。Webプロセスが閉じられていないため、Webライセンスは使用されたままとなります。

ループの終了判断として使用している**Process aborted**関数は、タイムアウトが発生するとTrueを返すため、ループは中断されてWebライセンスが開放されます。

HTMLページを送るために使用できるメソッドを紹介します。コンパイルドモードでは、このループはタイムアウトが発生しても中断することができません。

```
While(True)
    SEND HTML FILE(HTMLFile)
End while
```

Process abortedコマンドを使用すると、同じ動作を可能にすると共に、タイムアウトが発生するとループを抜けWebライセンスを開放することができます。

```
While(Not(Process aborted))
    SEND HTML FILE(HTMLFile)
End while
```

参照

DELAY PROCESS、PAUSE PROCESS

Current process

Current process → プロセス参照番号

説明

Current process関数は、このコマンドを呼び出したプロセスのプロセス参照番号を返します。

DELAY PROCESSコマンドと**PROCESS PROPERTIES**コマンドの例を参照してください。

参照

Process number、PROCESS PROPERTIES、Process state

Process state

Process state(プロセス) → 数値

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号
戻り値	数値	←	プロセスのステータス

説明

Process state関数は、＜プロセス＞に指定した参照番号を持つプロセスのステータスコードを返します。

プロセスのステータスとしては以下のような定数があらかじめ定義されています。

定数	タイプ	値	説明
Aborted	倍長整数	-1	プロセスが中止された
Delayed	倍長整数	1	プロセスが遅延されている
Does not exist	倍長整数	-100	プロセスが存在しない
Executing	倍長整数	0	プロセスメソッドが実行中
Hidden modal dialog	倍長整数	6	モーダルダイアログを持つプロセスが隠れている
Paused	倍長整数	5	プロセスが停止されている
Waiting for input output	倍長整数	3	レコードのキャッシュ中
Waiting for internal flag	倍長整数	4	データベースアクセス待ち
Waiting for user event	倍長整数	2	一般的にデータ入力待ち

プロセスが存在しない場合（つまり、1から**Count tasks**までの番号を渡さなかった)、この関数は**Does not exist**(-100)を返します。

例題

▼以下の例は、各プロセスの名前とプロセス参照番号を配列“arProcName”と“arProcNum”に入れます。このメソッドは、プロセスが中止されたかを調べます。この場合、プロセス名と参照番号は配列に追加されません。

```
$タスク:= Count tasks
ARRAY STRING(15 ; arProcName ; $タスク)
ARRAY INTEGER(arProcNum ; $タスク)
C_INTEGER($i ; vCount ; vState ; vTime)
For($i ; 1 ; $タスク)
        If(Process state >= Executing)      ` プロセスが中止されていなければ
                vCount:=vCount + 1   ` 要素数をカウントする
                PROCESS PROPERTIES(arProcNum{$i} ; arProcName{$i} ;
                                                        vState ; vTime)
        End if
End for
        ` プロセスの数に応じて配列の大きさを変える
ARRAY STRING(15 ; arProcName ; vCount)
ARRAY INTEGER(arProcNum ; vCount)
```

参照

Count tasks、PROCESS PROPERTIES

PROCESS PROPERTIES

PROCESS PROPERTIES(プロセス ; 名前 ; ステータス ; 時間 {; 表示 {;ユニーク ID{; 派生元 }}})

引数	タイプ		説明
プロセス	倍長整数	→	プロセス番号
名前	文字列	←	プロセス名
ステータス	整数	←	プロセス状態
時間	整数	←	現在のプロセスの実行時間
表示	ブール	←	可視（True）または不可視（False）
ユニーク ID	整数	←	独自のプロセス番号
派生元	倍長整数	←	プロセスの派生原因

説明

このコマンドを呼び出すと、次のような情報が返されます。

■ 引数<名前>はプロセスの名前を返します。プロセス名で注意すべき点は、次のとおりです。

- プロセスが「ユーザ」モードの「メソッド実行」ダイアログボックスから(「新規プロセス」オプションを選択した状態で)起動された場合に、プロセスの名前は「P_」で始まり、その後に数字が続く形になります。
- 「プロセス開始」プロパティが選択された任意のカスタムメニューから起動されたプロセスの場合には、プロセスの名前は「M_」または「ML_」で始まり、その後に数字が続く形になります。
- プロセスがアボートしていた場合（そして、その「スロット」がまだ再使用されていない場合)、プロセスの名前はそのまま返されます。プロセスがアボートしたかどうかを検出するには、<ステータス>=-1を判定します（下記の表を参照)。

■ <ステータス>は、呼び出しを行ったときのプロセスの状態を返します。この引数は、次の定義済定数で提供される値のいずれか1つを返すことができます。

定数	タイプ	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

■ 引数<時間>は、プロセスが起動されたときから使用している累積時間をtick数（1／60秒）で返します。

■ オプション引数<表示>が指定された場合には、プロセスが表示されている場合はTRUE、隠されている場合はFALSEを返します。

■ ユニークIDは、独自のプロセス番号を返します。
バージョン6.5からは、各プロセスはプロセス番号とセッションごとの独自のプロセスを持っています。独自のプロセス番号は、2つのプロセス間または2つのプロセスセッションの間で区別することができるようにするものです。これは、4th Dimensionセッションの間に開始されたプロセス番号に対応するものです。

■ 派生元は、プロセスの発生原因を表わす値を返します。4th Dimensionは、下記の定数を定義しています（Process Typeテーマ内で）。

定数	タイプ	値
Web Process with Context	倍長整数	-11
Other 4D Process	倍長整数	-10
External Task	倍長整数	-9
Event Manager	倍長整数	-8
Apple Event Manager	倍長整数	-7
Serial Port Manager	倍長整数	-6
Indexing Process	倍長整数	-5
Cache Manager	倍長整数	-4
Web Process with no Context	倍長整数	-3
Design Process	倍長整数	-2
User or Custom Menus Process	倍長整数	-1
None	倍長整数	0
Created from Programming	倍長整数	1
Created from Menu Command	倍長整数	2
Created from User Mode	倍長整数	3
Other User Process	倍長整数	4

注：4Dの内部プロセスは負の値を返し、ユーザが作成したプロセスは正の値を返します。

プロセスが存在しない場合、つまり、1から**Count tasks**までの番号を渡さなかった場合には、**PROCESS PROPERTIES**コマンドは変数パラメータの変更は行いません。

例題

1. 次の例は、変数vName、vState、VtimeSpentに現在のプロセスのプロセス名、プロセスステータス、プロセス時間を返します。

```
C_STRING(80;vName)`Initialize the variables
C_INTEGER(vState)
C_INTEGER(vTime)
PROCESS PROPERTIES(Current process ;vName ;vState ;vTime)
```

2. 「On Exit」データベースメソッドの例を参照してください。

参照

Count tasks、Process state

Process number

Process number(プロセス名{; *})→ 数値

引数	タイプ		説明
プロセス名	文字	→	プロセス参照番号を取得するプロセス名
*	*	→	プロセス参照番号を返す
戻り値	数値	←	プロセス参照番号

説明

Process number関数は、引数<プロセス名>に指定した名前を持つプロセスの番号を返します。プロセスが見つからない場合には、**Process number**関数は0を返します。

オプションの引数<*>を指定すると、サーバ上で実行されたプロセス（ストアドプロシージャ）のプロセスIDを4D Clientから取得することができます。この場合、負の値が返されます。**PROCESS VARIABLE**や**SET PROCESS VARIABLE**コマンドを使用する際には、このオプションが特に役立ちます。詳細はそれぞれのコマンドの説明を参照してください。

サーバマシン上のプロセスから引数<*>を指定してこのコマンドを実行すると、正の値が返されます。

例題

独立したプロセスで実行するカスタムフローティングウインドウを作成します。このプロセスでは、「デザイン」モードでやり取りができる独自のツールを実装します。例えば、キーワードの階層リストで項目を選択するときに、「デザイン」モードの最前面ウインドウにテキストを貼り付けたいとします。これを実行するには、クリップボードを使用できますが、イベントの貼り付けは「デザインプロセス」の内部で発生する必要があります。以下の関数は、「デザインプロセス」（が実行している場合）のプロセス番号を返します。

```
` 「Design process number」プロジェクトメソッド
` Design process number → 倍長整数
` Design process number → 「デザインプロセス」のプロセス番号
$0:=Process number(Get indexed string(170 ; 3))
` 「デザインプロセス」の名前は、4D内部のSTR#リソースIDが170で、ストリング番号
`  が3の中に格納される
`  注：これは、将来リソースが変わると変更される可能性がある
```

この関数を使用して、下記のプロジェクトメソッドは、引数として受け取ったテキストを「デザイン」モードの最前面のウインドウに貼り付けます（適用可能な場合）。

```
`「PASTE TEXT TO DESIGN」プロジェクトメソッド
` PASTE TEXT TO DESIGN(テキスト)
` PASTE TEXT TO DESIGN(最前面ウインドウに貼り付けられるテキスト)
C_TEXT($1)
C_LONGINT($vlDesignPID ; $vlCount)
$vlDesignPID:= Design process number
If($vlDesignPID  #  0)
        `クリップボードにテキストを配置する
        SET TEXT TO CLIPBOARD($1)
        `「Ctrl-V」または「Command-V」イベントをポストする
        POST KEY(Ascii("v") ; Command key mask ; $vlDesignPID)
        ` DELAY PROCESSコマンドを繰り返し呼び出すことでスケジューラは
        `「デザインプロセス」にイベントを渡す機会を取得する
        For($vlCount ; 1 ; 5)
                DELAY PROCESS(Current process ; 1)
        End for
End if
```

参照

GET PROCESS VARIABLE、PROCESS PROPERTIES、Process state、SET PROCESS VARIABLE

Count users

Count users → 整数

説明

Count users関数は、サーバに接続されているユーザの数を返します。もし、サーバがいくつかのプロセスが蓄積されて実行されていたなら、**Count users**関数はユーザの数に+1して返します。

シングルユーザの4th Dimensionの場合は、**Count users**関数は1を返します。

参照

Count tasks、Count user processes

Count tasks

Count tasks → 整数

説明

Count tasks関数は、ワークステーション上またはシングルユーザの4th Dimension上で開いているプロセスの数を返します。

この数には、4th Dimensionが自動的に管理するものも含めてすべてのプロセスが含まれます。この中には、「ユーザ／カスタムメニュー」プロセス、「デザイン」プロセス、「キャッシュマネージャ」プロセス、「インデックス」プロセス、および「Webサーバ」プロセスが含まれます。

Count tasks関数により返される数にはアボートされたプロセスも含まれます。

Process stateおよび「On Exit」データベースメソッドの例題を参照してください。

参照

Count user processes、Count users、PROCESS PROPERTIES、Process state

Count user processes

Count user processes → 整数

説明

Count user processes関数は、4th Dimensionが自動的に管理するプロセス以外の開いているプロセスの数を返します。

「ユーザ／カスタムメニュー」プロセス、デザインプロセス、Webサーバプロセスはユーザがクローズするまで「ユーザ」プロセスとみなします。したがって、このプロセスも「ユーザ」プロセスの数に入ります。

参照

Count tasks、Count users

EXECUTE ON CLIENT

EXECUTE ON CLIENT(クライアント名 ; メソッド名 {; 引数 1; ...; 引数 N })

引数	タイプ		説明
クライアント名	文字列	→	4D Clientの登録名
メソッド名	文字列	→	実行するメソッドの名前
引数 1...N		→	メソッドの引数

説明

このコマンドは、名前がクライアント名である登録されている4D Clientで、引数1...引数Nを引数としたメソッド名メソッドを実行します。4D Clientの登録名は**REGISTER CLIENT**コマンドで定義されます。

このコマンドは、4D Clientまたは4D Serverのストアドプロシージャによって呼び出すことができます。

メソッドが1つまたはそれ以上の引数が必要な場合、メソッドの名前の後に渡してください。

指定した4D Client上でのメソッドの実行は、指定した4D Client上で自動的に作成されたグローバルプロセス内で行われます。そのプロセス名は4D Clientの登録名にもなっています。

このコマンドが、同じ4D Clientに対し複数の4D Clientから連続的に呼び出されると、実行の順序がスタックされます。したがって、メソッドは非同期モードで、次から次へと処理されます。スタックされたメソッドが増えると、4D Clientのワークロードも大きくなってしまいます。**GET REGISTERED CLIENTS**コマンドを使用して、各クライアントのワークロードの状態を知ることができます。

注：実行順序のスタックは、UNREGISTER CLIENTコマンドを使って4D Clientが登録解除されない限りは、変更することも止めることもできません。

登録された複数の4D Clientに対し、同じメソッドを同時に実行させる事ができます。これを実行するには、クライアント名引数にワイルドカード（@）を使用します。

システム変数は、4D Serverがメソッドの実行要求を正しく受け取った場合、OK1になりますが、これはメソッドが4D Clientによって正しく実行されたということを保証するものではありません。

例題

▼ "GenerateNums"メソッドを、"Client1"クライアントステーション上で実行したいと仮定してみましょう。

```
EXECUTE ON CLIENT("Client1";"GenerateNums";12;$a;"Text")
```

▼ すべてのクライアントに "EmptyTemp"メソッドを実行させたい場合には、

```
EXECUTE ON CLIENT("@";"EmptyTemp")
```

▼ **REGISTER CLIENT**コマンドの例を参照してください。

参照

REGISTER CLIENT、UNREGISTER CLIENT、GET REGISTERED CLIENTS.

REGISTER CLIENT

REGISTER CLIENT(クライアント名 {; ピリオド } {; *})

引数	タイプ		説明
クライアント名	文字列	→	4D クライアントセッションの名前
ピリオド	倍長整数	→	4D Serverへの問い合わせ間隔（秒単位）
*	*	→	ローカルプロセス

説明

このコマンドは、4Dクライアントステーションを、クライアント名内で指定した名前で4D Serverに「登録」し、他のクライアントもしくは4D Server（保存されたメソッドを使って）が、**EXECUTE ON CLIENT**コマンドを使ってメソッドを実行できるようにします。一旦登録されると、4D Serverは他のクライアント用に1つまたはそれ以上のメソッドを実行することができます。

注：データベースプロパティダイアログの "開始時にクライアント登録"オプションを使って、4D Serverに接続するクライアントステーションを自動的に登録することができます（詳細は、「スタートアップ時にクライアントを登録する（4DServerのみ）を参照してください)。

このコマンドが実行されると、クライアントステーション上にクライアント名という名のプロセスが作成されます。このプロセスは、**UNREGISTER CLIENT**コマンドによってのみアボートが可能です。

オプションの*引数を渡すと、作成されるプロセスはローカルプロセスになり、4Dは自動的にプロセス名の始めにドルマーク（$）を付け加えます。そうでない場合は、グローバルプロセスです。

コマンドの実行後、クライアントステーションは、他の4D Clientまたは4D Server自身が呼び出しをしていないかどうかを見るために、定期的に4D Serverに問い合わせます。デフォルトではこの問い合わせは2秒ごとに行われ、この間隔はピリオドを変更することによって変えることができます。最小値は1秒です。

注：4th Dimensionのシングルユーザバージョンでこのコマンドは何も行ないません。

一度コマンドが実行されると、実行中の4D Clientの名前または4D Serverへ問い合わせ間隔を変更することはできません。これを実行するには、**UNREGISTERCLIENT**コマンドを呼び出し、再度**REGISTER CLIENT**コマンドを呼び出します。

注：複数の4Dクライアントが同じ登録名を持つことができます。

4D Clientが正しく登録されると、システム変数OKは1になります。4D Clientが既に登録されていると、コマンドは何も行なわずOKは0になります。

例題

▼下記の例では、クライアントワークステーション同士の間で通信を行なうことができる、小さなメッセージシステムを作成してみます。

1. このメソッドRegistrationは4D Clientを登録して、他の4D Clientからのメッセージを受け取ることができるようにします。

```
`他の名前で登録する前に、登録の解除が必要です
UNREGISTER CLIENT
Repeat
    vPseudoName:=Request("登録名： ";"User";"OK";"Cancel")
Until((OK=0) |(vPseudoName # ""))
If(OK=0)
    ...`登録しない場合の処理
Else
    REGISTER CLIENT(vPseudoName)
End if
```

2. 以下の指示は、登録された4D Clientのリストを得ることができるようにするものです。これは、On Startupデータベースメソッド内に置くことができます。

```
PrClientList:=New process("4D Client List";32000;"List of registered clients")
```

3. 4D Client Listメソッドは、登録している全4D Clientの登録名リストを入手し表示リストを更新します。

```
If(Application type=4D Client)
        `下記のコードはClient & Serverモードでのみ有効です
    $Ref:=Open window(100;100;300;400;-(Palette window+
            Has window title);"List of registered clients")
    Repeat
        GET REGISTERED CLIENTS($ClientList;$ListeCharge)
        `$ClientList内の登録されたクライアントで更新する
        ERASE WINDOW($Ref)
        GOTO XY(0;0)
        For($p;1;Size of array($ClientList))
            MESSAGE($ClientList{$p}+Char(Carriage return))
        End for
            `1 秒延滞
        DELAY PROCESS(Current process;60)
```

```
        Until(False) ` 無限ループ
    End if
```

4. 下記のメソッドは、登録している他の4D Clientにメッセージを送ります。これは、送られた4D ClientでDisplay_Messageメソッドを呼び出します（下記参照）。

```
$Addressee :=Request("メッセージ送信先： ";"")
        ` On startup データベースメソッドで作成された表示リストに
        `表示されている名前を入力します
If(OK # 0)
        $Message:=Request("送信メッセージ： ") ` メッセージ入力
        If(OK # 0)
        EXECUTE ON CLIENT($Addressee;"Display_Message";
                                                $Message) ` メッセージを送る
        End if
End if
```

5. これは、Display_Message メソッドです。

```
C_TEXT($1)
ALERT($1)
```

6. 最後に、クライアントステーションが他の4D Clientから見えず、メッセージも受け取れなくなるようにします。

```
UNREGISTER CLIENT
```

システム変数とセット4D Clientが正常に処理されるとシステム変数OKには1になります。既に4D Clientが処理されている時は、コマンドは実行されず、システム変数OKは0になります。

参照

EXECUTE ON CLIENT、UNREGISTER CLIENT、GET REGISTERED CLIENT

UNREGISTER CLIENT

UNREGISTER CLIENT

このコマンドには、引数はありません。

説明

このコマンドは、クライアントステーションの登録を解除します。クライアントは**REGISTER CLIENT**コマンドによって既に登録されているものでなければなりません。

注：4D Clientは、ユーザがアプリケーションを終了すると自動的に登録を解除します。

クライアントステーションが前もって登録されていなかったり、コマンドをシングルユーザ4th Dimensionで実行しても、コマンドは何も行ないません。クライアントの登録が正しく解除されると、システム変数OKは1になり、クライアントが登録されていないと、OKは0になります。

例題

▼ **REGISTER CLIENT**コマンドの例を参照してください。

参照

REGISTER CLIENT、GET REGISTERED CLIENTS.

GET REGISTERED CLIENTS

GET REGISTERED CLIENTS(クライアントリスト ; メソッド)

引数	タイプ		説明
クライアントリスト	テキスト列	←	セーブされた4D Clientの配列
メソッド	倍長整数配列	←	実行待ちメソッド数の配列

説明

このコマンドは2つの配列を作成します。

■ クライアントリストは、**REGISTER CLIENT**コマンドを使って「登録された」クライアントの配列となります。

■ メソッドは、各クライアントの「ワークロード」の配列となります。ワークロードは、**EXECUTE ON CLIENT**コマンドを呼び出すことによって発生した、4D Clientがこれから実行しなければならないメソッドの数です（より詳しい情報は、**EXECUTE ON CLIENT**コマンドを参照してください）。

正常に処理された場合、システム変数OKは1になります。

例題

▼ すべての登録されたクライアントの配列と、まだ実行されずに残っているメソッド数の配列の作成を考えてみましょう。

```
ARRAY TEXT($clients;0)
ARRAY LONGINT($methods;0)
GET REGISTERED CLIENTS($clients;$methods)
```

▼ **REGISTER CLIENT**コマンドの例を参照してください。

参照

REGISTER CLIENT、EXECUTE ON CLIENT、UNREGISTER CLIENT

QUERY BY EXAMPLE

QUERY BY EXAMPLE({テーブル} {;*})

引数	タイプ		説明
テーブル	テーブル	→	レコードセレクションを求めるテーブルまたは、省略した場合、デフォルトテーブル
*	*	→	指定した場合、スクロールバーの非表示

説明

QUERY BY EXAMPLEコマンドは、「ユーザ」モードの「フォームによるクエリ」メニューと同じ処理を実行します。クエリウインドウとしてカレント入力フォームを表示します。

QUERY BY EXAMPLEコマンドは、クエリウインドウに入力された情報を使用して＜テーブル＞を検索します。このときに使用するフォームには、ユーザに検索させたいフィールドを納めなければなりません。この検索は最適化されています。つまり、クエリを最適化するためにインデックスフィールドが自動的に使用されます。

「ユーザ」モードの「フォームによるクエリ」メニューの使用方法に関する詳細は、『4th Dimensionユーザリファレンス』を参照してください。

例題

▼ 以下のメソッドの例は、まず“検索”という名前のフォームを表示します。次にユーザがフォームに検索データを入力し、検索の実行を要求した場合（システム変数OKに1が代入された場合）に、検索結果を表示するようにします。

```
INPUT FORM([従業員] ; "検索")    ` フォームを検索するために切り換える
QUERY BY EXAMPLE([従業員])    ` フォームを表示し、検索を実行する
If(OK=1)    ` ユーザが検索実行を指示した場合
    DISPLAY SELECTION([従業員])    ` レコードを表示する
End if
```

参照

ORDER BY、QUERY

システム変数とセット

「登録」ボタンをクリックするか、または“enter”キーを押すと、システム変数OKに1を代入し、検索を開始します。また「キャンセル」ボタンをクリックするか、またはキャンセルのキーコンビネーションを押すと、システム変数OKに0を代入し、検索を中止します。

QUERY

QUERY({テーブル} {;検索条件} {; *})

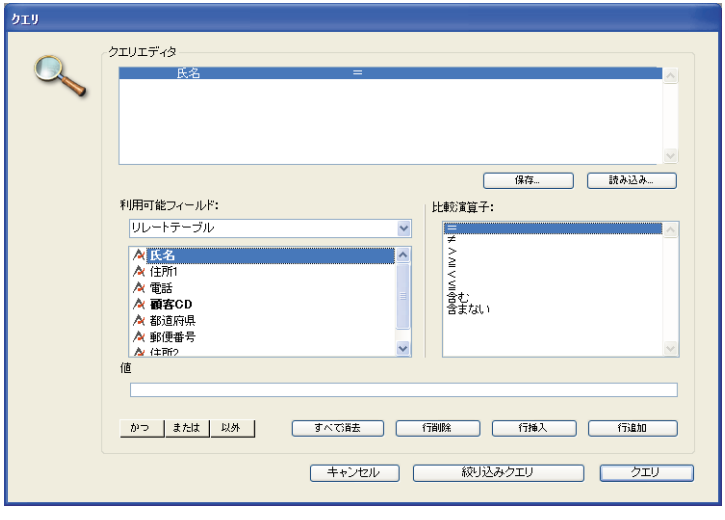
引数	タイプ		説明
テーブル	テーブル	→	レコードセレクションを求めるテーブルまたは、省略した場合、デフォルトテーブル
検索条件		→	検索条件
*	*	→	継続検索フラグ

説明

QUERYコマンドは、＜テーブル＞に対して＜検索条件＞に指定した条件に一致するレコードを検索し、検索結果をセレクションとして返します。このコマンドは、カレントプロセスの＜テーブル＞のカレントセレクションを変更します。そして、セレクションの先頭のレコードをカレントレコードにします。

引数＜テーブル＞を省略した場合、コマンドはデフォルトテーブルに対して適用されます。デフォルトテーブルが設定されていない場合には、エラーが発生します。

引数＜検索条件＞または＜*＞を指定しない場合、**QUERY**コマンドは＜テーブル＞用の「クエリ」エディタを表示します。以下の図は、「ユーザ」モードの「クエリ」エディタです。



「クエリ」エディタについての詳細は、『4th Dimension ユーザリファレンス』を参照してください。

ユーザはクエリを作成し、「クエリ」ボタンをクリックするか、または「絞り込みクエリ」ボタンをクリックして検索を実行します。検索が中断されずに実行された場合、システム変数OKには1が代入されます。ユーザが「キャンセル」をクリックするか、**QUERY**コマンドが中断されて実際には検索が行われなかった場合には、システム変数OKに0が代入されます。

例題

1. 以下の例は、[製品]テーブルに対する「クエリ」エディタを表示します。

```
QUERY([製品])
```

2. 以下の例は、デフォルトテーブル（設定されている場合）に対する「クエリ」エディタを表示します。

```
QUERY
```

引数＜検索条件＞を指定すると、標準の「クエリ」エディタは表示されず、クエリはプログラムから定義されます。単一クエリの場合（1つのフィールドだけを検索）、＜検索条件＞をもとに**QUERY**コマンドを1回コールします。複合検索の場合（複数フィールドに関する検索、または複合条件による検索）、＜検索条件＞を用いて必要な回数だけ**QUERY**コマンドをコールします。そして、最後の**QUERY**コマンドのコール以外の**QUERY**コマンドに対してオプション引数＜*＞を指定します。＜*＞を含まない最後のコールの後、実際の検索処理が実行されます。引数＜検索条件＞については、この節で更に詳しく説明します。

3. 以下の例は、[従業員]テーブルの名前が“山”で始まる人のレコードを検索します。

```
QUERY([従業員];[従業員]名字="山@")
```

4. 以下の例は、[従業員]テーブルの名字フィールドが“山”、または“川”で始まる人のレコードを検索します。

```
QUERY([従業員] ; [従業員]名字 = "山@";*) ` * は更に検索条件が続くことを示す
QUERY([従業員] ; | ; [従業員]名字 = "川@")
```

検索条件の指定方法

＜検索条件＞には、以下のような書式を使用します。

```
{論理演算子 ;} フィールド 比較演算子 比較値
```

■ ＜論理演算子＞は複合条件の検索を定義する際に、前に実行した**QUERY**コマンドの検索条件と次に実行する**QUERY**コマンドの検索条件を結合するために使用します。使用できる“論理演算子”は、「ユーザ」モードの「クエリ」エディタで使用できるものと同じです。“論理演算子”を次に示します。

検索用論理演算子	記号
かつ	&
または	¦
以外	#

＜論理演算子＞はオプションで、複合条件を使用して実行する検索処理の最初の**QUERY**コマンド上、および**QUERY**コマンドが1つしかない場合には使用することはできません。

■ ＜フィールド＞は、検索対象となるフィールドのことです。＜テーブル＞が自動リレートを持つ場合には、リレート先の1テーブルのフィールドも使用可能です。**QUERY**コマンドが適用されるテーブルは、nテーブルでなくてはなりません。

■ ＜比較演算子＞は、フィールドと比較値とを比較するために使用します。“比較演算子”を次に示します。

検索用比較演算子	記号
等号	=
不等号	#
より小さい	<
より大きい	>
より小さい、または等しい	<=
より大きい、または等しい	>=

■ ＜比較値＞は＜フィールド＞と比較するためのデータです。＜比較値＞は＜フィールド＞と同じデータタイプとして記述することも、文字列として記述することもできます。文字列として記述した場合には、自動的にフィールドのデータタイプと同じ形式に変換してから比較します。＜比較値＞は検索の初めに一度だけ評価されます。各レコードに対して評価されるわけではありません。文字列中に特定の文字列を含んでいるかどうかを検索する場合（包含検索）には、＜比較値＞にワイルドカード記号（@）を使用することができます。

複合条件検索を実行するための規則を次に示します。

■ 最初の**QUERY**コマンドは論理演算子を含んではいけません。

■ 最初以外の**QUERY**コマンドは論理演算子を含んでいなければいけません。

■ 最後の**QUERY**コマンド以外は引数＜*＞を指定しなければいけません。

■ 複合検索を実行するためには、最後の**QUERY**コマンドで引数＜*＞を指定してはいけません。代りに最後の**QUERY**コマンドで、テーブル以外に引数の全くない**QUERY**コマンドを実行することができます（「クエリ」エディタは表示されず、前の行までに定義した複合検索が実行されます）。

注：作成した各テーブル固有のカレントクエリは各テーブルにより管理されます。つまり各テーブルに1つの複合検索を同時に作成できます。引数＜テーブル＞を指定するか、DEFAULT TABLEコマンドを使用して、検索対象テーブルを明確に特定する必要があります。

検索の定義方法に関係なく、以下の処理が行われます。

■ 実際の検索処理に時間がかかる場合は、4th Dimensionは自動的にサーモメータで進捗状況のメッセージを表示します。**MESSAGES ON**コマンドと**MESSAGES OFF**コマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは「中止」ボタンをクリックして検索を中断することができます。検索が正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。

■ インデックスフィールドが指定された場合、検索処理は毎回最適化され（最初にインデックスフィールドが検索される）、検索は最小の時間で終了します。

例題

5. 以下の例は、“鈴木”という名字の人のレコードをすべて検索します。

```
QUERY([従業員] ; [従業員]名字 = "鈴木")
```

注：名字のフィールドにインデックスが設定されている場合、QUERYコマンドは高速な検索のために、自動的にインデックスを使用します。

注：この検索では、“鈴木”、“すずき”、“スズキ”等のレコードを検索します。大文字と小文字を区別するには、ASCIIコードを使用した検索を行ってください。

6. 以下の例は、“鈴木一郎”という名前の人のレコードをすべて検索します。名字のフィールドにはインデックスが設定されていますが、名前のフィールドにはインデックスが設定されていません。

```
QUERY([従業員] ; [従業員]名字 = "鈴木";*) ` 名字が鈴木で...
QUERY([従業員] ; &; [従業員]名前 = "一郎")      ` 名前が一郎という人を探す
```

検索を実行すると、まず、最初にインデックスを持った名字のフィールドを検索し、“鈴木”という名前の人のレコードのみに検索対象を絞ります。次に、このレコードセレクションの中から名前のフィールドに対してシーケンシャルな検索を行います。

7. 以下の例は、“鈴木”または“一郎”という名字の人のレコードをすべて検索します。名字のフィールドにはインデックスが設定されています。

```
QUERY([従業員] ; [従業員]名字;" = ";"鈴木";*)      ` 名字が鈴木という人か...
QUERY([従業員] ; | ; [従業員]名字;" = ";"一郎")    ` 名字が一郎という人を探す
```

QUERYコマンドは両方の検索にインデックスを使用します。2つの検索が実行され、その結果が内部セットに納められて、結合されます。

8. 以下の例は、会社名が設定されていないレコードをすべて検索します。これは空のフィールド(空の文字列)を検索することによって行います。

```
QUERY([従業員] ; [従業員]会社 = "")        ` 会社名のない人を探す
```

9. 以下の例は、“鈴木”という名字で“NY（ニューヨーク）”に会社のある人のレコードをすべて検索します。2番目の**QUERY**コマンドは、他のテーブルのフィールドを使用しています。これは、[従業員]テーブルから[会社]テーブルへリレートされているために可能になっています。

```
QUERY([従業員] ; [従業員]名字 = "鈴木";*) `名字が鈴木で...
QUERY([従業員] ; &; [会社]所在地 = "NY") `会社所在地がニューヨークの人を探す
```

10.以下の例は、名前のイニシャルがAからMの人のレコードをすべて検索します。

```
QUERY([従業員] ; [従業員]名前 < "n") ` 名前のイニシャルがAからMの人を探す
```

11.以下の例は、住所が“94（サンフランシスコ）”または“90（ロサンゼルス）”の人のレコードをすべて検索します（郵便番号が94または90で始まる）。

```
QUERY([従業員] ; [従業員]郵便番号 = "94@";*) ` 住所がサンフランシスコか...
QUERY([従業員] ; | ; [従業員]郵便番号 = "90@") ` ロサンゼルスの人を探す
```

12.以下の例は、インデックスの設定されたサブフィールドをすべて検索します。この検索は親レコード（[従業員]テーブルのレコード）のセレクションを返します。サブレコードのセレクションは返しません。検索結果は“太郎”という名前の子供がいる人のレコードすべてとなります。

```
` 名前が“太郎”という子供を探す
QUERY([従業員] ; [従業員]子供'名前 = "太郎")
```

13.以下の例は、「リクエスト」ダイアログボックスに入力された送り状の参照番号と一致するレコードを検索します。

```
v番号:=Request("送り状の参照番号を入力してください。")
                                        ` 送り状の参照番号を入力する

If(OK=1)` 検索実行？
` 入力された番号と同じ送り状番号を探す
      QUERY([送り状] ; [送り状]参照番号 = v番号)
End if
```

14.以下の例は、1996年に作成された送り状のレコードをすべて検索します。これは、1995年12月31日以降で1997年1月1日以前のレコードをすべて検索します。

```
QUERY([送り状] ; [送り状]作成日付 > !95.12.31!;*)` 作成日付が95年12月31日から...
QUERY([送り状] ;&; [送り状]作成日付 < !97.01.01!)` 97年7月1日までの送り状を探す
```

15.以下の例は、¥2,500,000≦給与＜¥5,000,000の条件に当てはまる従業員のレコードをすべて検索します。

```
` ¥2,500,000≦給与＜¥5,000,000の従業員を探す
QUERY([従業員] ; [従業員]給与 >= 2500000;*)
QUERY([[従業員] ; &[従業員]給与 < 5000000)
```

16 以下の例は、3,000,000円以上の給与を得ている営業部に所属する従業員のレコードをすべて検索します。“給与”フィールドにはインデックスが設定されているため最初に検索します。2番目の**QUERY**コマンドが、他のテーブルのフィールドを使用している点に注意してください。[従業員]テーブルから[部門]テーブルの“名称”フィールドへn対1の自動リレートが設定されているためにこのようなことが可能になっています。[従業員]テーブルのレコードの順番に従ってリレート処理を実行するために、[部門]テーブルの“名称”フィールドには、インデックスが設定されているにも関わらず、インデックスを使用した検索は行われません。

```
` 給与が3,000,000円よりも多い、営業部に所属する社員を探す
QUERY([従業員] ; [従業員]給与 > 3000000;*)
QUERY([従業員] ; &; [部門]名称 = "営業部")
```

17.以下の例は、変数“Var”に入力された情報を検索します。

```
QUERY([規定] ; [規定]テキスト=Var) `変数「Var」と一致するすべての値を探す
```

この検索処理では変数“Var”の内容次第で、多くの異なる検索結果が得られます。検索もまた異なる方法で実行されます。例えば、以下のようなものです。

■ 変数“Var”の内容が“@Copyright”の場合には、[規定]テーブルの“テキスト”フィールドが“Copyright”という文字列で終わるすべてのレコードを検索します。

■ 変数“Var”の内容が“Copyright@”の場合には、[規定]テーブルの“テキスト”フィールドが“Copyright”という文字列で始まるすべてのレコードを検索します。

■ 変数“Var”の内容が“@Copyright@”の場合には、[規定]テーブルのテキストフィールドに“Copyright”という文字列が最低1つ以上含まれるすべてのレコードを検索します。

参照

QUERY SELECTION

QUERY SELECTION

QUERY SELECTION({テーブル}{; 検索条件} {; *})

引数	タイプ		説明
テーブル	テーブル	→	レコードセレクションを求める テーブル、または 省略した場合、デフォルトテーブル
検索条件		→	検索条件
*	*	→	継続検索フラグ

説明

QUERY SELECTIONコマンドは、＜テーブル＞のレコードを検索します。**QUERY SELECTION**コマンドは、カレントプロセスの＜テーブル＞のカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY SELECTIONコマンドは、**QUERY**コマンドと同じような動作を実行します。相違点は、検索する範囲が異なるだけです。

■ **QUERY**コマンドは＜テーブル＞の全レコードを検索する。

■ **QUERY SELECTION**コマンドはテーブルのカレントセレクションのレコードを検索します。

詳細については、**QUERY**コマンドの説明を参照してください。

注：SET DATABASE PARAMETER関数は、QUERY SELECTION関数がindexを使っているかどうかを調べます。セレクトしたレコードの番号によります。

例題

以下の例は、**QUERY SELECTION**コマンドと**QUERY**コマンドの違いを表わしたものです。2種類の検索があります。

```
` 東京が所在地の会社をすべて検索する
QUERY([会社]; [会社]都市="東京")
` 証券を取り扱う会社をすべて検索する
`(所在地は問わない)
QUERY([会社]; [会社]業種="証券")
```

2番目の検索では、最初の検索結果が“無視”されている点に注意してください。以下の検索文と比較してください。

```
` 東京が所在地の会社をすべて検索する
QUERY([会社]; [会社]都市="東京")
` 証券を取り扱う会社をすべて検索する
` 会社の所在地は東京とする
QUERY SELECTION([会社]; [会社]業種="証券")
```

QUERY SELECTIONコマンドは、レコードセレクションに対する検索を行います。したがって、この例では所在地が東京である会社を対象に検索が行われます。

参照

QUERY、SET DATABASE PARAMETER

QUERY BY FORMULA

QUERY BY FORMULA({テーブル} {;} {検索用フォーミュラ})

引数	タイプ		説明
テーブル	テーブル	→	レコードセレクションを求めるテーブル、または省略した場合、デフォルトテーブル
検索用フォーミュラ	ブール	→	検索用フォーミュラ

説明

QUERY BY FORMULAコマンドは、＜テーブル＞からレコードを検索します。**QUERY BY FORMULA**コマンドは、カレントプロセスの＜テーブル＞のカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY BY FORMULAコマンドと**QUERY SELECTION BY FORMULA**コマンドは、全く同じように機能しますが、**QUERY BY FORMULA**コマンドがテーブルのすべてのレコードを検索対象とするのに対して、**QUERY SELECTION BY FORMULA**コマンドはカレントセレクションのレコードのみを検索対象とします。

両方のコマンドは、テーブルまたはセレクションの各レコードに対して＜検索用フォーミュラ＞を適用します。＜検索用フォーミュラ＞は、“True（真）”か“False（偽）”のいずれかの状態に評価されるブール式です。＜検索用フォーミュラ＞で“True（真）”に評価されたレコードを新しいセレクションに追加します。

＜検索用フォーミュラ＞は、フィールドと値とを比較するだけの単純なものから、計算、またはリレート先テーブルの情報を評価するような複雑なものまで処理します。＜検索条件式＞には、4th Dimensionの関数（コマンド）やユーザが作成した関数（メソッド）やファーミュラを使用することができます。文字フィールドやテキストフィールドに対して作業を実行する場合は、＜検索条件式＞にワイルドカードを使用することもできます。

検索処理が完了した時点で、新しいセレクションの最初のレコードがディスクからロードされ、その先頭のレコードをカレントレコードとします。

これらのコマンドではシーケンシャルな検索を行い、インデックスを使用した検索は行いません。**QUERY BY FORMULA**コマンドと**QUERY SELECTION BY FORMULA**コマンドは、インデックスが設定されたフィールドを検索する場合に、**QUERY**コマンドよりも処理速度は遅くなります。検索に要する時間は、テーブルやセレクション中のレコード数に比例します。

4D Server：サーバでは検索用フォーミュラは実行されません。各レコードがローカルのワークステーションへ送信され、ワークステーション上で検索用フォーミュラが評価されます。したがって、このコマンドを4D Serverで実行するとQUERYコマンドよりも効率が低下します。

例題

1. 以下の例は、すべての年度の12月に作成された送り状のレコードを検索します。これは、**Month of**関数を各レコードに適用して検索します。このような検索は月の情報を別のフィールドとして持たない限り、他の方法では実現できません。

```
` 12月に作成された送り状を探す
QUERY BY FORMULA([送り状] ; Month of([送り状]作成日付) = 12)
```

2. 以下の例は、名前が全角で5文字（半角で10文字）以上の人のレコードを検索します。

```
` 名前が全角で5文字以上の人を探す
QUERY  BY FORMULA([従業員] ; Length([従業員]名前) > 10)
```

参照

QUERY、QUERY SELECTION、QUERY SELECTION BY FORMULA

QUERY SELECTION BY FORMULA

QUERY SELECTION BY FORMULA({テーブル} {;} {;検索用フォーミュラ})

引数	タイプ		説明
テーブル	テーブル	→	レコードセレクションを求めるテーブル、または省略した場合、デフォルトテーブル
検索用フォーミュラ	ブール	→	検索用フォーミュラ

説明

QUERY SELECTION BY FORMULAコマンドは、＜テーブル＞からレコードを検索します。**QUERY SELECTION BY FORMULA**コマンドは、カレントプロセスの＜テーブル＞のカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY SELECTION BY FORMULAコマンドは、**QUERY BY FORMULA**コマンドと同じような動作を実行します。相違点は、検索する範囲が異なるだけです。

■ **QUERY BY FORMULA**コマンドは＜テーブル＞の全レコードを検索する。

■ **QUERY SELECTION BY FORMULA**コマンドはテーブルのカレントセレクションのレコードを検索します。

詳細については、**QUERY BY FORMULA**コマンドの説明を参照してください。

参照

QUERY、QUERY BY FORMULA、QUERY SELECTION

QUERY WITH ARRAY

QUERY WITH ARRAY(インデックスフィールド; 配列)

引数	タイプ		説明
インデックスフィールド	フィールド	→	値の比較の為に使用されるインデックスフィールド
配列	配列	→	検索される値の配列

説明

QUERY WITH ARRAYコマンドは、＜インデックスフィールド＞の値が＜配列＞の要素の値のうち少なくとも1つと等しいレコードをすべて検索します。見つかったレコードは、新しくカレントセレクションになります。

このコマンドを使用すると、複数の値に基づく検索を速やかかつ簡単に行えます。

注：

・このコマンドは、インデックスフィールドにのみ有効です。タイプがテキスト、ピクチャ、サブフィールドまたはBLOBのフィールドには使用できません。

・倍長整数タイプの配列は、時間タイプのフィールドと互換性があるということを覚えておいてください。

例題

以下の例題は、フランスおよびアメリカの顧客レコードを検索するものです。

```
ARRAY STRING(2;SearchArray;30)
SearchArray{1}:="FR"
SearchArray{2}:="US"
QUERY WITH ARRAY([Clients]Country;SearchArray)
```

SET QUERY DESTINATION

SET QUERY DESTINATION(配置先タイプ {; 配置先オブジェクト})

引数	タイプ		説明
配置先タイプ	数値	→	0=カレントセレクション 1=セット 2=命名セレクション 3=変数
配置先オブジェクト	文字列または変数	→	セット名、または命名セレクション名 変数名

説明

SET QUERY DESTINATIONコマンドを使い、カレントプロセスの以下のクエリの結果を配置する場所を4th Dimensionに指示することができます。

引数<配置先タイプ>に配置場所のタイプを指定します。4th Dimensionには、以下の表のようなあらかじめ定義された定数を持っています。

定数	タイプ	値
Into current selection	倍長整数	0
Into set	倍長整数	1
Into named selection	倍長整数	2
Into variable	倍長整数	3

以下の表にしたがって、オプション引数<配置先オブジェクト>にクエリ自身の配置先を指定します。

引数<配置先タイプ>	引数<配置先オブジェクト>
0（カレントセレクション）	引数を省略します。
1（セット）	（既存または作成した）セット名を渡します。
2（命名セレクション）	（既存または作成した）命名セレクション名を渡します。
3（変数）	（既存または作成した）変数名を渡します。

例えば、

```
SET QUERY DESTINATION(Into current selection)
```

以下のクエリで見つかったレコードは、最終的にはそのクエリの対象となるテーブルの新しいカレントセレクションの中に配置されます。

SET QUERY DESTINATION(Into set ; "MySet")

以下のクエリで見つかったレコードは、最終的にはセット“MySet”の中に配置されます。クエリの対象となるテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

注：クライアント／サーバでは、ローカル／クライアントのセット（$で名前が始まるもの）をクエリの配置先として使用することはできません。このタイプのセットはサーバでクエリが実行されたときにクライアントマシンで作られます。セットのタイプのについての詳細は第４８章「セットコマンド」を参照してください。

SET QUERY DESTINATION(Into named selection ; "MyName")

以下のクエリで見つかったレコードは、最終的には命名セレクション“MyName”の中に配置されます。クエリの対象となるテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

SET QUERY DESTINATION(Into variable ; $MyVar)

以下のクエリで見つかったレコードは、最終的には変数“$MyVar”の中に配置されます。クエリの対象となるテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

警告：SET QUERY DESTINATIONコマンドは、カレントプロセス内で行われた以下のクエリのすべてに影響を及ぼします。そのため、常にSET QUERY DESTINATION（配置先タイプ#0）の呼び出しは、通常のクエリを再実行するためにSET QUERY DESTINATION(0)の呼び出しと対になっていることを覚えておいてください。

SET QUERY DESTINATIONコマンドは、下記のクエリコマンドの動作を変更します。

■ **QUERY**

■ **QUERY SELECTION**

■ **QUERY BY EXAMPLE**

■ **QUERY BY FORMULA**

■ **QUERY SELECTION BY FORMULA**

■ **QUERY WITH ARRAY**

これに対して、**SET QUERY DESTINATION**コマンドは、**ALL RECORDS**コマンドや**RELATE MANY**コマンド等、テーブルのカレントセレクションを変更する他のコマンドには影響を与えません。

例題

1. 以下の例は[電話帳]テーブルのレコードを表示するフォームを作成して、そのフォームに（アルファベット26文字の）“asRolodex”と名付けられたタブコントロールと[電話帳]レコードを表示するサブフォームを作成します。タブコントロールから任意のタブを選択することにより、そのタブ上の文字で始まるレコードを表示することができます。

このアプリケーション例では、前回の検索結果を命名セレクションとしてを持っているため、任意のタブを選択するたびにクエリを実行する必要はありません。このように、検索に要する時間を節約することができます。

これを実行するために、必要な時に再利用できる命名セレクションの中にクエリを転送することができます。以下のコードは「asRolodex」タブコントロールのオブジェクトメソッドです。

```
`「asRolodex」タブコントロールのオブジェクトメソッド
Case of
    ¥(Form even=On Load)
        `フォームが画面上に表示される前に文字列配列とブール配列を初期化する。
        `これらの配列はタブ上の文字をクエリを実行する際に使用される。
        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)
        `タブコントロール上をクリックすると、そのタブ上の文字に対応する
                                クエリが実行されるかどうかをチェックする。t
        If(Not(abQueryDone{asRolodex}))
            `実行されない場合、命名セレクションに以下のクエリを転送する。
            SET QUERY DESTINATION(Into named selection ;
                                "Rolodex"+asRolodex{asRolodex})
            `クエリを実行する。
            QUERY([電話帳];[電話帳]名字=asRolodex{asRolodex}+"@")
            `通常のクエリ処理に戻す
            SET QUERY DESTINATION(Into current selection)
            `次回その文字を選択した場合は、クエリを再実行しません。
            abQueryDone{asRolodex}:=True
        End if
        `選択された文字に対応したレコードを表示する命名セレクションを
                                使用する。
```

```
            USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex})
        ¥(Form event=On Unload)
            `フォームが画面から見えなくなったら、作成した命名セレクション
                                                    を消去する。
            For($vlElem ; 1 ; 26)
                If(abQueryDone{$vlElem})
                    CLEAR NAMED SELECTION
                                                ("Rolodex"+asRolodex{$vlElem})
                End if
            End for
            `不要になった2つの配列をクリアする。
            CLEAR VARIABLE(asRolodex)
            CLEAR VARIABLE(abQueryDone)
    End case
```

2. 以下の“重複値検査”プロジェクトメソッドは、テーブル内のフィールド番号に重複する値がないかどうか検査することができます。カレントレコードは、既存または新しく作成したレコードです。

```
`「重複値検査」プロジェクトメソッド
`重複値検査(ポインタ ; ポインタ { ; ポインタ... } ) → ブール
`重複値検査( ->ブール ; ->フィールド1 { ; ->フィールド2... } ) → Yes or No
C_BOOLEAN($0 ; $2)
C_POINTER(${1})
C_LONGINT($vlField ; $vlNbFields ; $vlFound ; $vlCurrentRecord)
$vlNbFields:=Count parameters-1
$vlCurrentRecord:=Record number($1->)
If($vlNbFields>0)
    If($vlCurrentRecord#-1)
        If($vlCurrentRecord<0)
            `カレントレコードがまだ保存されていない
                    新規レコードの場合、1件のレコードが見つけると
                                        すぐに以下のクエリを中止できます。
            SET QUERY LIMIT(1)
        Else
            `カレントレコードが既存レコードの場合、
            `2件のレコードが見つけるとすぐに以下のクエリを
                                                    中止できます。
            SET QUERY LIMIT(2)
        End if
        `クエリはカレントレコードやカレントセレクションを
                                                変更することなく
```

```
`$viFoundの中にクエリ自身の結果を返します。
SET QUERY DESTINATION(Into variable ; $vlFound)
`指定されたフィールド番号によってクエリを作成する。
Case of
        ¥($vlNbFields=1)
                QUERY($1-> ; $2->=$2->)
        ¥($vlNbFields=2)
                QUERY($1->; $2->=$2-> ; *)
                QUERY($1->; & ; $3->=$3->)
Else
                QUERY($1-> ; $2->=$2-> ; *)
                For($vlField;2;$vlNbFields-1)
                        QUERY($1-> ; & ; $
                                {1+$vlField}->=${1+$vlField}->;*)
                End for
                QUERY($1->; & ; $
                        {1+$vlNbFields}->=${1+$vlNbFields}->)
End case
SET QUERY DESTINATION (Into current selection)
                                        `通常のクエリ処理に戻す。
SET QUERY LIMIT(0) `クエリに制限はありません。
`クエリ処理の結果
Case of
        ¥($vlFound=0
                $0:=True `重複した値がない場合
        ¥($vlFound=1)
                If($vlCurrentRecord<0)
                        $0:=False `新規レコードと同じ値を
                                持つ既存レコードが見つかった場合
                Else
                        $0:=True `重複した値がない場合
                End if
        ¥($vlFound=2)
                $0:=False ` どちらの場合であっても、
                                        値が重複している場合
        End case
Else
If(<>DebugOn) `開発バージョンの場合、
                                        デバッグウインドウを開く。
        TRACE `警告！重複しない値が
                                        カレントレコードなしで呼び出される。
```

```
        End if
        $0:=False `結果は保証されない。
    End if
    Else
        If(<>DebugOn) `開発バージョンの場合、デバッグウインドウを開く。
            TRACE `警告！重複しない値がクエリ条件なしで呼び出される。
        End if
        $0:=False `結果は保証されない。
    End if
```

このプロジェクトメソッドをアプリケーションに作成した後、以下のように記述します。

```
        ` ...
    If(重複値検査(->[交渉];->[交渉]会社;->[交渉]名字 ; ->[交渉]名前))
        `ここで、重複していない値を持つレコードに関する任意の処理を実行する。
    Else
        ALERT("この会社は既に交渉を行っています。.")
    End if
        ` ...
```

参照

QUERY、QUERY BY EXAMPLE、QUERY BY FORMULA、QUERY SELECTION、QUERY SELECTION BY FORMULA、SET QUERY LIMIT

SET QUERY LIMIT

SET QUERY LIMIT(制限)

引数	タイプ		説明
制限	数値	→	レコード数、0=制限なし

説明

SET QUERY LIMITコマンドは、引数＜制限＞で渡した数のレコードが見つかったらすぐにカレントプロセスの以下のクエリ（検索）を中止するように4th Dimensionに指示します。

例えば、＜制限＞に1を受け渡すと、以下のクエリはクエリ条件に一致した1件のレコードを見つけるとすぐにインデックスまたはデータファイルのブラウズ作業を中止します。

制限なしのクエリを再実行するには、＜制限＞に0を渡した**SET QUERY LIMIT**コマンドを再度呼び出します。

警告：SET QUERY LIMITコマンドは、カレントプロセス内で行われる以下のクエリのすべてに影響を及ぼします。そのため、常にSET QUERY LIMIT(制限)「制限>0」の呼び出しは、制限なしのクエリを再実行するためにSET QUERY LIMIT(0)の呼び出しと対になっていることを覚えておいてください。

SET QUERY LIMITコマンドは、下記のクエリコマンドの動作を変更します。

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY SELECTION BY FORMULA**

これに対して、**SET QUERY LIMIT**コマンドは、**ALL RECORDS**コマンドや**RELATE MANY**コマンド等のテーブルのカレントセレクションを変更する他のコマンドに影響を与えません。

例題

1. “100万円以上の売上を獲得している顧客10人を探せ”という要求に対応するクエリを実行するには、以下のように記述します。

```
SET QUERY LIMIT(10)
QUERY([顧客] ; [顧客]売上 > 1000000)
SET QUERY LIMIT(10)
```

2. **SET QUERY DESTINATION**コマンドの2番目の例を参照してください。

参照

QUERY、QUERY BY EXAMPLE、QUERY BY FORMULA、QUERY SELECTION、QUERY SELECTION BY FORMULA、SET QUERY DESTINATION

Find index key

Find index key(インデックスフィールド ; 値) → 倍長整数

引数	タイプ		説明
インデックスフィールド	フィールド	→	検索を実行するインデックスフィールド
値		→	検索する値
		←	見つかった値
戻り値	倍長整数	←	レコード番号 レコードが何も見つからなければ-1

説明

このコマンドは、そのインデックスフィールドが値と等しい最初のレコード番号を返します。何もレコードが見つからなければ、**Find index key**は-1を返します。

このコマンドを呼び出した後、値には見つかった値が挿入されます。これは、文字列フィールド上でワイルドカード（"@"）を使って検索し、見つかった値を知る事ができるようにします。

このコマンドは、カレントセレクションまたはカレントレコードを変更しません。

インデックスのみを使用しているため速く、特にレコード入力中に重複データの入力を防ぐのに役立ちます。

例題

▼ オーディオCD用のデータベースで、レコード入力中に既に登録されているシンガーかどうかを確認したいと仮定してみましょう。同姓同名も存在するため[Singer]Nameフィールドを重複不可にせず、入力フォームで[Singer]Nameフィールドのオブジェクトメソッドに下記のコードを書くことにします。

```
If(Form event=On Data Change)
    $RecNum:=Find index key([Singer]Name;[Singer]Name)
    If($RecNum # -1) `この名前が既に登録されている場合
        CONFIRM("同名のシンガーが既に存在します。レコードを
                                        見ますか？ ";"はい ";"いいえ ")
        If(OK=1)
            GOTO RECORD([Singer];$RecNum)
        End if
    End if
End if
```

ORDER BY

ORDER BY({テーブル}{;フィールド1}{; ソート種別1} {;...;フィールドN; ソート種別N}{;*})

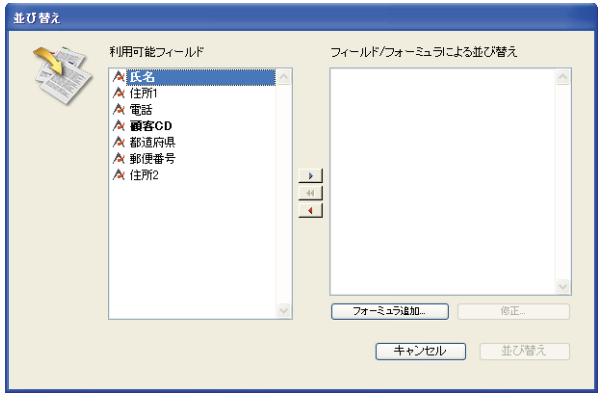
引数	タイプ		説明
テーブル	テーブル	→	ソート対象となるテーブル 省略した場合、デフォルトテーブル
フィールド	フィールド	→	ソートキーとなるフィールド
ソート種別	>または<	→	各レベルのソート種別 >昇順、<降順
*	*	→	ソート継続フラグ

説明

ORDER BYコマンドは、カレントプロセスの<テーブル>のカレントレコードセレクションをソートします。ソートが終了すると、セレクションの先頭レコードがカレントレコードとなります。

引数<テーブル>を省略した場合には、デフォルトテーブルがソートされます。デフォルトテーブルが設定されていない場合には、エラーが発生します。

引数<フィールド>や<ソート種別>を指定しない場合、**ORDER BY**コマンドは<テーブル>を対象とした「並び替え」エディタを表示します。次に「ユーザ」モードの「並び替え」エディタを示します。



「並び替え」エディタの使用に関する詳細は、『4th Dimension ユーザリファレンス』を参照してください。

ユーザはソートを組み立て、「並び替え」ボタンをクリックしてソートを実行します。ソートが中断されずに実行されると、システム変数OKに1がセットされます。ユーザが「キャンセル」ボタンをクリックすると、**ORDER BY**コマンドは中止されてソートは実際には実行されず、システム変数OKには0がセットされます。

例題

1. 以下の例は[住所]テーブルを対象とした「並び替え」エディタを表示します。

   ORDER BY([住所])

2. 以下の例は、デフォルトテーブルを対象とした「並び替え」エディタを表示します。

   ORDER BY

引数＜フィールド＞および＜ソート種別＞を指定すると、標準の「並び替え」エディタは表示されず、ソートはプログラムから定義されることになります。1つのレベル、または複数のレベルを用いてセレクションのソートを実行することができます。ソートレベルごとに、引数＜フィールド＞にフィールドを、＜ソート種別＞にはソート順を指定します。＜ソート種別＞が“>”の場合には、ソートを昇順に行います。＜ソート種別＞が“<”の場合には、ソートを降順に行います。

3. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドで昇順に並べ替えます。

   ORDER BY([住所] ; [住所]郵便番号 ; >)

4. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドで降順に並べ替えます。

   ORDER BY([住所] ; [住所]郵便番号 ; <)

5. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドを第1ソートキーに、“名字”フィールドを第2ソートキーに指定しそれぞれ昇順に並べ替えます。

   ORDER BY([住所] ; [住所]郵便番号 ; > ; [住所]名字 ; >)

6. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドを第1ソートキーに、“名字”フィールドを第2ソートキーに指定しそれぞれ降順に並べ替えます。

   ORDER BY([住所] ; [住所]郵便番号 ; < ; [住所]名字 ; <)

7. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドを昇順に、“名字”フィールドを降順に並べ替えます。

   ORDER BY([住所] ; [住所]郵便番号 ; > ; [住所]名字 ; <)

8. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドを降順に、“名字”フィールドを昇順に並べ替えます。

```
ORDER BY([住所] ; [住所]郵便番号 ; < ; [住所]名字 ; >)
```

＜ソート種別＞を省略した場合には、デフォルトとしてソートを昇順に行います。

9. 以下の例は、[住所]テーブルのカレントセレクションを“郵便番号”フィールドで昇順に並べ替えます。

```
ORDER BY([住所] ; [住所]郵便番号)
```

1つのフィールドのみの指定（1レベルのソート）で、そのフィールドにインデックスが設定されている場合は、インデックスを使用してソートします。インデックスが指定されていなかったり、複数のフィールドを指定した場合には、ソート処理はシーケンシャルに行われます。フィールドは、並び替えが行われた（セレクションの）テーブルに属していたり、または＜テーブル＞への自動リレートが設定された1テーブルに属している場合があります（**ORDER BY**コマンドが適用されるテーブルは必ずnテーブルでなくてはならないことを覚えておいてください）。この場合、ソート処理は常にシーケンシャルに行われます。

例題

10.以下の例は、[住所]テーブルの“郵便番号”フィールドにインデックスが設定されている場合は、インデックスソートを実行します。

```
ORDER BY([住所] ; [住所]郵便番号 ; > )
```

11.以下の例は、フィールドにインデックスが設定されていてもいなくても、シーケンシャルソートを実行します。

```
ORDER BY([住所] ; [住所]郵便番号 ; > ; [住所]名字 ; >)
```

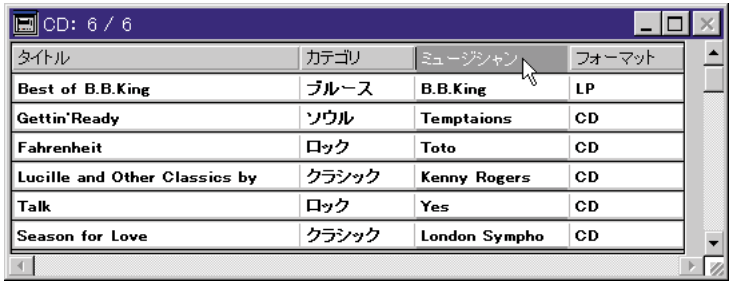
12.以下の例は、リレート先のフィールドを使用してシーケンシャルソートを実行します。

```
ORDER BY([送り状] ; [会社]会社名)
```

ソートの定義方法に関係なく、実際のソート処理に時間がかかる場合は、4th Dimensionは自動的にサーモメータで進捗状況のメッセージを表示します。**MESSAGES ON**コマンドと**MESSAGES OFF**コマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは「中止」ボタンをクリックしてソートを中止することができます。ソートが正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。

マルチソート（複数のフィールドによるソート）を実行するために、**ORDER BY**を何度も、必要なだけコールすることができ、実際のソートを始める最後の**ORDER BY**を除いて、オプション引数<*>を指定することによって実現します。この設定は、カスタマイズされたユーザインタフェースでの複数キーのソート管理に役立ちます。

警告：このシンタックスを利用した場合はORDER BYのコールごとに1つのソートレベル（フィールド）だけを渡すことができます。



13.カスタムメニューモードで表示される出力フォームで、ただ、縦の列ヘッダをクリックすることによって昇順にソートします。

shiftキーを押しながら、他の縦の列ヘッダをクリックすると、ソートは複数フィールドでソートが実行されます

各縦の列ヘッダには、以下のオブジェクトメソッドが埋め込まれたハイライトボタンを含んでいます。

MULTILEVEL(->[CDs]Title) `タイトルコラムヘッダのボタン

各ボタンは、縦の列フィールドに対応するポインタで、*MULTILEVEL*プロジェクトをコールします。*MULTILEVEL*プロジェクトメソッドは、以下の通りです。

```
` MULTILEVELプロジェクトメソッド
` MULTILEVEL(Pointer)
` MULTILEVEL(->[Table]Field)
C_POINTER($1) `Sort level(field)
C_LONGINT($lLevelNb)
`ソートレベルの取得
If(Not(Shift down)) `シンプルレベル(1フィールドによるソート)
        ARRAY POINTER(aPtrSortField;1)
        aPtrSortField{1}:=$1
Else
        $lLevelNb:=Find in array(aPtrSortField;$1) `このフィールドはすでに
                                                    ソートされているか?
        If($lLevelNb<0) `ソートされていないなら
```

```
                INSERT ELEMENT(aPtrSortField;Size of array(aPtrSortField)+1;1)
                aPtrSortField{Size of array(aPtrSortField)}:=$1
            End if
        End if
        `ソートの実行
        $lLevelNb:=Size of array(aPtrSortField)
        If($lLevelNb>0)  `少なくとも1つのソート条件があれば
            For($i;1;$lLevelNb)
                ORDER BY([CDs];(aPtrSortField{$i})->;>;*)  `ソート定義を構築する
            End for
                ORDER BY([CDs])  `*で終わらないソート定義で、現時点での
                                                        ソートを実行
        End if
```

参照

ORDER BY FORMULA

ORDER BY FORMULA

ORDER BY FORMULA(テーブル {; ソート条件式} {; ソート種別1}{ ;...; ソート条件式N ; ソート種別N })

引数	タイプ		説明
テーブル	テーブル	→	ソート対象となるテーブル
ソート条件式	文字または実数 整数、倍長整数、日付 時間、ブール	→	ソート条件式
ソート種別	>または<	→	各レベルのソート種別 >昇順、<降順

説明

ORDER BY FORMULAコマンドは、カレントプロセスの＜テーブル＞のカレントレコードセレクションをソートします。ソートが終了すると、セレクションの先頭レコードがカレントレコードとなります。

引数＜テーブル＞を必ず指定しなければならない点に注意してください。デフォルトテーブルを使用することはできません。

1つのレベル、または複数のレベルを用いてセレクションのソートを実行することができます。ソートレベルごとに、引数＜ソート条件式＞に式を、＜ソート種別＞にはソート順を指定します。＜ソート種別＞が“>”の場合には、ソートを昇順に行います。＜ソート種別＞が“<”の場合には、ソートを降順に行います。＜ソート種別＞を省略した場合には、デフォルトとしてソートを昇順に行います。

引数＜ソート条件式＞に指定できるタイプは、文字、実数、整数、倍長整数、日付、時間、ブールです。

ソートの定義方法に関係なく、実際のソート処理に時間がかかる場合は、4th Dimensionは自動的にサーモメータで進捗状況のメッセージを表示します。**MESSAGES ON**コマンドと**MESSAGES OFF**コマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは「中止」ボタンをクリックしてソートを中止することができます。ソートが正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。

4D Server：4D Serverでは＜ソート条件式＞を解釈することができないため、各レコードがローカルなワークステーションに送信されます。ソート順のフォーミュラはワークステーションで評価されます。これによりソートの効率は低下します。できる限りORDER BYコマンドを使用してください。

ORDER BYコマンドとは異なり、**ORDER BY FORMULA**コマンドは常にシーケンシャルソートを実行します。

例題

▼ 以下の例は、[従業員]テーブルを“名字”フィールドの長さをキーにして降順に並び替えます。最も長い名字を持った人がカレントセレクションの先頭になります。

```
ORDER BY FORMULA([従業員] ; Length([従業員]名字) ; <)
```

参照

ORDER BY

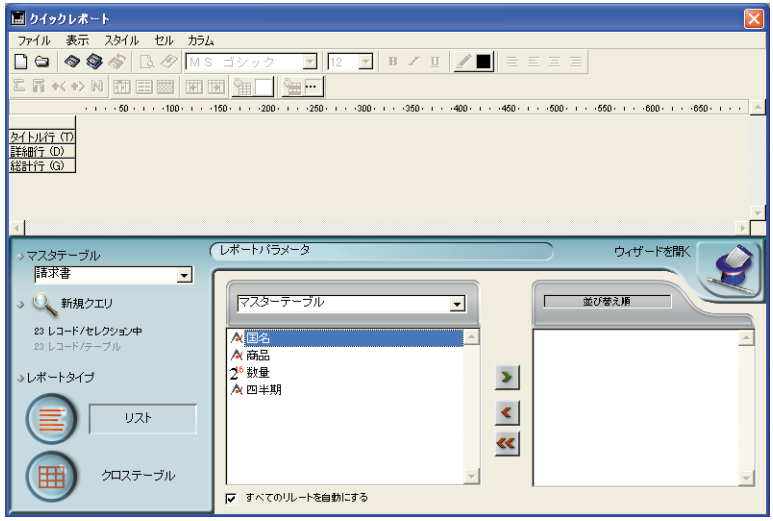
QR REPORT

QR REPORT({テーブル; }ドキュメント{; 階層型}{; ウィザード}{; 検索}{;*})

引数	タイプ		説明
テーブル	テーブル	→	レポートに使用するテーブル、または省略された場合にはデフォルトテーブル
ドキュメント	文字列	→	ロードするクイックレポートドキュメント
階層型	ブール	→	True= n リレートテーブルを表示する False= n リレートテーブルを表示しない（デフォルト値）
ウィザード	ブール	→	True=ダイアログにウィザードボタンを表示する False=ダイアログにウィザードボタンを表示しない（デフォルト値）
検索	ブール	→	True=検索ツールおよびマスターテーブル選択あり False=検索ツールおよびマスターテーブル選択なし（デフォルト値）
*		→	印刷ダイアログを表示しない

説明

QR REPORTコマンドは、次に示す「クイックレポート」エディタを用いて作成されたテーブルレポートを実行します。



「クイックレポート」エディタを使用すると、ユーザは独自のレポートを作成することができます。「クイックレポート」エディタが表示された際、「マスターテーブル」選択ドロップダウンリストや「新規クエリ」ボタンが存在しない可能性があることを除けば、ユーザは「ユーザ」モードでこのエディタが表示された場合と同じ状況で作業を行うことができます。つまり、ユーザはエディタに関するすべての制御を行えます。「クイックレポート」エディタを用いたレポートの作成に関する詳細は、『クイックレポートエディタリファレンス』マニュアルを参照してください。

注：

・テーブルが“非表示”として宣言されている場合、エディタは表示されません。

・QR REPORTコマンドによって表示されたクイックレポートエディタには、「すべてのリレートを自動にする」チェックボックスがありません。デベロッパは、SET AUTOMATIC RELATIONSおよびSET FIELD RELATIONコマンドを使用して、リレートの状態を制御することができます。

■ ドキュメント（文字列）

引数＜ドキュメント＞には、「クイックレポート」エディタを用いて作成され、ディスク上に保存されたレポートドキュメントを渡します。このドキュメントにはレポートに関する仕様が納められ、印刷されるレコードは含みません。

＜ドキュメント＞に空の文字列（""）が指定されると、**QR REPORT**コマンドは「ファイルを開く」ダイアログボックスを表示し、ユーザは印刷するレポートを選択することができます。

引数＜ドキュメント＞にドキュメントが指定され、そのドキュメントが存在しない場合（例えば、＜ドキュメント＞にChar(1)を指定）、「クイックレポート」エディタが表示されます。

■ 階層型（ブール）

引数＜階層型＞には、フィールド選択リストにｎテーブルを表示するかどうかを指定します。デフォルトとして、この値は0（ｎテーブルを表示しない）に設定されています。

■ ウィザード（ブール）

この引数には、「クイックレポート」エディタ上に「ウィザードを開く」ボタンを表示するかどうかを指定し、結果としてウィザードへのアクセスを許可または禁止します。デフォルトでは、この値は0（ウィザードへのアクセス不可）に設定されます。

■ 検索（ブール）

この引数には、「クイックレポート」エディタ上に「新規クエリ」ボタンと「マスターテーブル」ドロップダウンメニューを表示するかどうかを指定し、結果としてカレントテーブルとカレントマスターテーブルの変更を許可または禁止します。デフォルトでは、この値は0（変更不可）に設定されます。デフォルトとして、この値は“False（偽）”（検索ツールやマスターテーブルへアクセスしない）に設定されています。

引数＜*＞を指定しない場合、レポートの選択後に印刷ダイアログが表示されます。この引数を指定すると、これらのダイアログボックスは表示されず、レポートが印刷されます。

「クイックレポート」エディタを使用すると、ユーザは独自のレポートを作成することができます。「クイックレポート」エディタが表示された際、「マスターテーブル」ドロップダウンメニューと「新規クエリ」ボタンが存在しない可能性があることを除けば、ユーザは「ユーザ」モードでエディタが表示された場合と同じ状況で作業を行うことができます。つまり、ユーザはエディタに関するすべての制御を行えます。

「クイックレポート」エディタが起動されない場合に、レポートが印刷されるとシステム変数OKには1が、印刷されない場合（つまり、ユーザが印刷ダイアログで「キャンセル」をクリックした場合）には0（ゼロ）が代入されます。

4D Server：このコマンドは、ストアドプロシージャ内でのみ4D Server上で実行することができます。この状況では、次の制約があります。

・サーバマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。これを実現するには、コマンドを呼び出す際に、引数“*”または“>”を指定する必要があります。

・4D Serverでは、ラベルエディタを表示させる構文は動作しません。これを行った場合、システム変数OKは0に設定されます。

・プリンタ関連の問題が発生しても（用紙切れ、プリンタ接続切断など）、エラーメッセージは生成されません。

例題

1. 次の例題では、ユーザが[People]テーブルを検索した後に、“詳細項目一覧”というレポートが自動的に印刷されます。

```
QUERY([People])
If(OK=1)
    QR REPORT([People];"詳細項目一覧";False;False;False*)
End if
```

2. 次の例題では、ユーザは[People]テーブルを検索してから印刷するレポートを選択することができます。

```
QUERY([People])
If(OK=1)
    QR REPORT([People];"";False;False;False)
End if
```

3. 次の例題では、ユーザが[People]テーブルを検索した後に「クイックレポート」エディタが表示され、任意のレポートの設計、保存、ロード、印刷を行うことができます。

```
QUERY([People])
    If(OK=1)
    QR REPORT([People];Char(1);False;False;False)
End if
```

4. **SET FIELD RELATION**コマンドの例題を参照してください。

参照

PRINT LABEL、PRINT SELECTION

QR REPORT TO BLOB

QR REPORT TO BLOB(エリア;blob)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
blob	BLOB←		クイックレポートを格納するBlob

説明

QR REPORT TO BLOBコマンドは、＜エリア＞に渡された参照番号のレポートをBLOB（変数またはフィールド）に格納します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な＜blob＞引数を渡した場合、エラー番号64が生成されます（“ピクチャタイプのフィールドか変数が必要です。”）。

例題

次のコードは、“MyArea”に保存されたクイックレポートをBLOBフィールドに代入します。

```
QR REPORT TO BLOB(MyArea;[Table 1]Field4)
```

参照

QR BLOB TO REPORT

QR BLOB TO REPORT

QR BLOB TO REPORT(エリア;blob)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
blob	BLOB	→	レポートを格納するBLOB

説明

QR BLOB TO REPORTコマンドは、＜blob＞に格納されたレポートを＜エリア＞に渡されたクイックレポートエリアに配置します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な＜blob＞引数を渡した場合、エラー番号-9852が生成されます。

例題

1. 次のコードを使用し、データベースストラクチャと同じ階層にある“report.4qr”という名称のレポートファイルをMyAreaに表示することができます。このレポートファイルは、4D 2003で作成されている必要はなく、以前のバージョンのものであっても構いません。

```
C_BLOB($doc)
C_LONGINT(MyArea)
DOCUMENT TO BLOB("report.4qr";$doc)
QR BLOB TO REPORT(MyArea;$doc)
```

2. 次の例は、「Field4」に格納されたクイックレポートを取り出し、 MyAreaに表示します。

```
QR BLOB TO REPORT(MyArea;[Table 1]Field4)
```

参照

QR REPORT TO BLOB

QR New offscreen area

QR New offscreen area → 倍長整数

引数	タイプ		説明
戻り値	倍長整数	←	作成するエリアの参照番号

説明

QR New offscreen area関数は、新しくクイックレポートオフスクリーンエリアを作成し、その参照番号を返します。

参照

QR DELETE OFFSCREEN AREA

QR DELETE OFFSCREEN AREA

QR DELETE OFFSCREEN AREA(エリア)

引数	タイプ		説明
エリア	倍長整数	→	削除するエリアの参照番号

説明

QR DELETE OFFSCREEN AREAコマンドは、引数として渡された参照番号のクイックレポートオフスクリーンエリアをメモリから削除します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR New offscreen area

QR SET DESTINATION

QR SET DESTINATION(エリア; 出力先; {詳細})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
出力先	倍長整数	→	レポートの出力タイプ
詳細	文字列 ¦変数	→	出力タイプに関連する詳細

説明

QR SET DESTINATIONコマンドは、＜エリア＞に渡された参照番号のエリア用のレポート出力タイプを設定します。

次の表は、＜出力先＞および＜詳細＞の両引数へ渡すことができる値を示しています。

出力先	引数（値）	詳細
プリンタ	qr printer(1)	使用しません
テキストファイル	qr text file(2)	ファイルへのパス名
4D View	qr 4D View area(3)	使用しません
4D Chart	qr 4D Chart area(4)	使用しません
HTMLファイル	qr HTML file(5)	HTMLファイルへのパス名

■ テキストファイル(2)

ファイルのパス名に空の文字列を渡した場合には「ファイル保存」ダイアログが表示されます。それ以外の場合、ファイルはパス名で指定された場所に保存されます。

デフォルトのフィールド区切りはタブ記号（ASCII 9）です。また、デフォルトのレコード区切りはキャリッジリターン（ASCII 13）です。区切り文字用の2つのシステム変数（FldDelimitとRecDelimit）に値を指定して、これらのデフォルト値を変更することができます。Windowsにおいて、FldDelimitが13である場合、このキャリッジリターンの後にラインフィード（char 10）が付加されます。これらの変数は、**IMPORT TEXT**などの他のコマンドでも使用される点に注意してください。「クイックレポート」エディタのためにこれらの変数の値を変更すると、その変更はアプリケーションのあらゆる箇所に反映されます。

■ 4D View(3)

4D Viewがユーザから使用可能である場合、4D View外部ウインドウが作成され、クイックレポートエリアの現在の設定結果が納められます。

■ 4D Chart(4)

4D Chart外部ウインドウが作成され、クイックレポートエリアの現在の設定結果が納められます。この変換の実行方法については、「クイックレポート」エディタドキュメントの「ユーザリファレンス」の節を参照してください。

■ HTMLファイル(5)

SET HTML TEMPLATEで設定されたテンプレートを使用して、HTMLファイルが作成されます。この変換の実行方法については、「クイックレポート」エディタドキュメントの「ユーザリファレンス」の節を参照してください。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な出力先の値を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、出力先を“Mydoc.txt”というテキストファイルに設定し、クイックレポートを実行します。

```
QR SET DESTINATION(MyArea; 2; "MyDoc.txt")
QR RUN(MyArea)
```

参照

QR GET DESTINATION

QR GET DESTINATION

QR GET DESTINATION(エリア; 出力先; {詳細})

数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
出力先	倍長整数	←	レポートの出力タイプ
詳細	文字列 ¦変数	←	出力タイプに関連する詳細

説明

QR GET DESTINATIONコマンドは、＜エリア＞に渡した参照番号のエリア用のレポート出力タイプを取得します。

引数＜出力先＞の値をQR Destinationテーマの定数と比較することができます。

次の表は、＜出力先＞および＜詳細＞の両引数から取得できる値を示しています。

出力先	引数（値）	詳細
プリンタQR Printer(1)	使用できません	
テキストファイル	qr text file(2)	ファイルへのパス名
4D View	qr 4D View area(3)	使用できません
4D Chart	qr 4D Chart area(4)	使用できません
HTMLファイル	qr HTML file(5)	HTMLファイルへのパス名

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET DESTINATION

QR SET DOCUMENT PROPERTY

QR SET DOCUMENT PROPERTY(エリア;プロパティ;値)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
プロパティ	倍長整数	→	1=印刷ダイアログ 2=ドキュメントの単位
値	倍長整数	→	プロパティの値

説明

QR SET DOCUMENT PROPERTYコマンドを使用し、印刷ダイアログの表示、またはドキュメントで使用する単位の指定を行うことができます。

＜プロパティ＞には、**QR Document properties**定数テーマ内にある次の定数を渡すことができます。

定数	値
qr printing dialog	1
qr unit	2

■ ＜プロパティ＞が1の場合、コマンドは印刷ダイアログの表示に対して適用されます。

- ■ 値が1の場合、印刷の前に印刷ダイアログが表示されます。
- ■ 値が0の場合、印刷の前に印刷ダイアログが表示されません。

■ ＜プロパティ＞が2の場合、コマンドはドキュメントの単位に対して適用されます。

- ■ 値が0の場合、ドキュメントの単位はポイントです。
- ■ 値が1の場合、ドキュメントの単位はセンチです。
- ■ 値が2の場合、ドキュメントの単位はインチです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なプロパティの値を渡した場合、エラー番号-9852が生成されます。

参照

QR Get document property

QR Get document property

QR Get document property(エリア; プロパティ) → 倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
プロパティ	倍長整数	→	1=印刷ダイアログ 2=ドキュメントの単位
戻り値	倍長整数	←	プロパティの値

説明

QR Get document property関数を使用し、印刷ダイアログの表示ステータス、またはルーラーに用いる単位を取得することができます。

＜プロパティ＞には、**QR Document properties**定数テーマ内にある次の定数を渡すことができます。

定数	値
qr printing dialog	1
qr unit	2

■ ＜プロパティ＞が1の場合、コマンドは印刷ダイアログの表示ステータスに対して適用されます。

- ■ 戻り値が1の場合、印刷の前に印刷ダイアログが表示されます。
- ■ 戻り値が0の場合、印刷の前に印刷ダイアログが表示されません。

デフォルト値は1です。

■ ＜プロパティ＞が2の場合、コマンドはドキュメントの単位に対して適用されます。

- ■ 戻り値が0の場合、ドキュメントの単位はポイントです。
- ■ 戻り値が1の場合、ドキュメントの単位はセンチです。
- ■ 戻り値が2の場合、ドキュメントの単位はインチです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なプロパティの値を渡した場合、エラー番号-9852が生成されます。

参照

QR SET DOCUMENT PROPERTY

QR SET REPORT KIND

QR SET REPORT KIND(エリア;タイプ)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
タイプ	倍長整数	→	レポートのタイプ

説明

QR SET REPORT KINDコマンドは、＜エリア＞に渡した参照番号のエリアのレポートタイプを設定します。

■ タイプが1の場合、レポートはリストタイプです。

■ タイプが2の場合、レポートはクロスタブタイプです。

また、QR Report Typesテーマのqr List Reportおよびqr Cross Report定数を使用することもできます。

定数	値
qr list report	1
qr cross report	2

既存の現行レポートに新しいタイプを設定すると、以前の設定が削除されて新しく空のレポートが作成され、新たに設定を行うことができます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なタイプの値を渡した場合、エラー番号-9852が生成されます。

参照

QR Get report kind

QR Get report kind

QR Get report kind(エリア) → 倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
戻り値	倍長整数	←	レポートのタイプ

説明

QR Get report kind関数は、＜エリア＞に渡した参照番号のエリアのレポートタイプを取得します。

■ 戻り値が1の場合、レポートはリストタイプです。

■ 戻り値が2の場合、レポートはクロスタブタイプです。

また、この関数の戻り値をQR Report Typesテーマの定数と比較することもできます。

定数	値
qr list report	1
qr cross report	2

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET REPORT KIND

QR SET AREA PROPERTY

QR SET AREA PROPERTY(エリア;プロパティ;値)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
プロパティ	倍長整数	→	指定するパレット
引数	タイプ		説明
値	倍長整数	→	0=表示 1=非表示

説明

QR SET AREA PROPERTYコマンドを使用し、＜プロパティ＞に渡した参照番号のインタフェース要素（パレットやメニューバー）を表示、または非表示に設定することができます。

メニューバーおよびツールバーには1から6（上から下）までの番号が振られ、値 7はコンテキストメニュー専用です。

QR Area Propertiesテーマの定数を使用して、インタフェース要素を指定することができます。

プロパティ	定数	説明
Menu bar	qr view menubar(1)	メニューバーの表示ステータス（表示=1、非表示=0）
標準パレット	qr view standard toolbar(2)	標準パレットの表示ステータス（表示=1、非表示=0）
スタイルパレット	qr view style toolbar(3)	スタイルパレットの表示ステータス（表示=1、非表示=0）
関数パレット	qr view operators toolbar(4)	関数パレットの表示ステータス（表示=1、非表示=0）
カラムパレット	qr view column toolbar(5)	カラムパレットの表示ステータス（表示=1、非表示=0）
背景色パレット	qr view color toolbar(6)	背景色パレットの表示ステータス（表示=1、非表示=0）
コンテキストメニュー	qr view contextual menus(7)	コンテキストメニューの表示ステータス（表示=1、非表示=0）

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な＜プロパティ＞引数を渡した場合、エラー番号-9852が生成されます。

参照

QR Get area property

QR Get area property

QR Get area property(エリア; プロパティ)→　　倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
プロパティ	倍長整数	→	指定されたパレット
戻り値	倍長整数	←	0= 表示 1= 非表示

説明

QR Get area property関数は、＜プロパティ＞に渡されたインタフェース要素（パレットまたはメニューバー）が表示されていなければ0を、表示されていれば1を返します。

メニューバーとツールレットには1から6まで（上から下へ）の番号が付けられ、値7はコンテキストメニュー専用です。

QR Area Propertiesテーマの定数を使用し、インタフェース項目を指定することができます。

プロパティ	定数	説明
Menu bar	qr view menubar(1)	メニューバーの表示ステータス（表示=1、非表示=0）
Standard palette	qr view standard toolbar(2)	標準パレットの表示ステータス（表示=1、非表示=0）
Style palette	qr view style toolbar(3)	スタイルパレットの表示ステータス（表示=1、非表示=0）
Computation palette	qr view operators toolbar(4)	関数パレットの表示ステータス（表示=1、非表示=0）
Column palette	qr view column toolbar(5)	カラムパレットの表示ステータス（表示=1、非表示=0）
Color palett	qr view color toolbar(6)	カラーパレットの表示ステータス（表示=1、非表示=0）
Contextual menu	qr view contex-tual menus(7)	コンテキストメニューパレットの表示ステータス（表示=1、非表示=0）

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なプロパティ引数を渡した場合、エラー番号-9852が生成されます。

参照

QR SET AREA PROPERTY

QR SET REPORT TABLE

QR SET REPORT TABLE(エリア; テーブル)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
テーブル	倍長整数	→	テーブル番号

説明

QR SET REPORT TABLEコマンドは、＜テーブル＞に渡した番号のテーブルを＜エリア＞に渡した参照番号のレポートエリアのカレントテーブルに設定します。

レポートにテーブルを割り当てることが大切な理由は、レポートエディタではそのテーブルのカレントセレクションを使用し、必要に応じてデータの表示や計算の実行、リレートの設定を行うためです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なテーブル番号を渡した場合、エラー番号-9852が生成されます。

参照

QR Get report table

QR Get report table

QR Get report table(エリア)→倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
戻り値	倍長整数	←	テーブル番号

説明

QR Get report table関数は、＜エリア＞に渡した参照番号のレポートエリア用のカレントテーブルを返します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET REPORT TABLE

QR SET TEXT PROPERTY

QR SET TEXT PROPERTY(エリア;カラム番号; 行番号; プロパティ; 値)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
行番号	倍長整数	→	行番号
プロパティ	倍長整数	→	セルのプロパティを表わす値
値	倍長整数	→	選択したプロパティの値

説明

QR SET TEXT PROPERTYコマンドを使用し、＜カラム番号＞と＜行番号＞で指定されたセルのテキスト属性を設定することができます。

■ ＜エリア＞にはクイックレポートエリアの参照番号を渡します。

■ ＜カラム番号＞にはセルのカラム番号を渡します。

■ ＜行番号＞にはセルの行の参照番号を渡します。

- ■ ＜行番号＞が-1の場合、カラムタイトルを示します。
- ■ ＜行番号＞が-2の場合、詳細エリアを示します。
- ■ ＜行番号＞が-3の場合、カラムの総合計を示します。
- ■ ＜行番号＞が-4の場合、ページのヘッダを示します。
- ■ ＜行番号＞が-5の場合、ページのフッタを示します。

QR Rows for Propertiesテーマの定数を使用し、行アイテムを指定することができます。定数の値は、qr title（-1）、qr detail（-2）、qr grand total（-3）、qr header（-4）、qr footer（-5）です。

注：＜行番号＞に-4または-5を渡す場合、使用されなくても＜カラム番号＞を渡す必要があります。

- ■ ＜行番号＞が正の値である場合、対応する小計（ブレークレベル）を示します。

注：クロスタブモードでは、行の値が常に正数であることを除き、原則は同じです。

＜プロパティ＞には、割り当てるテキスト属性の値または定数を渡します。QR Text Propertiesテーマの定数を使用し、次の値を設定することができます。

定数（値）	設定する値
qr font(1)	Font numberを用いて返されるフォント番号
qr font size(2)	ポイント単位で表わされたフォントサイズ（9から255）
qr bold(3)	太字スタイル属性（0または1）
qr italic(4)	斜体スタイル属性（0または1）
qr underline(5)	フォントの下線スタイル属性（0または1）
qr text color(6)	フォントのカラー番号属性（倍長整数）
qr justification(7)	テキスト整列属性 （デフォルトは0、左揃えは1、中央揃えは2、右揃えは3）
qr background color(8)	背景色の番号
qr alternate background color(9)	代替背景色の番号

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効な行番号を渡した場合、エラー番号-9853が生成されます。

無効なプロパティ番号を渡した場合、エラー番号-9854が生成されます。

例題

このメソッドは、最初のカラムのタイトルに対して複数の属性を定義します。

```
`次のコールはフォントを“Times”に設定する。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr font;Font number("Times"))
`フォントサイズを10 ポイントに設定する。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr font size;10)
`“太字”属性を割り当てる。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr bold;1)
`“斜体”属性を割り当てる。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr italic;1)
`“下線”属性を割り当てる。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr underline;1)
`黄緑色を割り当てる。
⇒ QR SET TEXT PROPERTY(qr_area;1;-1;qr text color;0x0000FF00)
```

参照

QR Get text property

QR Get text property

QR Get text property(エリア;カラム番号; 行番号; プロパティ) → 倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
行番号	倍長整数	→	ブレーク番号
プロパティ	倍長整数	→	セルのプロパティを表わす値
戻り値	倍長整数	←	選択したプロパティの値

説明

QR Get text property関数は、＜カラム番号＞と＜行番号＞で指定されたセルのテキスト属性のプロパティ値を返します。

■ ＜エリア＞にはクイックレポートエリアの参照番号を渡します。

■ ＜カラム番号＞にはセルのカラム番号を渡します。

■ ＜行番号＞にはセルの行の参照番号を渡します。

- ■ ＜行番号＞が-1の場合、カラムタイトルを示します。
- ■ ＜行番号＞が-2の場合、詳細エリアを示します。
- ■ ＜行番号＞が-3の場合、カラムの総合計を示します。
- ■ ＜行番号＞が-4の場合、ページのヘッダを示します。
- ■ ＜行番号＞が-5の場合、ページのフッタを示します。

注：＜行番号＞に-4または-5を渡す場合、使用されなくても＜カラム番号＞を渡す必要があります。

- ■ ＜行番号＞が正の値である場合、対応する小計（ブレークレベル）を示します。

注：クロスタブモードでは、行の値が常に正数であることを除き、原則は同じです。

＜プロパティ＞には、取得するテキスト属性の値または定数を渡します。QR Text Propertiesテーマの定数を使用し、次の値を設定することができます。

定数（値）	設定する値
qr font(1)	Font number を用いて返されるフォント番号
qr font size(2)	ポイント単位で表わされたフォントサイズ（9 から255）
qr bold(3)	太字スタイル属性（0 または1）
qr italic(4)	斜体スタイル属性（0 または1）
qr underline(5)	フォントの下線スタイル属性（0 または1）
qr text color(6)	フォントのカラー番号属性（倍長整数）
qr justification(7)	テキスト整列属性 （デフォルトは0、左揃えは1、中央揃えは2、右揃えは3）
qr background color(8)	背景色
qr alternate background color(9)	代替背景色

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効な行番号を渡した場合、エラー番号-9853が生成されます。

無効なプロパティ番号を渡した場合、エラー番号-9854が生成されます。

参照

QR SET TEXT PROPERTY

QR RUN

QR RUN(エリア)

引数	タイプ		説明
エリア	倍長整数	→	実行するエリアの参照番号

説明

QR RUNコマンドは、出力先を含め、クイックレポートの現在の設定を使用して、引数として渡された参照番号のレポートエリアを実行します。

属するエリアのテーブルでテーブルレポートが実行されます。オフスクリーンエリアが選ばれている場合、**QR SET REPORT TABLE**コマンドで使用するテーブルを指定する必要があります。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

QR EXECUTE COMMAND

QR EXECUTE COMMAND(エリア;コマンド)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
コマンド	倍長整数	→	実行するメニューコマンド

説明

QR EXECUTE COMMANDコマンドは、＜コマンド＞に渡された参照番号のメニューコマンド、またはツールバーボタンを実行します。このコマンドの最も一般的な使い方としては、ユーザがコマンドを選択し、コード内で**QR ON COMMAND**コマンドを用いてそれを検知した後に、そのコマンドを実行することです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なコマンド番号を渡した場合、エラー番号-9852が生成されます。

参照

QR Get command status、QR ON COMMAND

QR Get command status

QR Get command status(エリア; コマンド{; 値}) →倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
コマンド	倍長整数	→	コマンド番号
値	テキスト¦倍長整数	←	選択したサブアイテムの値
戻り値	倍長整数	←	コマンドのステータス

説明

QR Get command status関数は、そのコマンドが使用不可であれば0を、使用可能であれば1を返します。

＜値＞は、選択されたサブアイテムがあれば、その値を示します。例えば、選択されたコマンドがフォントメニュー（1000）であり、選択されたフォントが“Arial”である場合、＜値＞には“Arial”が返されます。また、選択されたコマンドが色メニュー（1002,、1003、または1004）である場合、＜値＞にはカラー番号が返されます。

このコマンドは、次の2つの状況で使用することができます。

あるコマンドが使用可であるか使用不可であるかを調べる単純な判定文として。

QR ON COMMANDコマンドでインストールされたメソッドにおいてこのコマンドを使用すると、選択されたサブアイテムを知ることができます。そのメソッドでは、$1がエリアの参照番号となり、$2がコマンド番号となります。

コマンドおいて、値もしくは**QR Commands**定数テーマのいずれかの定数を渡すことができます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なコマンド番号を渡した場合、エラー番号-9852が生成されます。

参照

QR EXECUTE COMMAND、QR ON COMMAND

QR ON COMMAND

QR ON COMMAND(エリア;4D代替メソッド)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
4D代替メソッド	文字列	→	代替メソッドの名前

説明

QR ON COMMANDコマンドは、ユーザによるクイックレポートコマンドの起動時、またはメニューコマンドの選択時、あるいはボタンのクリック時に、＜4D代替メソッド＞に渡されたメソッドを実行します。＜エリア＞が0（ゼロ）の場合、データベースが閉じられるか、**QR ON COMMAND**(0;"")という構文で**QR ON COMMAND**コマンドが呼び出されるまで、＜4D代替メソッド＞が各クイックレポートエリアに適用されます。

＜4D代替メソッド＞は次の2つの引数を受け取ります。

$1はエリアの参照番号（倍長整数）です。

$2は選択されたコマンドのコマンド番号（倍長整数）です。

注：データベースをコンパイルする予定がある場合、$1と$2を使用しない場合でも、それぞれを倍長整数として定義する必要があります。

最初のコマンドを実行させたい場合には、呼び出されるメソッドに次の命令を記述する必要があります：**QR EXECUTE COMMAND**($1;$2)

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR EXECUTE COMMAND、QR Get command status

QR Find column

QR Find column(エリア; 表現式) → 倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
表現式	文字列 ¦ポインタ	→	カラムオブジェクト
戻り値	倍長整数	←	カラムの番号

説明

QR Find column関数は、引数で渡された表現式に一致する内容を持つ最初のカラムの番号を返します。

＜表現式＞には文字列またはポインタを渡します。

対象となるカラムが見つからない場合、-1が返されます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードは、[G.NQR Tests]Quarterフィールドが格納されているカラム番号を取得し、そのカラムを削除します。

```
$NumColumn:=QR Find column(MyArea;->[G.NQR Tests]Quarter)
```

または

```
$NumColumn:=QR Find column(MyArea; "[G.NQR Tests]Quarter")
```

このコードに続けて次のコードを実行します。

```
If($NumColumn#-1)
    QR DELETE COLUMN(MyArea; $NumColumn)
End if
```

QR SET SELECTION

QR SET SELECTION(エリア;左;上;右;下)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
左	倍長整数	→	左側境界
上	倍長整数	→	上側境界
右	倍長整数	→	右側境界
下	倍長整数	→	下側境界

説明

QR SET SELECTIONコマンドを使用すると、マウスクリックを実行した場合と同様に、セルや行、カラム、またはエリア全体を反転表示（選択）することができます。また、現在選択されている範囲を解除することもできます。

＜左＞には、左側境界となるカラムの番号を指定します。＜左＞が0の場合、行全体が選択されます。

＜上＞には、上側境界となる行番号を指定します。＜上＞が0の場合、カラム全体が選択されます。

＜右＞には、右側境界となるカラムの番号を指定します。

＜下＞には、下側境界となる行番号を指定します。

注：＜左＞と＜上＞の両方が0の場合、エリア全体が反転表示（選択）されます。

選択範囲を設定したくない場合、＜左＞、＜上＞、＜右＞、＜下＞に-1を渡します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR GET SELECTION

QR GET SELECTION

QR GET SELECTION(エリア; 左; 上{; 右}{; 下})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
左	倍長整数	←	左側境界
上	倍長整数	←	上側境界
右	倍長整数	←	右側境界
下	倍長整数	←	下側境界

説明

QR GET SELECTIONコマンドは、選択されたセルの座標を返します。

＜左＞には、選択範囲の左側境界となるカラムの番号が返されます。＜左＞が0の場合、行全体が選択されています。

＜上＞には、選択範囲の上側境界となる行番号が返されます。＜上＞が0の場合、カラム全体が選択されています。

注：＜左＞と＜上＞の両方が0の場合、エリア全体が反転表示（選択）されています。

＜右＞には、選択範囲の右側境界となるカラムの番号が返されます。

＜下＞には、選択範囲の下側境界となる行番号が返されます。

注：何も選択されていない場合、＜左＞、＜上＞、＜右＞、＜下＞には-1が代入されます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET SELECTION

QR SET HEADER AND FOOTER

QR SET HEADER AND FOOTER(エリア; セレクタ; 左タイトル; 中央タイトル; 右タイトル; 高さ{; ピクチャ}{; ピクチャ整列})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
セレクタ	倍長整数	→	1=ヘッダ 2=フッタ
左タイトル	文字列	→	左側に表示されるテキスト
中央タイトル	文字列	→	中央に表示されるテキスト
右タイトル	文字列	→	右側に表示されるテキスト
高さ	実数	→	ヘッダまたはフッタの高さ
ピクチャ	ピクチャ	→	表示するピクチャ
ピクチャ整列	倍長整数	→	ピクチャの整列属性

説明

QR GET HEADER AND FOOTERコマンドを使用し、ヘッダまたはフッタの内容とサイズを設定することができます。

＜セレクタ＞を使用して、ヘッダまたはフッタを選択します。＜セレクタ＞に1を指定すると、ヘッダに適用されます。また、＜セレクタ＞に2を指定すると、フッタに適用されます。

＜左タイトル＞、＜中央タイトル＞、＜右タイトル＞にはそれぞれ、左側、中央、右側にあるヘッダまたはフッタの値を指定します。

＜高さ＞には、そのレポートに対して選択した単位で表わされたヘッダまたはフッタの高さを指定します。

＜ピクチャ＞には、ヘッダまたはフッタに表示されるピクチャを指定します。

＜ピクチャ整列＞には、＜ピクチャ＞に渡されたピクチャの整列属性を指定します。

■ ＜ピクチャ整列＞が0の場合、そのピクチャは左揃えです。

■ ＜ピクチャ整列＞が1の場合、そのピクチャは中央揃えです。

■ ＜ピクチャ整列＞が2の場合、そのピクチャは右揃えです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なセレクタ値を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、“MyArea”のクイックレポートのヘッダタイトルとして“中央タイトル”を設定し、ヘッダの高さを200ピクセルに設定します。

```
QR SET HEADER AND FOOTER(MyArea; 1; ""; "中央タイトル";""; 200)
```

参照

QR GET HEADER AND FOOTER

QR GET HEADER AND FOOTER

QR GET HEADER AND FOOTER(エリア; セレクタ; 左タイトル; 中央タイトル; 右タイトル; 高さ{; ピクチャ}{; ピクチャ整列})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
セレクタ	倍長整数	→	1= ヘッダ 2= フッタ
左タイトル	文字列	←	左側に表示されるテキスト
中央タイトル	文字列	←	中央に表示されるテキスト
右タイトル	文字列	←	右側に表示されるテキスト
高さ	実数	←	ヘッダまたはフッタの高さ
ピクチャ	ピクチャ	←	表示するピクチャ
ピクチャ整列	倍長整数	←	ピクチャの整列属性

説明

QR GET HEADER AND FOOTERコマンドを使用し、ヘッダまたはフッタの内容とサイズを取得することができます。

＜セレクタ＞を使用して、ヘッダまたはフッタを選択します。＜セレクタ＞に1を指定すると、ヘッダ情報を取得できます。＜セレクタ＞に2を指定すると、フッタ情報を取得できます。

＜左タイトル＞、＜中央タイトル＞、＜右タイトル＞にはそれぞれ、左側、中央、右側にあるヘッダまたはフッタの値が返されます。

＜高さ＞には、そのレポートに対して選択した単位で表わされたヘッダまたはフッタの高さが返されます。

＜ピクチャ＞には、ヘッダまたはフッタに表示されるピクチャが返されます。

＜ピクチャ整列＞には、ヘッダまたはフッタに表示されるピクチャの整列属性が返されます。

■ ＜ピクチャ整列＞が0の場合、そのピクチャは左揃えです。

■ ＜ピクチャ整列＞が1の場合、そのピクチャは中央揃えです。

■ ＜ピクチャ整列＞が2の場合、そのピクチャは右揃えです。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なセレクタ値を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、ヘッダタイトルの値とヘッダサイズを取得し、それを警告として表示します。

```
⇒    QR GET HEADER AND FOOTER(MyArea;1;$LeftText;$CenterText;$RightText;
                                                                $height)
     Case of
          ¥($LeftText # "")
               ALERT("左側タイトル："+Char(34)+$LeftText+Char(34))
          ¥($CenterText # "")
               ALERT("中央タイトル："+Char(34)+$CenterText+Char(34))
          ¥($RightText # "")
               ALERT("右側タイトル："+Char(34)+$RightText+Char(34))
          Else
               ALERT("このレポートにはヘッダタイトルはありません.")
     End case
     ALERT("ヘッダの高さ："+string($height))
```

参照

QR SET HEADER AND FOOTER

QR SET BORDERS

QR SET BORDERS(エリア ; カラム ; 行; 罫線; 線{ ;色})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム	倍長整数	→	カラム番号
行	倍長整数	→	行番号
罫線	倍長整数	→	罫線の合成値
引数	タイプ		説明
線	倍長整数	→	線の太さ
色	倍長整数	→	罫線の色

説明

QR SET BORDERSコマンドを使用し、指定したセルの罫線スタイルを設定することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜カラム＞にはセルのカラム番号を渡します。

＜行＞にはセルの行番号を渡します。

■ ＜行＞が-1の場合、レポートのタイトル行に適用されます。

■ ＜行＞が-2の場合、レポートの詳細行に適用されます。

■ ＜行＞が-3の場合、レポートの総計行に適用されます。

■ ＜行＞が正の整数の場合、その数値は適用される小計（ブレーク）レベルを示します。

QR Rows for Propertiesテーマの定数を使用して、行項目を指定することができます（qr title=-1、qr detail=-2、qr grand total=-3）。

＜罫線＞には、適用するセルの罫線を示す合成値を渡します。

■ 1の場合、左側の罫線を示します。

■ 2の場合、上側の罫線を示します。

■ 4の場合、右側の罫線を示します。

■ 8の場合、下側の罫線を示します。

QR Bordersテーマの定数を使用して、罫線項目を指定することができます。

例えば、＜罫線＞に値5を渡すと、右側と左側の罫線に適用されます。

＜線＞にはその線の太さを渡します。

■ 0の場合、線は表示されません。

■ 1の場合、1/4ポイントの太さを示します。

■ 2の場合、1/2ポイントの太さを示します。

■ 3の場合、1ポイントの太さを示します。

■ 4の場合、2ポイントの太さを示します。

＜色＞はその線の色を渡します。

■ ＜色＞が正の値である場合、特定の色を示します。

■ ＜色＞が0の場合、黒色を示します。

■ ＜色＞が-1の場合、色は変更されません。

注：デフォルトの色は黒です。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効な行番号を渡した場合、エラー番号-9853が生成されます。

無効な罫線引数を渡した場合、エラー番号-9854が生成されます。

参照

QR GET BORDERS

QR GET BORDERS

QR GET BORDERS(エリア; カラム; 行; 罫線; 線{;色})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム	倍長整数	→	カラム番号
行	倍長整数	→	行番号
罫線	倍長整数	←	罫線の値
線	倍長整数	←	線の太さ
色	倍長整数	←	罫線の色

説明

QR SET BORDERSコマンドを使用し、指定したセルの罫線のスタイルを取得することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜カラム＞にはセルのカラム番号を渡します。

＜行＞にはセルの行番号を渡します。

■ 行番号が-1の場合、レポートのタイトルに適用されます。

■ 行番号が-2の場合、レポートの詳細行に適用されます。

■ 行番号が-3の場合、レポートの総計行に適用されます。

■ 行番号が正数の場合、その数字は適用する詳細（ブレーク）レベルを指定します。

QR Rows for Propertiesテーマの定数を使用し、行アイテムを指定することができます（qr title= -1、qr detail=-2、qr grand total=-3）。

＜罫線＞は、適用するセルの罫線を示す値を表わします。

■ 1の場合、左側の罫線を示します。

■ 2の場合、上側の罫線を示します。

■ 4の場合、右側の罫線を示します。

■ 8の場合、下側の罫線を示します。

■ 16の場合、内部の縦罫線を示します。

■ 32の場合、内部の横罫線を示します。

QR Bordersテーマの定数を使用し、その罫線アイテムを指定することができます。

注：QR SET BORDERSコマンドとは異なり、QR GET BORDERSコマンドは累計値を受け付けません。

＜線＞はその線の太さを表わします。

■ 0の場合、線は表示されません。

■ 1の場合、1/4ポイントの太さを示します。

■ 2の場合、1/2ポイントの太さを示します。

■ 3の場合、1ポイントの太さを示します。

■ 4の場合、2ポイントの太さを示します。

＜色＞はその線の色を表わし、その線分に適用された色の値を返します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効な行番号を渡した場合、エラー番号-9853が生成されます。

無効な罫線引数を渡した場合、エラー番号-9854が生成されます。

参照

QR SET BORDERS

QR SET INFO COLUMN

QR SET INFO COLUMN(エリア; カラム番号; タイトル; オブジェクト; 非表示; サイズ; 同一値; 表示フォーマット)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
タイトル	文字列	→	カラムのタイトル
オブジェクト	フィールド名¦変数	→	カラムに割り当てられるオブジェクト
非表示	倍長整数	→	0=表示 1=非表示
サイズ	倍長整数	→	カラムサイズ
同一値	倍長整数	→	0=同一値を印刷しない 1=同一値も印刷する
表示フォーマット	文字列	→	そのデータの表示フォーマット

説明

リストモード

QR SET INFO COLUMNコマンドを使用して、既存のカラムに関するパラメータを設定することができます。

＜エリア＞には、クイックレポートエリアの参照番号を指定します。

＜カラム番号＞には、修正するカラムの番号を指定します。

＜タイトル＞には、カラムのヘッダに表示されるタイトルを指定します。

＜オブジェクト＞には、そのカラムの実際のオブジェクト（変数、フィールド名、またはフォーミュラ）を指定します。

＜非表示＞には、カラムを表示するか、あるいは非表示にするかを指定します。

■ ＜非表示＞に1を指定すると、カラムは非表示に設定されます。

■ ＜非表示＞に0を指定すると、カラムは表示に設定されます。

＜サイズ＞には、カラムに割り当てるサイズをピクセル単位で指定します。-1（マイナス1）は、カラムのサイズを自動設定にします。

＜同一値＞には、データの反復印刷の状態を指定します。例えば、あるレコードから別のレコードまでの間にフィールドや変数の値が変わらない場合に、そのデータを繰り返し印刷するか、または印刷しないかを指定します。

■ ＜同一値＞が0の場合、値は繰り返し印刷されません。

■ ＜同一値＞が1の場合、値は繰り返し印刷されます。

＜表示フォーマット＞には、表示フォーマットを指定します。この表示フォーマットは、表示されるデータに対応した4Dフォーマットです。

次のコードは、カラム番号1のタイトルとして“タイトル”をセットし、カラム自体の内容としてField2をセットし、幅150ピクセルでカラムを表示してフォーマットを“###.##”に設定します。

QR SET INFO COLUMN(area; 1; "タイトル"; "[Table 1]Field2"; 0;150;0;"###,##")

クロスタブモード

QR SET INFO COLUMNコマンドを使用して、同様のパラメータを設定できますが、設定しようとするパラメータにより、適用するエリアの参照番号が異なります。第一に、このコマンドをクロスタブモードで用いた場合、引数＜タイトル＞、＜非表示＞、＜同一値＞は使用されません。設定したい値がカラムサイズ、データソース、表示フォーマットのいずれであるかによって、カラムに使用する値が変わります。

■ カラムサイズ

これは“視覚的”な属性であり、下図のようにカラムは左から右へと番号が付けられています。



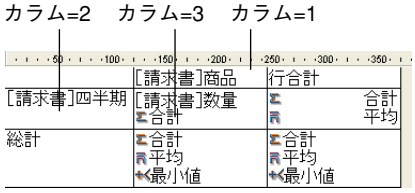
次のコードは、クロスタブレポートのすべてのカラムに対してサイズを自動に設定し、その他の要素は以前のまま変更しません。

```
For($i;1;3)
    QR GET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
    QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

お分かりのように、カラムサイズだけを変更したいため、**QR GET INFO COLUMN**を使用してカラムのプロパティを取得し、それを**QR SET INFO COLUMN**に渡してカラムサイズ以外の項目は変更していません。

■ データソース（オブジェクト）と表示フォーマット

この場合、カラム番号は次の図のように作用します。

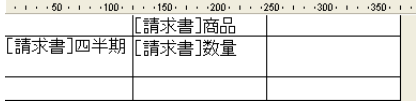


QR SET INFO COLUMNコマンドを使用しても、すべてのセルに対応できないことにお気付きでしょう。上図の中で番号が付けられていないセルに関しては、**QR SET TOTALS DATA**コマンドを用いて対処します。

次のコードは、基本的なクロスタブレポートの作成に必要となる3つのセルに対し、データソースを割り当てます。

```
QR SET REPORT TABLE(qr_area;Table(->[Invoices]))
ALL RECORDS([Invoices])
QR SET REPORT KIND(qr_area;2)
QR SET INFO COLUMN(qr_area;1;"";->[Invoices]Item;1;-1;1;"")
QR SET INFO COLUMN(qr_area;2;"";->[Invoices]Quarter;1;-1;1;"")
QR SET INFO COLUMN(qr_area;3;"";->[Invoices]Quantity;1;-1;1;"")
```

この結果、レポートエリアは次のようになります。



無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

参照

QR GET INFO COLUMN、QR Get info row、QR SET INFO ROW

QR GET INFO COLUMN

QR GET INFO COLUMN(エリア; カラム番号; タイトル; オブジェクト; 非表示; サイズ; 同一値; 表示フォーマット)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
タイトル	文字列	←	カラムのタイトル
オブジェクト	フィールド名¦変数	←	カラムに割り当てられたオブジェクト
非表示	倍長整数	←	0= 表示 1= 非表示
サイズ	倍長整数	←	カラムサイズ
同一値	倍長整数	←	0= 同一値を印刷しない 1= 同一値も印刷する
表示フォーマット	テキスト	←	そのデータの表示フォーマット

説明

リストモード

QR GET INFO COLUMNコマンドを使用して、既存のカラムに関するパラメータを取得することができます。

＜エリア＞には、クイックレポートエリアの参照番号を指定します。

＜カラム番号＞には、取得するカラムの番号を指定します。

＜タイトル＞には、カラムのヘッダに表示されるタイトルが返されます。

＜オブジェクト＞には、そのカラムの実際のオブジェクト（変数、フィールド名、またはフォーミュラ）が返されます。

＜非表示＞には、カラムを表示するか、あるいは非表示にするかを指定します。

■ ＜非表示＞に1を指定すると、カラムは非表示に設定されます。

■ ＜非表示＞に0を指定すると、カラムは表示に設定されます。

＜サイズ＞には、カラムサイズがピクセル単位で返されます。＜サイズ＞が -1の場合、サイズは自動に設定されています。

＜同一値＞には、データの反復印刷の状態が返されます。例えば、あるレコードから別のレコードまでの間にフィールドや変数の値が変わらない場合に、そのデータを繰り返し印刷する、または印刷しないという情報が返されます。＜同一値＞が0の場合、値は繰り返し印刷されません。＜同一値＞が1の場合、値は繰り返し印刷されます。

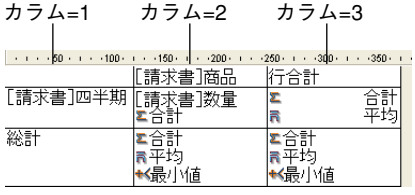
＜表示フォーマット＞には、表示フォーマットが返されます。この表示フォーマットは表示されるデータに対応する4Dフォーマットです。

クロスタブモード

QR GET INFO COLUMNコマンドを使用して、同様のパラメータを取得できますが、設定しようとするパラメータにより、適用するエリアの参照番号が異なります。第一に、このコマンドをクロスタブモードで使用した場合、引数＜タイトル＞、＜非表示＞、＜同一値＞は意味を持ちません。取得したい値がカラムサイズ、データソース、表示フォーマットのいずれであるかによって、カラムに使用する値が変わります。

■ カラムサイズ

これは“視覚的”な属性であり、下図のようにカラムは左から右へと番号が付けられています。



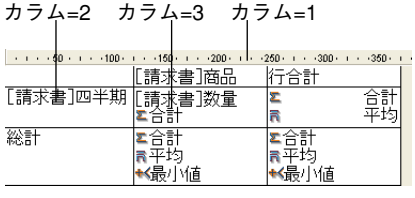
次のコードは、クロスタブレポートのすべてのカラムに対してサイズを自動に設定し、その他の要素は以前のまま変更しません。

```
For($i;1;3)
    QR GET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
    QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

お分かりのように、カラムサイズだけを変更したいため、**QR GET INFO COLUMN**を使用してカラムのプロパティを取得し、それを**QR SET INFO COLUMN**に渡してカラムサイズ以外の項目は変更していません。

■ データソース（オブジェクト）と表示フォーマット

この場合、カラムの番号は次の図のように付けられます。



無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

参照

QR Get info row、QR SET INFO COLUMN、QR SET INFO ROW

QR SET INFO ROW

QR SET INFO ROW(エリア; 行; 非表示)

引数	タイプ		説明
エリア	倍長整数	→	作成するエリアの参照番号
行	倍長整数	→	行を示す値
非表示	倍長整数	→	0=非表示 1=表示

説明

QR SET INFO ROWコマンドは、＜行＞に渡した参照番号の行を表示/非表示に設定します。

＜行＞には、コマンドが適用される行を指定します。

■ ＜行＞が-1の場合、レポートのタイトルに適用されます。

■ ＜行＞が-2の場合、レポートの詳細に適用されます。

■ ＜行＞が-3の場合、レポートの総計に適用されます。

■ ＜行＞が正の整数である場合、適用される小計（ブレーク）レベルを示します。

QR Rows for Propertiesテーマの定数を使用し、行アイテムを指定することができます（qr title=-1、qr detail=-2、qr grand total=-3）。

＜非表示＞には、行を表示するか、または非表示にするかを示す値を指定します。＜非表示＞が1の場合、行は非表示に設定され、0の場合、行は表示に設定されます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な行の値を渡した場合、エラー番号-9852が生成されます。

次のコードは、詳細行を隠します。

```
QR SET INFO ROW(area; -2; 1)
```

参照

QR GET INFO COLUMN、QR Get info row、QR SET INFO COLUMN

QR Get info row

QR Get info row(エリア; 行) → 倍長整数

引数	タイプ		説明
エリア	倍長整数	→	作成されたエリアの参照番号
行	倍長整数	→	行を示す値
戻り値	倍長整数	←	0= 表示 1= 非表示

説明

QR Get info row関数は、＜行＞に渡した参照番号の行に関する表示ステータスを取得します。

＜行＞には、コマンドが適用される行を指定します。

- ＜行＞が-1の場合、タイトルの表示属性を取得します。
- ＜行＞が-2の場合、詳細の表示属性を取得します。
- ＜行＞が-3の場合、総計の表示属性を取得します。
- ＜行＞が正の整数である場合、表示属性を取得する小計（ブレークレベル）を示します。

QR Rows for Propertiesテーマの定数を使用し、行アイテムを指定することができます（qr title=-1、qr detail=-2、qr grand total=-3）。

戻り値として、行が表示されるか、または非表示であるかを示す値が返されます。戻り値が1の場合、行は非表示に設定され、0の場合、行は表示に設定されています。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な行の値を渡した場合、エラー番号-9852が生成されます。

参照

QR GET INFO COLUMN、QR SET INFO COLUMN、QR SET INFO ROW

QR SET SORTS

QR SET SORTS(エリア;カラム配列;{並び替え順配列})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム配列	実数配列	→	カラム
並び替え順配列	実数配列	→	並び替え順

説明

QR SET SORTSコマンドを使用し、＜エリア＞に渡した参照番号のレポート内にある＜カラム＞の並び替え順を設定することができます。

■ カラム配列

この配列には、並び替え順を割り当てようとする各カラムのカラム番号を格納する必要があります。

■ 並び替え順配列

この配列の各要素には、＜カラム配列＞配列内の対応するカラムの並び替え順を格納しなくてはなりません。

- ■ 並び替え順配列{$i}が1の場合、並び替え順は昇順です。
- ■ 並び替え順配列{$i}が-1の場合、並び替え順は降順です。

クロスタブモード

クロスタブモードの場合、2つより多くの項目を配列に納めることはできません。並び替えられるのは、カラム（1）と行（2）だけです。データ（カラムと行の交差する場所に位置する）を並び替えることはできません。

次に示すコードは、クロスタブレポートにおいて行だけを並び替えます。

```
ARRAY REAL($aColumns;1)
$aColumns{1}:=2
ARRAY REAL($aOrders;1)
$aOrders{1}:=-1`行をアルファベット順に並び替える
⇒ QR SET SORTS(qr_area;$aColumns;$aOrders)
```

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR GET SORTS

QR GET SORTS

QR GET SORTS(エリア;カラム配列;{並び替え順配列})

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム配列	実数配列	←	カラム
並び替え順配列	実数配列	←	並び替え順

説明

QR GET SORTSコマンドは、次の2つの配列に値を代入します。

■ カラム配列
この配列には、並び替え順が設定されているすべてのカラムが格納されます。

■ 並び替え順配列
この配列の各要素には、対応するカラムの並び替え順が格納されます。

- 並び替え順配列{$i}が1の場合、並び替え順は昇順です。
- 並び替え順配列{$i}が-1の場合、並び替え順は降順です。

クロスタブモード

クロスタブモードの場合、結果の配列に格納される項目は2つ以下となります。これは、並び替えが実行されるのが、カラム（1）と行（2）だけであるためです（＜カラム配列＞の値）。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET SORTS

QR SET TOTALS DATA

QR SET TOTALS DATA(エリア;カラム番号; ブレーク番号; 演算子)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
ブレーク番号	倍長整数	→	ブレーク番号
演算子¦値	倍長整数¦文字列	→	セルまたはセル内容の演算子を示す値

説明

注：このコマンドで小計を作成することはできません。

リストモード

QR SET TOTALS DATAコマンドを使用し、特定のブレーク（総計または小計）に関する詳細を設定することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜カラム番号＞にはデータをセットするカラムの番号を渡します。

＜ブレーク番号＞には、データをセットするブレークの番号（小計または総計）を渡します。小計の場合、＜ブレーク番号＞には並び替え順の番号を指定します。総計の場合、＜ブレーク番号＞には-3または定数qr grand totalを指定します。

＜演算子＞には、セル内に存在するすべての演算子の和を指定します。QR operatorsテーマの定数を使用して、＜演算子＞を指定することができます。

定数	値
qr sum	1
qr average	2
qr min	4
qr max	8
qr count	16

＜演算子＞が0の場合、演算子は存在しません。

＜値＞にはセルに格納するテキストを指定します。

注：＜演算子＞と＜テキスト＞は相互に排他的な引数です、したがって＜演算子＞または＜テキスト＞のいずれか一方だけを設定してください。

次の値を渡すことができます。

■ ＜値＞に“#”を指定すると、ブレークまたは小計を対象とします。

■ “##S”は合計で置き換えられます。

■ “##A”は平均で置き換えられます。

■ “##C”はカウントで置き換えられます。

■ “##X”は最大値で置き換えられます。

■ “##N”は最小値で置き換えられます。

■ “##D”は標準偏差で置き換えられます。

■ “##xx”の“xx”にはカラム番号を指定し、その番号のカラムのフォーマットを用いて、カラムの値で置き換えられます。このカラムが存在しない場合、置き換えは実行されません。

クロスタブモード

QR SET TOTALS DATAコマンドを使用し、特定のセルの詳細を設定することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

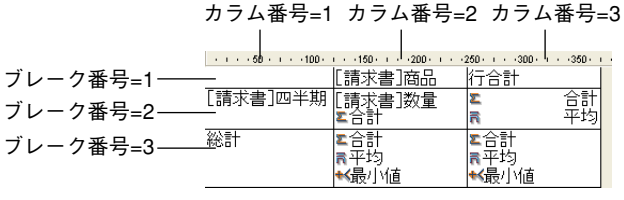
＜カラム番号＞には、データを設定するセルのカラム番号を渡します。

＜ブレーク番号＞には、データを設定するセルの行番号を渡します。

＜演算子＞には、セル内に存在するすべての演算子の和を指定します（上記参照）。

＜値＞にはセルに格納するテキストを指定します。

次の図は、クロスタブモードでの引数＜カラム番号＞と＜ブレーク番号＞の組み合わせ方について示しています。



サポートされるデータのタイプ：

渡すことができるデータのタイプは、基本的に2種類です。

■ タイトル

タイトルは、引数＜値＞を用いて渡します。実際のところ、＜値＞には文字列を指定し、次に示すセルに対してのみ渡すことができます。

カラム=3、ブレーク番号=1およびカラム=1、ブレーク番号=3

■ 演算子

単独の演算子、または演算子の組み合わせ（上図を参照）は、次のセルに対して渡すことができます。

■ カラム=2、ブレーク番号 =2

■ カラム=3、ブレーク番号 =2

■ カラム=2、ブレーク番号 =3

これら最後の2つの値は、セル（カラム3：行3）に対しても影響を与える点に注意してください。セル（カラム2：行3）に集計計算を定義する場合、セル（カラム2：行3）の内容は常にセル（カラム3：行3）をも定義します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効なブレーク番号を渡した場合、エラー番号-9853が生成されます。

参照

QR GET TOTALS DATA

QR GET TOTALS DATA

QR GET TOTALS DATA(エリア;カラム番号; ブレーク番号; 演算子; テキスト)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
ブレーク番号	倍長整数	→	ブレーク番号
演算子	倍長整数	←	セルの演算子を示す値
テキスト	文字列	←	セルの内容

説明

リストモード

QR GET TOTALS DATAコマンドを使用し、特定のブレークに関する詳細を取得することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜ブレーク番号＞には、データを取得するブレークの番号（小計または総計）を渡します。

小計：1から小計/並び替え項目数までの数字。

総計：-3または定数qr grand total。

＜カラム番号＞にはデータを取得するカラムの番号を渡します。

＜演算子＞には、セル内に存在するすべての演算子の和が返されます。QR operatorsテーマの定数を使用して、＜演算子＞を指定することができます。

定数	値
qr sum	1
qr average	2
qr min	4
qr max	8
qr count	16
qr standard deviation	32

＜演算子＞が0の場合、演算子は存在しません。

＜テキスト＞にはセル内のテキストが返されます。

注：＜演算子＞と＜テキスト＞は相互に排他的な引数です、したがって＜演算子＞または＜テキスト＞のいずれか一方にしか結果が返されません。

クロスタブモード

QR GET TOTALS DATAコマンドを使用し、特定のセルの詳細を取得することができます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

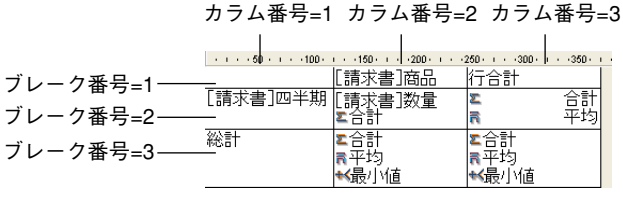
＜カラム番号＞には、データを取得するセルのカラム番号を渡します。

＜ブレーク番号＞には、データを取得するセルの行番号を渡します。

＜演算子＞には、セル内に存在するすべての演算子の和が返されます。QR Operatorsテーマの定数を使用し、返された値を処理することができます（上記参照）。

＜テキスト＞にはセル内のテキストが返されます。

次の図は、クロスタブモードでの＜カラム番号＞と＜ブレーク番号＞の組み合わせ方について示しています。



無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

無効なブレーク番号を渡した場合、エラー番号-9853が生成されます。

参照

QR SET TOTALS DATA

QR SET TOTALS SPACING

QR SET TOTALS SPACING(エリア; 小計番号;値)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
小計番号	倍長整数	→	カラム番号

引数	タイプ		説明
値	実数配列	→	0：間隔なし 32000：ページブレーク挿入 正の値：ブレークレベルの上部に 追加する間隔 負の値：比例増加

説明

QR SET TOTALS SPACINGコマンドを使用し、小計行の上部の間隔を設定することができます。このコマンドはリストモードにのみ適用されます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜小計番号＞には適用される小計レベル（ブレークレベル）を渡します。

＜値＞は間隔に関する値を指定します。

■ ＜値＞が0の場合、間隔は追加されません。

■ ＜値＞が32000の場合、ページブレークが挿入されます。

■ ＜値＞が正の値の場合、間隔を表わす値をピクセル単位で示します。

■ ＜値＞が負の値の場合、間隔を表わす値を小計行との比率で示します。例えば、-100であれば小計行の上に100%の間隔が設定されます。

注：小計行の上部の間隔により、行が次のページへ“押し出される”場合、そのページの行の上部には間隔が追加されません。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な小計番号を渡した場合、エラー番号-9852が生成されます。

参照

QR GET TOTALS SPACING

QR GET TOTALS SPACING

QR GET TOTALS SPACING(エリア; 小計番号;値)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
小計番号	倍長整数	→	小計番号
値	倍長整数	←	0：間隔なし 32000：ページブレークの挿入 正の値：ブレークレベルの上部に追加された間隔 負の値：比例増加

説明

QR GET TOTALS SPACINGコマンドを使用し、小計行の上部の間隔を取得することができます。このコマンドはリストモードにのみ適用されます。

＜エリア＞にはクイックレポートエリアの参照番号を渡します。

＜小計番号＞には適用される小計レベル（ブレークレベル）を渡します。＜小計＞は、1から小計／並び替えの数までの値となります。

＜値＞は間隔に関する値を示します。

- ＜値＞が0の場合、間隔は追加されていません。
- ＜値＞が32000の場合、ページブレークが挿入されています。
- ＜値＞が正の値の場合、間隔を表わす値がピクセル単位で示されます。
- ＜値＞が負の値の場合、間隔を表わす値が小計行との比率で示されます。例えば-100であれば小計行の上に100%の間隔が設定されています。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効な小計番号を渡した場合、エラー番号-9852が生成されます。

参照

QR SET TOTALS SPACING

QR INSERT COLUMN

QR INSERT COLUMN(エリア;カラム番号; オブジェクト)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号
オブジェクト	フィールド名¦変数¦ポインタ	→	カラムに挿入するオブジェクト

説明

QR INSERT COLUMNコマンドは指定された場所にカラムの作成または挿入を行います。これに応じて、挿入された場所の右にあるカラムはすべて右側へ移動します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

＜カラム番号＞には、左から右へ順に付加されたカラム番号を渡します。

カラムのデフォルトタイトルは＜オブジェクト＞に指定して渡します。

例題

次のコードはクイックレポートエリアの一番目のカラムを挿入（または作成）し、カラムタイトルに“Field1”を設定し（デフォルト動作）、Field1の値を本文の内容として代入します。

```
QR INSERT COLUMN(MyArea;1;->[Table 1]Field1)
```

参照

QR DELETE COLUMN

QR Get drop column

QR Get drop column(エリア)→倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
戻り値	倍長整数	←	ドロップの値

説明

QR Get drop column関数は、ドロップ動作を実行した場所に対応する値を返します。

■ 戻り値が負の場合、カラム番号を示します（つまり、カラム番号3でドロップを実行した場合には-3）。

■ 戻り値が正の場合、そのカラムの前にあるセパレータ上でドロップ動作が実行されたことを示します（つまり、カラム2の後ろにドロップされた場合には3）。ドロップ動作は、必ずしも既存のカラムの前で実行する必要はないことに留意してください。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR DELETE COLUMN

QR Count columns

QR Count columns(エリア)→倍長整数

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
戻り値	倍長整数	←	エリア内のカラム数

説明

QR Count columns関数は、クイックレポートエリアに存在するカラムの数を返します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードは、カラムの数を取得して、一番右端にある既存のカラムの右側にカラムを挿入します。

```
$ColNb:=QR Count columns(MyArea)
QR INSERT COLUMN(MyArea;$ColNb+1;->[Table 1]Field2)
```

参照

QR DELETE COLUMN、QR INSERT COLUMN

QR DELETE COLUMN

QR DELETE COLUMN(エリア;カラム番号)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
カラム番号	倍長整数	→	カラム番号

説明

QR DELETE COLUMNコマンドは、＜エリア＞にある＜カラム番号＞に渡された番号のカラムを削除します。このコマンドはクロスタブレポートに対しては適用されません。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

無効なカラム番号を渡した場合、エラー番号-9852が生成されます。

例題

次の例題は、レポートがリストレポートであることを確認し、3番目のカラムを削除します。

```
If(QR Get report kind(MyArea)=qr list report)
    QR DELETE COLUMN(MyArea;3)
End if
```

参照

QR INSERT COLUMN

QR SET HTML TEMPLATE

QR SET HTML TEMPLATE(エリア; テンプレート)

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
テンプレート	テキスト	→	HTMLテンプレート

説明

QR SET HTML TEMPLATEコマンドは、クイックレポートエリアに現在使用されるHTMLテンプレートを設定します。テンプレートは、HTML形式でレポートを作成する際に使用されます。テンプレートは一連のタグを用いて、元のレポートに近いレイアウトを維持したり、独自のカスタムHTMLを採用するためにデータを処理します。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

注：出力先をHTMLファイルに設定するため、まず初めにQR SET DESTINATIONコマンドを呼び出す必要がある点に注意してください。

HTMLタグ

■ <!--#4DQRheader--> ... <!--/#4DQRheader-->
カラムタイトルを元にして、これらのタグに挟まれたHTMLの内容を設定します。通常、これらのタグはレポートのタイトル行を定義するために使用します。

■ <!--#4DQRrow--> ... <!--/#4DQRrow-->
これらのタグに挟まれたHTMLの内容は、各データ行ごとに繰り返されます（詳細行と小計行を含む）。

■ <!--col--> ... <!--/#4DQRcol-->
これらのタグに挟まれたHTMLの内容は、1つの行の各データカラムごとに繰り返されます。カラムの順序は、レポート内の順序と同じです。“<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->”と一緒に使用した場合、“<!--#4DQRcol--> ... <!--/#4DQRcol-->”タグは、“<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->”を用いてその内容が挿入されたのではないカラムを対象とします。

例えば、5つのカラムから構成されるレポートの場合、“<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->”を使用して2番目のカラムからデータを挿入すると、“<!--#4DQRcol--> ... <!--/#4DQRcol-->”は、各行のカラム1、3、4、5を処理します。後者のタグは、“<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->”を用いてその内容が作成されたカラムを無視します。

■ <!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
これらのタグに挟まれたHTMLの内容は、番号が“n”であるレポートカラムから取り出されます。

例えば、3つのカラムから構成されるレポートにHTML出力として異なるカラム順を表示したい場合、次のように指定することができます。

<!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ... <!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->

この例では、カラムはレポートと逆の順序で挿入されます。

■ <!--#4DQRfont--> ... <!--/#4DQRfont-->
これらのタグに挟まれたHTMLの内容は、カレントカラムまたはセルのフォントおよびフォントサイズとして割り当てられます。

<!--#4DQRfont-->は、HTMLフォント定義に置き換えられ、<!--/#4DQRfont-->は対応する終了タグ（</font-->）に置き換えられます。

■ <!--#4DQRface--> ... <!--/#4DQRface-->
これらのタグに挟まれたHTMLの内容は、カレントカラムまたはセルのフォントスタイルとして割り当てられます。

<!--#4DQRface-->は、HTMLフェース定義に置き換えられ、<!--#4DQRface--> は対応する終了タグ（</face-->）に置き換えられます。

■ <!--#4DQRbgcolor-->
このカラータグは、カレントセルのカレントカラーで置き換えられます。

■ <!--#4DQRdata-->
このタグは、カレントセルのカレントデータで置き換えられます。

■ <!--#4DQRlHeader-->、<!--#4DQRcHeader-->、<!--#4DQRrHeader-->
これらのタグはそれぞれ、左ヘッダ、中央ヘッダ、右ヘッダのデータで置き換えられます。

■ <!--#4DQRlFooter-->, <!--#4DQRcFooter-->, and<!--#4DQRrFooter-->
これらのタグはそれぞれ、左フッタ、中央フッタ、右フッタのデータで置き換えられます。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR Get HTML template

QR Get HTML template

QR Get HTML template(エリア)→テキスト

引数	タイプ		説明
エリア	倍長整数	→	エリアの参照番号
戻り値	テキスト	←	テンプレートとして使用するHTMLコード

説明

QR Get HTML template関数は、クイックレポートエリアに現在使用されているHTMLテンプレートを返します。戻り値はテキスト値であり、HTMLテンプレートの全内容が納められます。

テンプレートが特に指定されていない場合、デフォルトのテンプレートが返されます。手動、またはプログラムにより出力先をHTMLファイルに設定した場合、テンプレートが返されない点に注意してください。

無効なエリア番号を渡した場合、エラー番号-9850が生成されます。

参照

QR SET HTML TEMPLATE

レコードロックコマンド 41

レコードのロック

4th Dimensionと4D Server／4D Clientは、マルチユーザまたはマルチプロセスのコンフリクトを防ぐことによってマルチユーザデータベースを自動的に管理します。2人のユーザまたは2つのプロセスが、同時に同じレコードやオブジェクトを修正することはできませんが、ロード（読み込む）することは可能です。

この章のマルチユーザコマンドを使用しなければならない状況がいくつかあります。

■ プログラミング言語を使用してレコードを更新する。

■ マルチユーザ環境でカスタムユーザインタフェースを使用する。

■ トランザクション内でリレート変更を保存する。

マルチユーザデータベースでコマンドを使用するときに注意すべき重要な概念が3つあります。

■ 各テーブルは、読み込み専用（リードオンリー）または読み書き可能（リードライト）のどちらかに設定される。

■ レコードは、ロードされた時点でロック状態となり、ロードされてない状態でロック解除状態になる。

■ ロックされたレコードは、更新することはできない。

この章では、マルチユーザデータベースを使用して作業を実行する人を“ローカルユーザ”と呼びます。マルチユーザデータベースを使用する他の人を“他のユーザ”と呼びます。この節では、ローカルユーザの観点で説明します。また、マルチプロセスの観点から、データベース上で処理を実行しているプロセスを“カレントプロセス”と呼びます。その他の実行しているプロセスは、“他のプロセス”と呼びます。この章では、カレントプロセスの観点で説明します。この章では、「ルーチン」エディタの「Record Locking」テーマ内にあるレコードロックコマンドについて説明します。

ロックされたレコード

ローカルユーザは、ロックされたレコードをロードすることは可能ですが、更新することはできません。他のユーザが、レコードを更新するためにレコードをロードした時点で、そのレコードはロックされます。この場合、レコードを更新しているユーザだけが、そのレコードをロックされていない状態で取り扱うことができます。その他のすべてのユーザは、レコードがロック状態になるため更新することはできません。ロックされていない状態でレコードをロードするには、テーブルを必ずリードライト状態に設定しなければなりません。

リードオンリー状態とリードライト状態

データベース上の各テーブルは、データベースの各ユーザおよびプロセスに対してリードオンリー状態とリードライト状態のいずれかになっています。リードオンリーとは、テーブルからレコードをロードすることはできるが更新することはできないという状態です。リードライトとは、テーブルのレコードをロードはロード可能であり、他のユーザが先にそのレコードをロックしていない場合には更新することができる状態です。

テーブルのステータスを変更すると、その変更が次からロードされるレコードに影響を及ぼすことに注意してください。そのため、テーブルステータスを変更する際にちょうどロードされたレコードがあっても、そのレコードはステータス変更によって影響を受けることはありません。

リードオンリー状態

テーブルがリードオンリー状態に設定されると、ロードしたレコードは常にロックされます。つまり、レコードの表示と印刷を実行することはできますが、更新することはできません。

このリードオンリー状態は、既存レコードの編集作業に対してのみ適用されることに注目してください。つまり、新規レコードを作成する場合には何の意味も持ちません。そのため、「ユーザ」モードの「更新」メニューにある「新規レコード」メニュー項目を選択する、あるいは**CREATE RECORD**コマンドや**ADD RECORD**コマンドを使ってリードオンリーテーブルにレコードを追加することができます。

4th Dimensionは、レコードに対する書き込み動作を伴わないコマンドが実行されるテーブルを自動的にリードオンリー状態にします。以下の表に、テーブルを自動的にリードオンリー状態にするコマンドを示します。

DISPLAY SELECTION **EXPORT TEXT** **REPORT**
DISTINCT VALUES **GRAPH TABLE** **SELECTION TO ARRAY**
EXPORT DIF **PRINT LABELS** **SELECTION RANGE TO ARRAY**
EXPORT SYLK **PRINT SELECTION**

これらのコマンドを実行する前に、4th Dimensionはカレントプロセスにおけるテーブルのそのときの状態（リードオンリーまたはリードライト）を保存します。コマンドを実行した後でテーブルの状態を復元します。

リードライト状態

リードライト状態のテーブルからロードされたレコードは、他のユーザがそのレコードを先にロックしていなければアンロックになります。レコードが他のユーザによってロックされている場合にはレコードをロードすることはできますが、更新することはできません。

テーブルがリードライト状態に設定され、ロードしたレコードがアンロックになった時点で初めて、そのレコードの更新が可能になります。

あるユーザがリードライト状態のテーブルからレコードをロードすると、他のユーザは修正するためにレコードをロードできません。しかし、他のユーザは「ユーザ」モード内で手動により、あるいは**CREATE RECORD**コマンドや**ADD RECORD**コマンドを使ってそのテーブルにレコードを追加することはできます。

データベースが開かれたり、または新規プロセスが開始された時点で、すべてのテーブルのデフォルト状態はリードライトです。

テーブルステータスの変更

READ ONLYコマンドおよび**READ WRITE**コマンドを使って、テーブルステータスを変更することができます。あるレコードをリードオンリー状態またはリードライト状態にするために任意テーブルのステータスを変更したい場合は、そのレコードをロードする前にコマンドを実行する必要があります。既にロードされたレコードは、この**READ ONLY**コマンドおよび**READ WRITE**コマンドによって影響を受けることはありません。

各プロセスは、データベース内の各テーブルに対する独自の状態（リードオンリーまたはリードライト）を持っています。

レコードのロード、更新、アンロード

ローカルユーザがレコードを更新するためには、テーブルがリードライト状態でかつ、ロードしたレコードがアンロックになっていなければなりません。

NEXT RECORD、**QUERY**、**ORDER BY**、**RELATE ONE**等のコマンドは、カレントレコードをロードし、そのレコードをロックまたはアンロックにします。

レコードは、テーブルのその時の状態（リードオンリーまたはリードライト）に応じてロードされます。レコードが自動リレートされている場合には、自動リレートを実行するコマンドによってリレート先テーブルからもレコードがロードされます。

テーブルがリードオンリー状態になっている場合には、テーブルからロードされたレコードはロックされます。ロックされたレコードは、保存および削除ができません。リードオンリー状態は他のユーザがレコードをロード、更新、保存等の処理を実行している時には望ましい状態です。

テーブルがリードライト状態になっている場合には、テーブルからロードされたレコードは他のユーザが先にロックしていない限りアンロックになります。アンロックされたレコードは、テーブルに保存することができます。テーブルは、レコードをロード、更新、保存等の処理を実行する前にリードライト状態にしなければなりません。

レコードを修正する場合、**Locked**関数を使って、他のユーザがレコードをロックしていないかどうかを調べます。レコードがロックされている場合（**Locked**関数がTrueの場合）には、レコードを**LOAD RECORD**コマンドでロードし、そのレコードがロックされているかどうかを再び調べます。レコードがアンロックになるまで（**Locked**関数がFalseになるまで）この処理を繰り返します。

レコードへの修正が完了したら、**UNLOAD RECORD**コマンドを使って、そのレコードを解放（他のユーザに対してアンロック）しなければなりません。レコードがアンロードされないと、他のカレントレコードが選択されるまで、すべての他のユーザに対してロックされた状態になります。テーブルのカレントレコードを変えると、前のカレントレコードは自動的にアンロックになります。カレントレコードを変更しない場合は、**UNLOAD RECORD**コマンドを明示的に呼び出す必要があります。これは既存レコードの場合だけであり、新しいレコードは作成されると、そのレコードが属するテーブルの状態に関係なく保存することができます。

LOCKED ATTRIBUTESコマンドを使用すると、レコードをロックしているユーザやプロセスを知ることができます。

アンロックされたレコードをロードするためのループ処理

以下の例は、アンロックされたレコードをロードするための最も単純なループ処理を示しています。

```
READ WRITE([顧客])          ` テーブルの状態をリードライトにする
Repeat          ` レコードがアンロックになるまで繰り返す
        LOAD RECORD([顧客])     ` ロック情報を得るためにレコードをロードする
Until(Not(Locked([顧客])))
        ` ここでレコードに対する処理を実行する
READ ONLY([顧客]) ` テーブルの状態をリードオンリーにする
```

このループ処理は、レコードがアンロックされるまで繰り返されます。

このようなループ処理は、ユーザがループが終了するまで待たされるため、レコードが他のユーザにロックされる可能性がほとんどない場合にのみ使用することができます。したがって、メソッドからのみレコードを更新するような場合以外では使用することはできません。

以下の例は、アンロックされたレコードをロードして、更新するためのループ処理を示しています。

```
READ WRITE([在庫])          ` テーブルの状態をリードライトにする
Repeat          ` レコードがアンロックになるまで繰り返す
 LOAD RECORD([在庫])    ` ロック情報を得るためにレコードをロードする
Until(Not(Locked([在庫])))
[在庫]部品数量:=[在庫]部品数量 - 1 ` レコードを更新
SAVE RECORD([在庫])       ` レコードを保存
UNLOAD RECORD([在庫])   ` 他のユーザが更新できるようにレコードをアンロック
READ ONLY([在庫]) ` テーブルの状態をリードオンリーにする
```

MODIFY RECORDコマンドは、自動的にレコードがロックされているかどうかを調べ、ロックされている場合には、それをユーザに通知します。以下の例は、最初に**Locked**関数を使用してレコードの状態を調べ、**MODIFY RECORD**コマンドの自動的な通知処理を無効にします。レコードがロックされている場合には、ユーザが処理を中断できるようにします。

以下の例は、[コマンド]テーブルのカレントレコードがロックされているかどうかを調べます。ロックされている場合は、プロセスがメソッドによって2秒間延期されます。この技法は、マルチユーザやマルチプロセスの場合に用いられます。

```
Repeat
      ` リードライト状態にする必要はない
   READ ONLY([コマンド])
   QUERY([コマンド])
         ` 検索がうまくいくと、レコードがいくつか返される
   If((OK=1) &(Records in selection([コマンド]) > 0))
         ` テーブルをリードライト状態に設定する
      READ WRITE([コマンド])
      ` レコードがアンロックの場合
      LOAD RECORD([コマンド])
      ` レコードがアンロックになるまでループする
      While(Locked([コマンド]) &(OK=1))
            ` どのユーザによってレコードがロックされているのかを調べる
         LOCKED ATTRIBUTES([コマンド] ; $ﾌﾟﾛｾｽ番号 ; $ユーザ ;
                                                $マシン ; $プロセス名)
         If($ﾌﾟﾛｾｽ番号=-1) ` レコードが削除されたか ?
            ALERT("レコードが合間に削除されてしまいました。")
            OK:=0
         Else   ` シングルユーザ環境 ?
            If($ユーザ="")
               $ユーザ:="あなた"
            End if
            CONFIRM("レコードは既に " + $ユーザ + " によって"
             + $プロセス名 +" プロセス内で使用されています。")
            If(OK=1)      ` もし、数秒間、延期したい場合
               DELAY PROCESS(Current process ; 120)
               ` レコードをロードしてみる
               LOAD RECORD([コマンド])
            End if
         End if
      End while
      ` レコードがアンロックされる
      If(OK=1)
            ` レコードを修正することができる
         MODIFY RECORD([コマンド])
         UNLOAD RECORD( [コマンド])
      End if
      ` リードオンリー状態に切り替える
```

```
            READ ONLY([コマンド])
            OK:=1
        End if
    Until(OK=0)
```

マルチユーザやマルチプロセスモードでのコマンドの使用

多くのコマンドは、レコードがロックされていることを検知した時点で、特定の処理を実行します。これらのコマンドは、レコードがロックされていない場合には、通常どおりの処理を遂行します。以下のリストに、レコードがロックされていることを検知した場合の各コマンドの処理を示します。

■ **MODIFY RECORD**コマンド：レコードが使用されていることを示すダイアログボックスを表示し、該当するレコードを表示しません。したがって、ユーザはレコードを修正することができません。「ユーザ」モードでは、レコードはリードオンリー状態で表わされています。

■ **MODIFY SELECTION**コマンド：ユーザがレコードをダブルクリックして修正しようとした場合を除いて通常どおりの処理を行います。**MODIFY SELECTION**コマンドは、レコードが使用されていることを示すダイアログボックスを表示し、リードオンリー状態でレコードのアクセスを許可します。

■ **APPLY TO SELECTION**コマンド：ロックされたレコードをロードしますが、そのレコードは更新されません。**APPLY TO SELECTION**コマンドは、特別な処置を行わずにテーブルからレコードを読み取ります。ロックされたレコードを検知すると、そのレコードをLockedSetと呼ばれるシステムセットに格納します。

■ **DELETE SELECTION**コマンド：ロックされたレコードを削除しないで読み飛ばします。ロックされているレコードを、LockedSetと呼ばれるシステムセットに格納します。

■ **DELETE RECORD**コマンド：レコードがロックされている場合には何も行いません。エラーも全く返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。

■ **SAVE RECORD**コマンド：レコードがロックされている場合には何も行いません。エラーも全く返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。

■ **ARRAY TO SELECTION**コマンド：ロックされたレコードは保存しません。ロックされたレコードを検知すると、そのレコードをLockedSetと呼ばれるシステムセットに格納します。

■ **GOTO RECORDコマンド：**マルチユーザ／マルチプロセスデータベース上では、他のユーザがレコードを削除、または追加することができます。したがって、レコード番号が変更される場合があるため、マルチユーザデータベース上でレコード番号を使用して直接レコードを参照する場合には、十分注意してください。

■ **セットコマンド：**セットに従っている情報を他のユーザやプロセスが変更する可能性があるため、セットの使用には細心の注意が必要です。

参照

LOAD RECORD、ロックされたレコード、LOCKED ATTRIBUTES、メソッド、READ ONLY、リードオンリー状態、READ WRITE、UNLOAD RECORD、変数

READ WRITE

READ WRITE {(テーブル | *)}

引数	タイプ		説明
テーブルまたは	テーブル	→	リードライト状態にするテーブルまたは
*	*		すべてのテーブル 省略した場合、デフォルトテーブル

説明

READ WRITEコマンドは、それが呼び出されるプロセス内の<テーブル>の状態をリードライトに変更します。オプション引数<*>を指定すると、すべてのテーブルはリードライト状態に変更されます。

READ WRITEコマンドを呼び出した後、レコードがロードされると、他のユーザがそのレコードをロックしていない場合にはアンロックになります。このコマンドは、現在ロードしているレコードの状態を変更するものではありません。その次以降にロードしたレコードに対してのみ適用されます。

すべてのテーブルのデフォルト状態は、リードライト状態です。

READ WRITEコマンドは、レコードを修正し、その結果を保存しなければならないときに使用します。また、レコードを修正しない場合でも、他のユーザに対してレコードをロックする必要のあるときにもこのコマンドを使用します。テーブルをリードライトモードに設定することにより、他のユーザによるそのテーブルに対する編集を防ぐことができます。ただし、他のユーザは新規レコードの作成は行えます。

注：このコマンドは、コマンドが実行された時点からさかのぼって適用されることはありません。レコードはロードされた時点におけるテーブルのリードライト状態に従ってロードされます。リードライトモード内でリードオンリーテーブルのレコードをロードするには、そのレコードをロードする前にそのテーブルステータスをリードライトモードに変更する必要があります。

参照

READ ONLY、Read only state

READ ONLY

READ ONLY {(テーブル | *)}

引数	タイプ		説明
テーブル	テーブル	→	リードオンリー状態にするテーブル またはすべてのテーブル 省略した場合、デフォルトテーブル

説明

READ ONLYコマンドは、それが呼び出されるプロセス内の＜テーブル＞の状態をリードオンリーに変更します。このコマンドを実行した後でロードしたレコードはすべてロックされ、変更することはできません。オプション引数＜*＞を指定すると、すべてのテーブルはリードオンリー状態に変更されます。

1つまたは複数のレコードを修正する必要のない場合に、**READ ONLY**コマンドを使用します。

注：このコマンドは、コマンドが実行された時点からさかのぼって適用されることはありません。レコードはロードされた時点におけるテーブルのリードライト状態に従ってロードされます。リードオンリーモード内でリードライトテーブルのレコードをロードするには、そのレコードをロードする前にそのテーブルステータスをリードオンリーモードに変更する必要があります。

参照

Read only state、READ WRITE

Read only state

Read only state {(テーブル)} → ブール

引数	タイプ		説明
テーブル	テーブル	→	読み込み専用状態かどうかを判断するテーブル 省略した場合、デフォルトテーブル
戻り値	ブール	←	リードオンリーモードの場合“True” リードライトモードの場合“False”

説明

この関数は、それが呼び出されるプロセス内の＜テーブル＞の状態がリードオンリー（読み込み専用）かどうかを調べます。＜テーブル＞が読み込み専用であれば、True（真）を返します。＜テーブル＞がリードライト状態であれば、False（偽）を返します。

例題

▼ 以下の例は、[送り状]テーブルの状態を判断するものです。[送り状]テーブルの状態が読み込み専用であれば、書き込み可能状態に設定し、カレントレコードを再度ロードします。

```
If(Read only state([送り状]))
    READ WRITE([送り状])
    LOAD RECORD([送り状])
End if
```

注：送り状のレコードを再度ロードすることにより、ユーザはレコードを修正できるようになります。前回読み込み専用でロードされたレコードは、書き込み可能状態で再度ロードされるまで、ロックされたままです。

参照

READ ONLY、READ WRITE

LOAD RECORD

LOAD RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	ロードするレコードの属するテーブル または省略した場合デフォルトテーブル

説明

LOAD RECORDコマンドは、＜テーブル＞のカレントレコードをロードします。カレントレコードが存在しない場合は、**LOAD RECORD**コマンドは何も行いません。

レコードの状態は、**Locked**関数で調べることができます。

- ＜テーブル＞が“リードオンリー”状態ならば、**Locked**関数は“True（真）”を返し、レコードを修正することはできません。
- ＜テーブル＞が“リードライト”状態でもレコードが既にロックされている場合は、レコードは“リードオンリー”状態になり、そのレコードを修正することはできません。
- ＜テーブル＞が“リードライト”状態で、しかもレコードがロックされていない場合は、カレントプロセス内のレコードを修正することができます。その際、**Locked**関数はその他のすべてのユーザとプロセスに対して“True（真）”を返します。

注：READ ONLYコマンドの後にLOAD RECORDコマンドを実行すると、UNLOAD RECORDコマンドを使用しなくても、レコードは自動的にアンロードされた後ロードされます。

一般に、**QUERY**コマンド、**NEXT RECORD**コマンド、**PREVIOUS RECORD**コマンド等のコマンドは自動的にカレントレコードをロードするため、**LOAD RECORD**コマンドを使用する必要はありません。

マルチユーザ環境やマルチプロセス環境において、既存レコードを修正するには、“リードライト”状態でテーブルをアクセスし、しかもそのレコードが他のユーザやプロセスによってロックされていないことが必要です。レコードがロックされており、ロードされない場合、**LOAD RECORD**コマンドは再度カレントレコードをロードしようと試みます。また、ループ内で**LOAD RECORD**コマンドを使用することにより、レコードが“リードライト”状態になるまで待機することができます。

Tips：LOAD RECORDコマンドを使用し、入力フォームにおいてカレントレコードを再ロードすることができます。すると、変更されたデータはすべて以前の値で置き換えられます。この場合、LOAD RECORDコマンドは、いわば一般的なデータ入力のキャンセル処理として振る舞います。

参照

Locked、UNLOAD RECORD

UNLOAD RECORD

UNLOAD RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	アンロードするレコードの属するテーブル または省略した場合、デフォルトテーブル

説明

UNLOAD RECORDコマンドは、＜テーブル＞のカレントレコードをアンロードします。

UNLOAD RECORDコマンドは、レコードがローカルユーザに対してアンロックされている場合（他のユーザに対してロックされている場合）は、他のユーザに対してレコードをアンロックします。

UNLOAD RECORDコマンドは、メモリからレコードを解放（アンロード）しますが、そのレコードはカレントレコードのままになります。他のレコードがカレントレコードになると、前のカレントレコードは自動的にアンロードされ、他のユーザに対してアンロックされます。レコードの修正が終わり、そのレコードを自分自身のカレントレコードとしたままで、他のユーザから使えるようにしたい場合は常にこのコマンドを実行します。

レコードに大きなデータやピクチャフィールドまたは4D Draw等の外部ドキュメントが含まれている場合に、修正の必要がなくなるまで、そのカレントレコードをメモリ内に保持したくないかもしれません。こういう場合、**UNLOAD RECORD**コマンドを使用して、メモリ内にレコードを持たずにカレントレコードを保持できるようにします。また、そのフィールドの値にアクセスすることはできませんが、そのレコードによって占有されていたメモリを解放することができます。後に、レコードの値へのアクセスが必要となった場合、**LOAD RECORD**コマンドを使用します。

参照

LOAD RECORD

Locked

Locked {(テーブル)} → ブール

引数	タイプ		説明
テーブル テーブル	テーブル	→	カレントレコードのロックを調べる 省略した場合、デフォルトテーブル
戻り値	ブール	←	ロックされている場合“True” アンロックされている場合“False”

説明

Locked関数は、＜テーブル＞のカレントレコードがロックされているかを調べます。この関数を使用してレコードがロックされているかどうかを調べた後、レコードが開放されるまで待機するか処理をスキップするかの選択をユーザに与える、等の適切な処理を行います。

Locked関数が“True”を返す場合、レコードは他のユーザまたはプロセスによりロックされており、レコードを保存することはできません。この場合には、**LOAD RECORD**コマンドを使用して、**Locked**関数が“False”を返すまでレコードのロードを繰り返します。

Locked関数が“False”を返す場合、レコードはアンロックされています。これはレコードが他のすべてのユーザに対してロックされることを意味します。ローカルユーザ、またはカレントプロセスだけがレコードを修正、保存することができます。レコードを修正するには、テーブルがリードライト状態でなければなりません。

削除されたレコードをロードしようとすると、**Locked**関数は“True”を返し続けます。存在しないレコードをこれ以上待機しないために、**LOCKED ATTRIBUTES**コマンドを使用します。レコードが削除されている場合には、**LOCKED ATTRIBUTES**コマンドはプロセス引数に-1を返します。

トランザクション処理の実行中にも、レコードがロックされているかどうかを調べるために**LOAD RECORD**コマンドと**Locked**関数を使用します。レコードがロックされている場合には、トランザクション処理をキャンセルするのが一般的です。

参照

LOAD RECORD、LOCKED ATTRIBUTES

LOCKED ATTRIBUTES

LOCKED ATTRIBUTES({テーブル ;} プロセス ; ユーザ ; マシン ; プロセス名)

引数	タイプ		説明
テーブル	テーブル	→	ロックされたレコードを調べるテーブル 省略した場合、デフォルトテーブル
プロセス	数値	←	プロセス参照番号
ユーザ	文字列	←	マルチユーザの場合、ユーザ名
マシン	文字列	←	マルチユーザの場合、マシン名
プロセス名	文字列	←	プロセス名

説明

LOCKED ATTRIBUTESコマンドは、レコードをロックしたユーザまたはプロセスに関する情報を返します。＜プロセス＞、＜ユーザ＞、＜マシン＞、および＜プロセス名＞の各変数にプロセス参照番号、ユーザ名、マシン名およびプロセス名を返します。レコードがロックされている場合には、この情報をダイアログボックスに表示して、ユーザに警告します。

レコードがロックされていない場合は、＜プロセス＞は0を返し、＜ユーザ＞、＜マシン＞、および＜プロセス名＞は空の文字列を返します。リードライト状態でロードしようとしたレコードが削除されている場合には、＜プロセス＞は-1を返し、＜ユーザ＞、＜マシン＞、および＜プロセス名＞は空の文字列を返します。

シングルユーザ環境では、このコマンドはレコードがロックされている場合にのみ＜プロセス＞と＜プロセス名＞に値を返します。この場合に、＜ユーザ＞と＜マシン＞は空の文字列を返します。

クライアント／サーバ環境では、コマンドから返されるプロセス番号は、サーバ上のプロセス番号です。

引数＜ユーザ＞には、たとえユーザ名が空白であっても、4th Dimensionのパスワードシステムからユーザ名が返されます。もし、パスワードシステムがない場合は、“管理者”が返されます。

引数＜マシン＞に返されるのは、オペレーションシステムのテーブル共有で設定されているオーナー名です。名前の変更は、マシンが再起動するまで行われません。

参照

Locked

DISPLAY RECORD

DISPLAY RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードを表示するテーブル 省略した場合、デフォルトテーブル

説明

DISPLAY RECORDコマンドは、カレント入力フォームを使って、<テーブル>のカレントレコードを表示します。このレコードは、イベントがウインドウを更新するまで表示されます。このイベントとは、**ADD RECORD**コマンドを実行する、または入力フォームへ戻ったり、メニューバーに戻ることです。**DISPLAY RECORD**コマンドは、カレントレコードが存在しない場合には何も行いません。

DISPLAY RECORDコマンドは、よくオリジナルの進捗メッセージを表示するために使用されます。また、自由に動くスライドショーを生成するために使用されることもあります。

フォームメソッドが存在する場合は、**On Load**イベントが発生します。

警告：DISPLAY RECORDコマンドはWeb接続プロセスからコールしないでください。このコマンドは、Webブラウザクライアントマシン上ではなく、4th DimensionのWebサーバマシン上で実行されるためです。

例題

▼ 以下の例は、レコードを次から次へ表示します。

```
ALL RECORDS([デモ])      ` すべてのレコードを選択
INPUT FORM([デモ] ; "表示")      ` 表示フォームの設定
For($i ; 1 ; Records in selection([デモ])) ` レコード全体をループ
    DISPLAY RECORD([デモ]) ` レコードの表示
    DELAY PROCESS(Current process; 180) ` 3秒間停止
    NEXT RECORD([デモ])      ` 以下のレコードへ移動
End for
```

参照

MESSAGE

CREATE RECORD

CREATE RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	新しいレコードを作成するテーブル 省略した場合、デフォルトテーブル

説明

CREATE RECORDコマンドは、＜テーブル＞に対して新しい空レコードを作成しますが、このレコードは表示されません。データ入力のために新しいレコードを作成、表示するには**ADD RECORD**コマンドを使用します。

CREATE RECORDコマンドは、レコードデータの割り当てをプログラミング言語上で実行する場合に、**ADD RECORD**コマンドの代わりに使用します。新しく作成されたレコードはカレントレコードとなり、さらにカレントセレクション（レコードが1件のみのカレントセレクション）になります。

新しいレコードは、同一テーブルに対する**SAVE RECORD**コマンドを実行するまではメモリ上にのみ存在します。新しいカレントレコードが保存される前に修正された場合（検索処理等によって）には、そのレコードは失われます。

例題

▼ 以下の例は、30日以上経過した[伝票]テーブルのデータを[台帳]テーブルに転記します。[台帳]テーブルに対し新しいレコードを登録することによって、これを行っています。転記が終了した時点で、転記し終わった[伝票]レコードを[伝票]テーブルから削除します。

```
` 本日から30日前までのレコードを探す
QUERY([伝票] ; [伝票]作成日付 <(Current date - 30))
For($i ; 1 ; Records in selection([伝票]))  ` 転記が終わるまで繰り返す
    CREATE RECORD([台帳])                  ` 新しい[台帳]レコードを作成
    [台帳]番号:=[伝票]番号                 ` [台帳]レコードへデータを転記
    [台帳]作成日付:=[伝票]作成日付
    [台帳]合計金額:=[伝票]合計金額
    SAVE RECORD([台帳])                    ` 新しい[台帳]レコード登録
    NEXT RECORD([伝票])                    ` 以下の[伝票]レコードへ移動
End for
DELETE SELECTION([伝票])                   ` [伝票]レコードを削除
```

参照

SAVE RECORD

DUPLICATE RECORD

DUPLICATE RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	複製元のカレントレコードの属する
テーブル			
			省略した場合、デフォルトテーブル

説明

DUPLICATE RECORDコマンドは、カレントレコードを複製して同じ＜テーブル＞内に新しいレコードを作成します。新しいレコードは、カレントレコードとなります。カレントレコードが存在しない場合には、**DUPLICATE RECORD**コマンドは何も行いません。新しいレコードを保存するには、**SAVE RECORD**コマンドを使用しなければなりません。

DUPLICATE RECORDコマンドは、データ入力中にも実行可能です。これにより、現在表示しているレコードのコピーを作ることができます。複写元のレコードへの変更を保存するために、**SAVE RECORD**コマンドを最初に実行しておくことを忘れないでください。

参照

SAVE RECORD

Is new record

Is new record({テーブル }) → ブール

引数	タイプ		説明
テーブル	テーブル	→	検査するレコードの属するテーブル。この引数が省略されている場合はデフォルトテーブル
戻り値	ブール	←	未登録の新規レコードの場合True、そうでなければFalse

説明

この関数は、指定されたテーブルのカレントレコードが未登録の新規レコードの場合にTrueを返します。

互換性について：Record numberコマンドが-3を返すかどうかで同じ情報を得ることができますが、Record numberの代りにIs new recordを使用することを強くお勧めします。実際、Is new recordコマンドは4th Dimensionの将来のバージョンとのより優れた互換性を保証します。

例題

▼ 下記の2つの方法は同一のものです。コードが4Dの将来のバージョンとの互換性を保つため、2番目の方法を強く推奨します。

```
If(Record number([Table])=-3)`この方法は推奨しません
   `...
End if
If(Is new record([Table])) `この方法を使用してください
   `...
End if
```

参照

Record number

Modified record

Modified record {(テーブル)} → ブール

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードが修正されたか判定するテーブル 省略した場合、デフォルトテーブル
戻り値	ブール	←	True：レコードは修正されている False：レコードは修正されていない

説明

Modified record関数は、＜テーブル＞のレコードが更新されたが保存されていない場合に“True（真）”を返します。それ以外は“False（偽）”を返します。この関数は、保存する必要があるレコードかどうかを判定する場合に使用します。これは、入力フォーム上で以下のレコードに進む前にカレントレコードを保存するかどうかチェックする際に特に有効です。**Modified record**関数は新規レコードについては、常に“True（真）”を返します。

例題

▼ 以下の例は、**Modified record**関数の一般的な使用方法です。

```
If(Modified record([テーブル1]))
    SAVE RECORD([テーブル1])
End if
```

参照

Modified、Old、SAVE RECORD

Is record loaded

Is record loaded({table}) → ブール

引数	タイプ		説明
テーブル	テーブル	→	検査するレコードの属するテーブル。この引数が省略されている場合はデフォルトテーブル
戻り値	ブール	←	レコードがロードされていればTrue、それ以外はFalse

説明

この関数は、指定テーブルのカレントレコードがカレントプロセス内にロードされていればTrueを返します。

例題

▼ “次レコード”または“前レコード”の自動動作を使用する代りに、これら2つのボタン用にオブジェクトメソッドを書いて、これらの作業を改善させることができます。"Next"ボタンは、セレクションの最後のレコードであればセレクションの最初のレコードを表示し、"Previous"ボタンは、セレクションの最初のレコードであればセレクションの最後のレコードを表示します。

```
` "Previous"ボタンのオブジェクトメソッド(自動動作無し）
If(Form event=On Clicked)
	PREVIOUS RECORD([Group])
	If(Not(Is record loaded([Group])))
		GOTO SELECTED RECORD([Group];Records in selection
		                                                ([Group]))
		`セレクションの最後のレコードに移行
	End if
End if
	`"Next"ボタンのオブジェクトメソッド(自動動作無し）
If(Form event=On Clicked)
	NEXT RECORD([Group])
	If(Not(Is record loaded([Group])))
		GOTO SELECTED RECORD([Groups];1)
		`セレクションの最初のレコードに移動
	End if
End if
```

SAVE RECORD

SAVE RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードを保存するテーブル 省略した場合、デフォルトテーブル

説明

SAVE RECORDコマンドは、カレントプロセスの＜テーブル＞のカレントレコードを保存します。カレントレコードが存在しない場合には、**SAVE RECORD**コマンドは何も行いません。

SAVE RECORDコマンドは、コードを使って新しく作成または修正されたレコードを保存するために使用します。フォームでユーザが修正、または確定したレコードは、**SAVE RECORD**コマンドで保存する必要はありません。しかし、ユーザによってフォーム中で修正されたレコードが取り消された場合でも、**SAVE RECORD**コマンドで保存することができます。

SAVE RECORDコマンドが必要とされる場合を次に示します。

■ **CREATE RECORD**コマンドまたは**DUPLICATE RECORD**コマンドで作成した新しいレコードを保存する場合

■ **RECEIVE RECORD**コマンドからのデータを保存する場合

■ メソッドによって修正したレコードを保存する場合

■ **ADD SUBRECORD**コマンド、**CREATE SUBRECORD**コマンド、**MODIFY SUBRECORD**コマンドによって新しく作成または修正したサブレコードを保存する場合

■ カレントレコードを変更するようなコマンドを実行する前に、データ入力途中で表示されているレコードを保存する場合

■ データ入力途中で新しいレコードを保存する場合

受け入れられたフォームの**On Validate**イベントで**SAVE RECORD**コマンドを実行してはいけません。もし、これを実行すると、レコードが2つ保存されてしまいます。

例題

以下の例は、メソッドの一部で、ドキュメントからレコードを読み込みます。コードのセグメントはレコードを受信し、この後受信が正常に行われると、レコードを保存します。

```
RECEIVE RECORD([顧客]) ` ディスクからレコードを受信する
If(OK= 1) ` レコードを正常に受信した場合
SAVE RECORD([顧客]) ` レコードを保存
End if
```

参照

CREATE RECORD、Locked、Triggers

DELETE RECORD

DELETE RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	削除するカレントレコードの属する
テーブル			
			省略した場合、デフォルトテーブル

説明

DELETE RECORDコマンドは、＜テーブル＞のカレントレコードを削除します。プロセスに＜テーブル＞のカレントレコードが存在しない場合には、**DELETE RECORD**コマンドは何も行いません。フォーム中では、このコマンドの代わりにレコード削除属性を持つボタンを作成することができます。レコードを削除すると、＜テーブル＞のカレントセレクションは空になります。

レコードの削除は、一度実行すると元に戻すことはできません。

レコードが削除されると、そのレコード番号は新しいレコードが作成される際に再利用されます。したがって、データベースからレコードを削除する場合は、レコード番号をレコードの識別に使用しないでください。

例題

▼ 以下の例は、1件の[従業員]レコードを削除します。まず、ユーザにどの従業員レコードを削除するのかを尋ね、[従業員]レコードを検索し、見つかったレコードを削除します。

```
v削除社員:=Request("社員番号を入力してください。") `社員番号を入力してもらう
If(OK=1)                    `検索、削除の実行？
    QUERY([従業員] ; [従業員]社員番号 = v削除社員)          `従業員を探す
    DELETE RECORD([従業員])                    `従業員を削除する
End if
```

参照

Locked、トリガ

Records in table

Records in table {(テーブル)} → 数値

引数	タイプ		説明
テーブル	テーブル	→	レコード数を求めるテーブル 省略した場合、デフォルトテーブル
戻り値	数値	←	テーブルの合計レコード数

説明

Records in table関数は、＜テーブル＞中の合計レコード数を返します。**Records in selection**関数は、カレントセレクションのレコード数を返します。**Records in table**関数がトランザクション内で使用される場合は、トランザクション中に作成されたレコードが考慮に入れられます。

例題

▼以下の例は、テーブルのレコードの数を示す警告を表示します。**Records in table**関数によって返されたレコード数を文字列に変換していることに注目してください。

```
ALERT("従業員テーブルには" + String(Records in table([従業員])) +
                                    "件のレコードがあります。")
```

参照

Records in selection

Record number

Record number {(テーブル)} → 数値

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードのレコード番号を求めるテーブル 省略した場合、デフォルトテーブル
戻り値	数値	←	カレントレコード番号

説明

Record number関数は、＜テーブル＞のカレントレコードの絶対レコード番号を返します。カレントレコードがない場合（レコードポインタがカレントセレクションの前後にある場合等）には-1を返します。カレントレコードが保存されていない新しいレコードの場合には-3を返します。

レコード番号は変更することができます。削除されたレコードのレコード番号は、再利用されます。4D Toolsを使用してデータベースを圧縮する、または「検査と修復」メニューの「データをタグにより複製した後、再作成する（数分）」オプションを使用して、データベースを修復すると、レコード番号は変更されます。トランザクション処理の間、新しく作成されたレコードは一時的なレコード番号を持ちます。トランザクション処理が受け入れられると、そのレコードは正式なレコード番号が割り当てられます。

例題

▼以下の例は、カレントレコードのレコード番号を変数に格納し、他に同じデータを持つレコードがないかを検索します。

```
$レコード番号:=Record number([従業員]) ` レコード番号を求める
QUERY([従業員] ; [従業員]名字 = [従業員]名字) ` 他に同じ名字がないかを検索する
      ` 同じ名字の従業員が見つかったら警告ボックスにそのレコード数を表示する
ALERT("同じ名字の従業員が" + String(Records in selection([従業員])) +
                                                    "件あります。")
GOTO RECORD([従業員] ; $レコード番号)      ` 元のレコードに戻る
```

参照

レコードに付けられた番号の使用、GOTO RECORD、Is new record、Selected record number、Sequence number

GOTO RECORD

GOTO RECORD({テーブル ;} レコード番号)

引数	タイプ		説明
テーブル	テーブル	→	移動するレコードの属するテーブル 省略した場合、デフォルトテーブル
レコード番号	数値	→	**Record number**関数で求めたレコード番号

説明

GOTO RECORDコマンドは、＜テーブル＞の特定のレコードをカレントレコードとして選択します。引数＜レコード番号＞は、**Record number**関数で求めたレコード番号です。このコマンドを実行するとセレクションは、選択されたレコード1件だけになります。

＜レコード番号＞がデータベースの中で最も小さいレコード番号よりも小さい場合や、データベースの中で最も大きいレコード番号よりも大きい場合は、4th Dimensionからレコード番号が範囲外である旨のエラーメッセージが表示されます。レコードが削除されたレコードのレコード番号と等しい場合には、セレクションは空になります。

注：このコマンドでは、トランザクション中に発行された仮のレコード番号を使用してはいけません。

前節の**Record number**関数の例を参照してください。

参照

レコードに付けられた番号の使用、Record number

Sequence number

Sequence number {(テーブル)} → 数値

引数	タイプ		説明
テーブル	テーブル	→	一連番号を求めるレコードの属するテーブル 省略した場合、デフォルトテーブル
戻り値	数値	←	一連番号

説明

Sequence number関数は、＜テーブル＞の以下の一連番号を返します。一連番号は、各テーブルに対して固有のものです。この番号は、＜テーブル＞に対して新しいレコードが追加されるたびに加算される、決して重複することのない番号です。番号は、1から始まります。

#N記号を使用する代わりに**Sequence number**関数を使用しなければならない場合を次に4つ示します。

■ 新しいレコードをフォームを使用せずに、メソッドを使用して作成した場合

■ 1以外の番号から始める必要がある場合

■ 番号の増分値に1以上の数を使用する必要がある場合

■ 一連番号を他のコードの一部に使用する場合（部品コードの一部に使用する場合等）

メソッドを使用して一連番号を各レコードに格納するには、テーブル上に倍長整数型のフィールドを作成し、そのフィールドに一連番号を代入します。

一連番号は、フォームのフィールドにデフォルト値として#N記号を設定した場合に得られる番号と同じです。デフォルト値の設定に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

一連番号が1以外の数値から始まる必要がある場合には、一連番号に対してその差を加算するだけで構いません。例えば、番号が1000から始まる必要がある場合には、以下のようなステートメントを使用します。

```
[テーブル]一連番号ﾌｨｰﾙﾄﾞ:= Sequence number([テーブル]) + 999
```

例題

▼以下の例は、フォームメソッドの一部です。このステートメントは、まずレコードが新しいものかどうか検査（請求書番号が空の文字列であるかどうかで判断）します。新しいレコードであれば、請求書番号を設定します。この請求書番号は、2つの情報から成り立っています。それは、一連番号とデータベースを開くときに入力された操作番号です。一連番号を、5桁の文字列として扱います。

```
If([請求書]請求書番号 = "")  ` 新しい請求書番号を設定？
        ` 新しい請求書番号を作成、請求書番号は文字化した一連番号と操作番号で構成する
    [請求書]請求書番号:= String(Sequence number ; "00000")
                                                + [請求書]操作番号
End if
```

参照

About Record Numbers、Record number、Selected record number

レコードに付けられた番号の使用

レコードに付けられた番号には、以下の3つがあります。

■ レコード番号

■ レコード位置番号

■ 一連番号

レコード番号

レコード番号は、レコードに対する絶対的な番号です。この番号は、レコードが作成されるごとに自動的に付けられます。レコードを削除するか、4D Toolsでテーブルを恒久的にソートするまで、レコード番号は変わることはありません。レコード番号はゼロから始まります。レコード番号は、削除されたレコード番号が新しいレコードに再利用されるため、それ独自のものではありません。また、レコード番号は4D Toolsを使用してテーブルを恒久的にソートする、またはデータベースを圧縮、修復すると変更されてしまいます。トランザクションに追加された新しいレコードには仮のレコード番号が割り当てられます。トランザクションが確定されたときに、正式なレコード番号が割り当てられます。

レコード位置番号

レコード位置番号は、カレントセレクション中のレコードの位置を示す番号です。したがって、この番号はカレントセレクションに依存します。セレクションを変更、またはソートした場合には、レコード位置番号が変更されることがあります。レコード位置番号に番号を付ける場合は1から始まります。

一連番号

一連番号は、レコードのフィールドに割り当てることができる、重複しない番号です。この番号は、各レコードに自動的に付けられるものではありません。一連番号は、1から始まり新しいレコードを作成するごとに加算されていきます。レコード番号と違って、一連番号はレコードが削除されたり、またはデータベースの圧縮、修復および恒久ソートの場合でも再利用されません。そのため、一連番号を使用して、レコードに重複しないID番号を持たせることができます。トランザクション処理中に一連番号が増えると、その番号はトランザクション処理が取り消されても、減少することはありません。

レコードに付けられた番号の例

次ページの表は、レコードに付けられた番号について説明しています。表の各行は、レコードについての情報を示しています。各行の順序は、レコードが出力フォームに表示される順序です。

■ データの列は、各レコードのデータの内容で、この例では人名を表しています。

■ **レコード番号の列**は、絶対的なレコード番号です。この番号は、**Record number**関数によって求められます。

■ **レコード位置番号の列**は、カレントセレクション中の位置を示す番号です。この番号は、**Selected record number**関数によって求められます。

■ **一連番号の列**は、各レコードに固有の一連番号です。この番号は、レコードが作成されたときに**Sequence number**関数によって返される番号です。この一連番号はフィールドに格納されています。

レコードの入力後

最初の表は、入力された後のレコードを示しています。

■ レコードのデフォルトの順序は、レコード番号により設定されます。

■ レコード番号は0から始まります。

■ レコード位置番号と一連番号は、ともに1から始まります。

データ	レコード番号	レコード位置番号	一連番号
芦沢	0	1	1
牧	1	2	2
寺島	2	3	3
北野	3	4	4
大杉	4	5	5

注：ソートを行わずにカレントセレクションを変更するコマンドの実行後、レコードのデフォルト順序は変わりません。例えば、「ユーザ」モードで「すべてを表示」メニューを選択した後やALL RECORDSコマンドを実行した後の場合です。

レコードのソート後

以下の表は、上の表と同じセレクションを名前でソートした後の状態を示しています。

■ レコード番号は、各レコードに付けられたままです。

■ レコード位置番号は、ソートされたセレクションの中の新しい位置を示しています。

■ 一連番号は、各レコードが作成されたときに付けられたままなため変わりません。

データ	レコード番号	レコード位置番号	一連番号
大杉	4	1	5
寺島	2	2	3
北野	3	3	4
牧	1	4	2
芦沢	0	5	1

レコードの削除後

以下の表は、“北野”のレコードが削除されたセレクションの状態を示しています。

■ レコード位置番号だけが変更されています。レコード位置番号はレコードが表示される順番を反映します。

データ	レコード番号	レコード位置番号	一連番号
大杉	4	1	5
寺島	2	2	3
牧	1	3	2
芦沢	0	4	1

レコードの追加後

以下の表は、“岸本”という新しいレコードが追加されたセレクションの状態を示しています。

■ 新しいレコードは、カレントセレクションの最後に加えられます。

■ “北野”のレコード番号が新しいレコード（“岸本”）に再使用されます。

■ 一連番号は加算され続けます。

データ	レコード番号	レコード位置番号	一連番号
芦沢	0	1	1
牧	1	2	2
寺島	2	3	3
大杉	4	4	5
岸本	3	5	6

レコードの変更およびソート後

以下の表は、3つのレコードが選択され、さらにソートされたセレクションの状態を示しています。

■ 各レコードのレコード位置番号だけが変更されています。

データ	レコード番号	レコード位置番号	一連番号
寺島	2	1	3
岸本	3	2	6
牧	1	3	2

参照

Record number、Selected record number、Sequence number

PUSH RECORD

PUSH RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	プッシュするレコードの属するテーブル 省略した場合、デフォルトテーブル

説明

PUSH RECORDコマンドは、＜テーブル＞のカレントレコード（それに付随するサブレコードも含めて）を、そのテーブルのレコードスタックにプッシュします。**PUSH RECORD**コマンドは、レコードがディスクに保存される前でも実行することができます。

アンロックされたレコードをプッシュした場合、レコードはポップするかアンロードされるまですべてのユーザやプロセスに対してロックされた状態になります。

例題

▼以下の例は、“顧客”テーブルのレコードをレコードスタックにプッシュします。

```
PUSH RECORD([顧客])      ` 顧客レコードをスタックにプッシュする
```

参照

POP RECORD、レコードスタックの使用

POP RECORD

POP RECORD({テーブル})

引数	タイプ		説明
テーブル	テーブル	→	ポップするレコードの属するテーブル 省略した場合、デフォルトテーブル

説明

POP RECORDコマンドは、＜テーブル＞のレコードを、そのテーブルのレコードスタックからポップし、そのレコードをカレントレコードにします。

レコードをプッシュして、プッシュしたレコードを含まないようにカレントセレクションを変更し、その後でレコードをポップした場合、カレントレコードはカレントセレクションに含まれません。ポップしたレコードをカレントセレクションのレコードにしたい場合には、**ONE RECORD SELECT**コマンドを実行しなければなりません。レコードを保存する前にレコードポインタを移動するようなコマンドを実行した場合は、メモリ上のコピーを失うことになります。

例題

▼以下の例は、“顧客”テーブルのレコードをポップしてレコードスタックからポップします。

```
PUSH RECORD([顧客])      `カレントレコードをプッシュする
QUERY([顧客])            `いくつかのレコードを検索する
POP RECORD([顧客])       `顧客レコードをスタックからポップする
ONE RECORD SELECT([顧客]) `ポップしたレコードをカレントレコードにする
```

参照

PUSH RECORD、レコードスタックの利用

レコードスタックの使用

PUSH RECORDコマンドと**POP RECORD**コマンドは、レコードをレコードスタックに格納（プッシュ、PUSH）、またはレコードをレコードスタックから削除（ポップ、POP）します。

プロセスごとに各テーブルは、固有のレコードスタックを持っています。4th Dimensionは、各レコードスタックを後入れ先出し法（Last-In-First-Out:LIFO）で管理します。レコードスタックの容量は、メモリの容量によって制限されます。

PUSH RECORDコマンドと**POP RECORD**コマンドを使用する場合には注意が必要です。プッシュされた各レコードは、メモリの空いている部分に格納されます。数多くのレコードをプッシュしすぎると、メモリの空きがなくなったり、または“スタックがいっぱいです”状態になります。

4th Dimensionは、メソッドを終了してメニューに戻った時点でレコードスタックにポップされないレコードがあれば、レコードスタックを消去します。

PUSH RECORDコマンドと**POP RECORD**コマンドは、データの入力中に同じテーブルの内容を調べるとき等に使用すると非常に便利です。これを実行するためには、まずレコードをプッシュします。次に検索を行い、ファイルのレコードを検査（例えば、フィールドの内容を変数にコピーする）します。最後にレコードを元の状態に戻すためにポップします。

バージョン3ユーザへの注意：レコード入力中に、複数のフィールドの重複しない値をチェックする必要がある場合、新しいコマンドであるSET QUERY DESTINATIONコマンドを使用してください。このコマンドにより、カレントレコードに入力したデータを保存しておくため、QUERYコマンドの前後にPUSH RECORDコマンドとPOP RECORDコマンドを呼び出す必要はなくなります。SET QUERY DESTINATIONコマンドを使用して、セレクションやカレントレコードを変更しない検索を実行することができます。

参照

POP RECORD、PUSH RECORD、SET QUERY DESTINATION

リレート

この章のコマンド、特に**RELATE ONE**コマンドと**RELATE MANY**コマンドは自動的および自動的でないテーブル間のリレートを設定、管理するものです。この章のコマンドを使用する前に、テーブル間のリレートの作成について『4th Dimensionデザインリファレンス』を参照してください。

コマンドを使用したテーブルの自動リレート

2つのテーブルは自動リレートで関連付けることができます。一般的にテーブルの自動リレートは、リレート先テーブル（nテーブル）のレコードをロード、または選択するために使用します。リレートを使用することで多くの処理を実行することができます。

以下のような処理が含まれます。

■ データ入力

■ 出力フォームによる画面上のリスト出力

■ 帳票の印刷

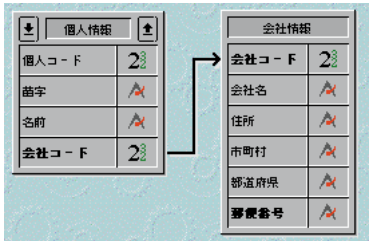
■ セレクションに対する検索、ソート、フォーミュラでの更新

性能を最大限に引き出すために、4th Dimensionはテーブルに対してカレントレコードとなるレコードのみ自動リレートを設定します。上記のリストに示した各操作を行ったときに、以下のルールに従って、リレートレコードがロードされます。

■ リレートテーブルの1レコードのみ選択するリレートの場合は、そのレコードはディスクからロードされます。

■ 複数のレコードを選択するリレートの場合には、新しいレコードセレクションがそのテーブルに作成され、セレクション内の先頭レコードがディスクからロードされます。

以下の図のテーブルストラクチャを例にとってみましょう。[個人]テーブルをデータ入力のためにロードした場合は、[会社]テーブルからリレート先レコードが1件選択され、ロードされます。同様に[会社]テーブルをロードしてデータ入力用に表示した場合には、[個人]テーブルから関連する複数のレコードが選択されます。



上図において、[個人]テーブルは“nテーブル”として参照され、[会社]テーブルは“1テーブル”として参照されます。これを解かりやすくするために、多くの人が1つの会社に関連しており、各会社には多くの人がいると考えてください。

同様に[個人]テーブルの“会社”フィールドはnフィールドとして参照され、[会社]テーブルの“会社名”フィールドは1フィールドとして参照されます。

リレートフィールドが、いつもユニーク（重複しない）とは限りません。例えば、“[会社]会社名”フィールドは同じ値を含んだ会社のレコードをいくつも持っているかもしれません。こういう場合は、常にユニークであるリレート先テーブルの別のフィールドでリレート作成することにより、簡単に処理することができます。このようなフィールドの例として、会社コードがあります。

以下の表に掲げたコマンドは、コマンドの実行中にリレート先のリレートレコードをロードするために自動リレートを使用します。これらのコマンドはすべて「n対1」のリレートを自動的に作成します。「1対n」の欄が○になっているコマンドのみが1対nのリレートに対応することができます。

コマンド	1対n
ADD RECORD	○
ADD SUBRECORD	×
APPLY TO SELECTION	×
DISPLAY SELECTION	×
EXPORT DIF	×
EXPORT SYLK	×
EXPORT TEXT	×
EXPORT DATA	×
MODIFY RECORD	○
MODIFY SUBRECORD	×
MODIFY SELECTION	○(データ入力時)
ORDER BY	×
ORDER BY FORMULA	×
PRINT LABEL	×
PRINT SELECTION	○

コマンド	1対n
QUERY	○
QUERY BY FORMULA	○
QUERY SELECTION	○
REPORT	×
SELECTION TO ARRAY	×
SELECTION RANGE TO ARRAY	×

テーブルリレートを実行するコマンド

自動リレートとは、コマンドがレコードをロードするたびに、そのテーブルに関連する1つまたは複数のレコードを自動的に選択するという意味ではありません。レコードをロードするコマンドを使用した後で、リレート先データにアクセスする必要がある場合には、**RELATE ONE**コマンドまたは**RELATE MANY**コマンドを使用して、リレート先レコードを明示的に選択することが必要なケースもあります。

前ページの表に掲げたコマンドの一部（**QUERY**コマンド等）は、処理の終了後にカレントレコードをロードします。この場合、ロードされたレコードは、リレート先の関連するレコードを自動的には選択しません。また、ここでも、リレート先データにアクセスする必要がある場合には、**RELATE ONE**コマンドまたは**RELATE MANY**コマンドを使用してリレート先レコードをを明示的に選択する必要があります。

参照

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

SET AUTOMATIC RELATIONS

SET AUTOMATIC RELATIONS(1リレート {; nリレート})

引数	タイプ		説明
1リレート	ブール	→	n対1のリレート
nリレート	ブール	→	1対nのリレート

説明

SET AUTOMATIC RELATIONSコマンドは、データベース全体のマニュアルリレートを1次的に自動リレートに変更します。リレートは、次に**SET AUTOMATIC RELATIONS**コマンドを使用するまで自動リレートのままになります。

＜1リレート＞がTrueの場合、すべてのn対1のマニュアルリレートを自動リレートに設定します。＜1リレート＞がFalseの場合、前もって自動リレートにしたすべてのn対1リレートがマニュアルリレートに戻ります。

＜nリレート＞も1対nリレートに対して同じように作用します。

このコマンドは、「デザイン」モードで既に自動リレートに設定されたものに対しては無効です。

このコマンドを使用すれば、「デザイン」モードでマニュアルに設定したリレートに対して自動リレートを必要とする処理の直前で自動リレートに切り替えることができます(例えば、リレーショナル検索やソート処理)。処理が終了した後で、再度マニュアルリレートに戻すことができます。

例題

▼以下の例は、すべてのn対1のマニュアルリレートを自動リレートに設定し、前もって自動リレートにした1対nリレートをマニュアルリレートに戻します。

```
SET AUTOMATIC RELATIONS(True ; False)
```

参照

GET AUTOMATIC RELATIONS、リレート、SELECTION RANGE TO ARRAY、SELECTION TO ARRAY、SET FIELD RELATION

GET AUTOMATIC RELATIONS

GET AUTOMATIC RELATIONS (1対1; n)

引数	タイプ		説明
1対1	ブール	←	すべてのn対1リレートのステータス
nブール	←		すべての1対nリレートのステータス

説明

GET AUTOMATIC RELATIONSコマンドにより、データベースで設定されたすべてのマニュアルn対1リレートおよび1対nリレートに関する自動／マニュアルのステータスがカレントプロセスにおいて変更されたかどうかを知ることができます。

■ 1対1：前回の**SET AUTOMATIC RELATIONS**コマンドの呼び出しにより、すべてのマニュアルn対1リレートが自動に設定された場合、この引数は“True”を返します（例えば**SET AUTOMATIC RELATIONS**(True;False)）。

また、**SET AUTOMATIC RELATIONS**コマンドがコールされなかった場合や、前回の実行によりマニュアルn対1リレートが変更されなかった場合（例えば**SET AUTOMATIC RELATIONS**(False;False)）には、“False”を返します。

■ 1対n：前回の**SET AUTOMATIC RELATIONS**コマンドの呼び出しにより、すべてのマニュアル1対nリレートが自動に設定された場合、この引数は“True”を返します（例えば**SET AUTOMATIC RELATIONS**(True;True)）。

また、**SET AUTOMATIC RELATIONS**コマンドがコールされなかった場合や、前回の実行によりマニュアル1対nリレートが変更されなかった場合（例えば**SET AUTOMATIC RELATIONS**(True;False)）は、“False”を返します。

例題

▼ **GET FIELD RELATION**コマンドの例題を参照してください。

参照

GET FIELD RELATION、GET RELATION PROPERTIES、SET AUTOMATIC RELATIONS.

SET FIELD RELATION

SET FIELD RELATION (nテーブル | 始点フィールド; 1対1; 1対n})

引数	タイプ		説明
nテーブル¦ 始点フィールド	テーブル¦フィールド	→	リレートの開始点となるテーブル、 またはリレートの開始点となる フィールド
1対1	倍長整数¦ブール	→	フィールドから開始するn対1の リレートのステータス、または テーブルのn対1のリレート
1対n	倍長整数¦ブール	→	フィールドから開始する1対nの リレートのステータス、または テーブルの1対nのリレート

説明

SET FIELD RELATIONコマンドを使用すると、「デザイン」モードの「リレートプロパティ」ウインドウで設定した初期ステータスが何であれ、データベースの各リレートの自動／マニュアルのステータスを個別に設定することができます。

■ 1番目の引数には、テーブルやフィールドの名前を渡します。

- ■ フィールド名（＜始点フィールド＞）を渡すと、コマンドは、指定した始点フィールドから開始するリレートに対してのみ適用されます。
- ■ テーブル名（＜nテーブル＞）を渡すと、コマンドは指定したnテーブルから開始するリレートに対してのみ適用されます。
- ■ ＜始点フィールド＞または＜nテーブル＞から開始するリレートが存在しない場合、引数＜1対1＞と＜1対n＞には0が返されます。シンタックスエラー番号16（“このフィールドにはリレートが設定されていません。”）が生成されて、システム変数OKには0が代入されます。

■ 引数＜1対n＞には、指定した1対nリレートに適用される自動／マニュアルのステータス変更を示す値を渡します。この値として、「Relations」テーマの定数を使用できます。

- ■ Do not modify(0) = 1対nリレートの現在のステータスを変更しない。
- ■ Structure configuration(1) = アプリケーションのストラクチャウインドウで指定された1対nリレートの設定を使用する。
- ■ Manual(2) = カレントプロセスに対し、1対nリレートをマニュアルに設定する。
- ■ Automatic(3) = カレントプロセスに対し、1対nリレートを自動に設定する。

注：このコマンドを使用して行った変更は、カレントプロセスに対してのみ適用されます。「リレートプロパティ」ウインドウのオプションを用いて指定されたリレート設定は変更されません。

例題

▼ このコマンドを使用することにより、クイックレポートのリレート管理がさらに容易になります。以前のバージョンの4th Dimensionでは、エディタ上でリレートを利用するには、すべてのリレートを自動に設定する必要がありました。次のコードを使用すると、有用なリレートだけを自動に設定できるようになります。

```
SET AUTOMATIC RELATIONS(False;False) `リレートのリセット
      `次のリレートだけが使用される
SET FIELD RELATION([Invoices]ID_Cus;Automatic;Automatic)
SET FIELD RELATION([Invoice_row]ID_Invoice;Automatic;Automatic)
QR REPORT([Invoices];Char(1);True;True;True)
```

参照

GET AUTOMATIC RELATIONS、GET FIELD RELATION、GET RELATION PROPERTIES、SET AUTOMATIC RELATIONS

GET FIELD RELATION

GET FIELD RELATION (始点フィールド; 1対1; 1対n{; *})

引数	タイプ		説明
始点フィールド	フィールド	→	リレートの開始点となるフィールド
1対1	倍長整数	←	n対1のリレートのステータス
1対n	倍長整数	←	1対nのリレートのステータス
*	*	→	渡した場合：＜1対1＞と＜1対n＞にはリレートのカレントステータスが返される（値“2”または“3”のみ） 省略した場合（デフォルト）：プログラムによりリレートが変更されていなければ、＜1対1＞と＜1対n＞には値“1”が返される

説明

GET FIELD RELATIONコマンドを使用すると、カレントプロセスの＜始点フィールド＞から開始するリレートの自動／マニュアルのステータスを調べることができます。「ストラクチャ」ウインドウで設定した自動リレートをはじめとして、あらゆるリレートを調べることができます。

■ ＜始点フィールド＞には、ステータスを調べようとするリレートが開始するnテーブルのフィールド名を渡します。フィールド＜始点フィールド＞から開始するリレートが存在しない場合、引数＜1対1＞と＜1対n＞には0が返されます。また、エラーが返されて、システム変数OKには0が代入されます（後述）。

■ コマンドの実行後、引数＜1対1＞には、指定したn対1リレートが自動に設定されているかどうかを示す値が格納されます。

- 0=＜始点フィールド＞から始まるリレートが存在しません。シンタックスエラー番号16（“このフィールドにはリレートが設定されていません。”）が生成され、システム変数OKには0が代入されます。
- 1=指定されたn対1リレートの自動／マニュアルのステータスは、「デザイン」モードの「リレートプロパティ」内の「自動1対1リレート」オプションにより設定されたものです（プログラムにより変更されていない）。
- 2=そのプロセスのn対1のリレートはマニュアルです。
- 3=そのプロセスのn対1のリレートは自動です。

■ コマンドの実行後、引数＜1対n＞には、指定した1対nリレートが自動として設定されているかどうかを示す値が格納されます。

■ 0=引数＜始点フィールド＞から始まるリレートが存在しません。シンタックスエラー番号16（“このフィールドにはリレートが設定されていません。”）が生成され、システム変数OKには0が代入されます。

■ 1=指定された1対nリレートの自動／マニュアルのステータスは、「デザイン」モードの「リレートプロパティ」にある「自動1対nリレート」オプションにより設定されたものです（プログラムにより変更されていない）。

■ 2=そのプロセスの1対nのリレートはマニュアルです。

■ 3=そのプロセスの1対nのリレートは自動です。

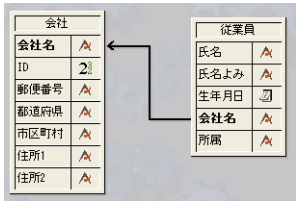
引数＜1対1＞および＜1対n＞に返された値は、「リレート」テーマの定数と比較することができます。

定数	タイプ	値
No relation	倍長整数	0
Structure configuration	倍長整数	1
Manual	倍長整数	2
Automatic	倍長整数	3

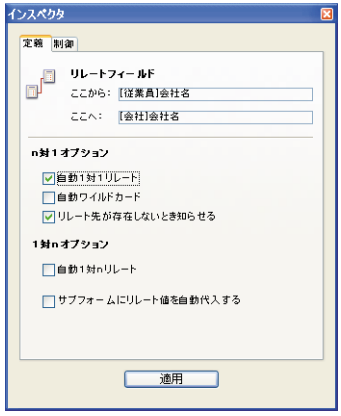
■ 任意の引数＜*＞を使用すると、プログラムから修正されていない場合でも、リレートのカレントステータスを“強制的に”読み込むことができます。言い換えれば、任意の引数＜*＞を渡した場合、引数＜1対1＞および＜1対n＞には値“2”または“3”だけが返されます。

例題

▼次のようなストラクチャについて検証しましょう。



[従業員]会社フィールドから[会社]名前フィールドへリンクするリレートのプロパティは次の通りです。



次のコードは、**GET FIELD RELATION**コマンドおよび**SET FIELD RELATION**コマンドにより提供されるさまざまな機能とともに、その効果を示します。

```
C_BOOLEAN($is_one;$is_many)
C_LONGINT(one;many)

GET AUTOMATIC RELATIONS($is_one;$is_many)  `False, Falseを返す
GET FIELD RELATION([従業員]会社名;one;many)  `1、1を返す
GET FIELD RELATION([従業員]会社名;one;many;*)  `3、2を返す
SET FIELD RELATION([従業員]会社名;2;0)

      `n 対 1 リレートをマニュアルに変更する
GET FIELD RELATION([従業員]会社名;one;many)  `2、1を返す
GET FIELD RELATION([従業員]会社名;one;many;*)  `2、2を返す
SET FIELD RELATION([従業員]会社名;1;0)

      `n 対 1 リレートに対し
      `「デザイン」モードで設定された引数を再設定する
GET FIELD RELATION([従業員]会社名;one;many) `1、1を返す
GET FIELD RELATION([従業員]会社名;one;many;*) `3、2を返す
SET AUTOMATIC RELATIONS(True;True) ` 全テーブルのすべてのリレートを
                                                    自動に変更する
GET AUTOMATIC RELATIONS($is_one;$is_many)  `True、True を返す
GET FIELD RELATION([従業員]会社名;one;many) `1、1を返す
GET FIELD RELATION([従業員]会社名;one;many;*) `3、3を返す
```

▼次は、**GET FIELD RELATION**と**SET FIELD RELATION**の2つのコマンドの代表的な使用方法を示す例題です。

```
C_LONGINT(one,many)
```

```
    `現在の引数を読み込む
GET FIELD RELATION([MyTable]MyField;one;many)
    `n対1リレートをマニュアルに変更し、1対nリレートは自動のままにする
SET FIELD RELATION([MyTable]MyField;3;2)
QUERY(...) ` クエリの実行
    `元の引数の再設定
SET FIELD RELATION([MyTable]MyField;one;many)
```

参照

GET AUTOMATIC RELATIONS、GET RELATION PROPERTIES、SET AUTOMATIC RELATIONS、SET FIELD RELATION

RELATE ONE

RELATE ONE(nテーブル | nフィールド {; 選択フィールド})

引数	タイプ		説明
nテーブル¦ または nフィールド	テーブル¦ フィールド	→	すべての自動リレートを設定するテーブル、 1テーブルにマニュアルリレートするフィールド
選択フィールド	フィールド	→	1テーブルから選択したフィールド

説明

RELATE ONEコマンドには、2つの形式があります。

第1の形式“**RELATE ONE**（nテーブル）”は、カレントプロセスのすべての自動的なn対1のリレートを＜nテーブル＞に対して設定します。つまり、このコマンドは、＜nテーブル＞の各フィールドに対して自動的なn対1のリレートを行い、各リレート先テーブルのリレートレコードを選択します。これは、そのプロセスのリレート先テーブルにあるカレントレコードを変更します。

第2の形式“**RELATE ONE**（nフィールド;選択フィールド）”は、＜nフィールド＞に関連するレコードを選択します。自動リレートである必要はありません。レコードが存在する場合、**RELATE ONE**コマンドは、リレート先レコードを1つメモリにロードし、これをそのテーブルのカレントレコードおよびカレントセレクションにします。

オプション引数の＜選択フィールド＞は、＜nフィールド＞が文字フィールドである場合にのみ指定することができます。＜選択フィールド＞は、リレート先のフィールドでなければなりません。＜選択フィールド＞は、文字フィールド、数値フィールド、日付フィールド、時間フィールド、またはブールフィールドでなければなりません。つまりテキストフィールド、ピクチャフィールド、BLOBまたはサブテーブルフィールドを選択することはできません。

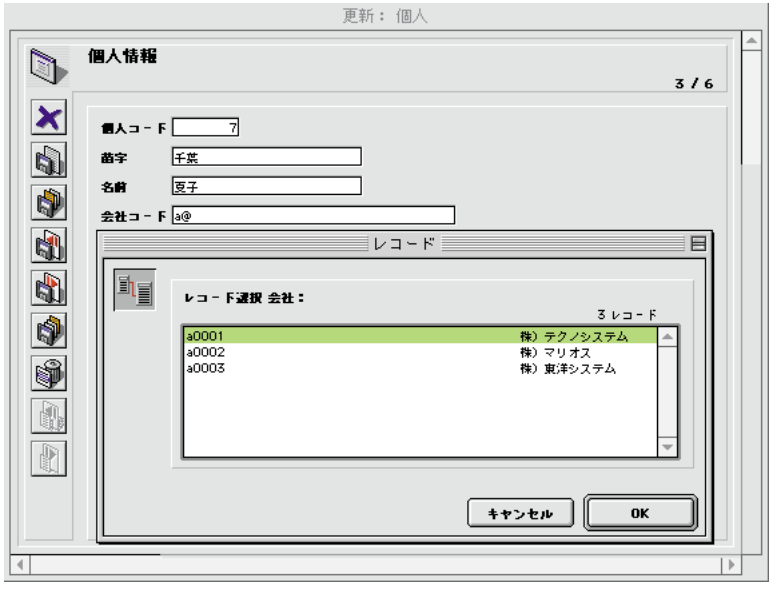
＜選択フィールド＞を指定し、リレート先テーブルで複数のレコードを発見した場合は、**RELATE ONE**コマンドは＜nフィールド＞の内容と一致するレコードを選択リストに表示します。この選択リストは、左の欄にリレート先フィールドの内容を、右の欄に選択フィールドの内容を表示します。

＜nフィールド＞の内容がワイルドカード記号（@）で終了する場合、複数のレコードが見つかる可能性があります。一致するレコードが1件しかなければ、選択リストは表示されません。＜選択フィールド＞を指定することは、テーブルのリレートを設定する時点でワイルドカード選択を指定するのと同じことです。ワイルドカード選択に関する詳細は、『4th Dimensionデザインリファレンス』を参照してください。

RELATE ONEコマンドは、サブテーブルに対するリレート機能もありますが、関連付けが適切に行われるように、親テーブルとサブテーブルのリレート先フィールドの両方にリレートを実行する必要があります。サブテーブルに対するリレートを使用する場合は、1回目の**RELATE ONE**コマンドでリレート先レコードをロードし、2回目の**RELATE ONE**コマンドで関連するサブレコードをロードします。

例題

1. 以下の図は、入力されたレコードと、その前面に表示された選択リストを示しています。

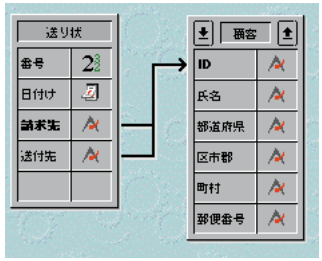


以下のコマンドを実行すると選択リストを表示します。

RELATE ONE([個人]名前 ; [会社]所在地)

“ACME”で始まる名称を持つすべての会社のリストを、その会社の所在地と共に表示させるために“ACME @”と入力しています。

2. 以下の例は、2つの自動的でないリレートで[送り状]テーブルと[顧客]テーブルが関連付けられています。1つのリレートは、“[送り状]請求先”フィールドから“[顧客]ID”フィールドに対するものです。もう1つのリレートは、“[送り状]送付先”フィールドから“[顧客]ID”フィールドに対するものです。



以下の図は、[送り状]テーブルの“請求先”と“送付先”を表示するための[送り状]テーブルに属するフォームを示しています。



両方とも同じテーブル（[顧客]テーブル）へのリレートであるため、“請求先”と“送付先”の情報を同時に取得することはできません。したがって、フォーム上にある両方の住所は、変数と**RELATE ONE**コマンドを使用して表示します。もし[顧客]テーブルのフィールドを使用して表示した場合には、一方のリレートから得られたデータしか表示しません。

以下の2つのメソッドは、“[送り状]請求先”フィールドと“[送り状]送付先”フィールドに対するオブジェクトメソッドです。次は、“[送り状]請求先”フィールドに対するオブジェクトメソッドです。

```
RELATE ONE([送り状]請求先 ; [顧客]都道府県)
v都道府県1:=[顧客]都道府県
v区市郡1:=[顧客]区市郡
v町村1:=[顧客]町村
v郵便番号1:=[顧客]郵便番号
```

次は、“[送り状]送付先”フィールドに対するオブジェクトメソッドです。

```
RELATE ONE([送り状]送付先 ; [顧客]都道府県)
v都道府県2:=[顧客]都道府県
v区市郡2:=[顧客]区市郡
v町村2:=[顧客]町村
v郵便番号2:=[顧客]郵便番号
```

参照

OLD RELATED ONE、RELATE MANY

RELATE MANY

RELATE MANY(1テーブル | 1フィールド)

引数	タイプ		説明
1テーブル ¦	テーブル ¦	→	すべての1対nの自動リレートを設定するテーブル、または
1フィールド	フィールド		1フィールド

説明

RELATE MANYコマンドには、2つの形式があります。

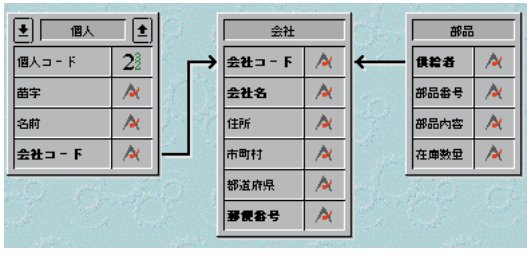
第1の形式“**RELATE MANY**（1テーブル）”は、＜1テーブル＞に対してすべての1対nの自動リレートを設定します。このコマンドは、＜1テーブル＞に対して1対nの自動リレートを持つ各テーブルのカレントセレクションを更新します。つまり、nテーブルのカレントセレクションは、1テーブルのそれぞれのリレート先フィールドの現在値を反映します。このコマンドが実行される時はいつでも、nテーブルのカレントセレクションが再構成されます。

第2の形式“**RELATE MANY**（1フィールド）”は、＜1フィールド＞に対して1対nのリレートを設定します。これは、＜1フィールド＞と関連を持つテーブルのみに対しカレントセレクションを変更します。つまり、リレート先テーブルの関連するレコードは、すべてnテーブルのカレントセレクションになることを意味します。

注：RELATE MANYコマンドを実行する際に、1テーブルのカレントセレクションが空の場合、このコマンドは何も行いません。

例題

以下の例は、3つのテーブルが自動リレートで関連付けられています。[個人]テーブルと[部品]テーブルは、両方とも[会社]テーブルに対してn対1のリレート関係にあります。



以下の図は、[個人]テーブルと[部品]テーブルの両方の関連するレコードを表示するための、[会社]テーブルのフォームを示しています。



フォームを表示すると、[個人]テーブルと[部品]テーブルから関連するレコードはロードされ、それぞれのテーブルのカレントセレクションとなります。しかし、[会社]テーブルをメソッド中のコマンドで選択する場合には、リレート先レコードはロードされません。このような場合には、**RELATE MANY**コマンドを使用する必要があります。

注：RELATE MANY関数は空白の選択に適用されたとき、コマンドは実行されずMANYテーブルのセレクトは変わりません。

以下のメソッドは、[会社]テーブルの各レコードに対し、警告ボックスを表示します。警告ボックスには、会社にいる人数（[個人]テーブル中の関連するレコードの数）、供給する部品の種類数（[部品]テーブル中の関連するレコードの数）を表示します。この例題では、印刷の関係上**ALERT**コマンドの引数が複数行にわたっています。**RELATE MANY**コマンドが、たとえ自動リレートの場合でも必要なことに注目してください。

```
ALL RECORDS([会社])   ` [会社]テーブルのレコードをすべて選択
ORDER BY([会社] ; [会社]会社名)   ` 会社名でソート
For($i ; 1 ; Records in Table([会社]))      ` [会社]レコード数だけ繰り返す
      RELATE MANY([会社]会社名)      ` リレートレコードを選択
      ALERT("会社：" + [会社]会社名 + Char(13) +"会社にいる人数：" + String
            (Records in selection([個人])) + Char(13) + "供給する部品の種類数："
                                          + String(Records in selection([部品])))
      NEXT RECORD([会社])      ` 以下の[会社]レコードへ移動
End for
```

参照

OLD RELATED MANY、RELATE ONE

CREATE RELATED ONE

CREATE RELATED ONE(フィールド)

引数	タイプ		説明
フィールド	フィールド	→	nフィールド

説明

CREATE RELATED ONEコマンドには、2つの機能があります。関連するレコードが<フィールド>に対して存在しない場合（つまり、<フィールド>の現在の値に一致するものがない場合）は、**CREATE RELATED ONE**コマンドは、新しくリレート先レコードを作成します。

適当なフィールドに値を保存するには、nフィールドから1フィールドへ値を割り当てます。**SAVE RELATED ONE**コマンドを実行して、この新しいレコードを保存します。

リレート先レコードが存在する場合は、**RELATE ONE**コマンドと全く同じようにそのレコードをロードします。

参照

SAVE RELATED ONE

SAVE RELATED ONE

SAVE RELATED ONE(フィールド)

引数	タイプ		説明
フィールド	フィールド	→	nフィールド

説明

SAVE RELATED ONEコマンドは、＜フィールド＞にリレートするレコードを保存します。**CREATE RELATED ONE**コマンドで新しく作成したレコードを更新する、または**RELATE ONE**コマンドでロードし修正したレコードを保存するには、このコマンドを実行します。

SAVE RELATED ONEコマンドは、サブテーブルに対しては適用できません。それは、親レコードを保存すればサブレコードも自動的に保存されるからです。

SAVE RELATED ONEコマンドは、ロックされたレコードは保存しません。このコマンドを使用する場合、最初にレコードがアンロックされているかどうかを確認する必要があります。レコードがロックされている場合には、このコマンドは無視され、レコードを保存せず、エラーも返しません。

参照

CREATE RELATED ONE、Locked、RELATE ONE、トリガ

OLD RELATED ONE

OLD RELATED ONE(フィールド)

引数	タイプ		説明
フィールド	フィールド	→	nフィールド

説明

OLD RELATED ONEコマンドは、**RELATE ONE**コマンドと同じ方法で処理を行います。ただし、**OLD RELATED ONE**コマンドは、＜フィールド＞の更新前の内容を使用してリレート処理を行います。

注：OLD RELATED ONEコマンドはOLD関数により返されるnフィールドの古い値を使用します。詳細は、OLD関数の説明を参照してください。

OLD RELATED ONEコマンドは、更新前のカレントレコードの内容に関連したレコードをロードし、その関連レコードにアクセスできるようにします。この更新前の関連レコードを修正し、保存したい場合には、**SAVE OLD RELATED ONE**コマンドを使用する必要があります。新しく作成されたレコードは、更新前の関連するレコードを持たないという点に注意してください。

参照

Old、OLD RELATED MANY、RELATE ONE、SAVE OLD RELATED ONE

OLD RELATED MANY

OLD RELATED MANY(フィールド)

引数	タイプ		説明
フィールド	フィールド	→	1フィールド

説明

OLD RELATED MANYコマンドは、1フィールドの更新前の内容を使用してリレート処理を実行する以外は、**RELATE MANY**コマンドと同じ方法で処理を行います。

注：OLD RELATED MANYコマンドはOLD関数により返されるnフィールドの古い値を使用します。詳細は、OLD関数の説明を参照してください。

OLD RELATED MANYコマンドは、リレートテーブルのセレクションを変更し、カレントレコードとしてそのセレクションの最初のレコードを選択します。

参照

OLD RELATED ONE、RELATE MANY

RELATE ONE SELECTION

RELATE ONE SELECTION(nテーブル ; 1テーブル)

引数	タイプ		説明
nテーブル	テーブル	→	nテーブルの名前（リレート元）
1テーブル	テーブル	→	1テーブルの名前（リレート先）

説明

RELATE ONE SELECTIONコマンドは、引数＜nテーブル＞のレコードセレクションをもとにして、＜1テーブル＞用のレコードセレクションを新たに作成します。

このコマンドは、＜nテーブル＞から＜1テーブル＞へのリレートがある場合に限って使用できます。**RELATE ONE SELECTION**コマンドは、リレートの複数レベルを対象に動作できます。＜nテーブル＞と＜1テーブル＞の間には、複数のリレートテーブルがある場合があります。これらのリレートは、マニュアルリレートまたは自動リレートになるように設定することもできます。

例題

以下の例では、今日が請求書の支払期日であるすべての顧客を検索しています。

次に示されているのは、[送り状]テーブル内のレコードの選択がある場合に、[顧客]テーブル内で選択を作成する方法の1つです。

```
CREATE EMPTY SET([顧客];"支払期日")
QUERY([送り状];[送り状]支払日 = Current date)
While(Not(End selection([送り状])))
    RELATE ONE([送り状]顧客ID)
    ADD TO SET([顧客];"支払期日")
    NEXT RECORD([送り状])
End while
```

以下の手法では、**RELATE ONE SELECTION**コマンドを使用して同じ結果を出しています。

```
QUERY([送り状];[送り状]支払日 = Current date)
RELATE ONE SELECTION([送り状];[顧客])
```

参照

QUERY、RELATE MANY SELECTION、RELATE ONE、セット

RELATE MANY SELECTION

RELATE MANY SELECTION(フィールド)

引数	タイプ		説明
フィールド	フィールド	→	nテーブルのフィールド（リレート元の）

説明

RELATE MANY SELECTIONコマンドは、1テーブルのレコードセレクションをもとにして、nテーブルのレコードセレクションを作成します。

注：RELATE MANY SELECTIONコマンドは、1テーブルのカレントレコードを変更します。

警告：RELATE MANY SELECTIONコマンドは、1テーブルのカレントレコードを変更します。

例題

以下の例では、貸出金が100万円以上の顧客を対象に作成されたすべての請求書を選択しています。[送り状]テーブルの[送り状]顧客 IDフィールドは、[顧客]テーブルの[顧客]ID番号フィールドにリレートしています。

```
` 顧客を選択する
QUERY([顧客];[顧客]貸出金>=1000000)
`任意の顧客にリレートしているすべての送り状を探す
RELATE MANY SELECTION([送り状]顧客ID)
```

参照

QUERY、RELATE ONE、RELATE ONE SELECTION

リソースの概念

リソースは、Windowsの「.RSR」ファイル、Macintoshファイルのリソースフォークに定義されたフォーマットで保持されるある種のデータです。一般的に、リソースは文字列、ピクチャ、アイコン等のデータを含んでいます。実際に、独自のタイプのリソースを作成して使用することができます。また、そのリソースの中にどんなデータでも格納することができます。

データフォークとリソースフォーク

Macintosh上では、各ファイルはデータフォークとリソースフォークを持つことができます。Macintoshファイルのデータフォークは、WindowsやUNIX上のファイルと同等のものです。Macintoshのリソースフォークは、そのファイルのMacintoshベースのリソースが含まれています。これは、WindowsやUNIX上に同等のものはありません。

Windowsベースのリソースは、ファイルの他のデータと混在する状態で格納されます。

例えば、Windowsのアプリケーションにおいて、「.EXE」ファイルはリソースデータとコードの両方を含むことができます。あなたの4Dアプリケーションをプラットフォームフォームに依存しない形式で管理するために、4th DimensionはMacintoshとWindowsの両方のプラットフォーム上でMacintoshベースのリソースを使って作業します。

リソースファイル

使用しているプラットフォームが何であれ、4Dデータベースのストラクチャファイルがリソースを持つ唯一のファイルタイプではありません。4Dアプリケーション自身がリソースを含んでいます。Macintosh上では、4Dリソースはそのアプリケーションのリソースフォークに保持されます。一方、Windows上では4Dリソースは「4D.RSR」ファイルに保持され、実行可能なコードを持つ4Dアプリケーションのリソース部分は「4D.EXE」ファイルに保持されます。

4Dプラグインもリソースを含むことができます。例えば、4D社の4D Drawプラグインはリソースを含んでいます。Macintosh上では、リソースは「4D Draw」のリソースフォークに保持されます。一方、Windows上では「4DDRAW.RSR」ファイルに保持されます。

また、4Dデータベースのデータファイルもリソースを含むことができます。例えば、（Customizer Plusユーティリティを使って）特定のストラクチャファイルでのみ使用するデータファイルをロックすると、この処理はストラクチャファイルとデータファイルの中に同一のWEDD（“Wedding”の“WEDD”）リソースを追加します。Macintosh上では、そのリソースはデータファイルのリソースフォークに追加されます。Windows上では、このリソースはデータファイルのリソースファイルである「.4DR」ファイルに保持されます。

注：Customizer Plusは、Windows版およびMacintosh版の4th Dimensionで提供されます。

Windows上では、データファイルの「.4DR」ファイルを除いて、通常、「.RSR」ファイル拡張子を持ったファイルとしてMacintoshベースのリソースを含んでいる標準の4Dファイルを検出します。**Create resource file**関数がデフォルトのファイル拡張子として「.RES」を使用することに注目してください。

ユーザ独自のリソースファイルの作成

4Dによって提供されるリソースファイルの他に、**Create resource file**関数と**Open resource file**関数を使って、ユーザ独自のリソースファイルを作成して使用することができます。この2つの関数は、開かれるリソースファイルを一意に識別する「リソースファイル参照番号」を返します。このリソース参照番号は、**Open document**関数等のシステムドキュメントコマンドによって返される一般ファイルのドキュメント参照番号と同等のものです。すべての4Dリソースコマンドは、任意にリソースファイル参照番号を求めます。リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドを使って、そのファイルを閉じることを忘れないでください。

リソースファイル連鎖

4Dのデータベースで作業する際、「現在開いているリソースファイルのすべて」または「特定のリソースファイル」のどちらか一方で作業することができます。

複数のリソースファイルを同時に開きますことができます。これは常に4Dデータベースの内部からの場合です。以下のファイルを開くことができます。

■ Macintosh上のシステムリソースファイル

■ Windows上の「ASIPORT.RSR」ファイル（Macintoshのシステムリソース部分を含む）

■ 4Dアプリケーションリソースファイル

■ データベースのストラクチャリソースファイル

■ リソースファイルを任意に開くことのできるデータベースのデータファイル

■ **Open resource file**関数を使って、開くことのできるユーザ独自のリソースファイル

上記の開くことのできるリソースファイル群のリストは、「リソースファイル連鎖」と呼ばれます。以下の2つの方法で所定のリソースを検索することができます。

■ 4Dリソースコマンドにリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。

■ 4Dリソースコマンドにリソースファイル参照番号を渡さない場合、そのリソースは現在開かれているすべてのリソースファイル内で検索されます。この場合、1番最後に開かれたファイルから1番最初に開いたファイルに向かって検索を行います。このように、リソースファイル連鎖は開かれた順序と逆の順番でブラウズされます。つまり、最後に開かれたリソースファイルが最初に検査されます。

以下の例題を見てください。

```
$vhResFile:=Create resource file("Just_a_file")
If(OK=1)
    ARRAY STRING(63 ; asSomeStrings ; 0)
    STRING LIST TO ARRAY(8 ; asSomeStrings ; $vhResFile)
    ALERT("配列のサイズは、"+String(Size of array(asSomeStrings))+
                                                    "要素です。")
    STRING LIST TO ARRAY(8 ; asSomeStrings)
    ALERT("配列のサイズは、"+String(Size of array(asSomeStrings))+
                                                    "要素です。")
    CLOSE RESOURCE FILE($vhResFile)
End if
```

このメソッドを実行すると、最初に“配列のサイズは、0要素です”という警告を表示します。次に、“配列のサイズは、634要素です”という警告を表示します。

1番目の呼び出しです。

```
STRING LIST TO ARRAY(8 ; asSomeStrings ; $vhResFile)
```

Create resource file関数により作成され開かれたリソースファイルの「"STR#" ID=8」リソース“のみ”を探します。このファイルは新規に作成されて空なため、そのリソースは見つかりません。

2番目の呼び出しです。

```
STRING LIST TO ARRAY(8 ; asSomeStrings )
```

“現在開いているすべて”のリソースファイルの「"STR#" ID=8」リソースを探します。作成され開かれたばかりのファイルは（**Create resource file**関数を使って）リソースを含んでいません。**STRING LIST TO ARRAY**コマンドは、データベースのストラクチャリソースファイルのリソースを探します。このリソースファイルはどちらもリソースを含んでいないため、**STRING LIST TO ARRAY**コマンドは次に4Dリソースファイルを調べて、このファイルのリソースを検出し、そのリソースを配列に割り当てます。

まとめ：リソースファイルを使って作業する際、もし特定のファイルにアクセスしたい場合は、4Dリソースコマンドにリソースファイル参照番号を渡すことを忘れないでください。そうしないと、4Dリソースコマンドは、リソースを検出する対象としていずれのファイルでもよいものとみなします。

リソースタイプ

リソースファイルは、高度に構造化されています。各リソースデータの他に、そのリソース内容を詳細に記述したヘッダとマップを含んでいます。

リソースは、タイプ別に分類されます。リソースタイプは、常に4バイトの文字列で表されます。リソースタイプは、大文字小文字、および発音区別記号付き文字を識別します。例えば、リソースタイプ“Hi_!”、“hi_!”、“HI_!”はすべて異なります。

重要：小文字のリソースタイプは、オペレーションシステム（OS）で使用するために予約されています。そのため、ユーザ独自のリソースタイプを小文字で設計するのは避けてください。

下記は、共通で使用されるリソースタイプを示したものです。

- “STR#”リソースタイプは、Pascal文字列のリストを含んでいるリソースです。このリソースは、「ストリングリストリソース」と呼ばれます。

- “STR ”（4バイト目にスペースがある点に注意）リソースタイプは、個別のPascal文字列を含んでいるリソースです。このリソースは、「ストリングリソース」と呼ばれます。

- “TEXT”リソースタイプは、長さ制限のないテキストストリングを含んでいるリソースです。このリソースは、「テキストリソース」と呼ばれます。

- “PICT”リソースタイプは、4Dを使って、MacintoshとWindowsの両方で使用および表示できるMacintoshベースのQuickDrawピクチャを含んでいるリソースです。このリソースは、「ピクチャリソース」と呼ばれます。

■ “cicn”リソースタイプは、4Dを使って、MacintoshとWindowsの両方で使用および表示できるMacintoshベースのカラーアイコンを含んでいるリソースです。このリソースは、「カラーアイコンリソース」と呼ばれます。例えば、“cicn”リソースは、**SET LIST ITEM PROPERTIES**コマンドを使って階層リストの項目に割り当てることができます。

標準のリソースタイプの他に、ユーザ独自のリソースタイプを作成することができます。例えば、“MTYP”（“My Type”の“MTYP”）リソースタイプを使って作業することができます。

開かれているすべてのリソースファイルまたは特定のリソースファイル内に現在存在するリソースタイプのリストを取得するには、**RESOURCE TYPE LIST**コマンドを使用します。また、開かれているすべてのリソースファイルまたは特定のリソースファイル内に現在存在するリソースの指定されたリソースタイプのリストを取得するには、**RESOURCE LIST**コマンドを使用します。このコマンドは、所定の全リソースタイプのID番号と名前を返します。

警告：ほとんどのアプリケーションはリソースの内容を使って作業するためにそのリソースタイプを信頼しています。例えば、アプリケーションが“STR#”リソースにアクセスする際には、そのリソース内でストリングリストが見つかるものとみなします。標準のリソースタイプ内に矛盾したデータを格納してはいけません。これを実行すると、4Dアプリケーションまたは他のアプリケーションにおいてシステムエラーを引き起こす原因になります。

警告：リソースは高度に構造化されたファイルです。そのため、リソースコマンド以外のコマンドを使ってリソースファイルにアクセスしてはいけません。例えば、SEND PACKET等のコマンドへリソースファイル参照番号（ドキュメント参照番号のような4D時間式）を渡すこともできますが、これを実行するとリソースファイルはおそらくダメージを受けます。

警告：リソースファイルは、個別に最大2700個のリソースを持つことができます。この制限は越えないようにしてください。この制限を越えると、リソースファイルがダメージを受けて、使用できなくなります。

リソース名とリソースID番号

リソースは、「リソース名」を持っています。リソース名は最大255バイトで指定することができ、発音区別記号は識別しますが、大文字小文字の区別は行いません。リソース名は任意のリソースを説明するのに有効ですが、実際はリソースのタイプとID番号を使って、任意のリソースにアクセスします。リソース名は、一意（ユニーク）ではありません。複数のリソースで同じ名前を持つことができます。

リソースは、「リソースID番号（短く言えば、リソースIDまたはID）」を持っています。このIDは、任意のリソースタイプおよびリソース内において一意です。例えば、

■ 1つのリソースファイルでは、リソース“ABCD”ID=1とリソース“EFGH”ID=1を持つことができます。

■ 2つのリソースファイルでは、同じタイプとIDのリソースを持つことができます。

4Dコマンドを使って任意のリソースにアクセスする場合、そのリソースのタイプとIDを指定します。このリソースを検索するためのリソースファイルを指定しなかった場合、4Dコマンドは最初に調べたリソースファイル内で見つかったリソースの出現数を返します。リソースファイルが開かれた逆の順番で検索されることを思い出してください。

リソースIDの範囲は、-32768から32767までです。

重要：15000から32767の範囲は、ユーザ独自のリソース用に使用できます。しかし、負数のリソースIDを使用してはいけません。これは、オペレーションシステム（OS）で使用されるためにあらかじめ予約されている番号です。また、0から14999の範囲のリソースIDを使用してもいけません。この範囲は、標準で使用されるためにあらかじめ予約されている番号です。

所定のリソースタイプのID番号と名前を取得するには、**RESOURCE LIST**コマンドを使用します。

個別のリソース名を取得するには、**Get resource name**関数を使用します。

個別のリソース名を変更するには、**SET RESOURCE NAME**コマンドを使用します。

4Dコンポーネントによりインストールされたリソースのためのカレント（事実上の）番号を取得するには、**Get component resource ID**コマンドを使用します。

各4Dコマンドはオプションでリソースファイル参照番号を受け付けるため、異なる2つのリソースファイル内で同じタイプとIDを持つリソースを簡単に取り扱うことができます。以下の例は、あるリソースから別のリソースにすべての“PICT”リソースをコピーします。

```
` 既存のリソースファイルを開く
$vhResFileA:=Open resource file("")
If(OK=1)
	` 新規リソースファイルを作成する
	$vhResFileB:=Create resource file("")
	If(OK=1)
		` "PICT"タイプのすべてのリソースのIDと名前のリストを取得し、
		` リソースファイルAの中に配置する
		RESOURCE LIST("PICT" ; $aiResID ; $asResName ; $vhResFileA)
		`各リソースファイルに対してループ
		For($vlElem ; 1 ; Size of array($aiResID))
			$viResID:=$aiResID{$vlElem}
			` Aファイルからリソースをロードする
```

```
                    GET RESOURCE("PICT" ; $viResID ; vxResData ;
                                                        $vhResFileA)
                    ` リソースがロードされた場合、
                    If(OK=1)
                        ` Bファイルの中にリソースを追加し書き込む
                        SET RESOURCE("PICT" ; $viResID ; vxResData ;
                                                        $vhResFileB)
                        ` リソースが追加および書き込まれた場合
                        If(OK=1)
                            `リソースの名前もコピーする
                            SET RESOURCE NAME("PICT ; $viResID ;
                                    $asResName{$vlElem}; $vhResFileB)
                            `リソースプロパティについては、次節の
                              「リソースプロパティ」を参照してください。
                            $vlResAttr:=Get resource properties("PICT";
                                                $viResID ; $vhResFileA)
                            SET RESOURCE PROPERTIES("PICT" ;
                                                $viResID ; $vlResAttr
                            ; $vhResFileB)
                        Else
                            ALERT("リソースPICT ID="+String
                                        ($viResID)+"は追加されません。")
                        End if
                    Else
                        ALERT("リソースPICT ID="+String($viResID)+
                                                "はロードされません。")
                    End if
                End if
                CLOSE RESOURCE FILE($vhResFileB)
            End if
            CLOSE RESOURCE FILE($vhResFileA)
        End if
```

リソースプロパティ

リソースは、タイプ、名前、ID番号の他に（属性と呼ばれる）追加プロパティを持っています。例えば、任意のリソースではパージ（消去）できるものもあれば、できないものもあります。この属性は、他のオブジェクトに割り当てるために空きメモリが必要とされている場合に、ロードされるリソースがメモリからパージできるかどうかオペレーションシステムに知らせます。上記の例で示しているように、任意のリソースを作成またはコピーする際、そのリソースのコピーだけでなく、そのリソースの名前およびプロパティもコピーすることが重要です。リソースプロパティに関する詳細は、**Get resource properties**関数および**SET RESOURCE PROPERIES**コマンドの説明を参照してください。

リソース内容の取り扱い

メモリの中に任意のリソースタイプをロードするには、BLOBの中にリソース内容を返す**GET RESOURCE**コマンドを呼び出します。

ディスク上に任意のリソースを追加、または書き込むには、ユーザが渡したBLOBの内容にリソース内容を設定できる**SET RESOURCE**コマンドを呼び出します。

既存のリソースを削除するには、**DELETE RESOURCE**コマンドを使用します。

一般的なリソースタイプを簡単に取り扱うために、4Dは下記のような内蔵コマンドを用意しています。

- **STRING LIST TO ARRAY**コマンドは、任意の文字列配列またはテキスト配列にストリングリストリソースに含まれる文字列を割り当てます。
- **ARRAY TO STRING LIST**コマンドは、文字列配列またはテキスト配列の要素を使ってストリングリストリソースを作成または上書きします。
- **Get index string**関数は、ストリングリストリソースから特定の文字列を返します。
- **Get string resource**関数は、ストリングリソースから文字列を返します。
- **SET STRING RESOURCE**コマンドは、ストリングリソースを作成または上書きします。
- **Get text resource**関数は、テキストリソースのテキストを返します。
- **SET TEXT RESOURCE**コマンドは、テキストリソースを作成または上書きします。
- **GET PICTURE RESOURCE**コマンドは、ピクチャリソースのピクチャを返します。
- **SET PICTURE RESOURCE**コマンドは、ピクチャリソースを作成または上書きします。
- **GET ICON RESOURCE**コマンドは、ピクチャとしてカラーアイコンリソースを返します。

上記のコマンドは一般的なリソースタイプを簡単に管理するために用意されていますが、これらのコマンドは、BLOBを使った**GET RESOURCE**コマンドや**SET RESOURCE**コマンドの使用を防止しない点に注意してください。例えば、以下のコードは、

```
ALERT(Get text resource(20000))
```

下記のコードと同等の処理を短くしたものです。

```
GET RESOURCE("TEXT" ; 20000 ; vxData)
If(OK=1)
    $vlOffset:=0
    ALERT(BLOB to text(vxData ; Text without length ; $vlOffset ;
                                                BLOB Size(vxData)))
End if
```

4Dコマンドとリソース

この章で説明するリソースコマンドの他に、リソースやリソースファイルを使って作業する4Dコマンドもあります。

■ Macintosh上では、**DOCUMENT TO BLOB**コマンドと**BLOB TO DOCUMENT**コマンドは任意のMacintoshファイルのリソースフォーク全体をロードして書き込むことができます。

■ **SET LIST ITEM PROPERTIES**コマンドと**SET LIST PROPERTIES**コマンドを使って、リストの項目にピクチャやカラーアイコンリソースを関連付けたり、リストのノードとしてカラーアイコンリソースを使用することができます。

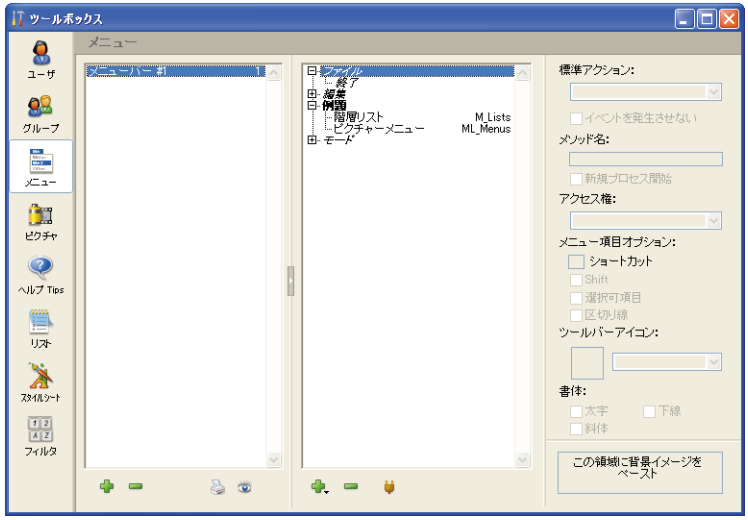
■ **PLAY**コマンドは、MacintoshとWindowsの両方で"snd"リソースをプレイします。

■ **SET CURSOR**コマンドは、"CURS"リソースを使って、マウスの外観を変更します。

リソースと4D Insider：例題

国際市場向けに、異なる言語で4Dデータベースの開発や管理を実行する場合、リソースはローカライズの問題を処理するにはたいへん便利です。

では例題をみていきましょう。以下の図はデータベースのメニューバーが日本語で表示されているところです。

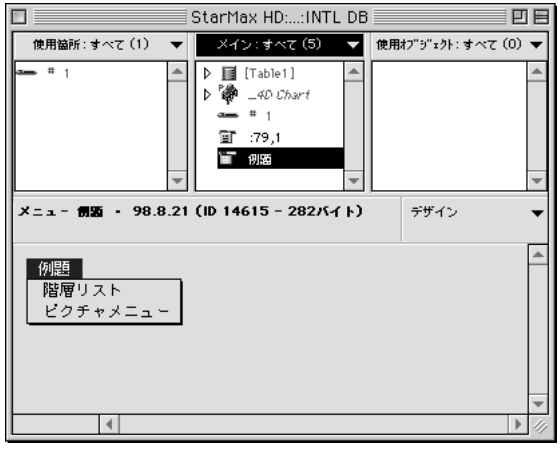


「ファイル」メニューのタイトルは、既にリソースを参照していますが、そのメニュー項目である「終了」は違います。「例題」メニューは「階層リスト」と「ピクチャメニュー」で構成されています。このメニューおよびメニュー項目は、リソースを参照しません。

4D Insiderを使用し、メニューバーの固定文字列を“STR#”リソースに格納されている文字列を参照するように変更することができます。この操作の実行手順について見ていきましょう。

注：4D Insiderは4Dの相互参照およびライブラリ管理用のツールです。

1. 4D Insiderでデータベースを開きます。以下の図は、4D Insiderのブラウザウインドウのメニューバーを示しています。



2. この時点で、このメニューバーが“STR#”リソースを参照するように変更できます。これを実行するには、4D Insiderの「ツール」メニューから「テキストからSTR#...」を選択します。

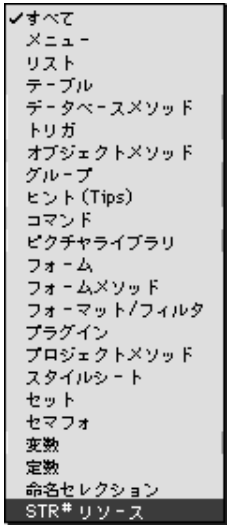


すると、「テキストからSTR#リソースへ」ダイアログボックスが表示されます。



3. リソース名と番号を入力し、「新規」ボタンをクリックします。この例では、“例題メニュー”がリソース名で、“20000”はリソース番号です。「設定」ボタンをクリックしてこのウインドウを閉じると、このリソースが作成されます。

4. ブラウザウインドウのメインリストのポップアップメニューから“STR#リソース”を選択します。



5. “STR# 20000”リストアイテムをダブルクリックして内容を表示します。

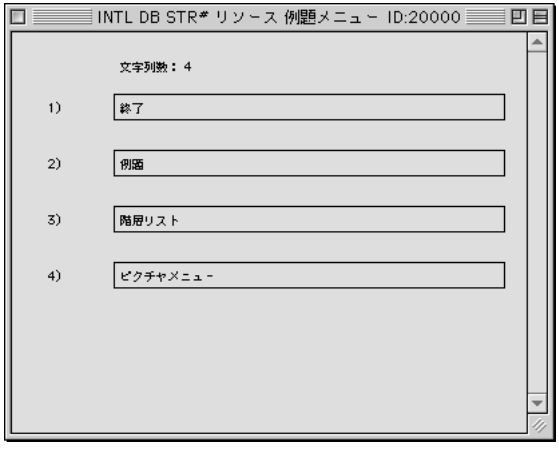


これらの文字列がリソースに保存され、データベース開発のロジックに影響を与えずに文字列の値を変更できるようになりました。

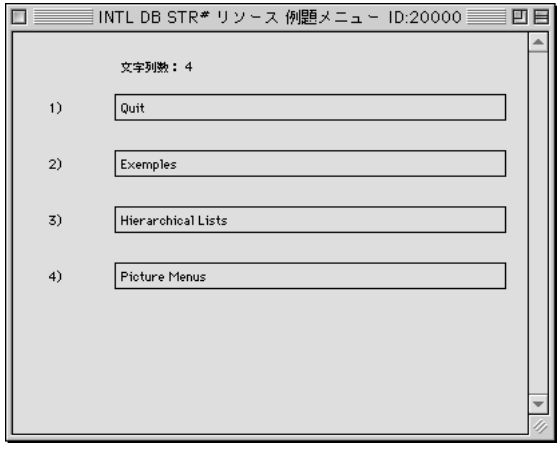
6. この値を変更するには、ブラウザウインドウのメインリストで“例題メニュー”が選択されている状態で、4D Insiderの「ツール」メニューから「STR#編集」を選択します。



すると、STR#リソースの編集用ウインドウが表示されます。



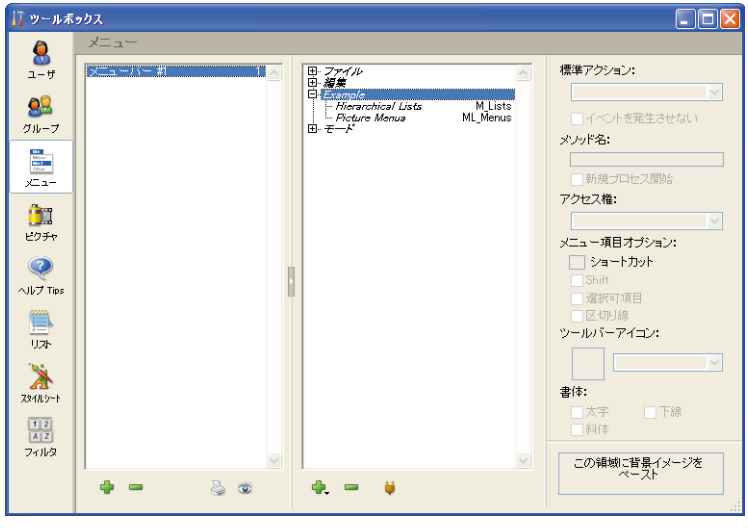
7. ここの文字列を他の言語に翻訳します。以下の図は、文字列が英語に翻訳されたところを表わしています。



8. 翻訳が終了したら、このウインドウを閉じます。確認ダイアログボックスで「はい」をクリックします。



9. この時点で4D Insiderを終了し、4th Dimensionでこのデータベースを開きます。「デザイン」モードの「メニューバーエディタ」には、英語のリソースを参照してメニューバーが表示されます。



この手順についての詳細は『4D Insiderリファレンス』マニュアルを参照してください。また、メニューバーおよびデータベースフォーム上のオブジェクトにおけるリソース参照の使用に関する詳細は、『4th Dimensionデザインリファレンス』マニュアルを参照してください。

4Dのリソースコマンドを利用して、4D Insiderで作成したリソースを使用することができます。以下のメソッドでは**STRING LIST TO ARRAY**コマンドを使用して“STR#”リソース（4D Insiderで作成）を配列にロードしています。



「デバッグ」ウインドウにおいて、4D Insiderで翻訳した文字列が配列に代入されているところを確認できます。



Open resource file

Open resource file(リソースファイル名 {; ファイルタイプ }) → DocRef

引数	タイプ		説明
リソースファイル名	文字列	→	リソースファイルのショート名またはロング名 空の文字列の場合、標準の「ファイルを開く」ダイアログボックスを表示
ファイルタイプ	文字列	→	Macintoshファイルタイプ（4バイトの文字列）、またはWindowsファイル拡張子（1~3バイトの文字列）、または、省略の場合、すべてのファイル
戻り値	DocRef	←	リソースファイル参照番号

説明

Open resource file関数は、引数<リソースファイル名>に渡した名前またはパス名を持つリソースファイルを開きます。

ファイル名を渡す場合、そのファイルはデータベースのストラクチャファイルと同じフォルダ内に配置されていなければなりません。他のフォルダ内に配置されているリソースを開くには、パス名を渡します。

引数<リソースファイル名>に空の文字列を渡すと、「ファイルを開く」ダイアログボックスが表示されます。このダイアログボックスでリソースを選択し、それを開くことができます。ダイアログボックスを取り消すと、リソースファイルは開かれません。この場合、**Open resource file**関数はヌルのDocRefを返し、システム変数OKに0を設定します。

リソースファイルが正常に開かれると、**Open resource file**関数はそのリソースファイル参照番号を返し、システム変数OKに1を設定します。リソースファイルが存在しない場合や開こうとしているファイルがリソースファイルではない場合、エラーが生成されます。

Macintosh上で、「ファイルを開く」ダイアログボックスを使用する場合、デフォルトとしてすべてのファイルが表示されます。ファイルの特定タイプを表示するには、オプション引数<ファイルタイプ>にそのファイルタイプを指定します。

Windows上で、「ファイルを開く」ダイアログボックスを使用する場合、デフォルトとしてすべてのファイルが表示されます。ファイルの特定タイプを表示するには、1から3バイトのWindowsファイル拡張子または**MAP FILE TYPES**コマンドを使ってマップされた任意のMacintoshファイルタイプをオプション引数<ファイルタイプ>に渡します。

リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドの呼び出しを忘れないでください。ただし、アプリケーションを終了する（または、他のデータベースを開く）場合は、4Dが**Open resource file**関数や**Create resource file**関数を使って開かれたすべてのリソースファイルを自動的に閉じます。

排他的なリード-ライトアクセスで任意のドキュメント（Macintosh上のデータフォーク）を開きます**Open document**関数と違って、**Open resource file**関数は4Dセッション内で既に開かれているリソースファイルをユーザが開くのを妨げません。例えば、**Open document**関数を使って同じドキュメントを2度開こうとすると、2度目の開く時にI／Oエラーが返されます。これに対して、4Dセッション内で既に開かれているリソースファイルを開こうとすると、**Open resource file**関数は既に開いているファイルのリソースファイル参照番号を返します。たとえ、何度も任意のリソースファイルを開いても、そのリソースファイルを閉じるにはたった1回**CLOSE RESOURCE FILE**コマンドを呼び出すだけで済みます。このことは、4Dセッション内でリソースファイルが開かれる場合にのみ有効である点に注意してください。他のアプリケーションで既に開かれているリソースファイルを開こうとすると、I／Oエラーが発生します。

この複数オープン機能により、通常の4D処理をむやみに変更することなく、4Dアプリケーションとデータベースリソースファイルの参照番号を容易に取得することができます。

警告

・4Dアプリケーションのリソースファイルにアクセスする場合は十分に注意してご使用ください。4Dアプリケーションのリソースを変更してはいけません。プログラムに予想できないダメージを与え、システムエラーを引き起こす原因になります。また、あなたのデータベースがさまざまな4D環境（4D、4D Runtime、4D Engine、4D Serverと4D Client）内で使用されることを覚えておいてください。

・データベースリソースファイルにアクセスして、プログラムでそのファイルのリソースを追加、削除、修正する場合、動作している4D環境を必ず調べてください。4D Serverでは、これはおそらく深刻な問題を引き起こします。例えば、（データベースメソッドやストアドプロシージャを使用して）サーバマシン上の任意のリソースファイルを修正する場合、ワークステーションにリソースを透過的に配付する内蔵の4D Server管理サービスに確実に影響を及ぼします。4D Clientでは、ストラクチャファイルがサーバマシン上に配置されているため、直接ストラクチャファイルにアクセスできない点に注意してください。

・この理由により、リソースを使用する場合、そのリソースをファイルに格納します。

*・オリジナルのリソースを使って作業している場合、負数のリソースIDを使用してはいけません。これは、オペレーションシステムで使用されるためにあらかじめ予約されている番号です。また、0から14999の範囲のリソースIDを使用してもいけません。この範囲は、4th Dimensionで使用されるためにあらかじめ予約されている番号です。オリジナルのリソースには、15000から32767の範囲の番号を使用してください。あるリソースファイルを開きますと、そのファイルがリソースファイル連鎖内で検索される最初のファイルとなることを覚えておいてください。システムまたは4Dリソースの範囲内のIDを持つリソースファイルに任意のリソースを格納する場合、このリソースは**GET RESOURCE**コマンドまたは4Dアプリケーションの内部ルーチンで見つけることができます。これが目的とする結果かもしれませんが、もしそうでない場合は、システムエラーを引き起こすかもしれないため、この範囲の値を使用してはいけません。*

*・リソースファイルは高度に構造化されたファイルで、しかも1ファイル当たり最高2700以上のリソースにアクセスすることはできません。数多くのリソースを含んでいるリソースファイルで作業する場合、そのリソースファイルに新しいリソースを追加する前にそのリソースの数を調べるのが得策です。これについては、**RESOURCE TYPE LIST**コマンドの例題の**Count resource**関数を参考にしてください。*

任意のリソースファイルを開きますと、**RESOURCE TYPE LIST**コマンドと**RESOURCE LIST**コマンドを使って、そのファイルの内容を解析することができます。

例題

1. 以下の例は、Windows上でデータベースフォルダに配置された“MyPrefs.res”リソースファイルを開きます。

```
$vhResFile:=Open resource file("MyPrefs" ; "res ")
```

   Macintosh上では、この例は“MyPrefs”ファイルを開きます。

2. 以下の例は、Windows上でデータベースフォルダに配置された“MyPrefs.rsr”リソースファイルを開きます。

```
$vhResFile:=Open resource file("MyPrefs" ; "rsr ")
```

   Macintosh上では、この例は“MyPrefs”ファイルを開きます。

3. 以下の例は、すべてのファイルタイプを表示する「ファイルを開く」ダイアログボックスを表示します。

```
$vhResFile:=Open resource file(" ")
```

4. 以下の例は、デフォルトファイルタイプを使って、**Create resource file**関数で作成されたファイルを表示する「ファイルを開く」ダイアログボックスを表示します。

```
$vhResFile:=Open resource file("" ; "res ")
If(OK=1)
```

```
        ALERT(Document+"を開きました。")
        CLOSE RESOURCE FILE($vhResFile)
    End if
```

5. 以下の例は、ローカル変数“$vhStructureResFile”にデータベースストラクチャリソースファイルの参照番号を返します。

```
    If(On Windows)
        $vhStructureResFile:=Open resource file(Replace string(Structure file ;
                                                ".4DB" ; ".RSR"))
    Else
        $vhStructureResFile:=Open resource file(Structure file)
    End if
```

6. 以下の例は、ローカル変数“$vhApplResFile”に4Dアプリケーションリソースファイルの参照番号を返します。

```
    If(On Windows)
        $vhApplResFile:=Open resource file(Replace string(Application file;
                                                "EXE";"RSR"))
    Else
        $vhApplResFile:=Open resource file(Application file)
    End if
```

参照

CLOSE RESOURCE FILE、Create resource file

システム変数とシステムセット

リソースファイルが正常に開かれた場合は、システム変数OKに1を設定します。リソースファイルが開くことができなかったり、ユーザが「ファイルを開く」ダイアログボックスの「キャンセル」ボタンをクリックした場合は、システム変数OKに0が設定されます。

リソースファイルが「ファイルを開く」ダイアログボックスを使って正常に開かれた場合は、システム変数Documentにそのリソースファイルのパス名を設定します。

エラー処理

リソースファイルが任意のリソースを開くことができなかったり、I／O（入出力）に問題が生じた場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Create resource file

Create resource file(リソースファイル名 {; ファイルタイプ }) → DocRef

引数	タイプ		説明
リソースファイル名	文字列	→	リソースファイルのショート名またはロング名 空の文字列の場合、標準の「ファイルを保存」ダイアログボックスを表示
ファイルタイプ	文字列	→	Macintoshファイルタイプ（4バイトの文字列）、またはWindowsファイル拡張子（1～3バイトの文字列）、 または省略の場合リソースドキュメント（"res"/.RES）
戻り値	DocRef	←	リソースファイル参照番号

説明

Create resource file関数は、引数<リソースファイル名>に渡した名前またはパス名を持つ新規リソースファイルを作成して開きます。

ファイル名を渡す場合、そのファイルはデータベースのストラクチャファイルと同じフォルダ内に配置されていなければなりません。他のフォルダ内に配置されているリソースを開くには、パス名を渡します。

ファイルが既に存在しており、現在開かれていない場合、**Create resource file**関数は空の新規リソースファイルでそれを上書きします。ファイルが現在開かれている場合は、I／Oエラーが返されます。

引数<リソースファイル名>に空の文字列を渡すと、「ファイルを保存」ダイアログボックスが表示されます。このダイアログボックスで作成されるリソースファイルの格納場所と名前を選択することができます。ダイアログボックスを取り消すと、リソースファイルは作成されません。この場合、**Create resource file**関数はヌルのDocRefを返し、システム変数OKに0を設定します。

リソースファイルが正常に作成され、開かれると、**Open resource file**関数はそのリソースファイル参照番号を返し、システム変数OKに1を設定します。リソースファイルが作成できない場合、エラーが生成されます。

Macintosh上では、**Create resource file**関数で作成されるリソースファイルのデフォルトファイルタイプは“res”です。Windows上では、デフォルトのファイル拡張子は“.RES”です。

他のファイルタイプを作成するには、

■ Macintosh上では、オプション引数<ファイルタイプ>にそのファイルタイプを渡します。

■ Windows上では、1から3バイトのWindowsファイル拡張子または**MAP FILE TYPES**コマンドを使ってマップされた任意のMacintoshファイルタイプをオプション引数<ファイルタイプ>に渡します。

リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドの呼び出しを忘れないでください。ただし、アプリケーションを終了する（または、他のデータベースを開く）場合、4Dが**Open resource file**関数や**Create resource file**関数を使って、開かれたすべてのリソースファイルを自動的に閉じます。

例題

1. 以下の例は、Windows上でデータベースフォルダに配置された“MyPrefs.res”リソースファイルを作成して開きます。

```
$vhResFile:=Create resource file("MyPrefs")
```

Macintosh上では、この例は“MyPrefs”ファイルを作成して開きます。

2. 以下の例は、Windows上でデータベースフォルダに配置された“MyPrefs.rsr”リソースファイルを作成して開きます。

```
$vhResFile:=Create resource file("MyPrefs" ; "rsr ")
```

Macintosh上では、この例は“MyPrefs”ファイルを作成して開きます。

3. 以下の例は、「ファイルを保存」ダイアログボックスを表示します。

```
$vhResFile:=Create resource file("" ; "res ")
If(OK=1)
    ALERT(Document+"を保存しました。")
    CLOSE RESOURCE FILE($vhResFile)
End if
```

参照

CLOSE RESOURCE FILE、Open resource file

システム変数とシステムセット

リソースファイルが正常に作成され開かれた場合は、システム変数OKに1を設定します。リソースファイルが作成できなかったり、ユーザが「ファイルを保存」ダイアログボックスの「キャンセル」ボタンをクリックした場合は、システム変数OKに0が設定されます。

リソースファイルが「ファイルを保存」ダイアログボックスを使って正常に作成され開かれた場合は、システム変数Documentにそのリソースファイルのパス名を設定します。

エラー処理

リソースファイルが任意のリソースを作成または開くことができなかったり、I／O（入出力）に問題が生じた場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

CLOSE RESOURCE FILE

CLOSE RESOURCE FILE(リソースファイル)

引数	タイプ		説明
リソースファイル	DocRef	→	リソースファイル参照番号

説明

CLOSE RESOURCE FILEコマンドは、引数<リソースファイル>に渡された参照番号を持つリソースファイルを閉じます。

たとえ、何度も同じリソースファイルを開いても、そのリソースファイルを閉じるには**CLOSE RESOURCE FILE**コマンドを1回呼び出すだけで済みます。

4Dアプリケーションやデータベースリソースファイルに**CLOSE RESOURCE FILE**コマンドを適用すると、このコマンドはそれを見つけて何も行いません。

無効なリソースファイル参照番号を渡すと、このコマンドは何も行いません。

Open resource file関数や**Create resource file**関数を使って開かれたリソースファイルを終了するために、最終的には**CLOSE RESOURCE FILE**コマンドを呼び出すことを忘れないでください。また、アプリケーションを終了する（または、他のデータベースを開く）場合、4Dが開かれたリソースファイルを自動的に閉じる点にも注目してください。

例題

▼以下の例は、任意のリソースファイルを作成し、そのファイルにストリングリソースを追加して閉じます。

```
$vhDocRef:=Create resource file("Just a file")
If(OK=1)
    SET STRING RESOURCE(20000 ; "Just a string" ; $vhDocRef)
    CLOSE RESOURCE FILE($vhDocRef)
End if
```

参照

Create resource file、Open resource file

システム変数とシステムセット

影響を及ぼすものはありません。

RESOURCE TYPE LIST

RESOURCE TYPE LIST(リソースタイプ {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列配列	←	利用できるリソースタイプのリスト
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

RESORCE TYPE LISTコマンドは、引数<リソースタイプ>の文字列配列に現在開いているリソースファイルの中に存在するリソースのリソースタイプを代入します。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが一覧表示されます。<リソースファイル>を省略した場合は、現在開いているリソースファイルのすべてのリソースが一覧表示されます。

RESORCE TYPE LISTコマンドを呼び出す前に、引数<リソースタイプ>配列を任意の文字列配列またはテキスト配列としてあらかじめ定義することができます。配列の事前定義を行わない場合、このコマンドはテキスト配列の<リソースタイプ>を作成します。

このコマンドの実行後、**Size of array**関数を引数<リソースタイプ>配列に対して実行し、見つかったリソースタイプの数を調べることができます。

例題

1. 以下の例は、現在開いているすべてのリソースファイルに存在するリソースのリソースタイプを配列“atResType”に割り当てます。

```
RESOURCE TYPE LIST(atResType)
```

2. 以下の例は、使用しているMacintoshの4Dストラクチャファイルが Windows上でデータベースを使用するためにアップデートする必要のある旧バージョンの4Dプラグインを含んでいるかどうかをユーザに知らせます。

```
$vhResFile:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResType ; $vhResFile)
If(Find in array(atResType ; "4DEX")>0)
        ALERT("このデータベースは旧モデルのMacintoshに対応した
                4Dプラグインが含まれています。"+(Char(13)*2) +"Windows上で
                        このデータベースを使用するには、このプラグインを
                                アップデートする必要があります。")
End if
```

注：ストラクチャファイルが旧バージョンの4Dプラグインを格納できる唯一のファイルではありません。データベースは“Proc.EXT”ファイルも含むことができます。

3. 以下のプロジェクトメソッドは、任意のリソースファイルの中に存在するリソースの数を返します。

```
` 「Count resources」プロジェクトメソッド
` Count resources(時間) → 倍長整数
` Count resources(DocRef) → リソースの数
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
```

このメソッドをデータベースに組み込むと、以下のように記述できます。

```
$vhResFile:=Open resource file("")
If(OK=1)
    ALERT(Document+"ファイルには、"+String(Count resources
                                    ($vhResFile))+" 個のリソースが存在します。")
    CLOSE RESOURCE FILE($vhResFile)
End if
```

参照

RESOURCE LIST

RESOURCE LIST

RESOURCE LIST(リソースタイプ ; リソースID ; リソース名 {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトリソースタイプ
リソースID	倍長整数配列	←	指定したリソースタイプのリソースID番号
リソース名	文字列配列	←	指定したリソースタイプのリソース名
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

RESORCE LISTコマンドは、引数＜リソースID＞と＜リソース名＞の配列に引数＜リソースタイプ＞に渡したタイプのリソースID番号とリソース名を割り当てます。

重要：＜リソースタイプ＞には、4バイトの文字列を受け渡す必要があります。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが一覧表示されます。＜リソースファイル＞を省略した場合は、現在開いているリソースファイルのすべてのリソースが一覧表示されます。

RESORCE LISTコマンドを呼び出す前に配列を事前定義する場合は、引数＜リソースID＞配列に任意の倍長整数配列、引数＜リソースタイプ＞配列に任意の文字列配列またはテキスト配列を前もって定義する必要があります。配列の事前定義を行わない場合、このコマンドは倍長整数配列の＜リソースID＞、テキスト配列の＜リソースタイプ＞を作成します。

このコマンドの実行後、**Size of array**関数を引数＜リソースID＞と＜リソース名＞の配列に対して適用し、見つかったリソースの数を調べることができます。

例題

1. 以下の例は、配列“$alResID”と“$atResName”にデータベースのストラクチャファイル内に存在するストリングリストリソースのIDと名前を割り当てます。

```
If(On Windows)
    $vhStructureResFile:=Open resource file(Replace string(Structurefile ;
                                                ".4DB" ; ".RSR"))
Else
    $vhStructureResFile:=Open resource file(Structure file)
End if
If(OK=1)
```

```
        RESOURCE LIST("STR#" ; $alResID ; $atResName ; $vhStructureResFile)
    End if
```

2. 以下の例は、現在開かれているリソースファイル内にあるピクチャリソースをデータベースのピクチャライブラリの中にコピーします。

```
RESOURCE LIST("PICT" ; $alResID ; $atResName)
$Ref : = Open window(50 ; 50 ; 550 ; 120 ; 5 ; "PICTリソースコピー中...")
For($vlElem ; 1 ; Size of array($alResID))
    GET PICTURE RESOURCE($alResID{$vlElem} ; $vgPicture)
    If(OK=1)
        $vsName:=$atResName{$vlElem}
        If($vsName="")
            $vsName:="PICTリソースID="+String($alResID{$vlElem})
        End if
        ERASE WINDOW
        GOTO XY(2 ; 1)
        MESSAGE("DBピクチャライブラリに"+$vsName+"を追加します。")
        SET PICTURE TO LIBRARY($vgPicture ; $alResID{$vlElem} ;
                                                    $vsName)
    End if
End for
CLOSE WINDOW
```

参照

RESOURCE TYPE LIST

システム変数とシステムセット

影響を及ぼすものはありません。

STRING LIST TO ARRAY

STRING LIST TO ARRAY(リソースID ; ストリング {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
ストリング列	文字列配列	→	ストリングを取り出すための文字列配列またはテキスト配列
		←	STR#リソースのストリング
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

STRING LIST TO ARRAYコマンドは、引数＜リソースID＞に渡されるIDを持つストリングリスト（"STR#"）リソース内に格納されているストリングを引数＜ストリング＞配列に割り当てます。

リソースが見つからない場合は、＜ストリング＞配列はそのまま変更されず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

STRING LIST TO ARRAYコマンドを呼び出す前に、引数＜ストリング＞配列に任意の文字列配列またはテキスト配列を前もって定義することができます。配列の事前定義を行わない場合、このコマンドはテキスト配列の＜ストリング＞を作成します。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

Tips：ストリングリストリソースは、合計で32Kまで使用できます。1リソース当たり最大100個程度のストリングにします。

例題

▼ **ARRAY TO STRING LIST**コマンドの例を参照してください。

参照

ARRAY TO STRING LIST、Get indexed string、Get string resource、Get text resource

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

ARRAY TO STRING LIST

ARRAY TO STRING LIST(ストリング ; リソースID {; リソースファイル})

引数	タイプ		説明
ストリング	文字列配列	→	文字列配列またはテキスト配列 （STR#リソースの新しい内容）
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

ARRAY TO STRING LISTコマンドは、引数＜リソースID＞に渡されるIDを持つストリングリスト（"STR#"）リソースを作成または上書きします。＜ストリング＞配列に渡した文字列をもとに、リソース内容が作成されます。この配列には、文字列配列またはテキスト配列を指定できます。

リソースが追加されなかった場合は、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖の最上位にある現在のファイル（最後に開いたリソースファイル）に追加されます。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

Tips：ストリングリストリソースは、合計で32Kまで使用できます。1リソース当たり最大100個程度のストリングにします。

例題

ご使用のデータベースでは、所定のフォントセットを使用しているものとします。

その場合、「On Exit」データベースメソッドにおいて、以下のように記述します。

```
`「On Exit」データベースメソッド
    If(<>bFontsAreOK)
        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
```

「On Startup」データベースメソッドにおいて、以下のように記述します。

```
    `「On Startup」データベースメソッド
    <>bFontsAreOK:=False
    FONT LIST($atNewFont)
    If(Test path name("FontSet")#Is a document)
        $vhResFile:=Create resource file("FontSet")
    Else
        $vhResFile:=Create resource file("FontSet")
    End if
    If(OK=1)
        STRING LIST TO ARRAY(15000 ; $atOldFont ; $vhResFile)
        If(OK=1)
            <>bFontsAreOK:=True
            For($vlElem ; 1 ; Size of array($atNewFont))
                If($atNewFont{$vlElem}#$atOldFont{$vlElem}))
                    $vlElem:=MAXLONG
                    <>bFontsAreOK:=False
                End if
            End for
        Else
            <>bFontsAreOK:=True
        End if
        CLOSE RESOURCE FILE($vhResFile)
    End if
    If(Not(<>bFontsAreOK))
        CONFIRM("同じフォントセットを使用していません。OK?")
        If(OK=1)
            <>bFontsAreOK:=True
        Else
            QUIT 4D
        End if
    End if
```

参照

SET STRING RESOURCE、SET TEXT RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが上書きされた場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

Get indexed string

Get indexed string(リソースID ; ストリングID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースID	数値	→	リソースID番号
ストリングID	数値	→	ストリング番号
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル
戻り値	文字列	←	インデックス付きのストリングの値

説明

Get indexed string関数は、引数＜リソースID＞に渡されるIDを持つストリングリスト（"STR#"）リソースに格納されている1つのストリングを返します。

引数＜ストリングID＞にストリングの番号を渡します。ストリングリストリソースのストリングは1からNの順に番号が振られます。ストリングリストリソースのすべてのストリング（およびそのストリング番号）を取得するには、**STRING LIST TO ARRAY**コマンドを使用します。

リソースまたはそのリソース内のストリングが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

例題

▼ **Month of**関数の例を参照してください。

参照

Get string resource、Get text resource、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

Get string resource

Get string resource(リソースID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル
戻り値	文字列	←	STRリソースの内容

説明

Get string resource関数は、引数<リソースID>に渡されるIDを持つストリングリスト("STR#")リソースに格納されているストリングを返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ストリングリソースは、最大255バイトの文字を使用することができます。

例題

▼以下の例は、ストリングリソースID=20911の内容を表示します。このIDは、現在開かれているリソースファイルの少なくとも1つに配置されている必要があります。

```
ALERT(Get string resource(20911))
```

参照

Get indexed string、Get text resource、SET STRING RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

SET STRING RESOURCE

SET STRING RESOURCE(リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	文字列	→	STR リソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合現在のリソースファイル

説明

SET STRING RESOURCEコマンドは、引数＜リソースデータ＞に渡されるストリングを使って、引数＜リソースID＞に渡されるIDを持つストリング（"STR "）リソースを作成または上書きします。

リソースが追加できなかった場合、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後に開いたリソースファイル）の上部にある現在のファイルに追加されます。

注：ストリングリソースは、最大255バイトの文字を使用することができます。

参照

Get string resource、SET TEXT RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

Get text resource

Get text resource(リソースID {; リソースファイル}) → テキスト

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル
戻り値	テキスト	←	TEXTリソースの内容

説明

Get text resource関数は、引数<リソースID>に渡されるIDを持つテキスト（"TEXT"）リソースに格納されているテキストを返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：テキストリソースは、最大32000バイトの文字を使用することができます。

例題

▼以下の例は、テキストリソースID=20800の内容を表示します。このIDは、現在開かれているリソースファイルの少なくとも1つに配置されている必要があります。

```
ALERT(Get text resource(20800))
```

参照

Get indexed string、Get string resource、SET STRING RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

SET TEXT RESOURCE

SET TEXT RESOURCE(リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	文字列	→	TEXTリソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

SET TEXT RESOURCEコマンドは、引数＜リソースデータ＞に渡されるテキストまたは文字列を使って、引数＜リソースID＞に渡されるIDを持つテキスト（"TEXT"）リソースを作成または上書きします。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖の最上位にあるファイル（最後に開いたリソースファイル）に追加されます。

注：テキストリソースは、最大32000バイトの文字を使用することができます。

参照

Get text resource、SET STRING RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

GET PICTURE RESOURCE

GET PICTURE RESOURCE(リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	フィールドまたは変数	→	ピクチャを受け取るピクチャフィールドまたはピクチャ変数
		←	PICTリソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

GET PICTURE RESOURCEコマンドは、引数＜リソースID＞に渡されるIDを持つピクチャ（"PICT"）リソースに格納されているピクチャを引数＜リソースデータ＞のピクチャフィールドまたはピクチャ変数に返します。

リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ピクチャリソースは、少なくとも数メガバイトのサイズになる可能性があります。

例題

▼ **RESOURCE LIST**コマンドの例を参照してください。

参照

GET ICON RESOURCE、ON ERR CALL、SET PICTURE RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

エラー処理

ピクチャをロードするための十分なメモリがない場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET PICTURE RESOURCE

SET PICTURE RESOURCE(リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	ピクチャ	→	PICTリソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

SET PICTURE RESOURCEコマンドは、引数＜リソースデータ＞に渡されるピクチャを使って、引数＜リソースID＞に渡されるIDを持つピクチャ（"PICT"）リソースを作成または上書きします。

リソースを追加できなかった場合、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖の最上位にある現在のファイル（最後に開いたリソースファイル）に追加されます。

引数＜リソースデータ＞に空のピクチャフィールドまたは変数を渡した場合、このコマンドは何も行わず、システム変数OKが0に設定されます。

注：ピクチャリソースは、少なくとも数メガバイトのサイズになる可能性があります。

参照

GET PICTURE RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

GET ICON RESOURCE

GET ICON RESOURCE(リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	アイコンリソースID番号
リソースデータ	フィールドまたは変数	→	ピクチャを受け取るピクチャフィールドまたはピクチャ変数
		←	cicnリソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

GET ICON RESOURCEコマンドは、引数<リソースID>に渡されるIDを持つカラーアイコン（"cicn"）リソースに格納されているアイコンを引数<リソースデータ>のピクチャフィールドまたはピクチャ変数に返します。

リソースが見つからなかった場合、<リソースデータ>はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

例題

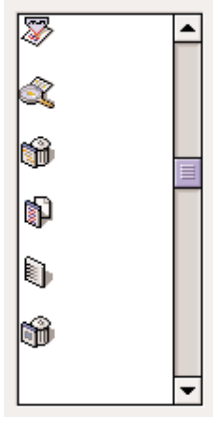
▼ 以下の例は、稼働している4Dアプリケーション内に配置されているカラーアイコンをピクチャ配列にロードします。

```
If(On 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

このコードを実行すると、配列がフォーム内に表示されると、以下のように見えます。



参照

GET PICTURE RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

GET RESOURCE

GET RESOURCE(リソースタイプ ; リソースID ; リソースデータ

{; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースデータ	BLOB	→	データを受け取るBLOBフィールドまたはBLOB変数
		←	リソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル

説明

GET RESOURCEコマンドは、引数＜リソースタイプ＞と＜リソースID＞に渡されるタイプとIDを持つリソースの内容を引数＜リソースデータ＞のBLOBフィールドまたはBLOB変数に返します。

重要：＜リソースタイプ＞には、4バイトの文字列を渡す必要があります。

リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：リソースは、少なくとも数メガバイトのサイズになる可能性があります。

プラットフォームの独立性：Macintoshベースのリソースを使って作業していることを忘れないでください。プラットフォームが何であろうと、倍長整数のような内部リソースデータはBLOB用の「Macintosh byte ordering」定義済み定数を使って格納されます。Windows上では、（ストリングリストリソースおよびピクチャリソース等の）標準リソースのデータは必要なときに自動的にバイトスワップされます。これに対して、ユーザ自身の内部データストラクチャを作成および使用する場合は、BLOBから取り出すデータをユーザの責任のもとでバイトスワップします（つまり、**BLOB to longint**関数へ「Macintosh byte ordering」定数を渡して）。

例題

▼ **SET RESOURCE**コマンドの例を参照してください。

参照

BLOBコマンド、リソース、SET RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

エラー処理

リソースをロードするための十分なメモリがない場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET RESOURCE

SET RESOURCE(リソースタイプ ; リソースID ; リソースデータ

{; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースデータ	BLOB	→	リソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

SET RESOURCEコマンドは、引数<リソースデータ>BLOBに渡されるデータを使って引数<リソースタイプ>と<リソースID>に渡されるタイプとIDを持つリソースを作成または上書きします。

重要：<リソースタイプ>には、4バイトの文字列を受け渡す必要があります。

リソースを書き込めなかった場合、システム変数OKに0が設定されます。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。<リソースファイル>を省略した場合は、リソースはリソースファイル連鎖の最上位にある現在のファイル（最後に開いたリソースファイル）に追加されます。

注：リソースは、少なくとも数メガバイトのサイズになる可能性があります。

プラットフォームの独立性：Macintoshベースのリソースを使って作業していることを忘れないでください。プラットフォームが何であろうと、倍長整数のような内部リソースデータはBLOB用の「Macintosh byte ordering」定義済み定数を使って格納されます。Windows上では、（ストリングリストリソースおよびピクチャリソース等の）標準リソースのデータは必要なときに自動的にバイトスワップされます。これに対して、ユーザ自身の内部データストラクチャを作成および使用する場合は、BLOBから取り出すデータをユーザの責任のもとでバイトスワップします（つまり、BLOB to longint関数「Macintosh byte ordering」定数を渡して）。

例題

4Dセッション中にインタープロセス変数でユーザのプレファレンス（環境設定）を管理すると仮定します。この環境設定をセッション間で保存するには、以下のように記述します。

1. **SAVE VARIABLES**コマンドと**LOAD VARIABLES**コマンドを使って、ディスク上の変数ドキュメントにその変数を保存して取り出します。

2. **VARIABLE TO BLOB**コマンド、**BLOB TO DOCUMENT**コマンド、**DOCUMEN T TO BLOB**、**BLOB TO VARIABLE**コマンドを使って、ディスク上のBLOBドキュメントにその変数を保存して取り出します。

3. **VARIABLE TO BLOB**コマンド、**SET RESOURCE**コマンド、**GET RESOURCE**コマンド、**BLOB TO VARIABLE**コマンドを使って、ディスク上のリソースファイルにその変数を保存して取り出します。

   以下の例は、3番目のメソッドの例です。「On Exit」データベースメソッドに記述します。

```
`「On Exit」データベースメソッド
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(<>bAutoRepeat ; $vxPrefData)
        VARIABLE TO BLOB(<>lCurTable ; $vxPrefData ; *)
        VARIABLE TO BLOB(<>sDfltOption ; $vxPrefData ; *)
        `...
        SET RESOURCE("PREF" ; 26500 ; $vxPrefData ; $vhResFile)
        CLOSE RESOURCE FILE($vhResFile)
End if
```

「On Startup」データベースに以下のように記述します。

```
`「On Startup」データベースメソッド
C_BOOLEAN(<>bAutoRepeat)
C_LONGINT(<>lCurTable)
$vbDone:=False
$vhResFile:=Open resource file("DB_Prefs")
If(OK=1)
        GET RESOURCE("PREF" ; 26500 ; $vxPrefData ; $vhResFile)
        If(OK=1)
                $vlOffset:=0
                BLOB TO VARIABLE($vxPrefData ; <>bAutoRepeat ; $vlOffset)
                BLOB TO VARIABLE($vxPrefData ; <>lCurTable ; $vlOffset)
                BLOB TO VARIABLE($vxPrefData ; <>sDfltOption ; $vlOffset)
```

```
            `...
            $vbDone:=False
        End if
        CLOSE RESOURCE FILE($vhResFile)
    End if
    If(Not($vbDone))
        <>bAutoRepeat:=False
        <>lCurTable:=0
        ARRY STRING(127 ; <>sDfltOption ; 0)
    End if
```

参照

BLOB Commands、GET RESOURCE

システム変数とシステムセット

リソースを書き込めた場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

Get resource name

Get resource name(リソースタイプ ; リソースID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開いているすべてのリソースファイル
戻り値	文字列	←	リソースの名前

説明

Get resource name関数は、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの名前を返します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在開かれているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource name**関数は空の文字列を返し、システム変数OKに0が設定されます。

例題

▼以下の例は、任意のリソースをコピーするプロジェクトメソッドです。そして、あるリソースファイルから別のリソースファイルにそのリソース名と属性をコピーします。

```
`「COPY RESOURCE」プロジェクトメソッド
` COPY RESOURCE(文字列 ; 倍長整数 ; 時間 ; 時間 )
` COPY RESOURCE( リソースタイプ ; リソースID ; コピー元ファイル ;
                                                    コピー先ファイル)
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
```

このメソッドをあなたのアプリケーションに組み込むと、以下のように記述できます。

```
` ファイルAからファイルBにリソース 'DATA' ID = 15000をコピーする
COPY RESOURCE("DATA" ; 15000 ; $vhResFileA ; $vhResFileB)
```

参照

SET RESOURCE PROPERTIES

システム変数とシステムセット

リソースが存在する場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

SET RESOURCE NAME

SET RESOURCE NAME(リソースタイプ ; リソースID ; リソース名

{; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソース名	文字列	→	リソースの新しい名前
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

SET RESOURCE NAMEコマンドは、引数<リソースタイプ>に渡されるタイプかつ引数<リソースID>に渡されるIDを持つリソースの名前を変更します。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、現在開かれているリソースファイル内で検索されます。

リソースが存在しない場合、**SET RESOURCE NAME**コマンドは何も行わずに、システム変数OKに0が設定されます。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

注：リソース名は、最大255バイトまで指定することができます。この名前は大文字小文字の区別を行いませんが、発音記号の区別は行います。

例題

▼ **Get resource name**関数の例を参照してください。

参照

SET RESOURCE PROPERTIES

システム変数とシステムセット

リソースが存在する場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

Get resource properties

Get resource properties(リソースタイプ ; リソースID {; リソースファイル})

→ 数値

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、開かれているすべてのリソースファイル
戻り値	数値	←	リソースの属性

説明

Get resource properties関数は、引数<リソースタイプ>に渡されるタイプかつ引数<リソースID>に渡されるIDを持つリソースの属性を返します。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、現在開かれているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource properties**関数は0を返し、システム変数OKに0が設定されます。

Get resource properties関数によって返される数値は、ビットが特別の意味を持っているビットフィールド値として理解する必要があります。リソース属性およびその影響に関する詳細は、**SET RESOURCE PROPERTIES**コマンドの節を参照してください。

例題

▼ **Get resource name**関数の例を参照してください。

参照

SET RESOURCE NAME

システム変数とシステムセット

リソースが存在する場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

SET RESOURCE PROPERTIES

SET RESOURCE PROPERTIES(リソースタイプ ; リソースID ; リソース属性
{; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソース属性	数値	→	リソースの新しい属性
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

SET RESOURCE PROPERTIESコマンドは、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの属性を変更します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在開かれているリソースファイル内で検索されます。

リソースが存在しない場合、**SET RESOURCE PROPERTIES**コマンドは何も行わずに、システム変数OKに0が設定されます。

引数＜リソース属性＞に渡す数値は、ビットが特別の意味を持っているビットフィールド値として表す必要があります。下記は、4th Dimensionによってあらかじめ用意されている定義済みの定数です。

定数	タイプ	値
System heap resource mask	倍長整数	64
System heap resource bit	倍長整数	6
Purgeable resource mask	倍長整数	32
Purgeable resource bit	倍長整数	5
Locked resource mask	倍長整数	16
Locked resource bit	倍長整数	4
Protected resource mask	倍長整数	8
Protected resource bit	倍長整数	3
Preloaded resource mask	倍長整数	4
Preloaded resource bit	倍長整数	2
Changed resource mask	倍長整数	2
Changed resource bit	倍長整数	1

上記の定数を使って、任意のリソース属性を構築することができます。後述の例を参照してください。

リソース属性とその及ぼす影響について

■ System heap

この属性を設定すると、リソースは4Dメモリ内ではなくシステムメモリの中にロードされます。もし、何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

■ Purgeable

この属性を設定すると、リソースがロードされた後、他のデータを割り当てるために空き領域が必要な場合、そのリソースをメモリからパージ（消去）することができます。4D BLOBの中にリソースをロードするため、メモリの使用を減らすためにユーザ独自のリソースをすべてパージ可能にすることは良い方法です。ただし、作業セッション中にこのリソースに頻繁にアクセスする場合、パージされたリソースの再ロードのディスクアクセスを減らすためにそれをパージ不可にしたい場合があるかもしれません。

■ Locked

この属性を設定すると、リソースがメモリの中にロードされた後、そのリソースを再配置することができなくなります（移動不可）。ロックされたリソースは、たとえパージ可能であっても、パージされません。リソースのロックは、メモリ領域のフラグメントという望ましくない影響を及ぼします。もし、何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

■ Protected

この属性を設定すると、リソースの名前、ID番号、およびリソースの内容を変更することができなくなります。また、このリソースを削除することもできなくなります。ただし、**SET RESOURCE PROPERTIES**コマンドを呼び出して、この属性を消去することができます。その後で再度、このリソースを変更、または削除することができるようになります。ほとんど、この属性を使用することはありません。

注：この属性は、Windows上では機能しません。

■ Preloaded

この属性を設定すると、リソースが配置されているリソースファイルが開かれると、そのリソースは自動的にメモリの中にロードされます。この属性は、任意のリソースファイルが開かれる際にロードされるリソースの最適化に有効です。ほとんど、この属性を使用することはありません。

■ Changed

この属性を設定すると、リソースが配置されているリソースファイルが閉じられると、“ディスク上に保存される必要がある”というマークが付けられます。**SET RESOURCE**コマンドはリソースの書き込みおよび上書きを実行することができるため、何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

例題

1. **Get resource name**関数の例を参照してください。

2. 以下の例は、リソース「"STR#" ID=17000」をパージ可能にします。ただし、その他の属性はそのまま変更しません。

```
$vlResAttr:=Get resource properties('STR#' ; 17000 ; $vhResFile)
SET RESOURCE PROPERTIES('STR#' ; 17000 ; $vlResAttr ?
                                + Purgeable resource bit ; $vhMyResFile)
```

3. 以下の例は、リソース「"STR#" ID=17000」の属性を「preloaded」と「purgeable」不可にします。

```
SET RESOURCE PROPERTIES('STR#' ; 17000 ; Preloaded resource mask ;
                                $vhResFile)
```

4. 以下の例は、リソース「"STR#" ID=17000」の属性を「preloaded」で「purgeable」にします。

```
SET RESOURCE PROPERTIES('STR#' ; 17000 ; Preloaded resource mask
                                + Purgeable resource mask ; $vhResFile)
```

参照

SET RESOURCE NAME

システム変数とシステムセット

リソースが存在する場合は、システム変数OKに1が設定されます。それ以外は0が設定されます。

DELETE RESOURCE

DELETE RESOURCE(リソースタイプ ; リソースID {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

説明

DELETE RESOURCEコマンドは、引数<リソースタイプ>に渡されるタイプかつ引数<リソースID>に渡されるIDを持つリソースを削除します。

オプション引数<リソースファイル>に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。<リソースファイル>を省略した場合は、現在開かれているリソースファイル内で検索されます。

リソースが存在しない場合、**DELETE RESOURCE**コマンドは何も行わずに、システム変数OKに0が設定されます。リソースが見つかって削除された場合は、システム変数OKに1が設定されます。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

例題

1. 以下の例は、リソース「"STR# ID=20000」を削除します。

```
`この例では現在開いているすべてのリソースの中で最初に見つかった
`「"STR# ID=20000」リソースを削除する点に注目してください
DELETE RESOURCE("STR#" ; 20000)
```

2. 以下の例は、指定されたリソースファイルの中で見つかったリソース「"STR# ID=20000」を削除します。

```
`この例ではローカル変数"$vhResFile"によって指定されたリソースファイル内にそのリソースが存在する場合にのみ「"STR# ID=20000」リソースが削除する点に注目してください
DELETE RESOURCE("STR#" ; 20000 ; $vhResFile)
`$vhResFile"によって指定された以外の現在開かれているリソースファイルの中に
`このようなリソースがある場合は、このリソースは何も手をつけられない点に注目
                                                    してください
```

3. プロジェクトメソッド「DELETE RESOURCE OF TYPE」は、(1番目の引数で）指定されたリソースファイルから（2番目の引数で）指定されたタイプのすべてのリソースを削除します。

```
`「DELETE RESOURCES OF TYPE」プロジェクトメソッド
` DELETE RESOURCES OF TYPE(時間 ; 文字列 )
` DELETE RESOURCES OF TYPE(リソースファイル ; リソースタイプ)
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
```

このプロジェクトメソッドをデータベースに組み込むと、以下のように記述できます。

```
`リソースファイル「$vhResFile」から“PREF”タイプのすべてのリソースを
                                                削除する
DELETE RESOURCES OF TYPE($vhResFile ; "PREF")
```

4. プロジェクトメソッド「DELETE RESOURCE BY NAME」は、リソースファイル名がわかっている（特定タイプの）リソースを削除します。

```
`「DELETE RESOURCE BY NAME」プロジェクトメソッド
` DELETE RESOURCE BY NAME(時間 ; 文字列 ; 文字列)
` DELETE RESOURCE BY NAME(リソースファイル ; リソースタイプ ; リソース名)
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
```

このプロジェクトメソッドをデータベースに組み込むと、以下のように記述できます。

```
`リソースファイル「$vhResFile」から“標準設定”という名前を持つ“PREF”
                                                リソースを削除する
DELETE RESOURCE BY NAME($vhResFile ; "PREF" ; "標準設定")
```

参照

RESOURCE LIST、SET RESOURCE PROPERTIES

システム変数とシステムセット

リソースが存在しない場合は、システム変数OKに1が設定されます。リソースが削除された場合は、システム変数OKに1が設定されます。

Get component resource ID

Get component resource ID(コンポーネント名; タイプ; オリジナル番号) →数値

引数	タイプ		説明
コンポーネント名	文字列(32)	→	リソースを参照するコンポーネント名
タイプ	文字列(4)	→	リソースタイプ（4桁） PICTまたはSTR#
オリジナル番号	数値	→	コンポーネントをインストール する前のリソースのオリジナル番号
戻り値	数値	←	現在のリソース番号

説明

Get component resource ID関数を使用すると、コンポーネントの開発者は、コンポーネントのインストール中にリソース番号が変更された場合でも、カスタマイズしたPICTやSTR#リソースの呼び出しが正しく実行されることが保証されます。

実際、独自のリソースを必要とするコンポーネントが4D Insiderによりインストールされる際、データベースリソースの中に同じIDを持つものが既に存在すれば、アプリケーションはこれらの新しいリソースの番号を自動的に振り替えることができます。

注：4th Dimensionのコンポーネントに関する詳細は、『4D Insiderリファレンス』を参照してください。

Get component resource ID関数は、リソースのタイプとオリジナルの番号をもとにして、コンポーネントにより使用される各リソースの現在の（実際の）番号を示します。

引数＜コンポーネント名＞には、所定のリソースを使用するコンポーネントの名前を渡します。

引数＜タイプ＞にはリソースタイプ（4桁のみ）を渡します。**Get component resource ID**関数は、PICTおよびSTR#タイプのリソースだけを受け付けます。

注：ピクチャライブラリに保存されたピクチャは、Get component resource ID関数によって処理されません。ピクチャライブラリのピクチャを4Dコンポーネントで使用するには、GET PICTURE FROM LIBRARYコマンドを呼び出し、最初の引数として文字列（ピクチャ名）を渡さなくてはなりません。詳細は、GET PICTURE FROM LIBRARYコマンドの説明を参照してください。

引数＜オリジナル番号＞には元のリソース番号、つまりデザインの時点で定義した番号を渡します。

関数は、データベースで使用される現在のリソースIDを返します。

＜オリジナル番号＞とコンフリクトするリソースがない場合、**Get component resource ID**は＜オリジナル番号＞として渡された値を返します。

例題

次のコード例では、リソース呼び出しが正しく行われるかどうか保証されません。

```
`リソース番号が変更された場合、この呼び出しは正しくなくなる
vResNumb := 15000
STRING LIST TO ARRAY(vResNumb; stringArray;resFile)
```

したがって、次のようなコードを使用するようお勧めします。

```
`この呼び出しは常に正しくなる
vResNumb :=Get component resource ID("Mycomp";"STR#";15000)
STRING LIST TO ARRAY(vResNumb; stringArray; resFile)
```

参照

GET SERIAL INFORMATION

GENERATE ENCRYPTION KEYPAIR

GENERATE ENCRYPTION KEYPAIR(秘密鍵; 公開鍵 {; 長さ})

引数	タイプ		説明
秘密鍵	BLOB	←	秘密鍵を納めるBLOB
公開鍵	BLOB	←	公開鍵を納めるBLOB
長さ	倍長整数	→	鍵の長さ（ビット単位）[386...1024] デフォルト値 = 512

説明

GENERATE ENCRYPTION KEYPAIRコマンドは、新しく1対のRSA鍵を生成します。4Dで提供されるセキュリティシステムは、情報の暗号化／解読のために設計されたこれらの鍵をベースにしています。これらの鍵は、SSLプロトコル、4D Webサーバ（暗号化および暗号化通信）、およびすべてのデータベース（データの暗号化）で使用されます。

注：4th Dimensionでは、この鍵の生成に適したRC4（128ビット）暗号を使用しています。

コマンドが実行されると、引数＜秘密鍵＞と＜公開鍵＞に渡されたBLOBには新しい1対の暗号化鍵が納められます。

オプションの引数＜長さ＞を使用して、鍵のサイズ（ビット単位）を設定することができます。鍵が大きいほど、暗号コードの解読は困難になります。ただし、鍵が大きくなると、実行時間や応答時間が長くなり、特にSSL接続ではこれが顕著です。

デフォルトでは（引数＜長さ＞を省略した場合）、生成される鍵のサイズは512ビットに設定されますが、このサイズはセキュリティ／効率性の割合からみてもほどよい大きさです。セキュリティ要素を増すには、例えば6ヶ月ごとに変える、というように、鍵を頻繁に変更します。

暗号の安全性を高めるために1,024ビットの鍵を生成できますが、Webアプリケーションの接続速度は低下します。

注：

・SSL証明書リクエストを発行するために鍵の生成を行う場合、512ビットおよび1,024ビットの長さの鍵しか認められない点に注意してください。

・大部分のブラウザでは、512ビットを超える鍵は受け付けられません。

このコマンドは、PEM（Privacy Enhanced Mail）フォーマットで鍵を生成します。PEMフォーマットでは、変更せずにその内容を電子メールへコピー＆ペーストできます。一対の鍵が一度生成されると、テキストドキュメントを作成することができ（例えば**BLOB TO DOCUMENT**コマンドを使用）、これらの鍵を安全な場所に保管できます。

常に、秘密鍵は知られないようにしてください。

RSA、秘密鍵と公開鍵

GENERATE ENCRYPTION KEYPAIRコマンドで使用するRSA暗号方式は、秘密鍵と公開鍵という二重鍵暗号システムに基づいています。その名が示す通り、公開鍵は第三者に渡され、情報の解読に使用されます。公開鍵は、情報の暗号化に使われるユニークな秘密鍵と合わされます。このように、秘密鍵は暗号化に使用され、公開鍵は解読に使用されます（またはその逆）。一方の鍵を使って暗号化された情報は、もう一方の鍵を使用しなければ解読することはできません。

SSLプロトコルの暗号化機能は、この原理に基づいており、証明書に納められた公開鍵がブラウザに送信されます。（「Webサービス：SSLプロトコルの使用」の節を参照）。

この暗号化モードは、**ENCRYPT BLOB**コマンドおよび**DECRYPT BLOB**コマンドの1番目のシンタックスでも使用されています。このシンタックスで用いる公開鍵は極秘に発行してください。

受信者が解読できる唯一の人となり、かつ送信者が暗号化を行った唯一の人となるように、2人の公開鍵と秘密鍵を合わせて情報の暗号化を行うことができます。この原理は、**ENCRYPT BLOB**コマンドと**DECRYPT BLOB**コマンドの2番目のシンタックスで示されています。

例題

ENCRYPT BLOBコマンドの例題を参照してください。

参照

DECRYPT BLOB、ENCRYPT BLOB、GENERATE CERTIFICATE REQUEST

GENERATE CERTIFICATE REQUEST

GENERATE CERTIFICATE REQUEST(秘密鍵; 証明書リクエスト; コード配列; 名称配列)

引数	タイプ		説明
秘密鍵	BLOB	→	秘密鍵を納めるBLOB
証明書リクエスト	BLOB	←	証明書リクエストを受け取るBLOB
コード配列	倍長整数配列	→	情報コードリスト
名称配列	文字列配列	→	名称リスト

説明

GENERATE CERTIFICATE REQUESTコマンドは、PEM（Privacy Enhanced Mail）フォーマットで証明書リクエストを生成します。このフォーマットは、Verisign®等の認証局により直接使用されています。証明書は、SSL暗号化プロトコルの重要な部分を務めます。これはSSLモードで接続している各ブラウザに送信され、Webサイトの“IDカード”（このコマンドに指定した情報をもとに作成）とともに、ブラウザが受信情報の解読に使用できる公開鍵も納められています。さらに、この証明書には、整合性を保証する認証局により加えられた各種情報も納められます。

注：4D Webサーバで使用するSSLプロトコルに関する詳細は、「Webサービス：SSLプロトコルの使用」の節を参照してください。

証明書リクエストには、**GENERATE ENCRYPTION KEYPAIR**コマンドで生成した一対の鍵が使用され、各種情報が納められます。認証局では、このリクエストと他の引数を組み合わせて証明書を作成します。

＜秘密鍵＞には、**GENERATE ENCRYPTION KEYPAIR**コマンドで生成した秘密鍵を納めたBLOBを渡します。

＜証明書リクエスト＞には空のBLOBを渡します。コマンドが実行されると、この引数には証明書リクエストがPEMフォーマットで納められます。このリクエストは、認証局へ提出する目的に、例えば**BLOB TO DOCUMENT**コマンドを使用して、テキストファイルへ保存することができます。

警告：秘密鍵はリクエストの作成に使用しますが、認証局へ送信しないでください。

配列＜コード配列＞（倍長整数）および＜名称配列＞（文字列）にはそれぞれ、認証局で必要となるコード番号と情報内容を納めます。

必要とされるコードおよび名称は、認証局や証明書の用法によって変わる場合があります。しかし、証明書の通常の用途であれば（SSL経由でのWebサーバ接続）、この配列には以下の項目を納める必要があります。

提供する情報	コード配列	名称配列（例）
CommonName	13	www.4D-Japan.com
CountryName（2桁）	14	JP
LocalityName	15	世田谷
StateOrProvinceName	16	東京
OrganizationName	17	フォーディー・ジャパン
OrganizationUnit	18	Web 管理者

コードと情報内容の入力順は問いませんが、これら2つの配列は同期していなくてはなりません。つまり、＜コード配列＞の3番目の項目の値が15（都市名）であれば、＜名称配列＞の3番目の項目にはその情報を納める必要があります。ここの例題では世田谷になります。

例題

“証明書リクエスト”フォームには、標準の証明書リクエストで必要となる6つのフィールドを納めます。「Generate」ボタンにより、証明書リクエストを納めたドキュメントがディスク上に作成されます。秘密鍵（**GENERATE ENCRYPTION KEYPAIR**コマンドで作成）を納めた“Privatekey.txt”もディスク上に存在している必要があります。



次は「Generate」ボタンのメソッドです。

```
` bGenerate オブジェクトメソッド
C_BLOB($vbprivateKey;$vbcertifRequest)
C_LONGINT($arrayNber)
ARRAY LONGINT($tLCodes;6)
ARRAY STRING(80;$tSInfos;6)
$tableNum:=Table(Current form table)
For($i;1;6)
    $tSInfos{$i}:= Field($tableNum;$i)->
    $tLCodes{$i}:=$i+12
End for
If(Find in array($tSInfos;"") # -1)
    ALERT("すべてのフィールドに入力してください。")
Else
    ALERT("秘密鍵を選択してください。")
    $vhDocRef:=Open document("")
    If(OK=1)
        CLOSE DOCUMENT($vhDocRef)
        DOCUMENT TO BLOB(Document;$vbprivateKey)
    End if
    GENERATE CERTIFICATE REQUEST($vbPrivateKey;$vbcertifRequest;
$tLCodes; $tSInfos)
    BLOB TO DOCUMENT("Request.txt";$vbcertifRequest)
End if
```

参照

GENERATE ENCRYPTION KEYPAIR

ALL RECORDS

ALL RECORDS {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	全レコードを選択するテーブル、または省略した場合、デフォルトテーブル

説明

ALL RECORDSコマンドは、カレントプロセスの<テーブル>の全レコードをカレントセレクションにします。**ALL RECORDS**コマンドは、先頭のレコードをディスクからロードし、カレントレコードに設定します。**ALL RECORDS**コマンドは、レコードの順序をデフォルトのレコード順序に戻します。これはレコードがディスクに保存されている順序です。

例題

▼ 以下の例は、[従業員]テーブルのすべてのレコードを表示します。

```
ALL RECORDS([従業員])    ` 従業員テーブルのすべてのレコードを
                                カレントセレクションにする
DISPLAY SELECTION([従業員])    ` 出力フォームにレコードを表示
```

参照

DISPLAY SELECTION、MODIFY SELECTION、ORDER BY、QUERY、Records in selection、Records in table

Records in selection

Records in selection{(テーブル)} → 数値

引数	タイプ		説明
テーブル	テーブル	→	カレントセレクションの数を求めるテーブル、または省略した場合、デフォルトテーブル
戻り値	数値	←	テーブルのセレクション中のレコード数

説明

Records in selection関数は、＜テーブル＞のカレントセレクションのレコード数を返します。一方、**Records in table**関数はテーブルの全レコード数を返します。

例題

▼以下の例は、カレントセレクションのすべてのレコード間を移動するのによく使用されるループ処理です。**APPLY TO SELECTION**コマンドを代用してもよいでしょう。

```
FIRST RECORD([従業員])  ` カレントセレクションのレコード数だけ繰り返す
    `カレントセレクションを1つずつループ
For($i ; 1 ; Records in selection([従業員]))
    Do Something        ` レコードに対して何らかの処理を実行
    NEXT RECORD([従業員])  ` 以下のレコードに移動
End for
```

参照

Records in table

DELETE SELECTION

DELETE SELECTION {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	カレントセレクションを削除するテーブル 省略した場合、デフォルトテーブル

説明

DELETE SELECTIONコマンドは、＜テーブル＞のカレントセレクションに含まれるすべてのレコードを削除します。カレントセレクションが空の場合は、**DELETE SELECTION**コマンドは何も行いません。レコードが削除された後、カレントセレクションは空になります。トランザクション処理中に削除されたレコードは、トランザクション処理が実行または取り消されるまで、他のユーザや他のプロセスに対してロックされます。

警告：レコードの削除は、恒久的な操作です。削除が実行されると元に戻すことはできません。

「テーブルプロパティ」ダイアログボックスの属性枠内の「完全に削除」オプションを選択することにより、**DELETE SELECTION**コマンドを使用する際、レコード削除処理を高速にすることができます。

例題

1. 以下の例は、[従業員]テーブルの、すべてのレコードを表示します。ユーザは、この中から削除したいものを選択します。この例には、2つのメソッドがあります。第1のメソッドは、レコードを表示するグローバルメソッドです。第2のメソッドは、“削除”というボタンのオブジェクトメソッドです。次に第1のメソッドを示します。

```
ALL RECORDS([従業員])  ` 全レコードをカレントセレクションにする
OUTPUT FORM(([従業員]) ; "リスト")  ` 出力フォームをセット
DISPLAY SELECTION([従業員])  ` すべてのレコードを表示
```

次に示すのは、「削除」ボタンのオブジェクトメソッドです。このボタンは出力フォームのフッタエリアに表示されます。このオブジェクトメソッドでは、セレクションを削除するために、ユーザが選択したレコード（Userset）を使用します。ユーザがレコードを1件も選択しなかった場合、**DELETE SELECTION**コマンドは何も行わないという点に注意してください。

```
` ユーザがレコードを削除したいかどうかを確認
CONFIRM("削除対象となるのは"+String(Records in set("UserSet"))+"名です。"
      +Char(13)+"削除する場合は、OK をクリックしてください。")
If(OK=1)
      USE SET("UserSet") ` ユーザが選択したレコードをセレクションにする
      DELETE SELECTION([従業員]) ` セレクションのレコードを削除
End if
ALL RECORDS([従業員]) ` 全レコードをカレントセレクションにする
```

2. **DELETE SELECTION**コマンドの実行中にロックされたレコードが見つかると、そのレコードは削除されません。ロックされたレコードはすべて“LocedSet”というセットに納められます。**DELETE SELECTION**コマンドの実行後、LocedSetを調べて、ロックされているレコードが存在していたかどうかを知ることができます。次は、ループを使用してすべてのレコードを削除します。

```
Repeat      ` ロックされたレコードを繰り返す
      DELETE SELECTION([テーブル])
If(Records in set("lockedSet")#0) `レコードがロックされている場合
      USE SET("LockedSet")      ` ロックされたレコードだけを選択
End if
Until(Records in set("LockedSet")=0) ` ロックされたレコードがなくなるまで続ける
```

参照

DISPLAY SELECTION、MODIFY SELECTION、レコードロック、セット

Selected record number

Selected record number {(テーブル)} → 数値

引数	タイプ		説明
テーブル	テーブル	→	レコード位置番号を求めるテーブル 省略した場合、デフォルトテーブル
戻り値	数値	←	カレントレコードのレコード位置番号

説明

Selected record number関数は、＜テーブル＞のカレントセレクション内のカレントレコードの位置を返します。

セレクションが空ではなく、カレントレコードがそのセレクションに含まれる場合は、**Selected record number**関数は1から**Records in selection**までの値を返します。セレクションが空で、カレントレコードが存在しない場合には、この関数は0を返します。

レコード位置番号は、**Record number**関数で求めたレコード番号とは違います。**Record number**関数は、絶対レコード番号を返します。レコード位置番号は、カレントセレクションおよびカレントレコードに依存します。

例題

▼以下の例は、カレントレコードのレコード位置番号を変数に格納します。

```
v位置番号:=Selected record number([従業員])  ` レコード位置番号を求める
```

参照

レコードに付けられた番号の使用、GOTO SELECTED RECORD、Records in selection

GOTO SELECTED RECORD

GOTO SELECTED RECORD({テーブル ;} レコード位置番号)

引数	タイプ		説明
テーブル	テーブル	→	移動するレコードの属するテーブル 省略した場合、デフォルトテーブル
レコード位置番号	数値	→	セレクション内のレコード位置

説明

GOTO SELECTED RECORDコマンドは、＜テーブル＞のカレントセレクション内の指定されたレコードに移動し、そのレコードをカレントレコードにします。カレントセレクションは変更されません。引数＜レコード位置番号＞は、カレントセレクション内のレコードの位置を指定します。**Record number**関数で求められるレコード番号ではありません。このレコード位置は、セレクションの作成方法およびセレクションがソートされているかどうかによって変わります。

カレントセレクションにレコードが全く存在しない場合、あるいは指定したレコード位置番号がセレクション内に存在しない場合には、**GOTO SELECTED RECORD**コマンドは、何も行いません。

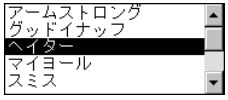
＜レコード位置番号＞に0を渡すと、＜テーブル＞のカレントレコードが存在しなくなります。これにより、例えばサブフォームに含まれる一覧上のすべてのレコードを取り除くことができます（空の一覧を作る）。

例題

▼以下の例は、[従業員]名字フィールドの内容を“a名字”という配列に取り込み、“a位置番号”という整数配列にレコード位置番号を取り込みます。両方の配列をソートし、その配列を使用してセレクション内のレコードを参照します。

```
` 名字を配列にコピーする。
SELECTION TO ARRAY([従業員]名字 ; a名字)
` レコード位置番号を格納する配列を定義する
$vlNbRecords:=Size of array(a名字)
ARRAY LONGINT(a位置番号 ; $vlNbRecords)
For($i ; 1 ; $vlNbRecords)   ` 配列にレコード位置番号を取り込む
    a位置番号{$i}:=$i
End for
SORT ARRAY(a名字 ; a位置番号 ; >)      ` 両方の配列をソートする。
```

配列“a名字”をスクロールエリアに表示すれば、ユーザは、その中の1つの項目をクリックすることによって選択することができます。ユーザが項目をクリックすると、その項目の位置が数値で配列の名前“a名字”に対して代入されます。以下の図の例では、3番目の項目が選択されています。したがって、“a名字”に3が代入されます。



“a名字”の値は、セレクションのレコードをロードするのに使用します。また、“a名字”の値は、配列“a位置番号”の配列要素をアクセスするのにも使用します。この場合の配列“a位置番号”の配列要素は、スクロールエリアでクリックされた項目に対応するレコードのレコード位置番号です。以下のメソッドは、スクロールエリア“a名字”のオブジェクトメソッドです。ユーザがクリックしたレコードをロードするために**GOTO SELECTED RECORD**コマンドを使用しています。

```
Case of
¥(Form event=On Clicked)
    If(a名字#0)
        GOTO SELECTED RECORD(a位置番号{a名字})
    End if
End case
```

参照

Selected record number

FIRST RECORD

FIRST RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	先頭のレコードに移動するテーブル 省略した場合、デフォルトテーブル

説明

FIRST RECORDコマンドは、＜テーブル＞のカレントセレクションの先頭のレコードをディスクからロードし、カレントレコードに設定します。検索、選択、ソートコマンドも先頭のレコードをカレントレコードに設定します。カレントセレクションが存在しない場合や、カレントレコードが既にセレクションの最初のレコードである場合、**FIRST RECORD**コマンドは何も行いません。

このコマンドは、最初のレコードから始めてレコードセレクションをループする処理に入る前に、**USE SET**コマンドの次でよく使用されます。しかし、カレントレコードが実際に先頭のレコードであるかどうかが確かではない場合、このコマンドをサブルーチンから呼び出すこともできます。

例題

▼以下の例は、[顧客]テーブルの最初のレコードを先頭レコードにします。

```
FIRST RECORD([顧客])
```

参照

Before selection、End selection、LAST RECORD、NEXT RECORD、PREVIOUS RECORD

Displayed line number

Displayed line number→ 倍長整数

引数	タイプ		説明
このコマンドには引数は必要ありません。			
戻り値	倍長整数	←	表示されている行の数

説明

Displayed line number関数は「On display detail」フォームイベントでのみ機能します。関数は、画面上にレコードリストが表示される間に処理される行の数を返します。**Displayed line number**関数はリスト表示以外の場面で呼び出されると、0を返します。

表示された行が空ではない場合（行がレコードに関連付けられている場合）、**Displayed line number**関数から返される値は**Selected record number**関数から返される値と同じです。

Selected record number関数と同様に、**Displayed line number**関数から返される値は1から始まります。このコマンドは、空の行も含め、画面上に表示されたリストフォームの各行を処理したい場合に役立ちます。

例題

▼次の例題により、画面上に表示されるリストフォームに対し、レコードが表示されない行に対しても代替色を割り当てることができます。

```
`リストフォームのメソッド
If(Form event=On display detail)
    If(Displayed line number% 2= 0)
        `偶数行のテキストに対しては白地に黒
        SET RGB COLORS([Table 1]Field1; -1; 0x00FFFFFF)
    Else
        `奇数行のテキストに対しては水色地に白
        SET RGB COLORS([Table 1]Field1; -1; 0x00E0E0FF)
    End if
End if
```

NEXT RECORD

NEXT RECORD({テーブル})

引数	タイプ		説明
テーブル	テーブル	→	次のレコードに移動するテーブル 省略した場合、デフォルトテーブル

説明

NEXT RECORDコマンドは、カレントプロセスの＜テーブル＞のカレンセレクションにある次のレコードへカレントレコードポインタを移動します。カレントセレクションが空の場合、あるいは、**Before Selection**関数や**End selection**関数が“True（真）”の場合は、**NEXT RECORD**コマンドは何も行いません。

NEXT RECORDコマンドで、カレントセレクションの最後を超えてカレントレコードポインタを移動した場合は、**End selection**関数は“True（真）”を返し、カレントレコードはなくなります。この場合、**FIRST RECORD**コマンド、**LAST RECORD**コマンド、**GOTO SELECTED RECORD**コマンドを使用して、カレントレコードポインタをカレントセレクション内に戻します。

参照

Before selection、End selection、FIRST RECORD、LAST RECORD、PREVIOUS RECORD

LAST RECORD

LAST RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	最後のレコードに移動するテーブル 省略した場合、デフォルトテーブル

説明

LAST RECORDコマンドは、＜テーブル＞のカレントセレクションの最終のレコードをディスクからロードし、カレントレコードに設定します。カレントセレクションが空の場合は、**LAST RECORD**コマンドは何も行いません。

例題

▼以下の例は、[従業員]テーブルの最後のレコードをカレントレコードにします。

```
LAST RECORD([従業員])
```

参照

Before selection、End selection、FIRST RECORD、NEXT RECORD、PREVIOUS RECORD

PREVIOUS RECORD

PREVIOUS RECORD {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	前のレコードに移動するテーブル 省略した場合、デフォルトテーブル

説明

PREVIOUS RECORDコマンドは、カレントプロセスの＜テーブル＞のカレントセレクションにある1つ前のレコードへカレントレコードポインタを移動します。カレントセレクションが空の場合、または**Before selection**関数や**End selection**関数が“True（真）”の場合は、**PREVIOUS RECORD**コマンドは何も行いません。

PREVIOUS RECORDコマンドで、カレントセレクションの前にカレントレコードポインタを移動する場合は、**Before selection**関数は“True（真）”を返し、カレントレコードはなくなります。この場合に、**FIRST RECORD**、**LAST RECORD**、**GOTO SELECTED RECORD**の各コマンドを使用して、カレントレコードポインタをカレントセレクション内に戻します。

参照

Before selection、End selection、FIRST RECORD、LAST RECORD、NEXT RECORD

Before selection

Before selection {(テーブル)} → ブール

引数	タイプ		説明
テーブル	テーブル	→	カレントセレクションの前を判定する
テーブル			
			省略した場合、デフォルトテーブル
戻り値	ブール	←	True：Yes、False：No

説明

Before selection関数は、カレントレコードポインタが＜テーブル＞のカレントセレクションの前にある場合は“True（真）”を返します。**Before selection**関数は、一般に**PREVIOUS RECORD**コマンドで、カレントレコードポインタを先頭のレコードの前に移動したかどうかを調べるために使用します。カレントセレクションが空の場合には、**Before selection**関数は“True（真）”を返します。

カレントレコードポインタをセレクションに内に戻すには、**LAST RECORD**、**FIRST RECORD**、**GOTO SELECTED RECORD**の各コマンドを使用します。**NEXT RECORD**コマンドではポインタはセレクション内に戻りません。

PRINT SELECTIONコマンド、または「ユーザ」モードの「プリント...」メニューを選択してレポートを印刷する場合も、**Before selection**関数は最初のヘッダで“True（真）”を返します。以下のステートメントを使用して、最初のヘッダを判定し、先頭ページに特殊なヘッダを印刷することができます。

```
` 集計レポート用出力フォームとして使用されるフォームのメソッド
$vpFormTable:=Current form table
Case of
` ...
    ¥(Form event=On Header)
    ` ヘッダエリアが印刷される
        Case of
            ¥(Before selection($vpFormTable->))
                ` 最初のブレークヘッダのコードをここに書く
                ` ...
        End case
End case
```

例題

以下の例は、レポートの印刷中に使用します。変数“vタイトル”を設定し、先頭ページのヘッダエリアに印刷します。

```
` 集計フォームのフォームメソッド
Case of
` ...
        ¥(Form event=On Header)
                Case of
                        ¥(Before selection([探傷検査))
                                vタイトル="検査報告書 1996"        ` 最先頭ページの
                                                                    タイトルをセット
                Else
                                vタイトル:="" ` 他のページのタイトルをクリア
                End case
End case
```

参照

End selection、FIRST RECORD、Form event、PREVIOUS RECORD、PRINT SELECTION

End selection

End selection{(テーブル)} → ブール

引数	タイプ		説明
テーブル	テーブル	→	カレントセレクションの後を判定するテーブル 省略した場合、デフォルトテーブル
戻り値	ブール	←	True：Yes、False：No

説明

End selection関数は、カレントレコードポインタが<テーブル>のカレントセレクションの後にある場合は“True（真）”を返します。一般に**End selection**関数は、**NEXT RECORD**コマンドで、カレントレコードポインタを最後のレコードの後に移動したかどうかをチェックするために使用します。カレントセレクションが空の場合には、**End selection**関数は“True（真）”を返します。

カレントレコードのポインタをセレクション内に戻すには、**LAST RECORD**、**FIRST RECORD**、**GOTO SELECTED RECORD**の各コマンドのいずれかを使用します。**PREVIOUS RECORD**コマンドでは、ポインタはセレクション内に戻りません。

PRINT SELECTIONコマンド、または「ユーザ」モードの「プリント...」メニューを選択してレポートを印刷する場合は、**End selection**関数は最後のフッタで“True（真）”を返します。以下のステートメントを使用して、最後のフッタを判定して最終ページに特殊なフッタを印刷することができます。

```
` 集計レポート用出力フォームとして使用されるフォームのメソッド
$vpFormTable:=Current form table
Case of
	` ...
	¥(Form event=On Printing Footer)
		` フッタが印刷される
		If(End selection($vpFormTable->))
			` 最終フッタのコードをここに書く
		Else
			` フッタのコードをここに書く
		End if
End case
```

例題

以下の例はレポートの印刷中に使用します。変数“vフッタ”を設定し、最終ページのフッタエリアに印刷します。

```
` 集計フォームのフォームメソッド
Case of
` ...
        ¥(Form event=On Printing Footer)
                If(End selection([探傷検査]))
                        vフッタ:="江田超音波試験株式会社" ` 最終ページのタイトルをセット
                Else
                        vフッタ:=""    ` 他のページのタイトルをクリアする
                End if
End case
```

参照

Before selection、Form event、LAST RECORD、NEXT RECORD、PRINT SELECTION

DISPLAY SELECTION

DISPLAY SELECTION({テーブル} {; 選択モード}{; リスト更新}{; *{; *})

引数	タイプ		説明
テーブル	テーブル	→	表示するテーブル、または 省略した場合、デフォルトテーブル
選択モード	倍長整数	→	選択モード
リスト更新	ブール	→	リスト更新可オプションを許可する
*		→	単一レコードからなるセレクションの場合に出力フォームを使用し、かつ入力フォームのスクロールバーを隠す
*		→	入力フォームのスクロールバーを表示する（=1番目の<*>引数の後者の機能を無効にする）

説明

DISPLAY SELECTIONコマンドは、出力フォームを使用して、<テーブル>のカレントセレクションを表示します。レコードは、「ユーザ」モードの出力に類似のスクロール可能なリストとして表示されます。ユーザがレコードをダブルクリックすると、デフォルトとしてそのレコードはカレント入力フォーム上に表示されます。リストは最前面にあるウインドウに表示されます。

セレクションを表示し、さらににレコードをダブルクリックしてカレント入力フォーム上で修正するには（「ユーザ」モードのウインドウで実行する場合と同様）、**DISPLAY SELECTION**コマンドではなく**MODIFY SELECTION**コマンドを使用してください。

次に説明する情報は、レコードの修正に関する情報を除き、すべて**DISPLAY SELECTION**コマンドと**MODIFY SELECTION**コマンドの両方に適用されます。

DISPLAY SELECTIONコマンドを実行した直後は、カレントレコードが存在しない場合があります。カレントレコードを必要とする場合は、**FIRST RECORD**コマンドまたは**LAST RECORD**コマンドを使用してください。

引数<選択モード>を使用し、マウスを用いてリスト上のレコードを選択できるように設定することができます。この引数には、「Form Options」テーマのいずれかの定数を渡すことができます。

■ 「No Selection(0)」を渡すと、リスト上のレコードを選択することはできません。

■ 「Single Selection(1)」を渡すと、一度に1レコードだけを選択できます。

■「Multiple Selection(2)」を渡すと、複数レコードを同時に選択することができます。連続しているレコードを選択するには、選択する最初のレコードをクリックし、次にShiftキーを押しながらセレクションに含めたい最後のレコードをクリックします。連続していないレコードを選択するには、Ctrlキー（Windows）または、Commandキー（Mac OS）を押したまま各レコードを個別にクリックします。

引数<選択モード>を渡さない場合は、デフォルトとして「Multiple Selection」モードが使用されます。

引数<リスト更新>を使用すると、表示されるリストの「リスト更新可」オプションを許可することができます。これにより、ユーザは直接出力フォーム上でレコードの値を選択し、変更できるようになります。このモードを有効にするには“True”を渡し、無効にするには“False”を渡します。引数<リスト更新>を渡さない場合は、デフォルトとして「リスト更新可」オプションが無効になります。

DISPLAY SELECTIONコマンドでは、この引数によりリストの値の選択が許可されるだけであり、変更は許可されないということを覚えておいてください。実際、**DISPLAY SELECTION**コマンドにより、カレントテーブルはリードオンリー（読み込みのみ）になります。**MODIFY SELECTION**コマンドを使用した場合にのみ、実際に値を入力することができます。

注：SET ENTERABLEコマンドを使用し、オンザフライで「リスト更新可」オプションを有効、または無効に設定することができます。

オプションの<*>引数に関する規則を次に説明します。

■ セレクションにレコードが1件しか存在しないときに、1番目のオプションのアスタリスク（*）を指定していなければ、そのレコードは出力フォームではなく入力フォーム上に表示されます。

■ 1番目のオプションのアスタリスク（*）を指定した場合は、セレクションに1件しかレコードが存在しない場合でも出力フォームを使用します（入力フォームを使用しない）。

■ 1番目のオプションのアスタリスク（*）を指定し、ユーザがレコードをダブルクリックしてそれを入力フォームに表示した場合には、入力フォームのスクロールバーは表示されません。これを無効にするには、2番目のオプションのアスタリスクを指定します。

DISPLAY SELECTIONコマンドの実行を終了するため、出力フォームのフッタまたはヘッダエリアにカスタムボタンを配置することができます。出力フォームのフッタエリアにこのボタンを作成ことができます。また、レコード表示を終了させるために「登録」ボタンや「キャンセル」ボタンを使用、あるいは**ACCEPT**や**CANCEL**コマンドを呼び出すオブジェクトメソッドを利用することもできます。**DISPLAY SELECTION**コマンドにより呼び出された出力フォームにボタンが存在しない場合は、Escapeキー（Windows）またはEscキー（Mac OS）を使用するとリストを終了することができます。

DISPLAY SELECTIONコマンドの実行後、ユーザが選択したレコードは“UserSet”という名前のセットに格納されます。UserSetは、ボタンがクリックされるか、メニュー項目が選択された際に、オブジェクトメソッドに対して表示されたセレクション内で使用できます。また、コマンド終了後に**DISPLAY SELECTION**コマンドを呼び出すプロジェクトメソッドからも利用できます。

例題

1. 以下の例は、最初に、[従業員]テーブルの全レコードをカレントセレクションにします。。次に、**DISPLAY SELECTION**コマンドを使用してレコードを表示し、ユーザがプリントするレコードを選択します。最後に、選択されたレコードを**USE SET**コマンドでカレントセレクションにし、**PRINT SELECTION**コマンドでそのレコードを印刷します。

```
ALL RECORDS([従業員])  ` 全レコードをカレントセレクションにする
DISPLAY SELECTION([従業員] ; *)      ` レコードの表示
USE SET("UserSet") ` ユーザが選択したレコードだけを使用
PRINT SELECTION([従業員])      ` ユーザが選択したレコードの印刷
```

2. **Form event**関数の6番目の例題を参照してください。この例題では、**DISPLAY SELECTION**コマンドの実行中に発生するイベントをすべて監視するためのあらゆるチェックが示されています。

3. 「カスタム」モードで**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドを使用する際に、例えば、「ユーザ」モードの「クエリ」メニューで提供される機能を再現するには、以下の手順にしたがってください。

a. 「デザイン」モードで必要なメニューを備えたメニューバーを作成します。例えば、“すべて表示”、“クエリ”、“並び替え”等のメニューです。

b. このメニューバーに（負のメニューバー番号を使用して）、**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドを使用した出力フォームを関連付けます。

c. 以下のプロジェクトメソッドをメニューに関連付けます。

```
` M_SHOW_ALL(「すべて表示」メニュー項目に付属)
$vpCurTable:=Current form table
ALL RECORDS($vpCurTable->)

` M_QUERY(「クエリ」メニュー項目に付属)
$vpCurTable:=Current form table
QUERY($vpCurTable->)

` M_ORDDER_BY(「並び替え」メニュー項目に付属)
$vpCurTable:=Current form table
```

ORDER BY($vpCurTable->)

「カスタム」モードでセレクションの表示や修正を実行するたび、“標準”メニューオプションをすべて提供するために、**PRINT SELECTION**、**QR REPORT**等、他のコマンドも使用することができます。**Current form table**コマンドを使用すれば、これらのメソッドは汎用コードとなり、このメニューバーをあらゆるテーブルのあらゆる出力フォームに関連付けることができます。

参照

Form event、MODIFY SELECTION、セット

MODIFY SELECTION

MODIFY SELECTION({テーブル} {; 選択モード}{; リスト更新} {; *{; *}})

引数	タイプ		説明
テーブル	テーブル	→	表示し、変更するテーブル、または省略した場合、デフォルトテーブル
選択モード	倍長整数	→	選択モード
リスト更新	ブール	→	リスト更新可オプションを許可する
*		→	単一レコードからなるセレクションの場合に出力フォームを使用し、かつ入力フォームのスクロールバーを隠す
*		→	入力フォームのスクロールバーを表示する（=1番目の<*>引数の後者の機能を無効にする）

説明

MODIFY SELECTIONコマンドは、**DISPLAY SELECTION**コマンドとほぼ同じ機能を提供します。詳細については、**DISPLAY SELECTION**コマンドの説明を参照してください。2つのコマンドの違いを次にあげます。

1. **DISPLAY SELECTION**コマンドはカレントレコードセレクションをリストモードで表示します。また、ユーザがレコードをダブルクリックすると、そのレコードは入力フォーム上に表示されます。一方、**MODIFY SELECTION**コマンドを使用した場合は、別のプロセスやユーザ、または「リスト更新可」モード（許可されている場合）でレコードが既に使用されていなければ、ダブルクリックするとそのレコードを修正することができます。

2. **DISPLAY SELECTION**コマンドは自動的にテーブルを“リードオンリー”（読み込み専用）状態に切り替えます。一方、**MODIFY SELECTION**コマンドは自動的にテーブルを“リードライト”（読み書き両用）状態に切り替えます。両コマンドともに、実行後にはテーブルの状態を元に戻します。

参照

DISPLAY SELECTION、Form event、セット

APPLY TO SELECTION

APPLY TO SELECTION({テーブル ;} ステートメント)

引数	タイプ		説明
テーブル	テーブル	→	ステートメントを適用するテーブル 省略した場合、デフォルトテーブル
ステートメント	ステートメント	→	1行のステートメントまたはグローバルメソッド

説明

APPLY TO SELECTIONコマンドは、＜テーブル＞のカレントセレクションに対して＜ステートメント＞を適用します。＜ステートメント＞は、1行のステートメントまたはプロジェクトメソッドのどちらでも構いません。＜ステートメント＞が＜テーブル＞のレコードを修正した場合は、そのレコードをディスクに保存しますが、レコードを修正しない場合には保存しません。また、カレントセレクションが空の場合は、**APPLY TO SELECTION**コマンドは何も行いません。自動リレートの場合に、＜ステートメント＞はリレート先のテーブルのフィールドを含むことができます。

APPLY TO SELECTIONコマンドは、カレントセレクションの情報（例えば、合計等）を求めるため、あるいはカレントセレクションを修正するため（例えば、フィールドの頭文字を大文字に変える等）に使用します。このコマンドをトランザクション処理内で使用する場合に、トランザクション処理が取り消されると、すべての変更は無視されます。

4D Server：＜ステートメント＞に渡されるコマンドはサーバでは実行されません。セレクションの各レコードは修正のためローカルのワークステーションに送り返されます。

APPLY TO SELECTIONコマンドを実行している間、処理の進捗を表すサーモメータが表示されます。**APPLY TO SELECTION**コマンドを呼び出す前に、**MESSAGES OFF**コマンドを使用して、サーモメータの表示を取り消すことができます。サーモメータが表示されると、ユーザは処理を取り消すことができます。

例題

1. 以下の例は、[従業員]テーブル上のすべての（アルファベットの）名前を大文字に変換します。

```
APPLY TO SELECTION([従業員] ; [従業員]名前:=Uppercase([従業員]名前))
```

2. **APPLY TO SELECTION**コマンドの実行中にレコードがロックされており、そのレコードが修正された場合、レコードは保存されません。ロックされたレコードは、すべて“LockedSet”と呼ばれるセットに格納されます。**APPLY TO SELECTION**コマンドが実行された後で、LockedSetを判定し、ロックされているレコードがあるかどうかを調べます。以下のループは、すべてのレコードを修正するまで実行します。

```
Repeat      ` ロックされたレコードがなくなるまでループ
   APPLY TO SELECTION([従業員] ; [従業員]名前:=Uppercase([従業員]名前))
   USE SET("LockedSet") ` ロックされたレコードを選択
Until(Records in  table([従業員]) = 0)      ` すべてのレコードがなくなるまで
```

3. メソッドを使用する例

```
ALL RECORDS([従業員])
APPLY TO SELECTION([ 従業員];M_Cap)
```

システム変数とセット

進捗サーモメータでユーザが「中止」ボタンをクリックすると、システム変数OKに0が代入されます。それ以外の場合には、システム変数OKに1がセットされます。

参照

セット

REDUCE SELECTION

REDUCE SELECTION({テーブル ;} レコード数)

引数	タイプ		説明
テーブル	テーブル	→	セレクションを抽出するテーブル 省略した場合、デフォルトテーブル
レコード数	数値	→	カレントセレクションに保持する レコード数

説明

REDUCE SELECTIONコマンドは、＜テーブル＞から新しいレコードのセレクションを作成します。このコマンドは、＜テーブル＞のカレントセレクションの先頭から＜レコード数＞個のレコードを含むセレクションを作成します。**REDUCE SELECTION**コマンドは、カレントプロセスの＜テーブル＞のカレントセレクションに適用されます。**REDUCE SELECTION**コマンドはカレントプロセスの＜テーブル＞のカレントセレクションを変更し、新しいセレクションの先頭のレコードをカレントレコードにします。

注：REDUCE SELECTION(0)という命令が実行されると、テーブルのセレクションおよびカレントレコードはなくなります。

例題

以下の例では、最初に20カ国にわたる販売店を対象にしたコンテストの正確な統計を検索します。国ごとに、500万円以上の製品売上を記録した上位3店と、全世界で上位100店に含まれる販売店に対し、賞が送られます。ほんの数行のコードで、この複雑な処理がインデックス検索を利用して実行されます。

```
CREATE EMPTY SET([販売店];"入賞決定") ` 空のセットを作成
SCAN INDEX([販売店]製品売上;100;<) ` インデックスの終わりから調べる
CREATE SET([販売店];"上位100 店") ` 選択されたレコードをセットに納める
For($国;1;Records in table([国])) ` 各国に対して
   ` 売上高が500万円以上の販売店をこの国で検索
   QUERY([販売店];[販売店]国=[国]国名;*)
   QUERY(&;[販売店];[販売店]売上高>=5000000)
   CREATE SET([販売店];"入賞販売店") ` セットに格納
   ` 上位 100 店のグループにいれる
   INTERSECTION("入賞販売店";"上位100 店";"入賞販売店")
   USE SET("入賞販売店") ` その国の入賞候補
   ` 降順で結果をソートする
   ORDER BY([販売店];[販売店]売上高;<)
   REDUCE SELECTION([販売店];3) ` 上位3店を取りだす
```

```
        CREATE SET([販売店];"入賞販売店") ` その国の入賞者
        ` 全世界での入賞者リストに入れる
        UNION("入賞販売店";"入賞決定";"入賞決定")
    End for
    CLEAR SET("上位100 店") ` このセットはもう必要ない
    CLEAR SET("入賞販売店") ` このセットはもう必要ない
    USE SET("入賞決定") ` 入賞者
    CLEAR SET("The Winners") ` このセットはもう必要ない
    OUTPUT FORM([販売店];"賞状") ` フォームを選択
    PRINT SELECTION([販売店]) ` フォームを印刷
```

参照

ORDER BY、QUERY、SCAN INDEX、セット

SCAN INDEX

SCAN INDEX(フィールド ; レコード数 {; 方向})

引数	タイプ		説明
フィールド	フィールド	→	インデックスを読み取るインデックスフィールド
レコード数	数値	→	取出すレコード数
方向		→	<インデックスの最後から >インデックスの最初から

説明

SCAN INDEXコマンドは、テーブルから＜レコード数＞個のレコードのセレクションを作成します。＜方向＞が“<”の場合、**SCAN INDEX**コマンドはインデックスの最後から＜レコード数＞個のセレクション作成します。＜方向＞が“>”の場合、**SCAN INDEX**コマンドはインデックスの先頭から＜レコード数＞個のセレクションを作成します。このコマンドは、インデックスを用いた処理なため非常に効率が良くなります。

SCAN INDEXコマンドは、インデックスフィールドにのみ使用できます。このコマンドは、カレントプロセスのテーブルのカレントセレクションを変更しますが、カレントレコードはありません。

例題

テーブル内のレコード数より多くのレコードを指定した場合には、**SCAN INDEX**コマンドはすべてのレコードを含むセレクションを作成します。

▼以下の例は、ワースト50の顧客とベスト50の顧客に手紙を出します。

```
SCAN INDEX([顧客]合計金額 ; 50 ; <)      ` ワースト50の顧客を取り出す
ORDER BY([顧客]郵便番号 ; >)     ` 郵便番号順にソート
OUTPUT FORM([顧客] ; "奮起の手紙")
PRINT SELECTION([顧客]) ` 手紙を印刷
      ` ベスト50の顧客
SCAN INDEX([顧客]合計金額 ; 50 ; >)      ` ベスト50の顧客を取り出す
ORDER BY([顧客]郵便番号 ; >)     ` 郵便番号順にソート
OUTPUT FORM([顧客] ; "感謝の手紙")
PRINT SELECTION([顧客]) ` 手紙を印刷
```

参照

ORDER BY、QUERY、REDUCE SELECTION

ONE RECORD SELECT

ONE RECORD SELECT {(テーブル)}

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードの属するテーブル 省略した場合、デフォルトテーブル

説明

ONE RECORD SELECTコマンドは、＜テーブル＞のカレントセレクションをカレントレコードだけにします。カレントセレクションにレコードが存在しない場合には、**ONE RECORD SELECT**コマンドは何も行いません。

以前のバージョンに関する注意
このコマンドは、テーブルのセレクションが変更される際、レコードスタックから“プッシュ”して“ポップ”したレコードをセレクションに“戻す”ために使用できます。バージョン6では、SET QUERY DESTINATIONコマンドを使用すると、テーブルのセレクションやカレントレコードを変更せずに検索を実行することができます。したがって、テーブル検索のためにカレントレコードの“プッシュ”と“ポップ”を実行する必要はなくなりました。この結果、テーブルのセレクションをカレントレコードだけにしたい場合を除き、ONE RECORD SELECTコマンドはあまり使われません。

HIGHLIGHT RECORDS

HIGHLIGHT RECORDS({{テーブル; }セット名}{;* }))

引数	タイプ		説明
テーブル	テーブル	→	レコードをハイライト表示するテーブル 省略した場合、カレントフォームの テーブル
セット名	文字列	→	強調表示する一連のレコード、または 省略した場合はUserSet
*	*	→	リストの自動スクロールを無効にする

説明

このコマンドは、出力フォーム内で指定された複数のレコードをハイライト表示させる事ができます。この動作は、クリックまたはShift+クリック、Ctrl+クリック（Windows）やCommand+クリック（Mac OS）キーコンビネーションを使用し、リストモードでレコードを手動で選択する場合と同じです。「選択された」レコードはハイライト表示されますが、カレントセレクションは変更されません。

注：UserSetセットはレコードリストの再表示で更新されます。すなわち、呼び出したメソッドの全体を実行した後であり、HIGHLIGHT RECORDSを実行した直後ではありません。

引数＜テーブル＞を使用し、“ハイライト表示”されるレコードが属すテーブルを指定します。この引数は、特にカレントテーブルに属していない組み込みサブフォームのレコードをハイライト表示するために使用することができます。

■ ＜セット名＞に有効なセット名を渡すと、コマンドはカレントテーブルのそのセット内のレコードに適用されます。

■ 引数＜セット名＞を省略すると、コマンドはカレントテーブルの現在のUserSet内のレコードだけをハイライト表示します。

引数＜*＞を渡すと、ハイライト表示されたレコードが表示されていない場合に、そのリストの自動スクロール機能が無効になります。このメカニズムにより、新しい**SCROLL LINES**コマンドを使用して、独自のスクロール管理を行えるようになります。

例題

▼ **MODIFY SELECTION**コマンドによって表示される出力フォーム内で、カレントセレクションを変更することなく、ユーザが検索を実行できるようにしたいとします。これを実行するには、フォーム内に検索ボタンを置いて、押された時に下記のメソッドを実行します。

SET QUERY DESTINATION(Into Set;"UserSet")
QUERY
SET QUERY DESTINATION(Into Current Selection)
HIGHLIGHT RECORDS

ユーザがボタンをクリックすると標準の検索ダイアログボックスが表示され、検索が実行された後、カレントセレクションを変更することなく、見つかったレコードを反転表示する事ができます。

参照

GET HIGHLIGHTED RECORDS、SCROLL LINES

GET HIGHLIGHTED RECORDS

GET HIGHLIGHTED RECORDS({テーブル; }セット名)

引数	タイプ		説明
テーブル	テーブル	→	対象となるテーブル 省略した場合、カレントフォームのテーブル
セット名	文字列	→	ハイライトされたレコードを納めるセット

説明

GET HIGHLIGHTED RECORDSコマンドは、引数＜テーブル＞がリスト表示し、ハイライトされているレコードのセットを引数＜セット名＞に格納します。引数＜テーブル＞を省略すると、カレントフォームやサブフォームのテーブルが使用されます。

注意：テーブルのシステムセットUserSetには、出力フォームにおいてユーザが“ハイライト表示”したレコードがすべて納められます。テーブルおよびプロセスごとにUserSetは1つだけ存在します。詳細は「セットコマンド」の章を参照してください。

このコマンドにより、ハイライト表示されたレコードを受け取るテーブルを選定することができるため、**GET HIGHLIGHTED RECORDS**コマンドはサブフォーム内のレコードセレクションの管理を行うこともできます。この場合、サブフォームのセレクションは別のテーブルのものでも構いません。以前のバージョンの4th Dimensionでは、ユーザ定義のセレクションの管理は**UserSet**を用いてのみ行われ、従ってカレントテーブルに限定されていました。

また、**GET HIGHLIGHTED RECORDS**コマンドはフォーム以外のコンテキストでも呼び出せますが、その場合は空のセットが返されます。

＜セット名＞で指定されたセットは、ローカルプロセスやクライアントプロセス、またはインタープロセスのいずれでも構いません。

例題

▼次のメソッドは、テーブル（[CDs]）のレコードを表示するサブフォームにおいて、選択されているレコードの数を示します。

```
GET HIGHLIGHTED RECORDS([CDs];"$highlight")
ALERT(String(Records in set("$highlight"))+" レコードが選択されています。")
CLEAR SET("$highlight")
```

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

参照

HIGHLIGHT RECORDS

セット

セットを使用すれば、レコードセレクションの操作をさらに迅速かつ効率的に行えるようになります。作成したセットに対して、カレントセレクションとのリレート、保存、読み込み、消去の処理が行えるだけでなく、4th Dimensionは3つの基本機能をセットに与えています。

■ 集合交差（**INTERSECTION**）

■ 集合結合（**UNION**）

■ 集合差異（**DIFFERENCE**）

セットとカレントセレクション

セットは、レコードセレクションを簡潔に表現します。セットの概念は、カレントセレクションと密接な関連があります。

セットは、一般的に以下のような場合に使用します。

■ 並び順（ソート結果）を必要としない、セレクションの保存と復元を実行する場合

■ ユーザが画面上で作成したセレクションにアクセスする場合（UserSet）

■ セレクション間で論理演算を実行する場合

カレントセレクションは、現在選択されている各レコードを指し示す参照のリストです。リストは、メモリ上に存在します。現在選択されているレコードのみがリストに含まれます。セレクションは、実際にレコードを含んでいるわけではなく、レコードに対する参照リストを保持しているだけです。レコードに対する参照はそれぞれ、1レコードに対してメモリを4バイト使用します。テーブルに対して作業を実行する場合にも、常にカレントセレクションのレコード用いて作業を行います。また、セレクションをソートした場合でも、参照のリストがソートされるだけです。カレントセレクションは、1つのプロセス内の各テーブルに対して1つしか存在しません。

カレントセレクションと同様に、セットはレコードセレクションを表わします。セットは、各レコードを非常に簡潔に表現します。1レコードに対してメモリを1ビット（1/8バイト）使用します。コンピュータは、ビットに対する演算を非常に高速に行うため、セットを使った処理は高速に行われます。セットは、セット内にレコードを含んでいるかどうかに関係なく、テーブル上に存在するすべてのレコードに対して1ビットずつ使用します。実際、各ビットは1または0であり、この値はレコードがセット内にあるかどうかに依存します。

セットはRAMスペースの面から見ると、非常に経済的です。テーブルのレコード件数を8で割れば、そのテーブルのセットサイズをバイト数で求めることができます。例えば、10,000件のレコードを持つテーブルに対してセットを作成すれば、セットはRAMを1,250バイト（約1.2K）使用します。

各テーブルに対してたくさんのセットを持つことができます。またデータベースとは別にセットをディスクに保存することもできます。セットに属するレコードを変更するには、最初にセットをカレントセレクションとして使用し、それから1つまたは複数のレコードを修正します。インタープロセスセットの名前はデータベース内で重複してはいけません。

セットは、ソートした順番には決してなりません。レコードがセットに含まれるか含まれないかだけを示します。これに対して、命名セレクションはソートの順番を保持することができますが、多くのメモリを必要とします。命名セレクションに関する詳細は、第34章を参照してください。

セットは、セットが作成された時点のカレントレコードを“記憶”しています。

以下の表は、カレントセレクションとセットの概念を比較したものです。

比較項目	カレントセレクション	セット
テーブルに対して持つことのできる数	1	0～多数
ソート	○	×
ディスクへの保存	×	○
1レコード当たりのＲＡＭ消費量	4バイト	1/8バイト
論理演算による関係づけ	×	○
カレントレコードを含む	○	○（セットの作成時点）

セットは、それを作成したテーブルに属します。セットの演算は、同じテーブルに属するセット間でのみ実行することができます。

セットは、実在するデータとは別に存在します。これは、テーブルを更新を行した後では、セットが正確でなくなる可能性があることを意味します。セットが不正確になる可能性のある処理は数多くあります。例えば、すべての東京出身の人でセットを作成した後でその中の1つのレコードを大阪出身に修正しても、セットは更新されません。これは、セットがレコードのセレクションを表現しているに過ぎないためです。レコードを削除した後で新しいレコードを追加した場合には、セットは新しいレコードを含み、元のレコードを含まなくなります。セットは、その対応するセレクションが更新されていない場合にのみ正確なものであることが保証されます。

プロセスセットとインタープロセスセット

3種類のセットを持つことができます。

■「プロセスセット」：プロセスセットは、それを作成したプロセス内でのみアクセスすることができます。“UserSet”と“LockedSet”もプロセスセットです。プロセスセットは、プロセスが終了しだい消去されます。プロセスセットは、その名前に特別な接頭辞を必要としません。

■「インタープロセスセット」：インタープロセスセットを作成するには、その名前の前に“<>”を付けます。インタープロセスセットは、すべてのプロセスで共用されます。

```
USE SET(<>MySet)
```

■「ローカルセット／クライアントセット」：このセットは、バージョン6から新たに追加されました。ローカルセット／クライアントセットの名前の先頭には、ドル記号（$）が付きます。

セットとトランザクション

セットは、トランザクショの内部で作成することができます。トランザクションの内部で作成されたレコードのセット、およびトランザクションの外部で修正されたり、作成されたりしたレコードのセットを作成することができます。トランザクションを終了する際、特にトランザクション処理を取り消した場合、セットはレコードを正確に表示できないためトランザクション中に作成したセットを消去する必要があります。

セットの例

▼ 以下の例は、重複するレコードをテーブルから削除します。このテーブルは従業員に関する情報を含んでいます。**For**ループは、カレントレコードと1つ前のレコードの内容を比較する処理を、すべてのレコードに対して行います。名前、住所、郵便番号がすべて一致する場合には、そのレコードをセットに追加します。ループが終了したところでセットをカレントセレクションにし、カレントセレクションを削除します。

```
CREATE EMPTY SET([従業員] ; "重複")
        ` 重複レコードを格納する空のセットを定義
ALL RECORDS([従業員])
                ` すべてのレコードを対象とする
                ` 重複データを探すために
                ` 郵便番号,住所,名前でソート
ORDER BY([従業員];[従業員]郵便番号 ; >; [従業員]住所 ; > ; [従業員]名前 ; >)
$名前:=[従業員]名前 ` フィールドの内容を保管する変数を
$住所:=[従業員]住所 ` 1番目のレコードの値で初期化する
$郵便番号:=[従業員]郵便番号
NEXT RECORD([従業員])   ` 2番目のレコードから比較を開始する
For($i ; 2 ; Records in file([従業員]))
        ` 2番目のレコードから最後のレコードまで繰り返す
                ` 郵便番号、住所、名前がすべて一つ前のレコードと一致したら
                                                        重複レコード
        If(([従業員]名前=$名前) &([従業員]住所=$住所) &([従業員]郵便番号=
                                                        $郵便番号))
                ADD TO SET([従業員] ; "重複")
        ` セットにカレントレコード(重複レコード)を追加する
        Else
                $名前:=[従業員]名前 ` 郵便番号,住所,名前を以下のレコードと
                $住所:=[従業員]住所 ` 比較するために変数に格納する
                $郵便番号:=[従業員]郵便番号
        End if
        NEXT RECORD([従業員])   ` 以下のレコードに移動する
End for
USE SET("重複")
        ` セット"重複"をカレントセレクションにする
DELETE SELECTION([従業員])    ` 重複レコードをすべて削除する
CLEAR SET("重複") ` セットをメモリ上から消去する
```

メソッドの最後でセットを削除する代わりに、セット内のレコードを画面上に表示するか、あるいは印刷すれば、さらに詳細な比較を実行することができます。

システムセット：UserSet

4th Dimensionは、“UserSet”というシステムセットを持っています。UserSetはユーザによって画面上で最後に選択されたセレクションを自動的に保持します。したがって、**MODIFY SELECTION**コマンドまたは**DISPLAY SELECTION**コマンドでセレクションを表示し、ユーザはそれから必要なレコードを選択します。その選択結果でセットまたはセレクションを作成します。

UserSetは1つのプロセスに対して1つしかありません。また、各テーブルに固有のUserSetを持つことはできません。UserSetを持つことができるのは、その時点でセレクションを表示しているテーブルのみです。

▼ 以下のメソッドは、レコードの表示方法を示し、ユーザにレコードを選択してもらい、UserSetを使用してその選択されたレコードを表示します。

```
` すべてのレコードを表示し、ユーザにいくつかのレコードを選択してもらい、
                                それをUserSetを使用してカレントセレクションにする
OUTPUT FORM([従業員] ; "表示") ` 出力フォーム設定
ALL RECORDS([従業員])  ` すべての従業員を対象とする
ALERT("コマンドキーを押しながらクリックして従業員を選択してください。")
DISPLAY SELECTION([従業員])  ` 従業員を表示する
USE SET("UserSet") ` 選択された従業員を使用する。
ALERT("以下の従業員が選択されました。")
DISPLAY SELECTION([従業員])  ` 選択された従業員を表示する
```

注：UserSetを取得するには、DISPLAY SELECTIONコマンドまたはMODIFY SELECTIONコマンドのどちらかを実行する必要があります。

システムセット：LockedSet

APPLY TO SELECTIONコマンド、**ARRAY TO SELECTION**コマンドと**DELETE SELECTION**コマンドは、マルチプロセス環境で使用する場合に、“LockedSet”という名前のセットを作成します。LockedSetは、コマンド操作中にロックされたレコードを示します。

参照

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

CREATE EMPTY SET

CREATE EMPTY SET({テーブル ;} セット)

引数	タイプ		説明
テーブル	テーブル	→	空のセットを作成するテーブル、または省略した場合、デフォルトテーブル
セット	文字列	→	作成する空のセットの名前

説明

CREATE EMPTY SETコマンドは、＜テーブル＞に対して新しい空のセット＜セット＞を作成します。**ADD TO SET**コマンドを使って、このセットにレコードを追加することができます。既に同じ名前のセットが存在している場合には、そのセットを消去して新しい空のセットに置き換えます。

注：CREATE SETコマンドを使用する前には、CREATE EMPTY SETコマンドを使用する必要はありません。

例題

▼ セットの説明の例を参照ください。

参照

CLEAR SET、CREATE SET

CREATE SET

CREATE SET({テーブル ;} セット)

引数	タイプ		説明
テーブル	テーブル	→	新しいセットを作成するためのセレクションの属するテーブル、または省略した場合、デフォルトテーブル
セット	文字列	→	作成するセットの名前

説明

CREATE SETコマンドは、＜テーブル＞に対して新しいセット＜セット＞を作成し、＜セット＞にカレントセレクションの内容を代入します。そのテーブルのカレントレコードポインタは、＜セット＞に格納されます。＜セット＞に対して**USE SET**コマンドを使用すると、カレントセレクションとカレントレコードが復元されます。すべてのセットに対してソート順序は適用されません。＜セット＞には、デフォルトの順序が適用されるだけです。既に同じ名前のセットが存在している場合には、そのセットを消去し新しいセットに置き換えます。

例題

▼以下の例は、検索を行った後で新しいセットを作成し、それをディスクに保存します。

```
QUERY([従業員])    ` ユーザによる検索
CREATE SET([従業員] ; "検索セット")    ` 新しいセットを作成
SAVE SET("検索セット" ; "検索結果")    ` セットをディスクに保存
```

参照

CLEAR SET、CREATE EMPTY SET

CREATE SET FROM ARRAY

CREATE SET FROM ARRAY(テーブル ; レコード配列 ; {セット名})

引数	タイプ		説明
テーブル	テーブル	→	セットを作成するテーブル
レコード配列	倍長整数配列 または ブール配列	→	レコード番号の配列、またはブール配列 （True=レコードをセットに含める、 False=レコードをセットに含めない）
セット名	文字列	→	作成するセット名 省略された場合はUserSetに適用

説明

このコマンドは下記の方法でセット名を作成します。

■ テーブルの絶対レコード番号の配列から、またはブール配列から。この場合、配列の値は、テーブル内の各レコードをセット名に含めるか（True）含めないか（False）を表わします。

このコマンドを使って倍長整数配列をレコード配列へ渡す時、配列要素の内容はセット名に含めるレコードのレコード番号を表わしています。無効なレコード番号を含んでいる場合（例えば、まだレコードが作成されていない場合）、エラー-10503が発生します。

このコマンドを使ってブール配列をレコード配列へ渡す時、配列のＮ番目の要素は、レコード番号Ｎのレコードをセット名に含めるか（True）含めないか（False）を表わします。通常、配列のサイズはテーブル内のレコードの数と等しくなければなりません。レコードの数よりも配列が少ない場合は、配列に定義されたレコードだけを対象に判断します。

注：ブール配列の場合、このコマンドは配列要素0から配列要素N-1までを使用します。

セット名引数を省略、または空白を渡すと、コマンドはUserSetに適用されます。

参照

CREATE SELECTION FROM ARRAY、BOOLEAN ARRAY FROM SET、CREATE SELECTION FROM ARRAY

USE SET

USE SET(セット)

引数	タイプ		説明
セット	文字列	→	使用するセットの名前

説明

USE SETコマンドは、＜セット＞内のレコードをそのセットの属するテーブルのカレントセレクションにします。

セットを作成する場合、カレントレコードはそのセットによって“記憶”されています。

USE SETコマンドはセット上のカレントレコードの位置を復元し、そのレコードを新しいカレントレコードにします。**USE SET**コマンドを実行する前にこのレコードを削除すると、4th Dimensionはセットの先頭のレコードをカレントレコードに設定します。セットに対して**INTERSECTION**コマンド、**UNION**コマンド、**DIFFERENCE**コマンド、**ADD TO SET**コマンドを実行した場合には、カレントレコードは再設定されます。カレントレコードの位置を含まないセットを作成した場合にも、**USE SET**コマンドはセットの先頭のレコードをカレントレコードに設定します。

警告：セットは、そのセットが作成された時点のセレクションを表現しているという点に注意してください。セットに対応するレコードが更新されると、セットは正確なものでなくなります。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としては、セットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。

例題

▼以下の例は、**LOAD SET**コマンドを使用して、所在地が“東京都千代田区の会社”のセットを復元し、その復元したセットをカレントセレクションにします。

```
` セットをメモリ上に復元する
LOAD SET([会社] ; "東京都千代田区" ; "東京都千代田区のセット")
USE SET("東京都千代田区")        ` カレントセレクションを東京都千代田区にする
CLEAR SET("東京都千代田区")      ` セットをメモリから消去
```

参照

CLEAR SET、LOAD SET

ADD TO SET

ADD TO SET({テーブル ;} セット)

引数	タイプ		説明
テーブル	テーブル	→	セットに追加するレコードの属するテーブル 省略した場合、デフォルトテーブル
セット	文字列	→	カレントレコードを追加するセットの名前

説明

ADD TO SETコマンドは、＜セット＞に＜テーブル＞のカレントレコードを追加します。ここで使用するセットは、既に作成されているものでなければなりません。存在しないセット名を指定した場合は、エラーになります。＜テーブル＞にカレントレコードが存在しない場合には、**ADD TO SET**コマンドは何も行いません。

参照

REMOVE FROM SET

REMOVE FROM SET

REMOVE FROM SET({テーブル ;} セット)

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードのテーブル 省略した場合は、デフォルトテーブル
セット	文字列	→	カレントレコードを削除するためのセットの名前

説明

REMOVE FROM SETコマンドは、引数＜セット＞から＜テーブル＞のカレントレコードを削除します。セットは常に存在していなければならず、存在していない場合には、エラーが発生します。引数＜テーブル＞に対するカレントレコードがない場合には、**REMOVE FROM SET**コマンドは何も行いません。

参照

ADD TO SET

CLEAR SET

CLEAR SET(セット)

引数	タイプ	説明
セット	文字列	メモリから消去するセットの名前

説明

CLEAR SETコマンドは、メモリから＜セット＞を消去し、＜セット＞の占有していたメモリを解放します。**CLEAR SET**コマンドはテーブル、セレクション、レコードには影響を与えません。セットは、消去する前に**SAVE SET**コマンドを使用して保存することができます。セットはメモリを使用するため、必要のないセットは消去してください。

USE SETコマンドの例を参照してください。

参照

CREATE EMPTY SET、CREATE SET、LOAD SET

Is in set

Is in set(セット) → ブール

引数	タイプ		説明
セット	文字列	→	調べるセット
戻り値	ブール	←	セットの属するテーブルのカレントレコードがセットに含まれる場合“True（真）”、そうでない場合は“False（偽）”

説明

Is in set関数は、＜セット＞の属するテーブルのカレントレコードが＜セット＞に含まれているかどうかを調べます。**Is in set**関数は、カレントレコードが＜セット＞に含まれていれば“True（真）”を返し、含まれていなければ“False（偽）”を返します。

例題

▼以下の例は、ボタンのオブジェクトメソッドです。これは、現在表示されているレコードが“お得意様”のセットに含まれているかどうかを調べます。

```
If(Is in set("お得意様"))      ` お得意様かどうか調べる
      ALERT("このお客様は、お得意様です。")
Else
      ALERT("このお客様は、お得意様ではありません。")
End if
```

参照

ADD TO SET、REMOVE FROM SET

Records in set

Records in set(セット) → 数値

引数	タイプ		説明
セット	文字列	→	調べるセット
戻り値	数値	←	レコード数

説明

Records in set関数は、＜セット＞に含まれるレコードの数を返します。＜セット＞が存在しない場合、または＜セット＞にレコードがない場合には0を返します。

例題

▼以下の例は、全顧客の中に占めるお得意様の割合をアラートボックスに表示します。

```
` お得意様の割合を計算
$パーセント:=(Records in set("お得意様") / Records in table([顧客])) * 100
      ` お得意様の割合をパーセント表示
ALERT(String($パーセント ; "##0%") + "がお得意様です。")
```

参照

Records in selection、Records in table

SAVE SET

SAVE SET(セット ; ドキュメント)

引数	タイプ		説明
セット	文字列	→	保存するセットの名前
ドキュメント	文字列	→	セットを保存するディスクファイルの名前

説明

SAVE SETコマンドは、＜ドキュメント＞で指定した名前のドキュメントファイルとして＜セット＞をディスクに保存します。

＜ドキュメント＞は、セットと同じ名前である必要はありません。＜ドキュメント＞に対して空の文字列（ヌル）を指定すると、「ファイル作成」ダイアログボックスが表示されます。ユーザは、ここでファイルの名前を入力することができます。保存したセットは、**LOAD SET**コマンドを使用して復元することができます。

「ファイル作成」ダイアログボックスで「キャンセル」ボタンをクリックした場合や保存処理中にエラーが発生した場合には、システム変数OKに0が代入されます。それ以外の場合には1がセットされます。

SAVE SETコマンドは、時間のかかる検索の結果をディスクに保存するためによく使用されます。

警告：セットは、そのセットが作成された時点のセレクションを表現しているということに注意してください。セットに対応するレコードが更新されると、セットは正確なものではなくなります。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としては、セットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。また、セットはフィールド値を保存しないということに注意してください。

例題

▼以下の例は、ユーザがセットを含んだファイル名を入力するための「ファイル作成」ダイアログボックスを表示します。

```
SAVE SET("検索セット" ; "")
```

▼ 以下の例は、**SAVE SET**コマンドを使用してシーケンシャルな検索の結果を保存します。検索条件は“東京の千代田区にある会社すべて”です。結果のセットは、ユーザが指定したドキュメントファイルに保存されます。

```
QUERY([会社] ; [会社]市区町村="千代田区" ; *)    ` 第1の検索...
QUERY([会社] ;&[会社]都道府県="東京都")` 第2の検索
CREATE SET([会社] ; "東京都千代田区")` カレントセレクションからセットを作成
SAVE SET("東京都千代田区" ; "") ` ユーザの名付けたファイルに保存
CLEAR SET("東京都千代田区") ` セットをメモリから消去
```

システム変数とシステムセット

「ファイル作成」ダイアログボックスで「キャンセル」がクリックされたり、保存処理中にエラーが発生した場合には、システム変数OKに0が代入されます。それ以外の場合には1がセットされます。

参照

LOAD SET

LOAD SET

LOAD SET({テーブル ;} セット ; ドキュメント)

引数	タイプ		説明
テーブル	テーブル	→	セットの属するテーブル 省略した場合、デフォルトテーブル
セット	文字列	→	メモリに復元するセット
ドキュメント	文字列	→	セットを含むドキュメントファイル

説明

LOAD SETコマンドは、**SAVE SET**コマンドでディスクに保存した＜ドキュメント＞からセットをメモリに復元します。

＜ドキュメント＞に格納されたセットは、＜テーブル＞から作成されてなければいけません。メモリ内で作成されたセットは既に同じセットが存在すると上書きされます。

引数＜ドキュメント＞は、セットを保存したドキュメントファイルの名前です。ドキュメントはセットと同じ名前である必要はありません。＜ドキュメント＞に対して空の文字列（ヌルストリング）を指定すると、「ファイルを開く」ダイアログボックスが表示されます。ユーザは、ここで復元するセットを選択することができます。

セットは、そのセットが作成された時点のセレクションを表現しているという点に注意してください。セットに対応するレコードが更新されると、セットは正確なものでなくなります。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としては、セットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。

例題

▼ 以下の例は、**LOAD SET**コマンドを使用して“東京都千代田区”のセットを復元します。

```
` セットをメモリに復元
LOAD SET([会社] ; "東京都千代田区" ; "東京都千代田保存")
USE SET("東京都千代田区") ` セット"東京都千代田"をカレントセレクションにする
CLEAR SET("東京都千代田区")    ` セットをメモリから消去
```

システム変数とシステムセット

「ファイルを開く」ダイアログボックスで「キャンセル」ボタンをクリックした場合やロード中にエラーが発生した場合はシステム変数OKに0が代入されます。それ以外の場合には1が代入されます。

参照

SAVE SET

DIFFERENCE

DIFFERENCE(セット1 ; セット2 ; 結果セット)

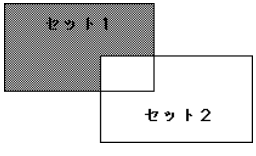
引数	タイプ		説明
セット1	文字列	→	オリジナルセット
セット2	文字列	→	排除するセット
結果セット	文字列	→	結果を格納するセット

説明

DIFFERENCEコマンドは、＜セット1＞と＜セット2＞を比較（集合差異演算）し、＜セット2＞と一致しない＜セット1＞のレコードだけを＜結果セット＞に格納します。つまり、＜セット1＞にだけ存在し、＜セット2＞には存在しないレコードのみを＜結果セット＞に格納します。以下の表に、集合差異演算の処理で考えられるすべての組み合わせを示します。

セット1	セット2	結果セット
○	×	○
○	○	×
×	○	×
×	×	×

以下の図に、集合差異演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。



＜結果セット＞は、**DIFFERENCE**コマンドで作成されます。＜結果セット＞と同じ名前のセット（セット1、セット2も含めて）が既に存在していた場合には、＜結果セット＞に置き換えられます。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞と＜セット2＞と同じテーブルに属します。

4D Server：クライアント／サーバ環境において、インタープロセスセットおよびプロセスセットはサーバマシン上に保持されます。一方、ローカルセットはクライアントマシン上で維持されます。DIFFERENCEコマンドでは3つのセットが同じマシン上に存在する必要があります。したがって、すべてのセットがローカルに存在するか、またはいずれのセットもローカルに存在しないかのいずれかでなくてはなりません。詳細は、『4D Serverリファレンス』マニュアルの4D Serverとセットに関する説明を参照してください。

例題

▼以下の例は、表示したセレクションからユーザが選択したレコードを排除します。このレコードリストは、以下のステートメントで画面に表示されます。

```
DISPLAY SELECTION([顧客])      `顧客のリストを表示
```

レコードリストの下部には、オブジェクトメソッド付きのボタンがあります。このオブジェクトメソッドはユーザが選択したレコード（UserSet）を排除し、新しいセットを表示します。

```
CREATE SET([顧客] ; "カレント") `カレントセレクションからセットを作成
`“UserSet”と一致するレコードを排除
DIFFERENCE("カレント" ; "UserSet" ; "カレント")
USE SET("カレント")        `新しいセットを使用
CLEAR SET("カレント")    `セットをメモリから消去
```

参照

INTERSECTION、UNION

INTERSECTION

INTERSECTION(セット1 ; セット2 ; 結果セット)

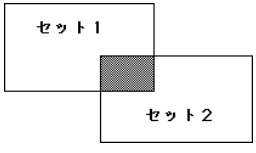
引数	タイプ		説明
セット1	文字列	→	第1のセット
セット2	文字列	→	第2のセット
結果セット	文字列	→	結果を格納するセット

説明

INTERSECTIONコマンドは、＜セット1＞と＜セット2＞を比較（集合交差演算）し、＜セット1＞と＜セット2＞の両方に存在するレコードだけを選択し、＜結果セット＞に格納します。下表に、集合交差演算の処理で考えられるすべての組み合わせを示します。

セット1	セット2	結果セット
○	×	×
○	○	○
×	○	×
×	×	×

以下の図に、集合交差演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。



＜結果セット＞は、**INTERSECTION**コマンドで作成されます。＜結果セット＞と同じ名前のセット（セット1、セット2も含めて）が既に存在する場合には、＜結果セット＞に置き換わります。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞と＜セット2＞と同じテーブルに属します。

4D Server：クライアント／サーバ環境において、インタープロセスセットおよびプロセスセットはサーバマシン上に保持されます。一方、ローカルセットはクライアントマシン上で維持されます。INTERSECTIONコマンドでは3つのセットが同じマシン上に存在する必要があります。したがって、すべてのセットがローカルに存在するか、またはいずれのセットもローカルに存在しないかのいずれかでなくてはなりません。詳細は、『4D Serverリファレンス』マニュアルの4D Serverとセットに関する説明を参照してください。

例題

▼以下の例は、“佐藤”と“吉野”という2人の販売担当者が重複して担当する顧客を検索します。販売担当者は、各自の顧客を表すセット“佐藤”と“吉野”を持っています。

```
INTERSECTION("佐藤" ; "吉野" ; "重複")  ` 重複して担当する顧客のセットを作成
USE SET("重複")      ` セットを使用
CLEAR SET("重複") ` セットをメモリから消去
DISPLAY SELECTION([顧客])      `2人が重複して担当する顧客のリストを表示
```

参照

DIFFERENCE、UNION

UNION

UNION(セット1 ; セット2 ; 結果セット)

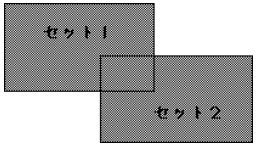
引数	タイプ		説明
セット1	文字列	→	第1のセット
セット2	文字列	→	第2のセット
結果セット	文字列	→	結果を格納するセット

説明

UNIONコマンドは、＜セット1＞と＜セット2＞を比較（集合結合演算）し、＜セット1＞と＜セット2＞に含まれるすべてのレコードを＜結果セット＞に格納します。下表に、集合結合演算の処理で考えられるすべての組み合わせを示します。

セット1	セット2	結果セット
○	×	○
○	○	○
×	○	○
×	×	×

以下の図に、集合結合演算の処理結果を図で示します。塗りつぶした箇所が結果セットの部分です。



＜結果セット＞は、**UNION**コマンドで作成されます。＜結果セット＞と同じ名前のセット（セット1、セット2も含めて）がすでに存在する場合には、＜結果セット＞に置き換えられます。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞、＜セット2＞と同じテーブルに属します。＜結果セット＞のカレントレコードは、＜セット1＞からのカレントレコードです。

4D Server：クライアント／サーバ環境において、インタープロセスセットおよびプロセスセットはサーバマシン上に保持されます。一方、ローカルセットはクライアントマシン上で維持されます。UNIONコマンドでは3つのセットが同じマシン上に存在する必要があります。したがって、すべてのセットがローカルに存在するか、またはいずれのセットもローカルに存在しないかのいずれかでなくてはなりません。詳細は、『4D Serverリファレンス』マニュアルの4D Serverとセットに関する説明を参照してください。

例題

▼ 以下の例は、“お得意様”のセットにレコードを追加します。すべてのレコードを表示した後で、“お得意様”のセットをディスクから復元します。そして、ユーザが選択したすべてのレコード（UserSet）を“お得意様”のセットに追加します。最後に新しい“お得意様”のセットをディスクに保存します。

```
ALL RECORDS([顧客])      ` すべての顧客を対象とする
DISPLAY SELECTION([顧客])      ` すべての顧客のリストを表示
LOAD SET("お得意様" ; "お得意様格納")  ` お得意様のセットを復元
UNION("お得意様" ; "UserSet" ; "お得意様")      ` 新しい顧客をセットに追加
SAVE SET("お得意様" ; "お得意様格納")  ` お得意様のセットを保存
```

参照

DIFFERENCE、INTERSECTION

COPY SET

COPY SET(コピー元 ; コピー先)

引数	タイプ		説明
コピー元	文字列	→	コピー元のソースセットの名前
コピー先	文字列	→	コピー先のセットの名前

説明

COPY SETコマンドは、引数<コピー先>セットの中に<コピー元>セットの内容をコピーします。

両方のセットとも、プロセスセット、インタープロセスセット、またはローカルセットにすることができます。

4D Server：ローカルセットはクライアントマシン上で管理されますが、クライアント／サーバにおいて、インタープロセスセットとプロセスセットはサーバマシン上で管理されます。COPY SETコマンドは2つのマシン間でセットをコピーすることができます。詳細は、『4D Serverリファレンス』マニュアルの4D Serverとセットに関する説明を参照してください。

例題

1. 以下の例は、クライアント／サーバにおいて、クライアントマシン上で管理されるローカルセット“$SetA”をサーバマシン上で管理されるプロセスセット“SetB”にコピーします。

```
COPY SET("$SetA" ; "SetB")
```

2. 以下の例は、クライアント／サーバにおいて、サーバマシン上で管理されるプロセスセット“SetA”をクライアント／サーバ上で管理されるローカルセット“$SetB”にコピーします。

```
COPY SET("SetA" ; "$SetB")
```

参照

セット

String

String(表現式 {; フォーマット}) → 文字列

引数	タイプ		説明
表現式		→	文字列に変換する表現式 実数、整数、倍長整数、日付、時間を指定可
フォーマット	文字列 ¦ 数値	→	変換に使用するフォーマット
戻り値	文字列	←	表現式の文字列形式

説明

String関数は、＜表現式＞に渡した数値、日付、時間を文字列に変換します。

オプションの＜フォーマット＞を指定しない場合、適当なデフォルトの形式で文字列が返されます。＜フォーマット＞を指定すると、結果の文字列は指定した形式になります。

数値式

＜表現式＞に数値（実数、整数、倍長整数）を渡した場合、オプションで文字列フォーマットを渡すことができます。次に例を示します。

例	結果
String(2^15) `デフォルトフォーマット使用	32768
String(2^15;"###,##0 Inhabitants")	32,768 Inhabitants
String(1/3;"##0.00000")	0.33333
String(1/3) ` デフォルトフォーマット使用	0.3333333333333333
String(Arctan(1)*4)	3.1415926535897931
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

フォーマットは、フォームの数値フォーマットと同じ方法で指定します。詳細に関しては、『4th Dimensionデザインリファレンス』を参照してください。カスタムフォーマットの書式名を＜フォーマット＞に渡すことができます。カスタムフォーマットの名前は“i”で始めなければなりません。

日付式

＜表現式＞が日付の場合、デフォルトフォーマット（YY.MM.DD）を使用して文字列が返されます。＜フォーマット＞を指定した場合、以下の表に示すフォーマットに従って変換します（フォーマット番号1は、コントロールパネルの「日付＆時刻」の「日付の書式」で年を4桁表示にするという指定にすると、年が4桁で表示されます）。

フォーマット	種類	例
1	Y.M.D	98.7.5
2	YYYY年M月D日(X)	1998年7月5日(日)
3	YYYY年M月D日X曜日	1998年7月5日日曜日
4	YY.MM.DD	98.07.05
5	Month Date, Year	July 5, 1998
6	Month Date, Year（短表記）	Jul 5, 1998
7	YYYY.MM.DD（強制的）	1998.07.05

4Dには以下のような定義済み定数が用意されています。

定数	タイプ	値
Short	倍長整数	1
Abbreviated	倍長整数	2
Long	倍長整数	3
MM DD YYYY	倍長整数	4
Month Day Year	倍長整数	5
Abbr Month Date	倍長整数	6
MM DD YYYY Forced	倍長整数	7

以下の例では、現在の日付が96/12/29であるものとします。

```
` $vsResult の結果は"96.12.29"です。
$vsResult:=String(Current date)
` $vsResult の結果は"December 29, 1996"です。
$vsResult:=String(Current date;Month Day Year)
```

時間式

＜表現式＞が時間の場合、デフォルトフォーマット（HH:MM:SS）を使用して文字列が返されます。＜フォーマット＞を指定した場合、以下の表に示すフォーマットに従って変換します。

フォーマット	種類	例
1	HH:MM:SS	01:02:03
2	HH:MM	01:02
3	H 時 M 分 S 秒	1時2分3秒
4	H 時 M 分	1時2分
5	H:MM AM / PM	1:02 AM

4Dには以下のような定義済み定数が用意されています。

定数	タイプ	値
HH MM SS	倍長整数	1
HH MM	倍長整数	2
Hour Min Sec	倍長整数	3
Hour Min	倍長整数	4
HH MM AM PM	倍長整数	5

以下の例は、現在時刻が5:30 PM45秒であるものとします。

```
$vsResult:=String(Current time) ` $vsResult の結果は"17:30:45"です。
$vsResult:=String(Current time;Hour Min Sec) ` $vsResult の結果は
                                                "17時30分45秒"です。
```

参照

Date、Num

Num

Num(表現式) → 数値

引数	タイプ		説明
表現式	文字列¦ブール	→	数値に変換する文字列、または0か1を返すブール値
戻り値	数値	←	文字列またはブール値に対する数値

説明

Num関数は＜表現式＞に渡した文字列、またはブール値を数値に変換します。

文字列式

＜文字列＞が数字をまったく含まない場合は、**Num**関数はゼロ（0）を返します。また、英文字と数字が混在する場合、**Num**関数は文字を無視します。したがって、**Num**関数は文字列“a1b2c3”を数値123に変換します。

注：文字列の最初の32バイトだけが評価されます。

ピリオド（.）、ハイフン（-）、e（またはE）の3種類の文字は、**Num**関数に対して特別な意味を持ち、それぞれ数値表現のフォーマット文字とみなされます。

■ ピリオドは、小数点を意味します。数字の文字列の間に置かなければなりません。

■ ハイフンは、数値や指数が負であることを意味します。数字の前、あるいは指数eの後に配置します。ハイフンが数字の間にあると、文字列は無効になり、**Num**関数はゼロ（0）を返します。例えば、**Num**("123-456")は0になります。また、**Num**(-9)は-9になります。

■ eまたはEがあると、その右側の数字をすべて指数として解釈します。eは、数字の文字列の間に置かなければなりません。**Num**("123e-2")は1.23になります。

ブール式

ブール式を渡した場合、式が“True（真）”の場合は1を返し、“False（偽）”の場合は0を返します。

例題

1. **Num**の使用例を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される値についての説明です。

```
v結果:=Num("ABCD")        ` v結果に0を代入
v結果:=Num("A1B2C3")      ` v結果に123を代入
v結果:=Num("123")  ` v結果に123を代入
v結果:=Num("123.4")       ` v結果に123.4を代入
v結果:=Num("-123") ` v結果に-123を代入
v結果:=Num("-123e2")      ` v結果に-12300を代入
```

2. 以下の例は、顧客の貸越について0または1を返すものです。文字列を反復する演算子、アスタリスク（*）を使用して顧客に対する評価を[顧客]危険率フィールドに納めます。このフィールドは、顧客の貸越残によって“低い”値または“高い”値が割り当てられます。

```
` 顧客の貸越残が1000以下なら危険率は低い
` 顧客の貸越残が1000以上なら危険率は高い
[顧客] 危険率:=("低い" * Num([顧客]貸越残<1000))+
                                ("高い" * Num([顧客]貸越残>=1000))
```

参照

論理演算子、String、文字列演算子

Position

Position(検索文字列 ; 文字列) → 数値

引数	タイプ		説明
検索文字列	文字列	→	探す文字列
文字列	文字列	→	元の文字列
戻り値	数値	←	最初の発見位置

説明

Position関数は、＜文字列＞の中で＜検索文字列＞が最初に現われる位置（バイト）を返します。

＜検索文字列＞が見つからない場合は、**Position**関数はゼロ（0）を返します。

＜検索文字列＞が見つかると、文字列の中に検索文字列が最初に表示された文字位置を返します。

空の文字列に対して空の検索文字列を指定すると、**Position**関数はゼロ（0）を返します。

警告：Position関数に対して“@”ワイルドカード記号を使用することはできません。例えば、＜検索文字列＞に“abc@”を渡すと、このコマンドは“abc”で始まる文字ではなく、単なる文字として“abc@”を検索します。

例題

1. **Position**関数の使用例を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される値についての説明です。

```
v結果:=Position("ll" ; "Willow") ` v結果に3を代入
v結果:=Position(変数1 ; 変数2)` v結果に変数2の中で変数1が最初に表示される位置を代入
```

2. **Substring**の例題を参照してください。

参照

比較演算子、Substring

Substring

Substring(文字列 ; 先頭文字位置 {; 文字数}) → 文字列

引数	タイプ		説明
文字列	文字列	→	この文字列から部分文字列（substring）を得る
先頭文字位置	数値	→	取り出す文字列の最初の文字の位置（バイト）
文字数	数値	→	取り出す文字列の長さ（バイト）
戻り値	文字列	←	文字列の部分文字列

説明

Substring関数は、＜先頭文字位置＞と＜文字数＞で指定した部分文字列(Substring)を＜文字列＞から取り出して返します。

＜先頭文字位置＞には文字列の中で取り出す文字列の最初の文字の位置を指定し、＜文字数＞には取り出す文字列の長さを指定します。

＜先頭文字位置＞と＜文字数＞の和が、文字列自身の文字数よりも大きい場合や＜文字数＞を指定しない場合には、先頭文字位置以降の文字列をすべて取り出します。＜先頭文字位置＞が文字列の長さより大きいと、**Substring**関数は空の文字列（""）を返します。

例題

1. **Substring**関数の使用例を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される内容についての説明です。

```
v結果:=Substring("62.04.08" ; 4 ; 2)          ` v結果に"04"を代入
v結果:=Substring("Emergency" ; 1 ; 6)   ` v結果に"Emerge"を代入
v結果:=Substring(var ; 2)                  `v結果に先頭の文字以外のすべての文字を代入
```

2. 以下のプロジェクトメソッドは、テキスト（最初の引数で指定）中に見つかった段落を文字列またはテキスト配列（2番目の引数としてポインタを渡す）に追加します。

```
` EXTRACT PARAGRAPHS
` EXTRACT PARAGRAPHS(テキスト; ポインタ)
` EXTRACT PARAGRAPHS(解析するテキスト ; -> 段落の配列)
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="")
```

参照

Position

Length

Length(文字列) → 数値

引数	タイプ		説明
文字列	文字列	→	長さを知りたい文字列
戻り値	数値	←	文字列の長さ

説明

Length関数は、＜文字列＞の長さをバイト数で返します。

注：「vtAnyText=""」というテストは、「If(Length(vtAnyText)=0)」と同じです。

例題

▼ **Length**関数の使用例を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される値についての説明です。

```
v結果:=Length("Japan")     ` v結果に5を代入
v結果:=Length("Citizen")   ` v結果に7を代入
```

Ascii

Ascii(文字) → 数値

引数	タイプ		説明
文字	文字列	→	ASCIIコードに変換する文字
戻り値	数値	←	文字のASCIIコード

説明

ASCII関数は、＜文字＞を**ASCII**コードに変換して返します。

＜文字＞が1文字より多い場合は、**Ascii**関数は最初の文字だけをコードに変換します。

Ascii関数の逆の変換を実行する関数が**Char**関数です。ASCIIコードで示す文字を返します。

重要：4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

例題

1. 通常、大文字と小文字は同じものとして扱われますが、Ascii関数を使用すれば大文字と小文字を区別することができます。以下のような場合の結果は“True（真）”になります。

```
("A" = "a")
```

一方、以下のような場合の結果は“False（偽）”になります。

```
(Ascii("A") = Ascii("a"))
```

2. 以下の例は、文字列“ABC”の最初の文字AのASCIIコードを返します。

```
GetAsc:=Ascii("ABC")        ` GetAscは65を得る(AのASCIIコード)
3.      以下の例は、キャリッジリターンとタブを検査します。
For($vlChar;1;Length(vtText))
    Case of
        ¥(vtText[[$vlChar]]=Char(Carriage return))
        ` 処理を実行
        ¥(vtText[[$vlChar]]=Char(Tab))
        ` 処理を実行
        ¥(...)
        ` ...
    End case
End for
```

サイズの大きなテキストに対して何度も実行する場合、以下のように記述した後コンパイルすると、この検査は高速に処理されます。

```
For($vlChar;1;Length(vtText))
    $vlAscii:=Ascii(vtText[[$vlChar]])
    Case of
        ¥($vlAscii=Carriage return)
        ` 処理を実行
        ¥($vlAscii=Tab)
        ` 処理を実行
        ¥(...)
        ` ...
    End case
End for
```

2番目の例題が高速に処理される理由は2つあります。つまり、ループでは1文字だけが参照され、キャリッジリターンやタブを検査する際に、文字列の比較ではなく倍長整数による比較が行われています。“CR”や“TAB”等の一般的なASCIIコードを使用して作業する場合には、この手法を利用してください。

参照

ASCIIコード、Char、文字列参照記号

Char

Char(ASCIIコード) → 文字列(1バイト)

引数	タイプ		説明
ASCIIコード	数値	→	0から255までのASCIIコード
戻り値	文字列	←	ASCIIコードで示された文字

説明

Char関数は、＜ASCIIコード＞で指定したASCIIコードをもつ文字を返します。

Tip：メソッド作成時、通常Char関数は通常キーボードから入力できない文字や、メソッドエディタでは編集コマンドとして解釈される文字を指定するために使用します。以下の表に、こうした文字の一部を示します。

コード	ASCIIコード	キー
3		enter（テンキー上の）
9	TAB	tab
13	キャリッジリターン（CR）	enter（キーボード上の）

重要：4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

例題

▼ 以下の例は、変数にキャリッジリターンを代入するために**Char**関数を使用します。そして警告を表示します。

```
CR:=Char(13)` 変数CRにキャリッジリターンを代入
ALERT("従業員："+ String(Records in file([従業員]))+CR+"OKをクリックして
                                                    ください。")
```

参照

Ascii、ASCIIコード、文字列参照記号

文字参照記号

文字列[[文字位置]] → 文字列(1バイト)

引数	タイプ		説明
文字列	文字列	→	この文字列から文字を取り出す
文字位置	数値	→	文字を取り出す位置（バイト）

説明

文字参照記号（[[...]]）は、＜文字列＞内の1文字（1バイト）を参照するために使用します。文字の位置を示す＜文字位置＞は、文字参照記号で囲んで指定します。この構文を使用し、テキスト変数、文字列変数、フィールドの文字を1文字ずつ参照することができます。

＜文字位置＞で指定した場所の文字を返します。

文字列の長さよりも大きな数値を指定した場合の結果は保証しません。

文字参照記号が代入演算子（:=）の左側にある場合は、文字列内の指定した位置に文字を代入します。以下の例は、vsNameが空の文字列ではない場合に、vsNameの最初の文字を大文字にします。

```
If(vsName#"")
    vsName[[1]]:=Uppercase(vsName[[1]])
End if
```

また、文字列参照記号が式に使用された場合、文字（参照される）は1バイトの文字列として返されます。

```
` 以下の例は vtTextの最後の桁が"@"記号であるかどうかを判定します。
If(vtText # "")
    If(Ascii(Substring(vtText;Length(vtText);1))=At Sign)
        ` ...
    End if
End if
` 文字列参照構文を使用し、以下のコードを作成
If(vtText # "")
    If(Ascii(vtText[[Length(vtText)]])=At Sign)
    ` ...
    End if
End if
```

無効な文字列参照に関する注意

文字列参照記号を使用する際、配列の既存の要素を使用するのと同じ要領で、文字列内の既存の文字を使用しなければなりません。例えば、文字列変数の20桁目の文字を使用する場合、この変数は必ずすくなくとも20桁の長さがなくてはなりません。

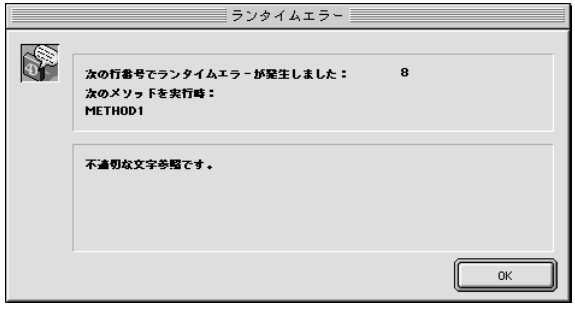
■ 長さが足りない場合、インタープリタモードでは構文エラーが発生します。

■ 長さが足りない場合、コンパイルモード（オプション指定なし）では、例えば文字列やテキストの終わりを越えた位置に文字を書き込んだ場合にクラッシュするおそれがあります。

■ 長さが足りない場合、コンパイルモードで範囲チェックオプションをオンにしてある場合に、例えば以下のようなコードを実行するとします。

```
vsAnyText:=""
vsAnyText[[1]]:="A"  ` 決してこれをしてははいけません
```

すると、ランタイムエラーが発生します。



例題

▼ 文字参照記号の使用例を次に示します。

```
v結果:="abcd"[[3]]    ` v結果に"c"を代入
v結果:=Last Name[[1]]      `v結果にLast Nameの先頭の文字を代入
v結果:=City[[$i]]      ` v結果にCityの$i番目の文字を代入
```

以下のサブルーチンは、文字列内の各単語の先頭の文字を大文字に変換する関数です。

```
$0:=$1` 返す文字列をコピー
$0[[1]]:=Uppercase($0[[1]]) ` 先頭は必ず大文字で始まる
For($i; 1 ; Length($0) - 1)   `先頭文字以外のすべての文字についてループ
      If(Position($0[[$i]] ; " !&()-{}:;<>?/,.=+*") > 0) ` 文字でなければ˘
            $0[[$i+1]]:=Uppercase($0[[$i+1]])   ` 次に表示された文字を大文字にする
      End if
End for
```

参照

Ascii、Char

Uppercase

Uppercase(文字列) → 文字列

引数	タイプ		説明
文字列	文字列	→	大文字に変換する文字列
戻り値	文字列	←	大文字で示された文字

説明

Uppercase関数は、アルファベット文字列をすべて大文字に変換して返します。

例題

Lowercase関数の例を参照してください。

参照

Lowercase

Lowercase

Lowercase(文字列) → 文字列

引数	タイプ		説明
文字列	文字列	→	小文字に変換する文字列
戻り値	文字列	←	小文字で示された文字

説明

Lowercase関数は、アルファベット文字列をすべて小文字に変換して返します。

例題

▼ 以下の例は、“Caps”という名前の関数です。与えられた文字列の最初の文字を大文字にして返すものです。例えば、「Name:=Caps("john")」の結果は、「Name="John"」となります。この例は、文字参照記号（[[...]]）を使用しています。

```
` 「Caps」プロジェクトメソッド
` Caps( 文字列) -> 文字列
` Caps( テキストまたは文字列 ) -> 大文字に変換したテキスト
$0:=Lowercase($1)
If(Length($0)>0)
        $0[[1]]:=Uppercase($0[[1]])
End if
```

参照

Uppercase

Change string

Change string(元の文字列 ; 修正文字列 ; 修正位置) → 文字列

引数	タイプ		説明
元の文字列	文字列	→	元の文字列
修正文字列	文字列	→	新しい文字列
修正位置	数値	→	修正開始位置
戻り値	文字列	←	結果文字列

説明

Change string関数は、＜元の文字列＞の中の文字グループを修正したものを返します。＜修正位置＞で指定された位置から、＜修正文字列＞で＜元の文字列＞を上書きします。

＜修正文字列＞が空の文字列（""）の場合は、**Change string**関数は＜元の文字列＞を加工しないで返します。**Change string**関数は常に＜元の文字列＞と同じ長さの文字列を返します。＜修正位置＞が＜元の文字列＞の長さ以下の場合や＜元の文字列＞の長さ以上の場合には、**Change string**関数は＜元の文字列＞を返します。

Change string関数は、文字を挿入しないで上書きするという点が**Insert string**関数とは異なります。

例題

▼ **Change string**関数の使用例を次に示します。結果を変数“v結果”に代入します。

```
v結果:=Change string("Acme" ; "CME" ; 2)          ` v結果に"ACME"を代入
v結果:=Change string("November" ; "Dec" ; 1)      ` v結果に"December"を代入
```

参照

Delete string、Insert string、Replace string

Insert string

Insert string(元の文字列 ; 挿入文字列 ; 挿入位置) → 文字列

引数	タイプ		説明
元の文字列	文字列	→	元の文字列
挿入文字列	文字列	→	新しい文字列
挿入位置	数値	→	挿入開始位置
戻り値	文字列	←	結果文字列

説明

Insert string関数は、＜元の文字列＞に文字列を挿入したものを返します。**Insert string**関数は、＜挿入位置＞で指定された位置の前に、＜挿入文字列＞を挿入します。

＜挿入文字列＞が空の文字列（""）であれば、**Insert string**関数は＜元の文字列＞を加工しないで返します。

＜挿入位置＞が、＜元の文字列＞の長さよりも大きい場合は、＜挿入文字列＞を＜元の文字列＞の後ろに追加します。＜挿入位置＞が1よりも小さい場合には、＜挿入文字列＞を＜元の文字列＞の前に挿入します。

Insert string関数は、文字を上書きしないで挿入するという点が**Change string**関数とは異なります。

例題

▼ **Insert string**関数の使用例を次に示します。結果を変数“v結果”に代入します。

```
v結果:=Insert string("The tree" ; "green" ; 4)      ` v結果に"The green tree"を代入
v結果:=Insert string("Shut" ; "o" ; 3)              ` v結果に"Shout"を代入
v結果:=Insert string("Indention" ; "ta" ; 6) ` v結果に"Indentation"を代入
```

参照

Change string、Delete string、Replace string

Delete string

Delete string(元の文字列 ; 削除位置 ; 文字数) → 文字列

引数	タイプ		説明
元の文字列	文字列	→	元の文字列
削除位置	数値	→	削除を開始する文字の位置（バイト）
文字数	数値	→	削除する文字数（バイト）
戻り値	文字列	←	結果文字列

説明

Delete string関数は、＜削除位置＞から＜文字数＞分の文字を＜元の文字列＞から削除した文字列を返します。

Delete string関数は、以下のような場合に＜元の文字列＞と同じ文字列を返します。

■ ＜元の文字列＞が空の文字列の場合

■ ＜削除位置＞がゼロ（0）か、ゼロより小さい場合

■ ＜削除位置＞が＜元の文字列＞の長さより大きい場合

■ ＜文字数＞がゼロ（0）の場合

＜削除位置＞が0より小さい場合、文字列の始めから文字が削除されます。

＜削除位置＞と＜文字数＞の和が＜元の文字列＞の長さと等しいかまたは大きい場合は、＜削除位置＞から文字列の最後まで文字を削除します。

例題

▼ **Delete string**関数の使用例を次に示します。結果を変数“v結果”に代入します。

```
v結果:=Delete string("Lamborghini" ; 6 ; 5)        ` v結果に"Lambo"を代入
v結果:=Delete string("Indentation" ; 6 ; 2) ` v結果に"Indention"を代入
v結果:=Delete string(var ; 3 ; 32000)        ` v結果にvarの最初の2文字を代入
```

参照

Change string、Insert string、Replace string

Replace string

Replace string(元の文字列 ; 対象文字列 ; 置き換え文字列 {; 回数}) → 文字列

引数	タイプ		説明
元の文字列	文字列	→	元の文字列
対象文字列	文字列	→	置き換えの対象となる文字列
置き換え文字列	文字列	→	置き換える文字列 空の文字列の場合、発見した 対象文字列をすべて削除
回数	数値	→	置き換える回数 省略した場合、すべて置き換え
戻り値	文字列	←	結果文字列

説明

Replace string関数は、＜元の文字列＞に存在するすべての＜対象文字列＞を＜置き換え文字列＞で＜回数＞分だけ置き換えます。

＜置き換え文字列＞が空の文字列（""）の場合は、**Replace string**関数は＜元の文字列＞の中の＜対象文字列＞をすべて削除します。

＜回数＞を指定した場合、**Replace string**関数は＜元の文字列＞の最初の文字から探して、その回数分だけ＜対象文字列＞で置き換えます。指定しない場合、発見した＜対象文字列＞をすべて置き換えます。

＜対象文字列＞が空の文字列の場合は、**Replace string**関数は変更前の文字列を返します。

例題

1. **Replace string**関数の使用例を次に示します。結果を変数“v結果”に代入します。コメントは、変数“v結果”に代入される内容についての説明です。

```
v結果:=Replace string("Willow" ; "ll" ; "d") ` v結果に"Widow"を代入
v結果:=Replace string("Shout" ; "o" ; "")  ` v結果に"Shut"を代入
v結果:=Replace string(var ; Char(9) ; ",") ` varの tabをすべてカンマに置き換える
```

2. 以下の例は、“v結果”のテキストからキャリッジリターンとタブを取り除きます。

```
v結果:=Replace string(Replace string(vtResult;Char(13) ;"");Char(9) ; "")
```

参照

Change string、Delete string、Insert string

Mac to Win

Mac to Win(テキスト)→ 文字列

引数	タイプ		説明
テキスト	文字列	→	Macintosh ASCIIマップを使用して表されたテキスト
戻り値	文字列	←	Windows ANSIマップを使用して表されたテキスト

説明

Mac to Win関数は、Macintosh ASCIIマップを使用して表され、引数<テキスト>に渡されるテキストと同等のテキストをWindows ANSIマップを使用して表して返します。

この関数には、Macintosh ASCIIマップを使用して表したテキスト引数が必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、**Mac to Win**関数の主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

注：このコマンドはCR（キャリッジリターン）文字をCRLF（キャリッジリターン+改行、Asciiコード13+10）に置き換えます。したがって文字数が元より多くなります。

例題

Windows**SEND PACKET**コマンドを使用してドキュメントに文字を書き込む時に、出力ASCIIマップを使用してMacintoshからWindowsへの文字のフィルタリングを実行（**USE ASCII MAP**コマンドを参照）していない場合には、手動でテキストをMacintoshからWindowsに変換する必要があります。この操作は、以下のように実行します。

```
SEND PACKET($vhDocRef ; Mac to Win(vtSomeText))
` ...
```

参照

SEND PACKET、USE ASCII MAP、Win to Mac

Win to Mac

Win to Mac(テキスト)→ 文字列

引数	タイプ		説明
テキスト	文字列	→	Windows ANSIマップを使用して表されたテキスト
戻り値	文字列	←	Macintosh ASCIIマップを使用して表されたテキスト

説明

Win to Mac関数は、Windows ANSIマップを使用して表され、引数<テキスト>に渡されるテキストと同等のテキストをMacintosh ASCIIマップを使用して表して返します。

このコマンドには、Windows ANSIマップを使用して表したテキスト引数が必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、**Win to Mac**関数の主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

注：このコマンドはCRLF（キャリッジリターン+改行、Asciiコード13+10）をCR（キャリッジリターン）文字に置き換えます。したがって文字数が元より少なくなります。

例題

Windows**SEND PACKET**コマンドを使用してドキュメントに文字を書き込む時に、入力ASCIIマップを使用してWindowsからMacintoshへの文字のフィルタリングを実行（**USE ASCII MAP**コマンドを参照）していない場合には、手動でテキストをWindowsからMacintoshに変換する必要があります。この操作は、以下のように実行します。

```
RECEIVE PACKET($vhDocRef ; vtSomeText ; 16*1024)
vtSomeText:=Win to Mac(vtSomeText)
` ...
```

参照

Mac to Win、RECEIVE PACKET、USE ASCII MAP

Mac to ISO

Mac to ISO(テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	Macintosh ASCIIマップを使用して表されたテキスト
戻り値	数値	←	ISO Latin-1文字マップを使用して表されたテキスト

説明

Mac to ISO関数は、Macintosh ASCIIマップを使用して表され、引数<テキスト>に渡されるテキストと同等のテキストをISO Latin-1文字マップを使用して表して返します。

通常は、このコマンドを使用する必要はありません。

この関数には、Macintosh ASCIIマップを使用して表したテキスト引数が必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMacintosh ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、この関数を使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

したがって、プラットフォームが何であっても、ISO Latin-1を使用してディスク上にHTMLドキュメントを作成したい場合には、**Mac to ISO**関数を使用して対象となるテキストを変換するだけですみます。これが、**Mac to ISO**関数の主要な目的です

注：Mac to ISO関数は、"Conversion"(MapC)が存在する場合は、それを使います。詳細は『Customizer Plusリファレンス』を参照してください。

例題

1. 以下のコードの行では、vtSomeTextに格納されている（仮想の）Macintoshでコード化されたテキストを、ISO Latin-1でコード化されたテキストに変換しています。

```
vtSomeText:=Mac to ISO(vtSomeText)
```

2. 4D Web Serverアプリケーションの開発中にHTMLドキュメントを作成し、後で**SEND HTML FILE**コマンドを使用してイントラネットまたはインターネット経由で送信しようとしています。これらのHTMLドキュメントの一部は他のドキュメントに参照やリンクを持っています。以下のプロジェクトメソッドでは、引数として受信したWindowsからのHTMLベースのパス名やMacintoshのパス名を求めています。

```
`「HTML Pathname」プロジェクトメソッド
` HTML Pathname(テキスト ) → テキスト
` HTML Pathname(ネイティブなファイルマネージャパス名) → HTMLパス名
C_TEXT($0 ; $1)
C_LONGINT($vlChar ; $vlAscii)
C_STRING(31 ; $vsChar)
$0:=""
If(On 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
```

注：「On Windows」プロジェクトメソッドは、第52章の「システムドキュメントコマンド」の中で示されています。

「On Window」プロジェクトメソッドをデータベースに作成した後、特定のディレクトリにあるドキュメントへのFTPリンクのリストを追加したい場合には、以下のように記述できます。

```
`「On Startup」データベースメソッドでインタープロセス変数を設定する
<>vsFTPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDirectory:="APS500:Spiders:" ` Macintoshファイルマネージャのパス名
` ...
` ...
ARRAY STRING(31 ; $asDocuments ; 0)
DOCUMENT LIST(...; $asDocuments)
$vlNbDocuments:=Size of array($asDocuments)
jsHandler:=...
For($vlDocument ; 1 ; $vlNbDocuments)
        vtHTMLCode:=vtHTMLCode+"<P><A HREF="+Char(34)+<>vsFTPURL
        + HTML Pathname(Substring($1+$asDocuments{$vlDocument}
        ; Length(<>vsFTPDirectory)+1))+Char(34)+jsHandler
        +"> "+$asDocuments{$vlDocument}+"</A></P>"+Char(13)
End for
` ...
```

参照

ISO to Mac、SEND HTML FILE、SEND PACKET、USE ASCII MAP

ISO to Mac

ISO to Mac(テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	ISO Latin-1文字マップを使用して表されたテキスト
戻り値	数値	←	Macintosh ASCIIマップを使用して表されたテキスト

説明

ISO to Mac関数は、ISO Latin-1文字マップを使用して表され、引数<テキスト>に渡されるテキストと同等のテキストをMacintosh ASCIIマップを使用して表して返します。

通常、このコマンドを使用する必要はありません。

このコマンドには、ISO Latin-1文字マップを使用して表したテキスト引数が必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMacintosh ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、ASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、4DはASCII変換を実行しません。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacintoshでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

したがって、プラットフォームが何であっても、**RECEIVE PACKET**コマンドを使用してディスクからISO Latin-1によるHTMLドキュメントを読み取りしたい場合には、**ISO to Mac**関数を使用して対象となるテキストを変換するだけですみます。これが、**ISO to Mac**関数の主要な目的です

注：ISO to Mac関数は、"Conversion"(MapC)が存在する場合は、それを使います。詳細は『Customizer Plusリファレンス』を参照してください。

例題

以下のコードの行では、vtSomeTextに格納されている（仮想の）ISO Latin-1でコード化されたテキストを、Macintoshでコード化されたテキストに変換しています。

```
`ISO Latin-1によるHTMLドキュメントからテキストを読み取る
RECEIVE PACKET($vhDocRef ; vtSomeText ; 16*1024)
vtSomeText:=ISO to Mac(vtSomeText)
```

参照

Mac to ISO、RECEIVE PACKET、USE ASCII MAP

ストラクチャアクセスコマンド 49

この章のコマンドは、データベースストラクチャの情報を返します。これらのコマンドは、テーブル数、各テーブルのフィールド数、テーブルの名前、フィールドの名前、各フィールドのタイプや属性を返します。

データベースストラクチャを特定すると、他のデータベースにコピーされる一連のプロジェクトメソッドやフォームを開発する、または使用している場合には非常に便利です。データベースストラクチャの情報を読み取ることによって、さまざまなデータベースで使用可能になり、移植性の高いコードを生成することができます。

参照

Count fields、Count tables、Field、GET FIELD PROPERTIES、Pointers、SET INDEX、Table、Table name

Count tables

Count tables → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	データベース中のテーブル数

説明

Count tables関数は、データベース中のテーブルの数を返します。テーブルは作成された順番に番号が付けられます。

例題

▼以下の例は、配列“テーブル配列”の配列要素を初期化します。この配列はフォーム上のポップアップメニューに割り当てられ、データベース内のテーブルのリストを表示するために使用されます。**For**文では、テーブルの数がこのメソッドを実行中に変更することができるため、**Count tables**関数の代わりに**Size of array**関数を使用しています。

```
ARRAY STRING(15 ; テーブル配列 ; Count tables) ` 配列を作成する
For($i ; 1 ; Size of array(テーブル配列)) ` 配列を通してループする
    テーブル配列 {$i}:=Table name($i) ` 配列要素に値を代入する
End for
```

参照

Count fields、Table name

Count fields

Count fields(テーブル番号 | テーブルポインタ) → 数値

引数	タイプ		説明
テーブル番号または	数値または	→	テーブル番号、または
テーブルポインタ	ポインタ	→	テーブルを指し示すポインタ
戻り値	数値	←	テーブルのフィールド数

説明

Count fields関数は、＜テーブル番号＞または＜テーブルポインタ＞で指定したテーブルのフィールドの数を返します。

フィールドは作成された順に番号が付けられています。

例題

▼以下の例は、引数$iに渡されるテーブルポインタで、ドロップダウン（ポップアップ）メニューにフィールドのリストが表示されます。まず、配列“フィールド配列”が作成され、**For**ループを通して配列要素に値を代入します。**For**ループでは、**Count fields**関数の代わりに**Size of array**関数を使用します。これは、以下のコードが実行されている間にテーブル内のフィールド数が変更されるためです。

```
C_LONGINT($TableNumber ; $HowMany ; $i)
$TableNumber :=Table($1)
      ` 配列を作成する
ARRAY STRING(15 ; arFields ; Count fields($TableNumber))
$HowMany:=Size of array(arFields)
For($i ; 1 ; $HowMany)              ` 配列を通してループする
      arFields {$i}:=Field name($TableNumber ; $i) ` 配列要素に値を代入する
End for
```

参照

Count tables、Field name、GET FIELD PROPERTIES

Table name

Table name(テーブル番号 | テーブルポインタ) → 文字列

引数	タイプ		説明
テーブル番号または	数値または	→	テーブル番号、または
テーブルポインタ	ポインタ	→	テーブルへのポインタ
戻り値	数値	←	テーブルのフィールド数

説明

Table name関数は、＜テーブル番号＞または＜テーブルポインタ＞で指定したテーブルの名前を返します。

▼ 以下の例は、あるテーブルのレコードを表示します。テーブルへの参照は、テーブルに対するポインタとして渡されます。**Table name**関数は、ウインドウのタイトルバーにテーブルの名前を表示するために使用されます。

```
` $1はテーブルに対するポインタ
SET WINDOW TITLE(Table name($1) + " のレコード")
` ウインドウタイトルを設定する
DISPLAY SELECTION($1->)        ` セレクションを表示する
```

参照

Count tables、Field name、Table

Field name

Field name((テーブル番号 ; フィールド番号) | フィールドポインタ) → 文字列

引数	タイプ		説明
テーブル番号	数値	→	テーブル番号
フィールド番号または	数値	→	テーブル番号が渡された場合、フィールド番号または
フィールドポインタ	ポインタ	→	フィールドへのポインタ
戻り値	文字列	←	フィールド名

説明

Field name関数は、＜テーブル番号＞と＜フィールド番号＞または＜フィールドポインタ＞で指定したフィールドの名前を返します。

例題

1. 以下の例は、配列“フィールド{1}”の2番目の要素にテーブル番号1に含まれるフィールド番号2のフィールドの名前を代入します。

```
フィールド{1}{2}:=Field name(1 ; 2)
```

2. 以下の例は、配列“フィールド{1}”の2番目の要素にフィールド“[私のテーブル]私のフィールド”の名前を代入します。この方法は、“[私のテーブル]私のフィールド”のフィールド名が変更された場合でも、新しいフィールド名が返されるため便利です。

```
フィールド{1}{2}:=Field name(->[私のテーブル]私のフィールド)
```

3. 以下の例は、フィールドに対してポインタを渡して、警告を表示します。

```
ALERT(Table name(Table($1)) + "テーブルの" + Field name($1)
                    + "フィールドのID番号は5文字以上でなければいけません")
```

参照

Count fields、Field、Table name

Table

Table(テーブル番号 | ポインタ) → ポインタ | 数値

引数	タイプ		説明
テーブル番号または	数値または	→	テーブル番号または
ポインタ	ポインタ	→	テーブルへのポインタまたは
		→	フィールドへのポインタ
戻り値	ポインタまたは	←	テーブル番号を渡した場合、テーブルへのポインタ
	数値		テーブルポインタを渡した場合、テーブル番号、フィールドポインタを渡した場合、テーブル番号

説明

Table関数には、3つの形式があります。

＜テーブル番号＞を指定した場合は、**Table**関数はテーブルへのポインタを返します。＜ポインタ＞にテーブルへのポインタを指定した場合は、**Table**関数はテーブル番号を返します。＜ポインタ＞にフィールドへのポインタを指定した場合には、**Table**関数はそのフィールドの属するテーブルのテーブル番号を返します。この形式は、フィールドポインタだけを指定してテーブル番号が求められます。

例題

1. 以下の例は変数“vテーブルポインタ”に3番目のテーブルに対するポインタを代入します。

   vテーブルポインタ:=**Table**(3)

2. 変数“vテーブルポインタ”を第2の形式に使用すると、Table関数は数値の3を返します。以下の例を実行すると変数“テーブル番号”に3を代入します。

   vテーブル番号:=**Table**(テーブルポインタ)

3. 以下の例は変数“vテーブル番号”にテーブル[テーブル3]のテーブル番号を代入します。

   vテーブル番号:=**Table**(->[テーブル3])

4. 以下の例は変数“vテーブル番号”にフィールド“[テーブル3]フィールド1”の属するテーブルのテーブル番号を代入します。

   vテーブル番号:=**Table**(->[テーブル3]フィールド1)

参照

Count tables、Field、Pointers、Table name

GET TABLE PROPERTIES

GET TABLE PROPERTIES(テーブルポインタ | テーブル番号; 非表示
{; 新規保存トリガ{; 既存保存トリガ{; 削除トリガ{; 読込みトリガ}}}})

引数	タイプ		説明
テーブルポインタ¦	ポインタ¦	→	テーブルポインタ、または
テーブル番号	倍長整数		テーブル番号
非表示	ブール	←	True=非表示、False=表示
新規保存トリガ	ブール	←	True=トリガ「新規レコード保存時」がアクティブ、そうでない場合はFalse
既存保存トリガ	ブール	←	True=トリガ「既存レコード保存時」がアクティブ、そうでない場合はFalse
削除トリガ	ブール	←	True=トリガ「レコード削除時」がアクティブ、そうでない場合はFalse
読込みトリガ	ブール	←	True=トリガ「レコード読込時」がアクティブ、そうでない場合はFalse

説明

GET TABLE PROPERTIESコマンドは、＜テーブルポインタ＞または＜テーブル番号＞で渡したテーブルのプロパティを返します。最初の引数としてテーブル番号またはテーブルへのポインタを渡すことができます。

このコマンドが実行されると

- このテーブルに対し“非表示”属性が設定されている場合、引数＜非表示＞にTrueが返され、そうでない場合Falseが返されます。非表示属性を指定することにより、4D標準のエディタ（ラベル、チャート等）の使用時にテーブルを隠すことができます。
- このテーブルに対し「新規レコード保存時」トリガが設定されている場合、引数＜新規保存トリガ＞にTrueが返され、そうでない場合Falseが返されます。
- このテーブルに対し「既存レコード保存時」トリガが設定されている場合、引数＜既存保存トリガ＞にTrueが返され、そうでない場合Falseが返されます。
- このテーブルに対し「レコード削除時」トリガが設定されている場合、引数＜削除トリガ＞にTrueが返され、そうでない場合Falseが返されます。
- このテーブルに対し「レコード読込時」トリガが設定されている場合、引数＜読込みトリガ＞にTrueが返され、そうでない場合Falseが返されます。

参照

GET FIELD ENTRY PROPERTIES、GET FIELD PROPERTIES、GET RELATION PROPERTIES

Field

Field(テーブル番号 | フィールドポインタ {; フィールド番号) → 数値 | ポインタ

引数	タイプ		説明
テーブル番号または フィールドポインタ	数値または ポインタ	→	テーブル番号、または フィールドへのポインタ
フィールド番号	数値	→	テーブル番号を渡した場合、フィールド番号
戻り値	数値または	←	フィールドポインタを渡した場合、フィールド番号
	ポインタ		テーブル番号とフィールド番号を渡した場合、フィールドポインタ

説明

Field関数には、2つの形式があります。

＜テーブル番号＞と＜フィールド番号＞を指定した場合は、**Field**関数はフィールドへのポインタを返します。

＜フィールドポインタ＞を指定した場合には、**Field**関数はフィールド番号を返します。

例題

1. 以下の例は、変数“vフィールドポインタ”に3番目のテーブルに含まれる2番目のフィールドへのポインタを代入します。

```
vフィールドポインタ:=Field(3 ; 2)
```

変数“vフィールドポインタ”を第2の形式に使用すると、**Field**関数は数値の2を返します。

2. 以下の例を実行すると変数“vフィールド番号”に2を代入します。

```
フィールド番号:=Field(vフィールドポインタ)
```

3. 以下の例は、変数“vフィールド番号”にフィールド“[テーブル3]フィールド2”のフィールド番号を代入します。

```
vフィールド番号:=Field(->[テーブル3]フィールド2)
```

参照

Count fields、Field name、GET FIELD PROPERTIES、Table

GET FIELD PROPERTIES

GET FIELD PROPERTIES(フィールドポインタ | テーブル番号 {; フィールド番号};
タイプ {; 長さ {;インデックス{;ユニーク{;非表示}}}})

引数	タイプ		説明
テーブル番号	数値	→	テーブル番号
フィールド番号	数値	→	フィールド番号
タイプ	数値	←	フィールドタイプ
長さ	数値	←	文字フィールドの長さ（バイト）
インデックス	ブールタイプ	←	False（偽）：インデックスなし、 True（真）：インデックスあり
ユニーク	ブールタイプ	←	False（偽）：ユニークではない True（真）：ユニーク
非表示	ブールタイプ	←	False（偽）：表示 True（真）：非表示

GET FIELD PROPERTIES(フィールドポインタ ; タイプ {; 長さ {;インデックス}})

引数	タイプ		説明
フィールドポインタ	ポインタ	→	フィールドポインタ
タイプ	数値	←	フィールドタイプ
長さ	数値	←	文字フィールドの長さ（バイト）
インデックス	ブール	←	False（偽）：インデックスなし、 True（真）：インデックスあり

説明

GET FIELD PROPERTIESコマンドは、＜テーブル番号＞と＜フィールド番号＞または＜フィールドポインタ＞で指定したフィールドの情報を返します。

以下のいずれかの引数を渡します。

■ 引数＜テーブル番号＞と＜フィールド番号＞、または

■ ＜フィールドポインタ＞にフィールドへのポインタ

コマンドの実行後、以下の情報が返されます。

■ ＜タイプ＞にはフィールドのタイプが返されます。以下のような4th Dimensionであらかじめ定義された定数の値が返されます。

定数	タイプ	値
Is Alpha Field	倍長整数	0
Is Text	倍長整数	2
Is Real	倍長整数	1
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Date	倍長整数	4
Is Time	倍長整数	11
Is Boolean	倍長整数	6
Is Picture	倍長整数	3
Is Subtable	倍長整数	7
Is BLOB	倍長整数	30

■ 引数＜長さ＞には、フィールドタイプが文字の場合（つまり、＜タイプ＞=Is Alpha Field）、フィールドの長さが返されます。その他のフィールドタイプに対しては、＜長さ＞の値は意味を持ちません。

■ 引数＜インデックス＞には、フィールドに＜インデックス＞が設定されていない場合は、“False（偽）”が、フィールドにインデックスが設定されている場合は、“True（真）”が返されます。＜インデックス＞の値は、フィールドタイプが文字、整数、倍長整数、実数、日付、時間、ブールの場合にだけ意味を持ちます。

■ 引数＜ユニーク＞には、フィールドがユニークに設定されているときはTrueが、そうでないときにはFalseが返されます。ユニーク設定はインデックスフィールドにのみできます。

■ 引数＜非表示＞には、フィールドが非表示に設定されているときにはTrueが、そうでないときにはFalseが返されます。非表示設定は4D標準のエディタの所定のフィールド（ラベルやチャートなど）を隠すために使うことができます。

例題

1. 以下の例は、変数“vタイプ”、“v長さ”、“vインデックス”、“vユニーク、“v非表示”に1番目のテーブルに含まれる3番目の属性を設定します。

```
GET FIELD PROPERTIES(1 ; 3 ; vタイプ ; v長さ ; vインデックス ; vユニーク ;
                                                    v非表示)
```

2. 以下の例は、変数“vタイプ”、“v長さ”、“vインデックス”、“vユニーク、“v非表示”にフィールド“[テーブル3]フィールド2”の属性を設定します。

```
GET FIELD PROPERTIES(->[テーブル3]フィールド2 ; vタイプ ; v長さ ;
                                vインデックス ; vユニーク ; v非表示)
```

参照

Field、Field name、SET INDEX

GET FIELD ENTRY PROPERTIES

GET FIELD ENTRY PROPERTIES(フィールドポインタ | テーブル番号
{; フィールド番号}; リスト{; 必須入力{; 表示のみ{; 修正不可}}})

引数	タイプ		説明
フィールドポインタ またはテーブル番号	ポインタ 倍長整数	→	フィールドポインタまたは テーブル番号
フィールド番号	倍長整数	→	1番目の引数にテーブル番号が渡された場合、フィールド番号
リスト	文字列	←	関連する選択リスト名または空の文字列
必須入力	ブール	←	True = 必須入力、False = 任意
表示のみ	ブール	←	True = 表示のみ、False = 入力可
修正不可	ブール	←	True = 修正不可、False = 修正可

説明

GET FIELD ENTRY PROPERTIESコマンドは、＜テーブル番号＞および＜フィールド番号＞、または＜フィールドポインタ＞で指定したフィールドのデータ入力プロパティを返します。

次のいずれかの引数を指定することができます。

■ ＜テーブル番号＞および＜フィールド番号＞に対し、テーブル番号とフィールド番号を指定

■ ＜フィールドポインタ＞にフィールドのポインタを指定

注：このコマンドは、ストラクチャウインドウレベルで定義したプロパティを返します。同様のプロパティはフォームレベルで定義できます。

このコマンドが実行されると

■ 引数＜リスト＞には、このフィールドに関連付けられた選択リスト名（存在する場合）が返されます。リストは次のタイプのフィールドに関連付けることができます：文字列、テキスト、実数、整数、倍長整数、日付、時間、ブール。

フィールドに関連付けられた選択リストが存在しない場合や、フィールドタイプが選択リスト用のものでない場合、空の文字列（""）が返されます。

■引数<必須入力>には、フィールドが“必須入力”であればTrueが、そうでない場合にはFalseが返されます。この必須入力属性は、サブテーブルとBLOBを除き、あらゆるフィールドタイプに設定することができます。

■引数<表示のみ>には、フィールドが“表示のみ”であればTrueが、そうでない場合にはFalseが返されます。入力不可のフィールドは、読み取りのみであり、データの入力はできません。この表示のみ属性は、サブテーブルとBLOBを除き、あらゆるフィールドタイプに設定することができます。

■引数<修正不可>には、フィールドが“修正不可”であればTrueが、そうでない場合にはFalseが返されます。修正不可のフィールドへの入力は一度しか行えず、以後修正はできません。この修正不可属性は、サブテーブルとBLOBを除き、あらゆるフィールドタイプに設定することができます。

参照

GET FIELD PROPERTIES、GET RELATION PROPERTIES、GET TABLE PROPERTIES

GET RELATION PROPERTIES

GET RELATION PROPERTIES(フィールドポインタ | テーブル番号
{; フィールド番号}; 1テーブル; 1フィールド{; 選択フィールド{; 自動1{; 自動n}}})

引数	タイプ		説明
フィールドポインタまたはテーブル番号	ポインタ 倍長整数	→	フィールドポインタまたはテーブル番号
フィールド番号	倍長整数	→	1番目の引数にテーブル番号が渡された場合、フィールド番号
1テーブル	倍長整数	←	1テーブルの番号、またはフィールドからリレートが定義されていない場合、0
1フィールド	倍長整数	←	1フィールドの番号、またはフィールドからリレートが定義されていない場合、0
選択フィールド	倍長整数	←	選択フィールド番号、または選択フィールドなしの場合 0
自動1	ブール	←	True = 1対1自動リレート False = 1対1マニュアルリレート
自動n	ブール	←	True = 1対n自動リレート False = 1対nマニュアルリレート

説明

GET RELATION PROPERTIESコマンドは、＜テーブル番号＞および＜フィールド番号＞、または＜フィールドポインタ＞で指定した元のフィールドを起点とするリレート（存在する場合）のプロパティを返します。

次のいずれかの引数を指定することができます。

■ ＜テーブル番号＞および＜フィールド番号＞に対し、テーブルとフィールドを指定

■ ＜フィールドポインタ＞にフィールドのポインタを指定

このコマンドが実行されると

■ 引数＜1テーブル＞および＜1フィールド＞にはそれぞれ、起点フィールドからのリレートが指し示すテーブル番号およびフィールド番号が納められます。このフィールドを起点とするリレートが存在しない場合、これらの引数には0が返されます。

■引数＜選択フィールド＞には、このリレートで定義されたターゲットテーブルのワイルドカード選択フィールド番号が納められます。このリレートに対しワイルドカード選択フィールドが設定されていない場合、または起点フィールドからのリレートがない場合、この引数には0が返されます。

■引数＜自動1＞および＜自動n＞にはそれぞれ、このリレートに対し「自動1対1リレート」および「自動1対nリレート」チェックボックスがチェックされている場合にTrueが、そうでない場合にはFalseが返されます。

注：引数＜自動1＞および＜自動n＞は、起点フィールドから始まるリレートが存在しない場合にもTrueを返します（この場合、返す値には意味がありません）。2つの引数＜1テーブル＞および＜1フィールド＞の値により、リレートが存在するかどうかを確認できます。

参照

GET FIELD ENTRY PROPERTIES、GET FIELD PROPERTIES、GET TABLE PROPERTIES

SET INDEX

SET INDEX(フィールド; インデックス{; モード}{; *})

引数	タイプ		説明
フィールド	フィールドまたは サブフィールド	→	インデックスを作成または 消去するフィールド
インデックス	ブール	→	インデックスを作成（True）または インデックスを消去（False）
モード	倍長整数	→	インデックスモード（パーセントで）
*	*	→	非同期にインデックスを作成

説明

SET INDEXコマンドは、引数＜フィールド＞に指定したフィールドまたはサブフィールド対してインデックスを作成または取り除きます。

フィールドまたはサブフィールドのインデックスを作成する場合、＜インデックス＞に“True（真）”を指定します。＜フィールド＞にインデックスが既に付いている場合は、このコマンドは何もしません。インデックスを削除するには、＜インデックス＞に“False（偽）”を指定します。インデックスが存在しない場合は、このコマンドは何もしません。

フィールドまたはサブフィールドのインデックスを作成する場合、＜インデックス＞に“False（偽）”の場合は、＜フィールド＞にインデックスが付いているとそのインデックスを取り除きます。

SET INDEXコマンドはロックされたレコードのインデックス付けは行いません。レコードがロック解除されるまで待機します。

バージョン6.5より2つのインデックスモードを選択できるように成りました。"従来"モードは以前の4Dバージョンを使ったモードで、新しい"高速"モードは『4th Dimensionデザインリファレンス』マニュアルを参照してください。

選択したインデックスモードを使って、モード引数を適用するか選択します。モード引数はコマンドにインデックス作成（インデックス引数がTrueの時）を指定した場合にのみモード引数が使われます。

■ モード引数を通さなかった場合、インデックスは"従来"モードで実行されます。この時、インデックスが別のプロセスで作成されていると、データベースは実行中でも利用できる状態になります。インデックスを使ったオペレーションはインデックスを生成中に実行してもインデックスは使えません。フィールドがインデックスされるか決定するのは**GET FIELD PROPERTIES**関数を使用します。

■ モード引数を通した場合､コマンドは"高速"モードで使用されます。この時、インデックスのプロセスの間にテーブルのデータを修正することはできません。

■ モード引数を適用せず、"従来"モードでする、インデックスの作成は別プロセスで行われるため、その間データベースを使用することができます。インデックスを作成中にインデックスを使用する処理をしてもインデックスは用いられません。＜フィールド＞にインデックスが付いているかどうかを調べるには**GET FIELD PROPERTIES**コマンドを使用します。

■ モード引数を適用して、"高速"モードを使ったコマンドの場合は、インデックスプロセスが実行中にテーブルデータの更新が可能となります。

パーセンテージを現す整数値をモード引数に設定する必要があります。この値は、インデックスを最も有効に活用するタイプを指示することができます。これは下記の間でなければいけません。

■ mode = 0：インデックスはレコードの追加や挿入用に最適化されます。

■ mode = 100：インデックスは検索用に最適化されます。

オプション引数＜*＞を用いて、非同期に（同時に）インデックスが付けられるようになりました。非同期なインデックス付けにより、インデックス付けが終了したかどうかに関係なく、呼び出し側のメソッドの実行をそのまま続行できます。ただし、インデックスが必要なコマンドを実行すると失敗します。

例題

▼ 以下の例は、従来モードでフィールド"顧客ID"にインデックスを付けます。

```
UNLOAD RECORD([顧客])
SET INDEX([顧客]顧客ID ;True)
```

▼ 下記のコードは、[Clients]Nameフィールドのインデックスを作成します。このフィールドは主に検索に使用されます。

```
SET INDEX([Clients]Name;True;100)
```

▼ [Prospects]Nameフィールドのインデックスを作成します。このフィールドは、レコードの追加や挿入および検索をする時に使用されますが、追加や挿入の方が検索よりも頻繁に行なわれます。

```
SET INDEX([Prospects]Name;True;30)
```

参照

GET FIELD PROPERTIES、ORDER BY、QUERY

Get database parameter

Get database parameter({テーブル;} セレクタ) →倍長整数

引数	タイプ		説明
テーブル	テーブル	→	属性の値を得るテーブル。この引数が省略されている場合はデフォルトテーブル
セレクタ	倍長整数	→	データベース属性コード
戻り値	倍長整数	←	属性の値

説明

このコマンドは、カレントプロセス用の4Dデータベース属性の値を読み込むことができます。

セレクタ引数は、読み込む属性を指定します。4th Dimensionは「データベース属性」のカテゴリー内に、前もって定義されている下記のような定数があります。

定数	タイプ	値
Seq Order Ratio	倍長整数	1
Seq Access Optimization	倍長整数	2
Seq Distinct Values Ratio	倍長整数	3
Index Compacting	倍長整数	4
Seq Query Select Ratio	倍長整数	5
Minimum Web Process	倍長整数	6
Maximum Web Process	倍長整数	7
Web Conversion Mode	倍長整数	8
Database Cache Size	倍長整数	9
4th Dimension Scheduler	倍長整数	10
4D Server Scheduler	倍長整数	11
4D Client Scheduler	倍長整数	12
4D Server Timeout	倍長整数	13
4D Client Timeout	倍長整数	14
Port ID	倍長整数	15
IP Addres to listen	倍長整数	16
Character set	倍長整数	17
Max Concurrent Web Processes	倍長整数	18
Client Minimum process Web	倍長整数	19
Client Maximum process Web	倍長整数	20
Client Maximum Web requests size	倍長整数	21
Client Port ID	倍長整数	22
Client IP Address to listen	倍長整数	23

Client Character set	倍長整数	24
Client Max Concurrent Web Proc	倍長整数	25
Cache Writing Mode	倍長整数	26
Maximum Web requests size	倍長整数	27
4D Server Log Recording	倍長整数	28
Web Log Recording	倍長整数	29
Client Web Log Recording	倍長整数	30
Table Sequence Number	倍長整数	31
Real Display Precision	倍長整数	32
Debug Log Recording	倍長整数	34

セレクタ1から8、および10から34に対して、この関数から返される値、ならびに各セレクタの有効範囲については、**SET DATABASE PARAMETER**コマンドの説明を参照してください。

データベースキャッシュサイズ（9）のセレクタはカレントのデータベースのメモリキャッシュのサイズを得ることができます。戻り値はバイトで示されます。最大キャッシュサイズはWindows、Macintoshの両方のプラットフォームでセットでき、最小キャッシュサイズはMacintoshのプラットフォームでのみセットできます。セットするには、データベースプロパティのダイアログボックスでします。実際のサイズはデータベースキャッシュはセットの仕方とカレントのシステムリソースの両方によって割り当てられます。**Get database parameter**関数は4Dによってデータベースキャッシュを割り当てたメモリの際のサイズを得ることができます。

注：ランゲージを使ってはデータベースキャッシュメモリのサイズをセットできません。つまり、データベースキャッスサイズセレクタはSET DATABASE PARAMETERコマンドを使ってもセットされません。

例題

(1)以下のメソッドにより、4Dスケジューラの現在の値を得ることができます。

```
C_LONGINT($ticksbtwcalls;$maxticks;$minticks;$lparams)
If(Application type=4th Dimension) ` 4D シングルユーザが使われているとき
$lparams:=Get database parameter(4th Dimension scheduler)
$ticksbtwcalls:=$lparams & 0x00ff
$maxticks:=($lparams>>8) & 0x00ff
$minticks:=($lparams>>16) & 0x00ff
End if
```

(2)セレクタ16（IP Address to listen）により、HTTPのリクエストを受けた4D Webサーバのアドレスを取得することができます。以下の例は16進化10進数への分割を行います。

C_LONGINT($a;$b;$c;$d)
C_LONGINT($addr)
$addr:=**Get database parameter**(IP Address to listen)
$a:=($addr>>24)&0x000000ff
$b:=($addr>>16)&0x000000ff
$c:=($addr>>8)&0x000000ff
$d:=$addr&0x000000ff

参照

DISTINCT VALUES、QUERY SELECTION、SET DATABASE PARAMETER

SET DATABASE PARAMETER

SET DATABASE PARAMETER({テーブル;} セレクタ;値)

引数	タイプ		説明
テーブル	テーブル	→	属性を設定するテーブル 引数が省略されている場合は デフォルトテーブル
セレクタ	倍長整数	→	変更するデータベース属性コード
値	倍長整数	→	属性の値

説明

このコマンドは、カレントプロセス用に4Dデータベース内部の様々な属性を変更することができます。

セレクタは、変更するデータベースの属性コードを指定します。4th Dimensionは「データベース属性」のカテゴリー内に、前もって定義されている下記のような定数があります。次の表は、各定数とその有効範囲を示しています。

セレクタ	値	有効範囲
Seq Order Ratio	1	カレントテーブルとプロセス
Seq Access Optimization	2	カレントテーブルとプロセス
Seq Distinct Values Ratio	3	カレントテーブルとプロセス
Index Compacting	4	4Dアプリケーション(*)
Seq Query Select Ratio	5	カレントテーブルとプロセス
Minimum Web Process	6	4th Dimension、4D Server(*)
Maximum Web Process	7	4th Dimension、4D Server(*)
Web conversion mode	8	カレントプロセス
Database cache size	9	4Dアプリケーション(*)(**)
4th Dimension Scheduler	10	4Dアプリケーション(*)

4D Server Scheduler	11	4Dアプリケーション(*)
4D Client Scheduler	12	4Dアプリケーション(*)
4D Server Timeout	13	4Dアプリケーション（正の値の場合）(***)
4D Client Timeout	14	4Dアプリケーション（正の値の場合）(***)
Port ID	15	4th Dimension、4D Server(*)
IP Address to listen	16	4th Dimension、4D Server(*)
Character set	17	4th Dimension、4D Server(*)
Max Concurrent Web Processes	18	4th Dimension、4D Server(*)
Client Minimum process Web	19	すべての4D Clientマシン(*)
Client Maximum process Web	20	すべての4D Clientマシン(*)
Client Max Web requests size	21	すべての4D Clientマシン(*)
Client Port ID	22	すべての4D Clientマシン(*)
Client IP Address to listen	23	すべての4D Clientマシン(*)
Client Character set	24	すべての4D Clientマシン(*)
Client Max Concurrent Web Proc	25	すべての4D Clientマシン(*)
Cache writing mode	26	4Dアプリケーション(*)
Maximum Web requests size	27	4th Dimension、4D Server(*)
4D Server Log Recording	28	4D Server (*)
Web Log Recording	29	4th Dimension, 4D Server (*)
Client Web Log Recording	30	All 4D Client machines (*)
Table Sequence Number	31	4D Application
Real Display Precision	32	4D Application
Debug Log Recording	34	4D アプリケーション

(*) この場合、引数＜テーブル＞は無視されます。
(**) このセレクタは読み込みのみ可能です（Get database parameter関数を参照）。
(***)引数＜値＞が負の値である場合、設定はカレントプロセスにのみ影響し、次回のリクエスト時にはリセットされます。

値は、属性の値を指定します。値の内容は変更しようとする属性によって違います。セレクタで指定する可能性のある値を示します。

■ セレクタ＝1(Seq Order Ratio)(シーケンシャルソートの実行)

値：0→100,000

内容：レコードの（セレクトされたレコードとレコードの合計数の間の）選択率。その率以下ではソートがシーケンシャルモードで実行されます。この率は、100,000分の1単位で表わされます。デフォルト値は9,000（=9%）です。

■ セレクタ＝2(Seq Access Optimization)(シーケンシャルソートの最適化)

値：0または1（0：最適化されず、1：最適化する）

内容：シーケンシャルアクセス（配列のソート、検索、選択）用の最適化モード。最適化モードでは、4Dはディスクからの多くのレコードを一度に読もうとしますが、これらをキャッシュ内には置きません。このモードは、キャッシュのサイズが低いためです。デフォルトでは、値は1になります（最適化モード）。

■ セレクタ＝3(Seq Distinct Values Ratio)(シーケンシャルDistinct Valuesの実行)

値：0→100,000

内容：レコードの（セレクトされたレコードとレコードの合計数の間の）選択率。その率以下では**DISTINCT VALUES**コマンドがシーケンシャルモードで実行されます。この率は、100,000分の1単位で表わされます。デフォルト値は0です。

■ セレクタ＝4(Index Compacting)(インデックスの圧縮)

値：0または1（0：no、1：yes）

内容：インデックスページ圧縮の可能または不可能。デフォルトでは値は1（インデックスは必要であればコンパクト化される）です。インデックスページは多くのインデックスやレコードを含むデータベースでは、4Dのメモリキャッシュを多量に使います。キャッシュがいっぱいで4Dが空きの追加を必要としていると、キャッシュ内のデータは正しくアンロードしません。データをアンロードする前に空きを増やさないと、プログラムはインデックスページの圧縮に空きスペースができるかチェックします。別の方法は後でデータを再ロードを避けることが可能です。

■ セレクタ＝5(Seq Query Select Ratio)(シーケンシャル検索の実行)

値：0→100,000

内容：レコードの（セレクトされたレコードとレコードの合計数の間の）選択率。その率以下では**QUERY SELECTION**コマンドがシーケンシャルモードで実行されます。この率は、100,000分の1単位で表わされます。デフォルト値は0です。

■ セレクタ=6(Minimum Web Process)(Webプロセスの最小数)

値：0→32,767

内容：4th Dimensionならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最少数。デフォルトでは、値は0になります（下記参照）。

■ セレクタ＝7(Maximum Web Process)(Webプロセスの最大数)

値：0→32,767

内容：4th Dimensionならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最少数。デフォルトでは、値は10になります。

Webサーバが、非コンテキストモードでプロセスの再利用をするために、4DはWebプロセスを5秒間延滞し、次に起こりうるHTTPリクエストの実行のために待機させます。能力の面で言えば、各問い合わせに新しいプロセスを作成するよりも、この原理はずっと利点の多いものです。Webプロセスが再利用されると、もう一度5秒間延滞させられます。5秒以内に何のリクエストも発生しない場合、Webプロセス数が指定した最小数でなければプロセスはアボートされ、最小数に達した場合は再度延滞させられます。

これらの引数は、リクエストの数やメモリ等に応じて、Webサーバの機能を調整できるようにするものです。

■ セレクタ＝8(Web conversion mode)(Web変換モード)

値：0、1，2または3

0＝（デフォルト）ブラウザが対応している場合はHTML 4.0フォーマットに変換し、対応していない場合はHTML3.2と配列を使います。

1＝6.0.x変換モード

2＝6.5変換モード

3＝HTML4.0フォーマット+CSS-Pに変換（バージョン6.5.2から）

説明：4th Dimensionならびに4D Serverで使用するWeb用の4Dフォームの変換モード。デフォルトでは、4D Web ServerはCSS1（cacading style sheets）を使用し、4th Dimensionで表示される4Dフォームと同様のHTMLページを生成します。この機能を使用すると、バージョン 6.7より前の4Dで作成されたデータベースに関しては、フォームが正しく変換されない可能性があります。このため、フォーム変換モードを設定する必要があるかもしれません。

このモードは、**SET DATABASE PARAMETER**が呼び出されたプロセス（Webコンテキスト）に対してのみ設定することができます。このコマンドは、「On Web Connection」データベースメソッド内で呼び出してデータベースのすべてのフォームを確実に統一したり、あるいは特定のフォームを表示する前にのみ呼び出すこともできます。このコマンドは、コンテストモード、またはWebプロセス以外の場所から呼び出すと、何も行いません。

注：セレクタの追加はGet database parameter関数のデーターベースキャッシュサイズ（9）でできます。このセレクタはSET DATABASE PARAMETER関数では出来ません。さらに詳しい説明はGet database parameterを参照してください。

■ セレクタ ＝ 10 (4th Dimension Scheduler)

■ セレクタ ＝ 11 (4D Server Scheduler)

■ セレクタ ＝ 12 (4D Client Scheduler)

値：これら3つのセレクタに対し、引数＜値＞は16進数、0x00aabbcc の形式で表わされます。詳細は次の通りです。

aa = システムへのコール毎の最小tick数（0 ～ 100）

bb = システムへのコール毎の最大tick数（0 ～ 100）

cc = システムへのコール間のtick数（0 ～ 20）

これらの値のうち1つが範囲外であれば、4Dによって最大数に設定されます。引数＜値＞には、次の定義済標準値のうちいずれかを渡すことができます。

値 = -1：4Dに割り当てられた最高優先度

値 = -2：4Dに割り当てられた平均優先度

値 = -3：4Dに割り当てられた最低優先度

説明：この引数を使用して、4Dシステム内部コールをダイナミックに設定することができます。セレクタの値に応じて、スケジューラの値は次のアプリケーションのために設定されます。

■ このコマンドが4th Dimensionから呼び出された場合、4th Dimension（シングルユーザ）および4D Tools（セレクタ = 10）。

■ このコマンドが4D Serverから呼び出された場合、4D Server（セレクタ = 11）。

■ このコマンドが4D Clientから呼び出された場合、4D Client（セレクタ = 12）。

（例題1を参照）

■ セレクタ = 13（4D Server Timeout）

説明：この引数を使用して、4D Serverのタイムアウトの値を変更することができます。4D Serverのタイムアウトのデフォルト値は、サーバ側の「データベースプロパティ」ダイアログボックスの「接続設定」ページで定義します。

セレクタ「4D Server Timeout」により、対応する引数＜値＞に新しいタイムアウト（分単位で指定）を設定できます。この機能は、クライアント側でCPUを占有する時間がかかる処理を実行する前に、タイムアウト設定を長くしたい場合は特に便利です。例えば、膨大なページの印刷などは、予期しないタイムアウトになる可能性があります。

また、2種類のオプションがあります。

■ 引数＜値＞に正の値を渡すと、グローバルかつ永続的なタイムアウトが設定されます。この新しい値はすべてのプロセスに対して適用され、4Dアプリケーションの初期設定に保存されます（「データベースプロパティ」ダイアログボックスで変更した場合と同じ）。

■ 引数＜値＞に負の値を渡すと、ローカルで一時的なタイムアウトが設定されます。この新しい値は呼び出し元のプロセスに対してのみ適用され（他のプロセスではデフォルトの値を維持）、例えば処理の終了時のように、クライアントが動作していることを示す信号をサーバが受信すると即座に、デフォルト値へリセットされます。このオプションは、4Dプラグインにより開始された時間のかかる処理を管理する際に便利です。

“タイムアウトしない”オプションを設定するには、＜値＞に 0 を渡します。（例題2を参照）

■ セレクタ = 14（4D Client Timeout）

説明：この引数を使用して、4D Clientのタイムアウトの値を変更することができます。4D Clientのタイムアウトのデフォルト値は、クライアント側の「データベースプロパティ」ダイアログボックスの「接続設定」ページで定義します。

このセレクタに関する詳細は、「4D Server Timeout」の説明（13）を参照してください。

4D Clientのタイムアウトは、非常に特殊な状況において変更されます。

■ セレクタ = 15（Port ID）

説明：この引数を使用して、4th Dimensionおよび4D Serverによる4D Webサーバが使用するTCPポートのIDをオンザフライで変更することができます。デフォルト値は80で、この値は「データベースプロパティ」ダイアログボックスの「Web サーバI」ページで設定することができます。

セレクタ「Port ID」は、コンパイルして4D Runtimeを組み込んだ4D Webサーバで役立ちます（この場合、「デザイン」モードへのアクセス手段がありません）。TCPポートIDに関する詳細は、「Webサービス：システム設定」の節を参照してください。

■ セレクタ = 16（IP Address to listen）

説明：この引数を使用して、4th Dimensionおよび4D Serverによる4D Webサーバが HTTPリクエストを受信するIPアドレスをユーザがオンザフライで変更することができます。デフォルトでは、特定のアドレスは定義されていません（＜値＞=0）。この引数は「データベースプロパティ」ダイアログボックスの「Webサーバ I」ページで設定することができます。

セレクタ「IP Address to listen」は、コンパイルして4D Runtimeを組み込んだ4D Webサーバで役立ちます（この場合、「デザイン」モードへのアクセス手段がありません）。

引数＜値＞には、16進数のIPアドレスを渡します。つまり、“a.b.c.d”のようなIPアドレスを指定するには、以下のようなコードを作成します。

```
C_LONGINT($addr)
$addr:=($a<<24)|($b<<16)|($c<<8)|$d
SET DATABASE PARAMETER(IP Address to listen;$addr)
```

例題3も参照してください。IPアドレスの設定方法に関する詳細は「Webサービス：Webサーバセッティング」の節を参照してください。

■ セレクタ = 17（Character set）

値：

0：Western European（西ヨーロッパ）

1：Japanese（日本語）

2：Chinese（中国語）

3：Korean（韓国語）

4：User-defined（ユーザ定義）

5：Reserved（予備）

6：Central European（中央ヨーロッパ）

7：Cyrillic（キリル文字）

8：Arabic（アラビア語）

9：Greek（ギリシャ語）

10：Hebrew（ヘブライ語）

11：Turkish（トルコ語）

12：Baltic（バルト語）

説明：この引数を使用して、ユーザはデータベースに接続しているブラウザとの通信に、4D Webサーバ（4th Dimensionならびに4D Serverを使用）が使用する文字セットをオンザフライで変更することができます。実際のところ、デフォルト値はOSの言語に依存します。

この引数は「データベースプロパティ」ダイアログボックスの「WebサーバII」ページで設定することができます。セレクタ「Character set」は、コンパイルして4D Runtimeを組み込んだ4D Webサーバで役立ちます（この場合、「デザイン」モードへのアクセス手段がありません）。

■ セレクタ = 18（Max Concurrent Web Processes）

値：デフォルト値は32,000ですが、10から32,000までの任意の値を渡すことができます。

説明：この引数を使用して、4th Dimensionならびに4D Serverを用いた4D Webサーバでサポートされる任意のタイプの同時Webプロセス上限数（コンテキスト、非コンテキスト、または“プロセス再利用”に属するプロセス－セレクタ7、「Webプロセスの最大数」を参照）を正確に設定することができます。この上限数（マイナス1）に達した場合、4Dはそれ以上プロセスを作成しなくなり、HTTPステータス503（「Service Unavailable to all new requests」すべての新しいリクエストへのサービス不可）を返します。

この引数により、同時に行われる非常に膨大な数のリクエストやコンテキスト作成に関する過大な要求の結果として、サーバが飽和状態になることを防げます。また、この引数は「データベースプロパティ」ダイアログボックスでも設定することができます（「Webサービス：Webサーバセッティング」の節を参照）。

理論上、Webプロセスの最大数は次の計算式の結果になります。使用可能メモリまたはWebプロセスのスタックサイズ。別の解決策は、ランタイムエクスプローラに表示されるWebプロセス情報を示す方法です。つまり現在のWebプロセス数およびWebサーバの開始以降に達した最大数が示されている情報です。

注：“プロセスの再利用”の上限数より小さい値を渡した場合、この上限数はセレクタ 18の値に合わせるために減らされます。必要であれば、再利用の下限数（セレクタ6、Webプロセスの最小数）も変更できます。

■ セレクタ= 19(Client Minimum process Web)

■ セレクタ= 20(Client Maximum process Web)

■ セレクタ= 21(Client Max Web requests size)

■ セレクタ= 22(Client Port ID)

■ セレクタ= 23(Client IP Address to listen)

■ セレクタ= 24(Client Character set)

■ セレクタ = 25（Client Max Concurrent Web Proc）

値：4th Dimensionや4D Serverの対応するセレクタと同じ（セレクタ6から8、15から18、および27を参照）

説明：これらのセレクタを使用して、Webサーバとして使用する4D Clientマシンの操作用パラメータを指定することができます。

これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの「環境設定」ダイアログボックスを使用してください。

■ セレクタ= 26（Cache writing mode）

値：0または1（0=無効、1=有効）

説明：最適化されたキャッシュ書き込みモードを有効、または無効に設定します。デフォルトでは、この値は1になります（有効モード）。

バージョン2003の4th Dimensionより、最適化されたキャッシュ書き込みモードはデフォルトとして有効に設定され、特にMac OSにおいて4Dアプリケーションの処理速度が著しく向上します。

■ セレクタ= 27（Maximum Web requests size）

値：500,000から2,147,483,648

説明：Webサーバが処理を許可された受信HTTPリクエスト（POST）の最大数（バイト単位）。デフォルトでは、この値は2,000,000になります。最大値（2,147,483,648）を渡すと、実際上は制限がなくなります。

この制限を使用し、受信するリクエストが多すぎるためにWebサービスが限界に達してしまう危険性を回避します。リクエストがこの制限に達すると、4D Webサービスはリクエストを拒否します。

■ セレクタ=28（4D Server Log Recording）

値：0または 1 （0=記録しない、1=記録する）

説明：4D Serverが受け取る標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）に設定されます。サーバマシンから送られた各リクエストは、ログファイルに記録されます。このメカニズムが有効になると、データベースストラクチャと同じ階層にログファイルが作成されます。ファイルには"4DRequestsLog"で始まる名前が付けられます。このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストのサイズ、プロセスにかかる時間など）がシンプルな表形式のフォーマットで保存されます。この情報は、アプリケーションの微調節を行う場合や統計の目的で利用する場合に特に役立ちます。この情報を、例えばスプレッドシートソフトウェアに読み込んで処理することもできます。

ファイルのサイズが急速に増加してしまう恐れがあるので、このオプションは限られた時間内でのみ有効にすることをお薦めします。

■ セレクタ=29（Web Log Recording）

値：0または1（0=記録しない、1=記録する）

説明：4th Dimensionまたは4D ServerのWebサーバが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0（リクエストを記録しない）に設定されます。Webリクエストの記録は"logweb.txt"という名前のテキストファイルに保存されます。このファイルは自動的にデータベースストラクチャファイルと同じ階層に置かれます。このファイルはCLF（Common LogFile Format：共通ログフォーマット）またはNCSAのフォーマットで保存されます。これは、ほとんどのWeb解析ツールで認識することができます。

このファイルは、4th Dimensionの環境設定内の"Web/詳細"ページからも有効にすることができます。

■ セレクタ=30（Client Web Log Recording）

値：0または1（0=記録しない、1=記録する）

説明：クライアントマシンのWebサーバが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。

このセレクタの動作はセレクタ29と同一ですが、Webサーバとして使用されるすべての4D Clientマシンに適用されます。特定のクライアントマシンのみに値を設定したい場合には、4D Clientの環境設定ダイアログボックスで設定を行ってください。

■ セレクタ=31（Table Sequence Number）

値：すべての倍長整数

説明：このセレクタは、引数としてテーブルのレコードに代入されたカレント番号（Current unique number）を変更または取得するために使用します。"カレント番号"とは"最後に使われた番号"を意味します。**SET DATABASE PARAMETER**コマンドを使用してこの値を変更すると、渡された値+1からなる番号で次のレコードが作成されます（この新しい番号は、**Sequence number**関数によって返されます）。

デフォルトでは、4th Dimensionがこの固有の番号を設定し、レコードの作成順に返します。詳細は、**Sequence number**関数のドキュメントを参照してください。

■ セレクタ=32（Real Display Precision）

値：すべての正数の倍長整数

説明：このセレクタを使用すると、実際の画面表示のアルゴリズムに基づき末端何桁かが切り捨てられた有効数字以外の数の変更または取得が可能になります。この値は、カレントアプリケーションおよびセッションに合わせて設定されます。

デフォルトでは、このオプションの値は、4です。0の値は、デフォルト値が使用され、セッション中にパラメータが修正されなかったことを示します。

歴史的理由から、4th Dimensionは実数を10バイトに格納しますが、処理中にこの数字を8バイトに変換します（「実数の表示について」の節を参照）。この処理は完全に透過的で計算にも影響しません。ただし、特定の計算結果が予期した通りには表示されない場合があります。例えば、4.1-4.09の計算で表示される結果は0.009999999999999780000ですが、0.01を検索する際には正しい値が検索されます。

4Dは次のような方法で実数を表示します。

例えば、計算から得られた値が8.97499999999996158だとします（期待する値は通常8.975）。

アルゴリズムに基づき、デフォルトでは末尾4桁（6158）を切り捨てた後、残った最後の数字が0もしくは9でないかを確認します。0の場合、アルゴリズムにより先頭の0以下すべての端数が切り捨てられます。値が9の場合、先頭の9の前の位の数字を繰り上げます。

例えば、この例題では、8.97499999999996158が8.975になります。

例えば8.97499999999986158のような結果が得られる場合もあります。この場合、末尾の数字（最後の4桁を切り捨てた後）が0でも9でもないため、値は正しく四捨五入されません。

特定のデータベースの特徴に合わせて、精度アルゴリズムを使用し四捨五入をする数の桁数を調整したいとします。この場合は、カスタムの値を渡してください。0（4Dの内部で選択される値）を除く数字は、精度アルゴリズムにより切り捨てられる桁数を表します。

この設定は数値の表示やその際の内部処理には影響しません。

例題

(1) シングルユーザ版の4Dを実行している場合、次のメソッドを使用して、スケジューラの値を定義することができます。

```
C_LONGINT($ticksbtwcalls;$maxticks;$minticks;$lparams)
If(Application type=4th Dimension) ` シングルユーザの4Dを使用
    $ticksbtwcalls:=12
    $maxticks:=20
    $minticks:=7
    $lparams:=($minticks<<16)|($maxticks<<8)|$ticksbtwcalls
    SET DATABASE PARAMETER(4th Dimension scheduler;$lparams)
End if
```

(2) 以下のコードでは、予期しないタイムアウトを回避しています。

```
`カレントプロセスに対してタイムアウトを3時間まで延長する
SET DATABASE PARAMETER(4D Server Timeout;-60*3)
`4Dの制御を受けずに、時間のかかる処理を実行する
```

```
...
WR PRINT MERGE(Area;3;0)
...
```

(3)IPアドレス192.193.194.195は、次のコードを使用して設定します。

```
SET DATABASE PARAMETER(IP Address to listen;0xC0C1C2C3)
```

参照

DISTINCT VALUES、GET database parameter、QUERY SELECTION

CREATE SUBRECORD

CREATE SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	新しいサブレコードを作成するサブテーブル

説明

CREATE SUBRECORDコマンドは、＜サブテーブル＞に対して新しいサブレコードを作成し、それをカレントサブレコードとします。新しいサブレコードは、親レコードが保存された場合にのみディスクに保存されます。親レコードは、**SAVE RECORD**コマンドのようなコマンドを実行するか、ユーザがレコードを登録すると保存されます。カレント親レコードが存在しない場合には、**CREATE SUBRECORD**コマンドは何も行いません。サブレコード入力フォームを使いサブレコードを追加する場合には**ADD SUBRECORD**コマンドを使用します。

例題

▼以下の例は、ボタンのオブジェクトメソッドです。ボタンがクリックされると、このオブジェクトメソッドを実行し、[従業員]テーブルの子供に対し新しいサブレコードを作成します。一度に複数の子供を追加できるように、**Repeat**ループで「キャンセル」ボタンがクリックされるまで繰り返すように制御しています。フォームに、サブフォームを使用して複数の子供を表示することができますが、「入力可」オプションをオフに設定しているため、サブテーブルに直接データを入力することはできません。

```
` 「キャンセル」ボタンがクリックされるまで繰り返す
Repeat
    v子供:=Request("子供の名前を入力してください。")
` 子供の名前を得る
    If(OK=1)      ` 実行？
        CREATE SUBRECORD([従業員]子供) ` 子供の新しいサブレコード作成
        [従業員]子供'名前:=v子供    ` 子供の名前をサブフィールドへ代入
```

```
            End if
        Until(OK=0)
```

参照

ADD SUBRECORD、DELETE SUBRECORD、SAVE RECORD

DELETE SUBRECORD

DELETE SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	カレントサブレコードを削除するサブテーブル

説明

DELETE SUBRECORDコマンドは、＜サブテーブル＞のカレントサブレコードを削除します。カレントサブレコードが存在しない場合には、**DELETE SUBRECORD**コマンドは何も行いません。サブレコードが削除されるとサブテーブルのカレントサブセレクションは空になります。そのため、**DELETE SUBRECORD**コマンドは、サブセレクション内のサブレコードに対して削除処理を繰り返し実行することはできません。

サブレコードの削除は、親レコードが保存されるまでは恒久的なものではありません。親レコードを削除した場合は、そのレコードに属するサブレコードはすべて削除されます。

サブセレクションを削除するには、削除しようとするサブセレクションを作成し、最初のサブレコードを削除し、次にまたサブセレクションを作成して最初のサブレコードを削除します。この作業を繰り返します。

例題

1. 以下の例は、サブテーブルのすべてのサブフィールドを削除します。

```
ALL SUBRECORDS([従業員]子供)
While(Records in subselection([従業員]子供) > 0)
DELETE SUBRECORD([従業員]子供)
        ALL SUBRECORDS([従業員]子供)
End while
```

2. 以下の例は、[子供]サブテーブルから年齢が12歳以上の子供のサブレコードをすべて削除します。

```
ALL RECORDS([従業員]) ` すべてのレコードをカレントセレクションにする
For($i ; 1 ; Records in selection([従業員])) ` 全セレクションに対して
        ` 検索条件に沿ってサブレコードを検索する
    QUERY SUBRECORDS([従業員]子供 ; [従業員]子供'年齢 >= 12) `
    ` レコード間をループする
    While(Records in subselection([従業員]子供) > 0)
        DELETE SUBRECORD([従業員]子供) ` サブレコードを削除する
        ` 以下のレコードを検索する
        QUERY SUBRECORDS([従業員]子供 ; [従業員]子供'年齢 >= 12)
    End while
    SAVE RECORD([従業員])    ` 親レコードを保存する
    NEXT RECORD([従業員])
End for
```

参照

ALL SUBRECORDS、QUERY SUBRECORDS、Records in subselection、SAVE RECORD

ALL SUBRECORDS

ALL SUBRECORDS(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	カレントサブレセレクションを作成するサブテーブル

説明

ALL SUBRECORDSコマンドは、<サブテーブル>のすべてのサブレコードをカレントサブセレクションにします。カレント親レコードが存在しない場合には、**ALL SUBRECORDS**コマンドは何も行いません。初めて親レコードがロードされた時は、サブセレクションはべてのサブレコードを含みますが、**ADD SUBRECORDS**、**QUERY SUBRECORDS**、**DELETE SUBRECORD**等の各コマンドを実行した後では、サブセレクションがすべてのサブレコードを含むとは限りません。

例題

▼以下の例は、すべてのサブレコードを選択し、その金額の合計を算出します。

```
ALL SUBRECORDS([地域]売上)
売上合計:=Sum([地域]売上'金額)
```

参照

QUERY SUBRECORDS、Records in subselection

Records in subselection

Records in subselection(サブテーブル) → 数値

引数	タイプ		説明
サブテーブル	サブテーブル	→	サブレコードの数を得るサブテーブル
戻り値	数値	←	カレントサブセレクションのサブレコード数

説明

Records in subselection関数は、＜サブテーブル＞のカレントサブセレクションに含まれるサブレコードの数を返します。**Records in subselection**関数は、カレントレコードのカレントサブセレクションに対してのみ適用されます。この関数は、**Records in selection**関数のサブテーブル版といえます。カレント親レコードが存在しない場合には、値として未定義（Undefined）を返します。

例題

▼以下の例は、すべてのサブレコードを選択してから、親レコードに対する子供の数を表示します。

```
ALL SUBRECORDS([従業員]子供)｀すべてのサブレコードを選択
ALERT("子供の数：" + String(Records in subselection([従業員]子供)))
```

参照

ALL SUBRECORDS、QUERY SUBRECORDS

APPLY TO SUBSELECTION

APPLY TO SUBSELECTION(サブテーブル ; ステートメント)

引数	タイプ		説明
サブテーブル	サブテーブル	→	フォーミュラで更新するサブテーブル
ステートメント	ステートメント	→	1行のステートメントまたはメソッド

説明

APPLY TO SUBSELECTIONコマンドは、＜サブテーブル＞のカレントサブセレクションのすべてのサブレコードに対して＜ステートメント＞を適用します。この＜ステートメント＞は、1行のステートメントまたはメソッドを指定します。ステートメントでサブレコードを修正した場合、その結果は親レコードが保存された時にのみディスクに保存されます。サブセレクションクションが空の場合は、**APPLY TO SUBSELECTION**コマンドは何も行いません。

APPLY TO SUBSELECTIONコマンドは、サブセレクションから情報を収集する、またはサブセレクションを修正するために使用します。

例題

▼以下の例は、すべてのサブレコードに対し単価と数量から売上金額を算出します。

```
ALL SUBRECORDS([請求書]明細) ` すべてのサブレコードを選択
APPLY TO SUBSELECTION([請求書]明細 ;[請求書]明細'売上金額 :=
                                    [請求書]明細'単価 * [請求書]明細'数量)
```

注：ページの都合上、ステートメントが複数行にわたって記述されています。

参照

ALL SUBRECORDS、QUERY SUBRECORDS、SAVE RECORD

FIRST SUBRECORD

FIRST SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	先頭サブレコードに移動するサブテーブル

説明

FIRST SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの先頭のサブレコードをカレントサブレコードにします。またすべての検索コマンド、選択コマンド、ソートコマンドも先頭のサブレコードをカレントサブレコードに設定します。カレントサブセレクションが空の場合は、**FIRST SUBRECORD**コマンドは何も行いません。

例題

▼以下の例は、サブテーブルに格納されている子供の名字と名前を連結します。連結した名前を“子供の名前”という配列にコピーします。

```
      ` 名前を保存する配列を宣言する
ARRAY TEXT(子供の名前 ; Records in subselection([従業員]子供))
FIRST SUBRECORD([従業員]子供)      ` 先頭のサブレコードへ移動
      ` 子供の数だけ繰り返す
For($i ; 1 ; Records in subselection([従業員]子供))
      子供の名前{$i}:=[従業員]子供'名字 + " " + [従業員]子供'名前
      NEXT SUBRECORD([従業員]子供)
End for
```

参照

LAST SUBRECORD、NEXT SUBRECORD、PREVIOUS SUBRECORD

LAST SUBRECORD

LAST SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	最終サブレコードに移動するサブテーブル

説明

LAST SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの最終のサブレコードをカレントサブレコードにします。カレントサブセレクションが空の場合は、**LAST SUBRECORD**コマンドは何も行いません。

例題

▼以下の例は、サブテーブルに格納されている子供の名字と名前を連結します。連結した名前を“子供の名前”という配列にコピーします。最終のサブレコードから先頭のサブレコードへ処理していく以外は、**FIRST SUBRECORD**コマンドの例と同じです。

```
        ` 名前を保存する配列を宣言する
ARRAY TEXT(子供の名前 ; Records in subselection([従業員]子供))
LAST SUBRECORD([従業員]子供)` 最終のサブレコードへ移動
        ` 子供の数だけ繰り返す
For($i ; 1 ; Records in subselection([従業員]子供))
        子供の名前{$i}:=[従業員]子供'名字 + " " + [従業員]子供'名前
        PREVIOUS SUBRECORD([従業員]子供)
End for
```

参照

FIRST SUBRECORD、NEXT SUBRECORD、PREVIOUS SUBRECORD

NEXT SUBRECORD

NEXT SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	以下のサブレコードに移動するサブテーブル

説明

NEXT SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの1つ以下のサブレコードにカレントサブレコードポインタを移動します。**NEXT SUBRECORD**コマンドが、最終のサブレコードを過ぎてポインタを移動した場合には、**End subselection**関数に“True（真）”を設定し、カレントサブレコードはなくなります。**End subselection**関数“True（真）”を返した場合には、ポインタをカレントサブレコードに戻すために、**FIRST SUBRECORD**コマンドまたは**LAST SUBRECORD**コマンドを使用します。カレントサブセレクションが空の場合や**Before subselection**関数が“True（真）”を返した場合には、**NEXT SUBRECORD**コマンドは何も行いません。

FIRST SUBRECORDコマンドについては、前節の例を参照してください。

参照

FIRST SUBRECORD、LAST SUBRECORD、PREVIOUS SUBRECORD

PREVIOUS SUBRECORD

PREVIOUS SUBRECORD(サブテーブル)

引数	タイプ		説明
サブテーブル	サブテーブル	→	前のサブレコードに移動するサブテーブル

説明

PREVIOUS SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの1つ前のサブレコードにカレントサブレコードポインタを移動します。**PREVIOUS SUBRECORD**コマンドが先頭のサブレコードを過ぎてポインタを移動した場合には、**Before subselection**関数に“True（真）”を設定し、カレントサブレコードはなくなります。**Before subselection**関数が“True（真）”を返した場合は、ポインタをカレントサブレコードに戻すために、**FIRST SUBRECORD**コマンドや**LAST SUBRECORD**コマンドを使用します。カレントサブセレクションが空の場合や**End subselection**関数が“True（真）”を返した場合には、**PREVIOUS SUBRECORD**コマンドは何も行いません。

LAST SUBRECORDコマンドについては、前節の例を参照してください。

参照

FIRST SUBRECORD、LAST SUBRECORD、NEXT SUBRECORD

Before subselection

Before subselection(サブテーブル) → ブール

引数	タイプ		説明
サブテーブル	サブテーブル	→	ポインタがセレクションの前にあるか判定するサブテーブル
戻り値	ブール	←	True：Yes、False：No

説明

Before subselection関数は、＜サブテーブル＞のカレントサブレコードポインタが先頭のサブレコードよりも前にあるときに“True（真）”を返します。**Before subselection**関数は、**PREVIOUS SUBRECORD**コマンドでポインタを移動するときに、ポインタが先頭のサブレコードを過ぎたかどうかを調べるために使用します。カレントサブセレクションが空の場合にも、**Before subselection**関数は“True（真）”を返します。

例題

▼以下の例は、ボタンに対するオブジェクトメソッドです。ボタンがクリックされるとポインタが前のサブレコードへ移動します。ポインタが先頭のサブレコードの前を指した場合には、最終のサブレコードへ移動します。

```
PREVIOUS SUBRECORD([従業員]子供) ` 前のサブレコードへ移動
If(Before subselection([従業員]子供)) ` サブレコードの終わり？
    LAST SUBRECORD([従業員]子供) ` 最終のサブレコードへ移動
End if
```

参照

PREVIOUS SUBRECORD

End subselection

End subselection(サブテーブル) → ブール

引数	タイプ		説明
サブテーブル	サブテーブル	→	ポインタがセレクションの後にあるかを判定するサブテーブル
戻り値	ブール	←	True：Yes、False：No

説明

End subselection関数は、サブテーブルのカレントサブレコードポインタが最終のサブレコードよりも後にあるときに“True（真）”を返します。**End subselection**関数は、**NEXT SUBRECORD**コマンドでポインタを移動するときに、ポインタが最終のサブレコードを過ぎたかどうかを調べるに使用します。カレントサブセレクションが空の場合にも、**End subselection**関数は“True（真）”を返します。

例題

▼以下の例は、ボタンに対するオブジェクトメソッドです。ボタンがクリックされるとポインタが以下のサブレコードへ移動します。ポインタが最終のサブレコードの後を指した場合には、先頭のサブレコードへ移動します。

```
NEXT SUBRECORD([従業員]子供) ` 以下のサブレコードへ移動
If(End subselection([従業員]子供)) ` サブレコードの終わり？
    FIRST SUBRECORD([従業員]子供) ` 先頭のサブレコードへ移動
End if
```

参照

NEXT SUBRECORD

ORDER SUBRECORD BY

ORDER SUBRECORD BY(サブテーブル;サブフィールド1{;ソート種別1}
{; ... ;サブフィールドN;{ソート種別N}})

引数	タイプ		説明
サブテーブル	サブテーブル	→	ソート対象となるサブテーブル
サブフィールド	サブフィールド	→	各レベルのソートキーとなるサブフィールド
ソート種別	>または<	→	各レベルのソート順 >昇順、<降順

説明

ORDER SUBRECORD BYコマンドは、＜サブテーブル＞のカレントサブセレクションをソートします。このコマンドは、カレント親レコードに属するサブテーブルのサブセレクションのみをソートします。

＜ソート種別＞で昇順または降順にソートするかを指定します。＜ソート種別＞が（>）の場合は、ソートを昇順に行います。＜ソート種別＞が（<）の場合には、ソートを降順に行います。＜ソート種別＞を省略した場合には、ソートを昇順に行います。

1つのステートメント内に複数のソートキーとなるサブフィールドとソート種別を指定して、複数レベルのソートを実行できます。

ソート処理が完了した時点で、ソートしたサブセレクションの先頭のレコードをカレントサブレコードとします。サブレコードのソートは、一時的な処理です。サブレコードはソートした状態で保存されることはありません。カレントレコードが存在しない場合や上位レベルのサブレコードが存在しない場合は、**ORDER SUBRECORD BY**コマンドは何も行いません。

フォームが固定フレームで印刷されるサブフォームを含んでいる場合、このコマンドは親フォームのフォームメソッドのBeforeフェーズで印刷前に一度呼ばれる必要があります。

例題

▼ 以下の例は、金額サブフィールドをソートキーにして[地域]売上サブテーブルを昇順にソートします。

```
ORDER SUBRECORD BY([地域]売上 ; [地域]売上'金額 ; >)
```

参照

QUERY SUBRECORDS

QUERY SUBRECORDS

QUERY SUBRECORDS(サブテーブル ; 検索条件式)

引数	タイプ		説明
サブテーブル	サブテーブル	→	検索対象となるサブテーブル
検索条件式	ブール	→	検索条件式

説明

QUERY SUBRECORDSコマンドは、＜サブテーブル＞を検索し、新しいサブセレクションを生成します。このコマンドは、サブレコードを検索してサブレコードセレクションを返す機能を持つ唯一のコマンドです。＜検索条件式＞は、サブテーブルの各サブレコードに適用されます。＜検索条件式＞は、“True（真）”または“False（偽）”のいずれかの状態に評価されるブール式です。＜検索条件式＞で“True（真）”に評価されたサブレコードは、新しいサブセレクションに追加されます。検索処理が完了した時点で、**QUERY SUBRECORDS**コマンドは先頭のサブレコードを＜サブテーブル＞のカレントサブレコードとします。

QUERY SUBRECORDSコマンドは、現在選択されている親レコードに属すサブテーブルのサブレコードのみを検索し、親テーブルのレコードに関連するすべてのサブレコードを検索するわけではありません。**QUERY SUBRECORDS**コマンドは、カレント親レコードを変更しません。

＜検索条件式＞は、サブフィールドに対し比較演算子を使用して変数または定数との比較を行います。＜検索条件式＞同士は、論理演算子“かつ（&）”または論理演算子“または（¦）”を使用して結合することができます。また、＜検索条件式＞が関数であったり、または関数を含むこともできます。ワイルドカード記号（@）は、文字列を検索する場合にのみ使用できます。

カレントレコードが存在しない場合または上位のサブレコードが存在しない場合は、**QUERY SUBRECORDS**コマンドは何も行いません。

例題

▼ 以下の例は、10歳以上の子供を検索します。

```
QUERY SUBRECORDS([従業員]子供 ; [従業員]子供'年齢 > 10)
```

参照

ALL SUBRECORDS、ORDER SUBRECORDS BY、Records in subselection

システムドキュメントの概要

コンピュータで使用するすべてのドキュメントやアプリケーションは、コンピュータに接続、またはマウントされているハードディスク、フロッピーディスク、あるいはその他の記憶装置にファイルとして格納されています。4th Dimension内では、このようなドキュメントを区別せずにファイルまたはドキュメントと呼ぶことができます。ただし、この章のほとんどのコマンドは（アプリケーションやシステムファイルではなく）ディスク上にあるドキュメントにアクセスする目的で使用するため、ここでは、“ドキュメント”という用語を使用しています。

1つのハードディスクは、1つまたは複数のパーティションとしてフォーマットすることができます。個々のパーティションは、ボリュームと呼ばれます。2つのボリュームが同一のハードディスクに物理的に存在するかどうかに関係なく、4Dのレベルでは、通常、これら2つのボリュームを2つの同等なエンティティ（存在物）として扱います。

ボリュームは、コンピュータに物理的に接続されたり、NetBEUI（Windows）やAFP（Macintosh）のようなファイル共有プロトコルを使用してネットワーク経由でマウントされているハードディスク上にあります。どのような場合でも、4Dのレベルでは、システムドキュメントコマンドを使用する場合には、これらのボリュームを同等に扱います（方法について熟知しており、4Dのプラグインソフトウェアを使用して、そのドメインのアプリケーションの機能を拡張する場合は例外です）。

個々のボリュームには、ボリューム名があります。Windowsでは、ボリュームは文字の後にコロンを続けて表します。通常、A:およびB:が、5.25インチまたは3.5インチのフロッピーディスクドライブを表します。通常、C:は、システムをブートするために使用するボリュームを表します（コンピュータを他の方法で構成している場合は例外です）。したがって、D:～Z:までの文字を使用して、コンピュータに接続またはマウントされるボリューム（CD-ROMドライブ、追加ドライブ、ネットワークドライブ等）を表します。Macintoshでは、ボリュームには任意の名前を付けることができ、最長で31文字になります（Finderのレベルでデスクトップに表示される名前です）。

通常、ドキュメントはフォルダに分類します。フォルダには、任意のフォルダを含むこともできます。何百個あるいは何千個ものファイルをボリュームの同一のレベルに蓄積するのはあまりよい方法ではありません。まず、見栄えが悪く、次にシステムの速度を低下させてしまいます。Windowsでは、フォルダはディレクトリと呼ばれます（呼ばれていました）。Windows 95の登場以来、フォルダという用語が使用されています。Macintoshでは通常、フォルダと呼ばれます。

ドキュメントをユニークに識別するには、ドキュメント自体の名前の他に、そのドキュメントがあるボリュームの名前およびフォルダの名前も知っている必要があります。これらの名前をすべて連結すれば、そのドキュメントへのパス名になります。このパス名の中では、フォルダ名はディレクトリ（区切り）記号と呼ばれる特殊文字で区切られています。Windowsでは、この文字は円記号（¥）であり、Macintoshではコロン（:）です。

例で検討してみましょう。「Memo」というフォルダに「Important Memo」というドキュメントが入っており、Memo自体も「Document」というフォルダに、さらにDocumentは「Current Work」というフォルダに入っています。

Windowsでは、これらすべてが「C:」というドライブ（ボリューム）にある場合には、このドキュメントへのパス名は、以下のようになります。

C:¥Current Work¥Documents¥Memos¥Important Memo.TXT

Macintoshでは、これらすべてが「Internal Drive」というディスク（ボリューム）にある場合には、このドキュメントへのパス名は、以下のようになります。

Internal Drive:Current Work:Documents:Memos:Important Memo

Windowsでは、この例の場合、ドキュメントの名前にファイル拡張子「.TXT」がついていることに注意してください。理由はこの後で説明します。

プラットフォームが何であっても、ドキュメントへのパス名は、正式には以下のように表現できます。

ボリューム名 ディレクトリ記号 { ディレクトリ名 ディレクトリ記号 { ディレクトリ名 ディレクトリ記号 { ... } } } ドキュメント名

ボリュームにあるすべてのドキュメント（ファイル）には、通常、属性またはプロパティと呼ばれる複数の特性があります。つまり、ドキュメント自体の名前（Macintoshでは31文字以内、Windows 3.1では8文字以内、Windows 95およびWindows NTでは255文字以内）、タイプ、そしてクリエータです。

ドキュメントファイルのタイプとクリエータ

Windowsでは、ドキュメントにはタイプがあります。Macintoshでは、ドキュメントにはタイプとクリエータがあります。ドキュメントのタイプは、通常、ドキュメントの種類または内容を表します。例えば、テキストドキュメントにはテキストが含まれます（スタイル等は含まれません）。

Windowsでは、ドキュメントのタイプは、そのドキュメントの名前に追加されたファイル拡張子と呼ばれる接尾辞によって判断されます。例えば、「.TXT」はテキストドキュメントに対するWindowsでのファイル拡張子です。Macintoshでは、ドキュメントのタイプは、4文字の識別子（Finderのレベルでは表示されません）であるドキュメントのファイルタイププロパティによって判断されます。例えば、テキストドキュメントのファイルタイプは「TEXT」です。

さらにMacintoshでは、ドキュメントには、そのドキュメントを作成したアプリケーションを表すクリエータがあります。この概念は、Windowsにはありません。ドキュメントのクリエータは、4文字の識別子（ファインダのレベルでは表示されません）であるドキュメントのファイルクリエータプロパティによって判断されます。例えば、4Dバージョン6で作成されたドキュメントのファイルクリエータは「4D06」です。

DocRef：ドキュメント参照番号

ドキュメントは開く、または閉じることができます。内蔵の4Dコマンドを使用すると、ドキュメントは一度に1つのプロセスによってのみ開くことができます。1つのプロセスで複数のドキュメントを開くことができ、複数のプロセスではより多くのドキュメントを開くことができますが、同一のドキュメントを一度に2つ開くことはできません。

Open document関数、**Create document**関数、**Append document**関数を使用すると、ドキュメントが開きます。ドキュメントが開くと、そのドキュメントから文字を読み取ったり、文字を書き込むことができます（**RECEIVE PACKET**コマンドおよび**SEND PACKET**コマンドを参照してください）。ドキュメントでの作業を終了したら、通常、**CLOSE DOCUMENT**コマンドを使用してドキュメントを閉じます。

開いているすべてのドキュメントは、**Open document**関数、**Create document**関数、**Append document**関数で返されるDocRef式を使用して参照されます。DocRefは、開いているドキュメントをユニークに識別します。これは、正式には時間タイプの式です。開いているドキュメントを操作するすべてのコマンドでは、DocRefが引数（パラメータ）として必要です。このようなコマンドに間違ったDocRefを渡すと、ファイルマネージャエラーが発生します。

I／Oエラーの処理

ドキュメントにアクセスする（開く、閉じる、削除する、名前を変更する、コピーする）、ドキュメントのプロパティを変更する、あるいはドキュメントで文字を読み取る、または書き込むと、I／Oエラーが発生する場合があります。ドキュメントが見つからなかったり、ロックされていたり、既に開かれている場合等です。このようなエラーは、**ON ERR CALL**コマンドでインストールされているエラー処理メソッドを使用して検出することができます。システムドキュメントを使用している時に最も発生しやすいエラーについては、「OSファイルマネージャエラー」の節で説明しています。

Documentシステム変数

Open document関数、**Create document**関数、**Append document**関数、**Select document**関数を使用すると、標準の「ファイルを開く」または「ファイル保存」ダイアログボックスを用いてドキュメントにアクセスすることができます。

標準のダイアログを使用してドキュメントにアクセスすると、4DはDocumentシステム変数にそのドキュメントへの完全なパス名を返します。このシステム変数は、コマンドの引数リストに表示される引数＜ドキュメント＞とは区別する必要があります。

ドキュメント名またはドキュメントパス名の指定

この節内の、ドキュメント名が必要なほとんどのルーチンでは、ドキュメントの名前およびドキュメントへのパス名の両方を使用できます（その他の方法で指定された場合を除く）。名前を渡すと、コマンドはデータベースのフォルダ内にあるドキュメントを検索します。パス名を渡す場合は、パス名が有効なことを確認してください。

誤った名前やパス名を渡すとファイルマネージャエラーが発生します。これは、ON ERR CALLメソッドを使用して独自のエラー処理を記述することができます。

警告：引数＜ドキュメント＞の最大長は255文字です。これより長い文字数の名前を渡すと、余分な文字が切り捨てられ、ファイルマネージャーエラーが発生します。

Windowsでパス名およびエスケープシーケンスを利用する

4th Dimensionのメソッドエディタでは、エスケープシーケンスを利用することができます。エスケープシーケンスとは文字のセットで、"特殊"文字の代わりに使用することができます。このシーケンスは、¥（英語版ではバックスラッシュ＝日本語環境ではフォントの都合で¥通貨記号）で始まり後ろに文字が続きます。例えば、「¥t」はTabのエスケープシーケンスです。

また、Windows上では¥をパス名の区切り文字として使用することもあります。これは、メソッドエディタで¥を¥¥（ダブルバックスラッシュ）で置き換えて入力します。例えば、C:¥Folderの場合、C:¥¥Folderになります。

ただし、C:¥MyDocuments¥Newと記述した場合、4th DimensionはC:¥¥MyDocuments¥Newと表示します。この場合、2番目の¥が誤って¥N（既存のエスケープシーケンス）と解釈されたからです。このため、エスケープシーケンスとして認識されてしまうような文字¥の後ろに続く場合には、¥（バックスラッシュ）の代わりに¥¥（ダブルバックスラッシュ）を入力してください。

4th Dimension によって認識されるエスケープシーケンスは次の通りです。

エスケープシーケンス	代理の文字
¥n	LF（改行）
¥t	HT（水平タブ）
¥r	CR（キャリッジリターン）
¥¥	¥（バックスラッシュ）
¥"	"（引用符）

ディスク上にあるドキュメントを操作する場合に便利なプロジェクトメソッド

実行しているプラットフォームの検出

4th Dimensionには**MAP FILE TYPES**コマンドのような、プラットフォームの特殊性によるコーディングの変形を排除するためのコマンドがありますが、ディスク上のドキュメントを操作する際に下位レベルでの作業（プログラムによりパス名を取得するような）を開始すると、Macintoshプラットフォームまたは Windowsプラットフォームのどちらで動作しているのかを知る必要があります。

以下の「On Windows」プロジェクトメソッドは、データベースがWindows上で実行されているかどうかを示します。

```
` 「On windows」プロジェクトメソッド
` On windows → ブール
` On windows → Windowsの場合はTrue
C_BOOLEAN ($0)
C_LONGINT ($vlPlatform ; $vlSystem ; $vlMachine)
PLATFORM PROPERTIES ($vlPlatform ; $vlSystem ; $vlMachine)
$0:=($vlPlatform=Windows)
```

正しいディレクトリ区切り記号の使用

Windowsでは、ディレクトリレベルは円記号（¥）で表します。Macintoshでは、フォルダレベルはコロン（:）で表します。実行しているプラットフォームに従って、以下の「Directory symbol」プロジェクトメソッドでは、正しいディレクトリシンボル（文字）のASCIIコードを返します。

```
` 「Directory symbol」プロジェクトメソッド
` Directory symbol → 整数
` Directory symbol → "/"(Windows) または":"(Macintosh)文字のASCIIコード
C_INTEGER ($0)
If (On Windows)
	$0:=Ascii ("¥¥")
Else
	$0:=Ascii (":")
End if
```

ロングネームからのファイル名の抽出

ドキュメントへのロングネーム（パス名+ファイル名）を取得した後では、例えば、ウインドウのタイトルにドキュメントのファイル名を表示するために、そのロングネームからファイル名を抽出することが必要となる場合があります。「Long name to file name」プロジェクトメソッドは、この処理をWindowsおよびMacintoshの両方で実行します。

```
` 「Long name to file name 」プロジェクトメソッド
` Long name to file name (文字列) → 文字列
` Long name to file name (長いファイル名) → ファイル名
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) `On Startupデータベースメソッドではこの変数をTrueまたは
                                                    Falseに設定
        If ($0="")
                TRACE
        End if
End if
```

ロングネームからのパス名の抽出

ドキュメントへのロングネーム（パス名+ファイル名）を取得した後では、例えば、ドキュメントを追加して同じ位置に保存するために、そのロングネームからドキュメントが保存されているディレクトリへのパス名を抽出することが必要となる場合があります。「Long name to path name」プロジェクトメソッドは、この処理をWindowsおよびMacintoshの両方で実行します。

```
`「Long name to path name 」プロジェクトメソッド
` Long name to path name (文字列) → 文字列
` Long name to path name (長い名前) → パス名
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) `On Startupデータベースメソッドではこの変数をTrueまたは
                                                    Falseに設定
        If ($0="")
                TRACE
        End if
End if
```

参照

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

Document type

Document type (ドキュメント) → 文字列

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前
戻り値	文字列	←	Windowsファイル拡張子（1~3バイト）、または Macintoshファイルタイプ（4バイト）

説明

Document type関数は、引数＜ドキュメント＞に渡した名前またはパス名を持つドキュメントファイルのタイプを返します。

Windowsでは、**Document type**関数はドキュメントのファイル拡張子（つまりMicrosoft Wordの場合には"DOC"、実行可能ファイルの場合には"EXE"等）を返し、4th Dimensionによって（つまり"TXT"ファイル拡張子には"TEXT"）あるいは**MAP FILE TYPES**コマンドの呼び出しの前に、Windowsファイル拡張子に相当するMacintoshベースの4文字からなるファイルタイプにマップされていればこのようなファイルタイプを返します。

Macintoshでは、**Document type**関数は、4文字からなるドキュメントのファイルタイプ（つまりテキストドキュメントの場合には"TEXT"、ダブルクリック可能なアプリケーションの場合には"APPL"等）を返します。

参照

Document creator、GET DOCUMENT PROPERTIES、MAP FILE TYPES、SET DOCUMENT TYPE

SET DOCUMENT TYPE

SET DOCUMENT TYPE (ドキュメント ; ファイルタイプ)

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前 またはドキュメントファイルへの完全なパス名
ファイルタイプ	文字列	→	Windowsファイル拡張子（1~3バイト） Macintoshファイルタイプ（4バイト）

説明

SET DOCUMENT TYPEコマンドは、ドキュメントの名前またはパス名を引数<ドキュメント>に渡すドキュメントファイルのタイプを設定します。

ドキュメントの新しいタイプは、引数<ファイルタイプ>に渡します。

前述の「ドキュメントファイルのタイプとクリエータ」の節を参照してください。

Windowsでは、このコマンドはファイル拡張子を修正し、その結果<ドキュメント>の値が変わります。

```
SET DOCUMENT TYPE ("C:¥¥Docs¥¥Invoice.asc";"TEXT")
```

上記のコードを実行すると、“Invoice.asc”ファイルは“Invoice.txt”という名前に変わります。4Dでは、Macintosh の“TEXT”タイプはWindowsの“txt”タイプに相当します。

4Dに対応するタイプがない場合、拡張子を渡す必要があります。例えば、以下のコードは“Invoice.asc”ファイルを“Invoice.zip”という名前に変えます。

```
SET DOCUMENT TYPE ("C:¥¥Docs¥¥Invoice.asc";"zip")
```

参照

Document type、MAP FILE TYPES、SET DOCUMENT CREATOR、SET DOCUMENT PROPERTIES

Document creator

Document creator (ドキュメント) → 文字列

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前、または ドキュメントの完全なパス名
戻り値	文字列	←	空の文字列（Windows）、または ファイルクリエータ（Macintosh）

説明

Document creator関数は、引数<ドキュメント>に渡した名前またはパス名を持つドキュメントファイルのクリエータを返します。

Windowsでは、**Document creator**関数は空の文字列を返します。

参照

Document type、SET DOCUMENT CREATOR

SET DOCUMENT CREATOR

SET DOCUMENT CREATOR (ドキュメント ; ファイルクリエータ)

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前、または ドキュメントの完全なパス名
ファイルクリエータ	文字列	→	Macintoshファイルクリエータ（4バイト） または、空の文字列（Windows）

説明

SET DOCUMENT CREATORコマンドは、引数<ドキュメント>に渡した名前またはパス名を持つドキュメントファイルのクリエータを設定します。

ドキュメントの新しいクリエータは、引数<ファイルクリエータ>に渡します。

このコマンドは、Windowsでは無効です。

前述の「ドキュメントファイルのタイプとクリエータ」の節を参照してください。

参照

Document creator、SET DOCUMENT PROPERTIES、SET DOCUMENT TYPE

Open document

Open document (ドキュメント{; ファイルタイプ {;モード}}) → ドキュメントファイル参照番号

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントファイル名、または ドキュメントへの完全なパス名、または 空の文字列の場合、標準のファイル ダイアログボックス表示
ファイルタイプ	文字列	→	Macintoshファイルタイプ（4桁の文字）、 または Windowsファイル拡張子（1～3桁の文字）、 または省略した場合、 テキストドキュメント（.TXT）
モード	整数	→	ドキュメントを開くモード
戻り値	時間	←	ドキュメントファイル参照番号

説明

Open document関数は、＜ドキュメント＞に指定したドキュメント名またはパス名を持つドキュメントファイルを開きます。

＜ドキュメント＞に空の文字列（ヌル""）を指定した場合には、Windows版では「開く」（Macintosh版では、「ファイルを開く」）ダイアログボックスが表示されます。このダイアログをキャンセルすると、ドキュメントファイルは開かれず、**Open document**関数はドキュメントファイル参照番号にヌル値を返し、システム変数OKに0を代入します。

- ドキュメントが正しく開かれると、**Open document**関数はドキュメントファイル参照番号を返し、システム変数OKに1を代入します。
- ドキュメントが既に開かれており、引数＜モード＞が省略されている場合には、**Open document**関数はドキュメントをReadモードで開き、システム変数OKに1を代入します。
- ドキュメントが既に開かれており、Writeモードでそのドキュメントを開こうとした場合、エラーが発生します。
- ドキュメントが存在しなかったり、既に開かれている場合には、エラーが発生します。

Macintoshの場合、「ファイルオープン」ダイアログボックスにはデフォルトとしてすべてのドキュメントファイルが表示されます。別のタイプのドキュメントファイルを表示するには、オプション引数＜ファイルタイプ＞にドキュメントファイルのタイプを指定します。

Windowsの場合、「開く」ダイアログボックスにはデフォルトとしてすべてのドキュメントファイルタイプ（*.*）が表示されます。別のタイプのドキュメントファイルを表示するには、オプション引数＜ファイルタイプ＞に1から3文字のWindowsのファイル拡張子、または**MAP FILE TYPES**コマンドを使ってマップされるMacintoshのファイルタイプを指定します。

Windowsの場合、「開く」ダイアログボックスを使用しない場合でも、オプション引数＜ファイルタイプ＞に開こうとするドキュメントファイルのファイル拡張子を指定することがあります。デフォルトでは、**Open document**関数は.TXTという拡張子の付いたテキストドキュメントを開こうとします。＜ファイルタイプ＞を指定すると、**Open document**関数は“ドキュメント.ファイルタイプ”という名前のドキュメントファイルを開こうとします。

例えば、

```
vhDocRef:=Open document("C:¥¥Letter";"WRI")
```

という命令を実行すると、ディスク上の“C:¥Letter.WRI”というドキュメントファイルが開かれます。＜ファイルタイプ＞に3桁以上の文字を指定すると、**Open document**関数は最初の3桁だけを参照します。ドキュメントファイルのタイプを指定しないと、**Open document**関数はファイル拡張子のないドキュメントファイルを開こうとし、ファイルが見つからない場合、.TXTという拡張子を持つファイルを開こうとします。このファイルが見つからない場合、「ファイルが見つかりません。」というエラーを返します。

ドキュメントファイルが開かれると、**Open document**関数ははじめにファイルの位置をドキュメントの最初に設定しますが、**Append document**関数はドキュメントの終わりに設定します。

ドキュメントファイルを開いたら、**RECEIVE PACKET**や**SEND PACKET**コマンドを使用してドキュメントへの読み込みや書き込みを行うことができます。また、これらのコマンドと**Get file position**や**SET FILE POSITION**コマンドを組み合わせて、ドキュメントの一部に直接アクセスすることもできます。

オプションの引数モードは、ドキュメントがどのように開かれるかを指定することができるようにするものです。4つのオープンモードが指定可能です。4th Dimensionには下記の定数が定義されています。

定数	タイプ	値
Read and Write(デフォルト値)	整数	0
Write Mode	整数	1
Read Mode	整数	2
Get Pathname	整数	3

最後に、開かれたドキュメントファイルに対して**CLOSE DOCUMENT**を呼び出すことを忘れないようにしてください。

例題

▼以下の例は、ドキュメントファイル“ノート”を開き、それに“さようなら”という文字列を書き込み、ドキュメントファイルを閉じます。これで、前の内容の“こんにちは”は上書きされるため残りません。

```
C_TIME(vDoc)
vDoc:=Open document("ノート")`新しいドキュメントファイル"ノート"を開く
If(OK=1)
    SEND PACKET(vDoc ; "さようなら")`ドキュメントファイルに書き込む
    CLOSE DOCUMENT(vDoc)`ドキュメントファイルを閉じる
End if
```

▼**Write Mode**で既に開かれているファイルをリードすることができます。

```
vDoc:=Open document("PassFile";"TEXT")`ファイルのオープンファイルが閉じ
                                られる前に、読み出し専用モードで検査することができます。
vRef:=Open document("PassFile";"TEXT";Read Mode)
```

システム変数とセット

Documentが正しく開かれると、システム変数OKは1になります。それ以外は0となります。呼ばれた後、システム変数Documentはdocument名が入れられます。

モードに3を渡すと!!00:00:00!!（ドキュメントの参照無し）を返します。ファイルは開かれませんが、Documentとシステム変数OKは更新されます。

■ システム変数OKは1になります。

■ Documentには、ドキュメントに渡された値に応じて、名前またはドキュメントのフルパスのいずれかがセットされます（ファイル名を渡すとDocumentはこの名前になります。フルパスを渡すとDocumentはこのフルパスになります）。

注：＜ドキュメント＞に設定されたファイルが見付からない場合または＜ドキュメント＞に空白を渡した場合は、オープンファイルダイアログボックスが表示されます。これが受け入れられると、Documentシステム変数とシステム変数OKは上記のように更新されます。これが受け入れられないと、システム変数OKは0になります。

参照

Append document、Create document

Create document

Create document(ドキュメント {; タイプ}) → ドキュメントファイル参照番号

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントファイル名、または ドキュメントへの完全なパス名、または 空の文字列の場合、標準のファイル ダイアログボックス表示
タイプ	文字列	→	Macintoshファイルタイプ（4桁の文字）、 または Windowsファイル拡張子（1～3桁の文字）、 または省略した場合、 テキストドキュメント（.TXT）
戻り値	時間	←	ドキュメントファイル参照番号

説明

Create document関数は、＜ドキュメント＞で指定した名前の新しいドキュメントファイルを作成し、そのドキュメントファイルのドキュメントファイル参照番号を返します。

＜ドキュメント＞には新しいドキュメントファイルの名前、または完全なパス名を渡します。＜ドキュメント＞が既にディスク上に存在する場合には、そのドキュメントファイルを上書きします。しかし、＜ドキュメント＞がロックされていたり、既に開かれている場合には、エラーが発生します。

＜ドキュメント＞が空の文字列（ヌル""）の場合には、「ファイル作成」ダイアログボックスを開きます。ユーザは、ここで新しいドキュメントファイルの名前を入力することができます。このダイアログをキャンセルすると、ドキュメントファイルは作成されず、**Create document**関数はドキュメントファイル参照番号にヌル値を返し、システム変数OKに0を代入します。

ドキュメントが正常に作成され、開かれると、**Create document**関数はドキュメントファイル参照番号を返し、システム変数OKに1を代入します。また、システム変数Documentが更新され、作成されたドキュメントのアクセスパスを返します。

「ファイル作成」ダイアログボックスを使用するかしないかに関わらず、**Create document**関数はデフォルトとしてTEXT（Windows）またはTEXT（Macintosh）タイプのドキュメントファイを作成します。別のタイプのドキュメントファイルを作成するには、引数＜ファイルタイプ＞を指定します。

Macintoshでは、ファイルタイプを渡します。Windowsでは、1から3文字のWindowsのファイル拡張子、または**MAP FILE TYPES**コマンドを使ってマップされるMacintoshのファイルタイプを指定します。拡張子なしのドキュメント、いくつかの拡張子を含むドキュメント、または3文字を超える拡張子を含むドキュメントを作成したい場合は、引数＜タイプ＞を使わずにフルパスネームを引数＜ドキュメント＞へ渡して下さい。

ドキュメントファイルを作成し、開いた後は、**RECEIVE PACKET**や**SEND PACKET**コマンドを使用してドキュメントへの読み込みや書き込みを行えます。また、これらのコマンドと**Get file position**や**SET FILE POSITION**コマンドを組み合わせて、ドキュメントの一部に直接アクセスすることもできます。

最後に、開かれたドキュメントファイルに対して**CLOSE DOCUMENT**を呼び出すことを忘れないようにしてください。

例題

1. 以下の例は、新しいドキュメントファイル“ノート”を作成し、それに“こんにちは”という文字列を書き込み、ドキュメントファイルを閉じます。

```
C_TIME(vDoc)
vドキュメント:=Create document("ノート")` 新しいドキュメントファイル
                                                    "ノート"を作成
If(OK=1)
      SEND PACKET(vドキュメント ; "こんにちは") ` ドキュメントファイルに
                                                    書き込む
      CLOSE DOCUMENT(vドキュメント)      ` ドキュメントファイルを閉じる
End if
```

ドキュメントファイル“ノート”をワープロソフトウェアで開いてみると、“こんにちは”という文字列が含まれています。

2. 以下の例はWindows上で標準的ではない拡張子をともなうドキュメントを作成しています。

```
$vtMyDoc:=Create document("Doc.ext1.ext2") `Several extensions
$vtMyDoc:=Create document("Doc.shtml") `Long extension
$vtMyDoc:=Create document("Doc.") `No extension(the period "." is mandatory)
```

システム変数とセット

ドキュメントが正常に作成されると、システム変数OKには1が代入され、システム変数Documentには、＜ドキュメント＞に渡した値に応じて、＜ドキュメント＞の名前、またはフルパス名とドキュメント名が代入されます（ファイル名を渡した場合、Documentにはその名前が代入され、フルパス名を渡すと、Documentにはそのフルパス名が代入されます）。

参照

Append document、Open document

Append document

Append document(ドキュメント {; 形式}) → ドキュメントファイル参照番号

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントファイル名、または ドキュメントへの完全なパス名、または 空の文字列の場合、標準のファイル ダイアログボックス表示
タイプ	文字列	→	Macintoshファイルタイプ（4桁の文字）、 または Windowsファイル拡張子（1～3桁の文字）、 または省略した場合、 テキストドキュメント（.TXT）
戻り値	時間	←	ドキュメントファイル参照番号

説明

Open document関数は、ドキュメントの先頭から読み書きを実行するために<ドキュメント>で指定した既存のドキュメントファイルを開き、そのドキュメントファイル参照番号を返します。ドキュメントに書き出されたデータはそのドキュメントの先頭に書き出され、既存データを上書きします。

Append document関数は**Open document**関数と同様の処理を行います。このコマンドにより、ディスク上のドキュメントファイルを開くことができます。

異なる点としては、**Append document**関数はファイル位置をドキュメントファイルの最後に設定しますが、**Open document**関数はドキュメントファイルの最初に設定するところです。

Append document関数の使用に関する詳細は、**Open document**関数の説明を参照してください。

例題

▼以下の例は、ドキュメントファイル“ノート”を開き、このドキュメントファイルの最後に“また、会いましょう”という文字列とキャリッジリターンを書き加え、ドキュメントファイルを閉じます。このドキュメントに“さようなら”という文字列が既に含まれている場合、ドキュメントファイルの内容は、“さようなら”と改行、続けて“また、会いましょう”になります。

```
C_TIME(vDoc)
vDoc:=Append document("ノート")        `新しいドキュメントファイル"ノート"を開く
SEND PACKET(vDoc;"  また、会いましょう" +Char(13))`ドキュメントファイルに書き加える
CLOSE DOCUMENT(vDoc)`ドキュメントファイルを閉じる
```

参照

Create document、Open document

CLOSE DOCUMENT

CLOSE DOCUMENT(ドキュメントファイル参照番号)

引数	タイプ		説明
ドキュメント参照番号	時間	→	ドキュメントファイルの参照番号

説明

CLOSE DOCUMENTコマンドは、＜ドキュメントファイル参照番号＞で指定したドキュメントファイルを閉じます。

ドキュメントファイルを閉じることは、ファイルに書き込んだデータを確実に保存する唯一の方法です。**Open document**、**Create document**、**Append document**コマンドで開いたドキュメントファイルはすべて、必ずこのコマンドを使用して閉じなければなりません。

例題

▼以下の例は、新しいドキュメントファイル“ノート”を作成し、それに“こんにちは”という文字列を書き込み、ドキュメントファイルを閉じます。

```
C_TIME(vDoc)
ドキュメント:=Create document("ノート")｀新しいドキュメントファイル
                                                "ノート"を作成
If(OK=1)
    SEND PACKET(ドキュメント;"こんにちは")｀ドキュメントファイルに
                                                書き込む
    CLOSE DOCUMENT(ドキュメント)｀ドキュメントファイルを閉じる
End if
```

参照

Append document、Create document、Open document

COPY DOCUMENT

COPY DOCUMENT(コピー元 ; コピー先 {; *})

引数	タイプ		説明
コピー元	文字列	→	コピーするドキュメントの名前
コピー先	文字列	→	コピーされるドキュメントの名前
*	*	→	既存のドキュメントを上書き

説明

COPY DOCUMENTコマンドは、引数<コピー元>で指定されたドキュメントを<コピー先>で指定された位置にコピーします。

<コピー元>および<コピー先>は共に、データベースフォルダにあるドキュメントを表す名前、またはドキュメントをボリュームのルートレベルに相対的に表すパス名です。

オプション引数<*>を指定して、**COPY DOCUMENT**コマンドがコピー先ドキュメントを削除して上書きするように指示しないかぎり、<コピー先>という名前のドキュメントが既に存在する場合にはエラーが発生します。

例題

1. 以下の例では、ドキュメントをそのドキュメントのあるフォルダ内で複製します。

```
COPY DOCUMENT("C:¥¥FOLDER¥¥DocName" ; "C:¥¥FOLDER¥¥DocName2")
```

2. 以下の例では、ドキュメントをデータベースフォルダにコピーします（C:¥FOLDERがデータベースフォルダではない場合に限ります）。

```
COPY DOCUMENT("C:¥¥FOLDER¥¥DocName" ; "DocName")
```

3. 以下の例では、あるドキュメントをあるボリュームから別のボリュームへコピーします。

```
COPY DOCUMENT("C:¥¥FOLDER¥¥DocName" ; "F:¥¥Archives¥¥DocName.OLD")
```

4. 以下の例では、ドキュメントをそのドキュメントのあるフォルダ内で、既に存在する別のコピーを上書きして複製します。

```
COPY DOCUMENT("C:¥¥FOLDER¥¥DocName" ; "C:¥¥FOLDER¥¥DocName2" ; *)
```

参照

MOVE DOCUMENT

MOVE DOCUMENT

MOVE DOCUMENT(移動元パス名 ; 移動先パス名)

引数	タイプ		説明
移動元パス名	文字列	→	既存のドキュメントへの完全なパス名
移動先パス名	文字列	→	移動先のパス名

説明

MOVE DOCUMENTコマンドは、ドキュメントを移動、または名前を変更します。

引数＜移動元パス名＞にはドキュメントへの完全なパス名を指定し、＜移動先パス名＞にはドキュメントの新しい名前または新しい位置を指定します。

警告：MOVE DOCUMENTコマンドを使用すると、ドキュメントを同じボリューム上のどのディレクトリにも移動することができます。ドキュメントを2つの異なるボリュームの間で移動するには、COPY DOCUMENTコマンドを使用してドキュメントを「移動」してからDELETE DOCUMENTコマンドを使用して、オリジナルのドキュメントを削除します。

例題

1. 以下の例では、ドキュメント「DocName」の名前を変更します。

```
MOVE DOCUMENT("C:¥¥FOLDER¥¥DocName" ; "C:¥¥FOLDER¥¥NewDocName")
```

2. 以下の例では、ドキュメント「DocName」を移動し、名前を変更します。

```
MOVE DOCUMENT C:¥¥FOLDER1¥¥DocName" ; "C:¥¥FOLDER2¥¥NewDocName")
```

3. 以下の例では、ドキュメント「DocName」を移動しています。

```
MOVE DOCUMENT("C:¥¥FOLDER1¥¥DocName" ; "C:¥¥FOLDER2¥¥DocName")
```

注：後の2つの例では、移動先のフォルダ"C:¥FOLDER2"が存在しなければなりません。MOVE DOCUMENTコマンドはドキュメントを移動するだけで、フォルダは作成しません。

参照

COPY DOCUMENT

DELETE DOCUMENT

DELETE DOCUMENT(ドキュメント)

引数	タイプ		説明
ドキュメント	文字列	→	削除するドキュメントファイル

説明

DELETE DOCUMENTコマンドは、＜ドキュメント＞に渡した名前のドキュメントファイルを削除します。

指定したドキュメントファイルが存在しない場合でも、エラーにはなりませんが、現在開かれているドキュメントファイルを削除しようとすると、エラーが発生します。

＜ドキュメント＞に空の文字列（ヌル""）を指定することはできません。＜ドキュメント＞に空の文字列を指定すると、ダイアログボックスは表示されず、エラーが発生します。

警告：DELETE DOCUMENTコマンドは、ディスク上のあらゆるファイルを削除することができます。これらのファイルには、他のアプリケーションで作成されたドキュメントファイルやそのアプリケーションそのものも含まれます。DELETE DOCUMENTコマンドは、注意して使用してください。一度削除したファイルは復元できません。

例題

1. 以下の例は、ドキュメントファイル“ノート”を削除します。

```
DELETE DOCUMENT("ノート") ` ドキュメントファイルを削除
```

2. **APPEND TO CLIPBOARD**コマンドの例を参照してください。

システム変数とシステムセット

ドキュメントファイルの削除に成功すると、システム変数OKに1が代入されます。削除に失敗した場合には、システム変数OKに0が代入されます。

Select document

Select document(ディレクトリ; ファイルタイプ; タイトル; オプション
{; 選択項目})→文字列

引数	タイプ		説明
ディレクトリ	テキスト! 倍長整数	→	・ドキュメント選択ダイアログボックスにデフォルトとして表示するディレクトリのアクセスパス、または ・デフォルトのユーザフォルダを表示する場合は空の文字列（Windowsでは“My documents”、Mac OSでは“Documents”）、または ・保存されたアクセスパスの番号
ファイルタイプ	テキスト	→	ふるい分けするドキュメントタイプの一覧、またはドキュメントのふるい分けを行わない場合は“*”
タイトル	テキスト	→	選択ダイアログボックスのタイトル
オプション	倍長整数	→	選択オプション
選択項目	テキスト配列	←	選択したファイルのアクセスパスと名前の一覧を格納する配列
戻り値	文字列	←	選択したファイルの名前（複数項目が選択された場合は1番目のファイル）

説明

Select document関数は、標準の「ドキュメントを開く」ダイアログボックスを表示し、これを用いてユーザは1つ以上のファイルを指定し、その選択したファイルの名前やフルアクセスパスを返すことができます。

引数<ディレクトリ>はフォルダを表わし、最初にそのフォルダの内容がドキュメントを開くダイアログボックス上に表示されます。3種類の値を渡すことができます。

■ 表示するフォルダのフルアクセスパスを含むテキスト

■ 現行のオペレーティングシステムのデフォルトユーザフォルダを表示する場合は、空の文字列（""）（Windowsでは“My documents”、Mac OSでは“Documents”）

■ 関連するフォルダを表示するために保存されたアクセスパスの番号（1～32000）

この場合、ユーザが選択ボタンをクリックした際に開かれるフォルダのアクセスパス、つまり、ユーザにより選択されたフォルダをメモリ上に保存することができます。最初に任意の番号を呼び出すと（例えば、“5”）、コマンドはそのオペレーティングシステムのデフォルトユーザフォルダを表示します（空の文字列を渡す場合と同じ）。また、ユーザはハードディスク上のフォルダを閲覧することもできます。ユーザが選択ボタンをクリックすると、アクセスパスが保存されて番号“5”に関連付けられます。この後で番号“5”を呼び出すと、保存されたアクセスパスがデフォルトとして使用されます。新しい場所が選択された場合は、パス番号5が更新されます。

このメカニズムにより、32,000個までのアクセスパスを記憶しておくことができます。Windows上では、各パスはそのセッションでのみ保持されます。Mac OS上では、各パスはシステムによって保持され、セッションからセッションへと保存されます。

引数＜ファイルタイプ＞には、ファイルを開くダイアログボックス上で選択できるファイルタイプを渡します。複数のタイプを“;（セミコロン）”で区切ったリストを渡すことができます。定義された各タイプごとに、行がダイアログボックスのタイプ選択メニューに追加されます。

Mac OS上では、ふるい分けられるファイルのタイプを渡さなくてはなりません（TEXT、APPL、その他）

Windows上で、Mac OSタイプのファイル（4th Dimensionが内部的に変換を行います）やファイルの拡張子（.txt、.exe）を渡すこともできます。Windowsでは、ダイアログボックスに“*.*”と入力すると、すべてのドキュメントタイプを“強制的”に表示できる点に注意してください。しかしこの場合、4th Dimensionは選択されるファイルタイプの追加検証を実行します。ユーザが許可されていないタイプを選択すると、コマンドはエラーを返します。

表示されるファイルをいくつかのタイプに限定したくない場合、＜ファイルタイプ＞に値“*（アスタリスク）”または“.*”を渡します。

引数＜タイトル＞には、ダイアログボックスに表示しなくてはならないラベルを渡します。空の文字列を渡すと、“Open”というラベルが表示されます。

引数＜オプション＞を使用し、ファイルを開くダイアログボックスで許可される上級機能を指定することができます。4th Dimensionでは、次に示す「System documents」テーマの定義済み定数が用意されています。

定数	タイプ	値
Multiple files	倍長整数	1
Package open	倍長整数	2
Package selection	倍長整数	4
Alias selection	倍長整数	8
Use Sheet Window	倍長整数	16

定数のうちの一つを渡すか、あるいは定数を組み合わせて渡すことができます。

■ Multiple files：キーコンビネーションのShift+クリック（隣接する項目の選択）、Ctrl+クリック（Windows）、Command+クリック（Mac OS）を使用し、複数ファイルを同時に選択することを許可します。この場合、引数＜選択項目＞を渡すと、選択されたファイルの一覧がこの引数に代入されます。
この定数を使用しない場合、デフォルトとしてコマンドは複数ファイルの選択を許可しません。

■ Package open（Mac OSのみ）：パッケージのオープンならびにその内容の表示を許可します。
この定数を使用しない場合、デフォルトとしてコマンドはパッケージのオープンを許可しません。

■ Package selection（Mac OSのみ）：実体としてパッケージの選択を許可します。

この定数を使用しない場合、デフォルトとしてコマンドはソフトウェアパッケージをそのまま使用することを許可しません。この場合、ソフトウェアパッケージのオープンや選択は行えません（定数「Package open」を渡した場合でも）。

■ Alias selection：ショートカット（Windows）やエイリアス（Mac OS）をドキュメントとして選択することを許可します。
デフォルトでは、この定数を使用しない場合にショートカットやエイリアスを選択すると、コマンドは目的とする項目のアクセスパスを返します。

■ Use Sheet Window（Mac OSのみ）：選択用ダイアログボックスをシートウインドウ形式で表示します（Windowsにおいて、このオプションは無視されます）。
シートウインドウはMac OS X固有のインタフェースであり、グラフィックアニメーションを伴います（詳細については、「ウインドウタイプ」の節を参照してください）。
この定数が使用されない場合、デフォルトとしてコマンドは標準のダイアログボックスを表示します。

定数を渡した場合、コマンドはエイリアスやショートカット自体のアクセスパスを返します。

オプションをいずれも使用したくない場合、引数＜オプション＞に0を渡してください。

任意の引数＜選択項目＞を使用して、ユーザが選択した各ファイルのフルアクセスパス（アクセスパス + ファイルの名前）を取得することができます。コマンドは、ユーザの選択に応じて配列の作成、サイズの調整、値の代入を行います。この引数は、「Multiple files」を選択した場合や、選択したファイルへのアクセスパスを調べたい場合に役立ちます（コマンドにより返された配列の値からファイル名を取り除く）。ファイルが選択されない場合、コマンドは空の配列を返します。

コマンドは選択されたファイルの名前（Windowsでは名前＋拡張子）を返します。複数のファイルが選択された場合、コマンドは選択ファイルリスト内の最初のファイル名を返します。このファイル一覧は、任意の引数＜選択項目＞により取得することができます。ファイルが選択されない場合、コマンドは空の文字列を返します。

参照

Open Document、Select folder

システム変数とシステムセット

コマンドが正常に実行され、有効なドキュメントが選択されると、システム変数OKには1が代入され、システム変数Documentには選択されたファイルのフルアクセスパスが代入されます。

ファイルが選択されない場合（例えば、ファイルを開くダイアログボックスでユーザが「キャンセル」ボタンをクリックした場合）、システム変数OKには0が代入され、システム変数Documentは空白になります。

Test path name

Test path name(パス名) → 数値

引数	タイプ		説明
パス名	文字列	→	ディレクトリ、フォルダ、ドキュメントへのパス名
戻り値	数値	←	1=パス名は既存のドキュメントを表す 0=パス名は既存のディレクトリまたはフォルダを表す <0=無効なパス名、OSファイルマネージャエラーコード>

説明

Test path name関数は、引数<パス名>に渡された名前またはパス名を持つドキュメントまたはフォルダが、ディスク上に存在するかどうかをチェックします。

ドキュメントが見つかれば、**Test path name**関数は1を返します。フォルダが見つかれば、**Test path name**関数は0を返します。

4Dには、以下のようなあらかじめ定義された定数があります。

定数	タイプ	値
Is a documen	倍長整数	1
Is a directory	倍長整数	0

ドキュメントもフォルダも見つからない場合には、**Test path name**関数は負の値を返します（つまり、ファイルが見つからない場合には-43になります）。

例題

▼以下の例では、「Journal」というドキュメントがデータベースのフォルダにあるかどうかをテストし、見つからない場合にはこれを作成します。

```
If(Test path name("Journal") # Is a document)
    $vhDocRef:=Create document("Journal")
    If(OK=1)
        CLOSE DOCUMENT($vhDocRef)
    End if
End if
```

参照

Create document、CREATE FOLDER

SHOW ON DISK

SHOW ON DISK(パス名{; *})

引数	タイプ		説明
パス名	文字列	→	表示する項目のパス名
*		→	項目がフォルダである場合、その内容を表示する

説明

SHOW ON DISKコマンドは、引数<パス名>に渡されたパス名のファイルまたはフォルダをOSの標準ウインドウ上に表示します。

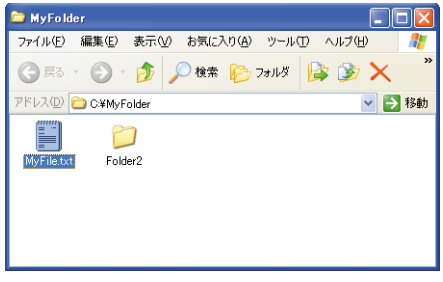
ユーザインタフェースでこのコマンドを使用すると、特定のファイルやフォルダの場所を指定することができます。

<パス名>にフォルダが指定された場合、デフォルトとしてコマンドはそのフォルダ自体の階層を表示します。任意の引数である<*>を渡すと、コマンドはそのフォルダを開き、ウインドウ上にフォルダ内容を表示します。<パス名>にファイルを指定した場合、引数<*>は無視されます。

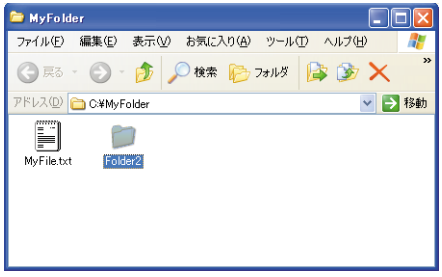
例題

▼ 次の例は、このコマンドの動作を表わしています。

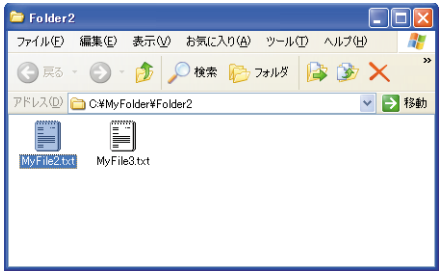
SHOW ON DISK("c:¥¥MyFolder¥¥MyFile.txt")
` 指定されたファイルを表示する



SHOW ON DISK("c:¥¥MyFolder¥¥Folder2")
` 指定されたフォルダを表示する



SHOW ON DISK("c:¥¥MyFolder¥¥Folder2";*)
`指定されたフォルダの内容を表示する



システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入されます。

CREATE FOLDER

CREATE FOLDER(パス名)

引数	タイプ		説明
パス名	文字列	→	作成する新しいフォルダへのパス名

説明

CREATE FOLDERコマンドは、引数<パス名>に渡すパス名にしたがって、フォルダを作成します。

名前を渡すと、データベースのフォルダにフォルダが作成されます。パス名を渡す場合には、作成したいフォルダの名前までの、ボリュームのルートレベルまたはデータベースフォルダのレベルから始まる有効なパス名を表していなければなりません。

フォルダが既に存在する場合には、エラーが発生します。

例題

1. 以下の例では、「Archives」というフォルダをデータベースのフォルダ内に作成します。

```
CREATE FOLDER("Archives")
```

2. 以下の例では、「Archives」というフォルダをデータベースのフォルダ内に作成し、「January」および「February」というサブフォルダを作成します。

```
CREATE FOLDER("Archives")
CREATE FOLDER("Archives¥¥January")
CREATE FOLDER("Archives¥¥February")
```

3. 以下の例では、「Archives」というフォルダをCボリュームのルートレベルに作成します。

```
CREATE FOLDER("C:¥¥Archives")
```

4. Cボリュームのルートレベルに「NewStuff」というフォルダがない場合には、以下の例は失敗します。

```
CREATE FOLDER("C:¥¥NewStuff¥¥Pictures")
                                `1回の呼び出しで2つのフォルダは作成できない
```

参照

FOLDER LIST、Test path name

Select folder

Select folder({メッセージ }) → 文字列

引数	タイプ		説明
メッセージ	文字列	→	表示されるフォルダ選択ダイアログ内のタイトル
戻り値	文字列	←	選択されたフォルダへのアクセスパス

説明

このコマンドは、フォルダ選択ダイアログボックスを表示し選択したフォルダのフルパスを返します。

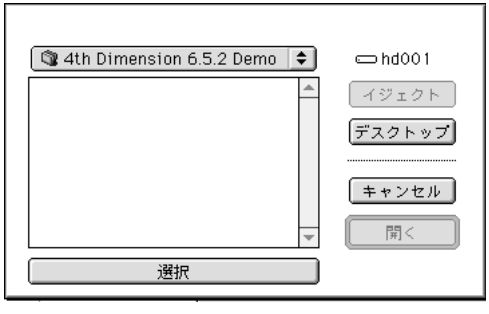
注：このコマンドは、4Dのカレントフォルダを変更しません。

Select folderコマンドはワークステーションのボリュームおよびフォルダ内をナビゲートするための標準のダイアログボックスを表示します。

オプションの引数メッセージを指定すると、ダイアログボックス内に表示されます。

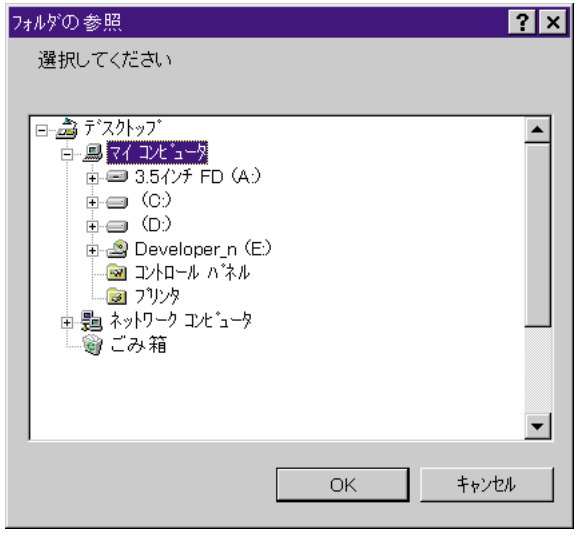
以下の例題は、"目的のフォルダを選択してください:"のメッセージです。

(Macintosh)



Select folder ("目的のフォルダを選択してください:") するとダイアログにメッセージが表示されます。

(Windows)



ユーザはフォルダを選択して、（Windowsでは）OKボタンをクリックするか、または（Macintoshでは）選択ボタンをクリックすると、フォルダへのフルパスが戻り値として返されます。

■ Windowsでのパスは、下記のフォーマットで返されます。

"C:¥Folder1¥Folder2¥SelectedFolder¥"

■ Macintoshでのパスは、下記のフォーマットで返されます。

"Hard Disk:Folder1:Folder2:SelectedFolder:"

注：Macintoshでは、ダイアログボックス内でフォルダの名前を選択しているかどうかで、帰ってくるパスは違うものになる場合があります。

4D Server：この関数は、クライアントワークステーションに接続されているボリュームを見ることができるようにするものです。ストアードプロシージャからこの関数を呼び出すことはできません。

ユーザがダイアログボックスを受け入れるとシステム変数OKは1になります。ユーザがキャンセルボタンをクリックするとシステム変数OKは0になり、関数は空白を返します。

注：Windowsでは、ユーザが何らかの誤ったエレメント（「ワークステーション」や「ごみ箱」等）を選択すると、ユーザがダイアログボックスを受け入れても、システム変数OKは0になります。

例題

▼下記の例では、フォルダを選択し、ピクチャライブラリ内のピクチャを保存します。

```
$PictFolder:=Select folder("ピクチャを保存するフォルダを選択してください ")
PICTURE LIBRARY LIST(pictRefs;pictNames)
For($n;1;Size of array(pictNames))
    $vRef:=Create document($PictFolder+pictNames{$n};"PICT")
    If(OK=1)
        GET PICTURE FROM LIBRARY(pictRefs{$n};$vStoredPict)
        SAVE PICTURE TO FILE($vRef;$vStoredPict)
        CLOSE DOCUMENT($vRef)
    End if
End for
```

参照

CREATE FOLDER、FOLDER LIST

DELETE FOLDER

DELETE FOLDER(フォルダ)

引数	タイプ		説明
フォルダ	文字列	→	削除するフォルダの名前、 またはフルパス

説明

DELETE FOLDERコマンドは、＜フォルダ＞に渡された名前またはフルパスを持つフォルダを削除します。

このコマンドでは、空のフォルダしか削除することはできません。

■ ファイルの入っているフォルダを削除しようとすると、-47のエラー（空ではないフォルダを削除しようとしました）が生成されます。

■ ＜フォルダ＞にファイルへのパスや空の文字列、または存在しないフォルダへのパスを渡した場合、コマンドは何も行わず-43のエラー（ファイルが見つかりません）が生成されます。

これらのエラーは、**ON ERR CALL**コマンドでインストールしたメソッドを使って検出することができます。

参照

DELETE DOCUMENT

CREATE ALIAS

CREATE ALIAS(ターゲットパス; エイリアスパス)

引数	タイプ		説明
ターゲットパス	文字列	→	エイリアス／ショートカット作成対象の名前またはアクセスパス
エイリアスパス	文字列	→	エイリアス／ショートカットの名前またはフルパス名

説明

CREATE ALIASコマンドは、＜ターゲットパス＞に渡した対象ファイル、またはフォルダのエイリアス（Windowsでは“ショートカット”と呼ばれる）を作成します。この名前と場所は、引数＜ターゲットパス＞で定義します。

エイリアスはあらゆる種類のドキュメントまたはフォルダに対して作成することができます。エイリアスのアイコンは、対象とするアイテムと同じものになります。このアイコンには、左側下部に小さな矢印が付きます。また、Mac OSではアイコンの名前がイタリック体で表示されます。

このコマンドは、デフォルトで名前の設定は行わないため、引数＜エイリアスパス＞に名前を渡す必要があります。この引数に名前だけを渡した場合、現在作業を行っているフォルダ（通常は、ストラクチャアイルがあるフォルダ）内にエイリアスが作成されます。

注：Windowsにおいて、ショートカットは“.LNK”という拡張子が付いたファイルになります。この拡張子を渡さないと、コマンドは自動的に拡張子を付加します。

＜ターゲットパス＞に空の文字列が渡されると、このコマンドは何も行いません。

例題

データベースで、“Report Number”という名前のテキストファイルをソートした状態でデータベースフォルダに作成します。ユーザはこれらのレポートのショートカットを作成して、使いやすい場所に保存しておきたいものとします。

```
`CREATE_REPORTメソッド
C_TEXT($vtRport)
C_STRING(250;$vtpath)
C_STRING(80;$vaname)
C_TIME(vDoc)
C_INTEGER($ReportNber)
$vTReport:=... `レポート作成
```

```
$ReportNber:=... `レポート番号の計算
$vaName:="Report"+String($ReportNber)+".txt" `ファイル名
vDoc:=Create document($vaName)
If(OK=1)
	SEND PACKET(vDoc;$vTReport)
	CLOSE DOCUMENT(vDoc)
	`エイリアスを追加
	CONFIRM("このレポートのエイリアスを作成しますか？")
	If(OK=1)
		$vtPath:=Select folder("エイリアスを何処に作成しますか？")
		If(OK=1)
			CREATE ALIAS($vaName;$vtPath+$vaName)
		End if
	End if
End if
```

参照

RESOLVE ALIAS

RESOLVE ALIAS

RESOLVE ALIAS(エイリアスパス; ターゲットパス)

引数	タイプ		説明
エイリアスパス	文字列	→	エイリアス／ショートカットの名前またはアクセスパス
ターゲットパス	文字列	←	エイリアス／ショートカット対象の名前またはアクセスパス

説明

RESOLVE ALIASコマンドは、エイリアス（Windowsでは“ショートカット”と呼ばれる）の作成対象となったファイル、またはフォルダのフルパスを返します。

エイリアスのフルパスを引数＜エイリアスパス＞に渡します。

このコマンドが実行されると、＜ターゲットパス＞に指定した変数にはエイリアスの対象ファイルまたはフォルダへのフルパスが納められます。

注：Windowsにおいて、ショートカットは“.LNK”という拡張子が付いたファイルになります。この拡張子を指定しない場合、コマンドは自動的に拡張子を付加します。

参照

CREATE ALIAS

VOLUME LIST

VOLUME LIST(ボリューム)

引数	タイプ		説明
ボリューム	配列	←	現在マウントされているボリュームの名前

説明

VOLUME LISTコマンドは、テキスト配列または文字列配列の＜ボリューム＞に、現在コンピュータに定義されているボリュームの名前（Windowsの場合）、またはマウントされているボリュームの名前（Macintoshの場合）を代入します。

Macintoshでは、Finderレベルに表示されるボリュームのリストが返されます。

一方、Windowsでは、現在定義されているボリュームが、それぞれのボリュームが物理的に存在するかどうかに関係なく返されます（つまり、フロッピーディスクドライブに実際にディスクが入っているかどうかに関係なく、ボリュームA:¥が返されます）。

例題

「asVolumes」というスクロール可能エリアを使用して、コンピュータに定義またはマウントされているボリュームのリストを表示するには、以下のように記述します。

```
Case of
	¥(Form event=On Load)
		ARRAY STRING(31 ; asVolumes ; 0)
		VOLUME LIST(asVolumes)
		` ...
End case
```

参照

DOCUMENT LIST、FOLDER LIST、VOLUME ATTRIBUTES

VOLUME ATTRIBUTES

VOLUME ATTRIBUTES(ボリューム ; サイズ ; 使用サイズ ; 空きサイズ)

引数	タイプ		説明
ボリューム	文字列	→	ボリュームの名前
サイズ	数値	←	ボリュームのサイズ（バイト単位）
使用サイズ	数値	←	使用サイズ（バイト単位）
空きサイズ	数値	←	空きサイズ（バイト単位）

説明

VOLUME ATTRIBUTESコマンドは、引数＜ボリューム＞に渡した名前を持つボリュームのサイズ、使用サイズおよび空きサイズを、バイト単位で表わして返します。

注：＜ボリューム＞により、マウントされていないリモートボリュームが指定された場合、システム変数OKには0が代入され、3つの引数は-1を返します。

例題

このアプリケーションには、夜間や週末に実行し、ディスク上に大規模な中間ファイルを格納するバッチ処理がいくつか含まれています。このプロセスをできる限り自動的かつ柔軟にするには、中間ファイルを格納するために十分な空きサイズがある最初のボリュームを自動的に見つけるルーチンを作成します。例えば、以下のようなプロジェクトメソッドを作成します。

```
`「Find volume for space」プロジェクトメソッド
` Find volume for space(倍長整数) → 文字列
` Find volume for space(必要なサイズ ) → ボリューム名または空の文字列
C_STRING(31 ; $0)
C_STRING(255 ; $vsDocName)
C_LONGINT($vlNbVolumes ; $vlVolume ; $vlSize ; $vlUsed ; $vlFree)
C_REAL($1 ; $vlSize ; $vlUserd ; $vlFree)
C_TIME($vhDocRef)
`戻り値を初期化する
$0:=""
`エラー阻止メソッドを使用して、すべてのI/O処理を保護する
ON ERR CALL("エラーメソッド")
` ボリュームのリストを取得する
ARRAY STRING(31 ; $asVolumes ; 0)
gError:=0
VOLUME LIST($asVolumes)
If(gError=0)
```

```
`Windowsで実行する場合には(通常の) 2つのフロッピーディスクドライブ
                                                は省略する
If(On Windows)
        $vlVolume:=Find in array($asVolumes ; "A:¥")
        If($vlVolume>0)
                DELETE ELEMENT($asVolumes ; $vlVolume)
        End if
        $vlVolume:=Find in array($asVolumes ; "B:¥")
        If($vlVolume>0)
                DELETE ELEMENT($asVolumes ; $vlVolume)
        End if
End if
$vlNbVolumes:=Size of array($asVolumes)
` For each volume
For($vlVolume ; 1 ; $vlNbVolumes)
        `使用サイズおよび空きサイズを取得する
        gError:=0
        VOLUME ATTRIBUTES($asVolumes{$vlVolume} ; $vlSize ;
                                                $vlUsed ; $vlFree)
        If(gError=0)
                `空きサイズは十分ありますか(必要なサイズに32KBを加え
                                                たサイズ)?
                If($vlFree>=($1+32768))
                        ` 十分であれば、ボリュームがアンロックされている
                                                かどうかをチェック...
                        $vsDocName:=$asVolumes{$vlVolume}+Char
                                                (Directory symbol )
                                +"XYZ"+String(Random)+".TXT"
                        $vhDocRef:=Create document($vsDocName)
                        If(OK=1)
                                CLOSE DOCUMENT($vhDocRef)
                                DELETE DOCUMENT($vsDocName)
                                `すべて問題がなければ、ボリュームの名前を返す
                                $0:=$asVolumes{$vlVolume}
                                $vlVolume:=$vlNbVolumes+1
                        End if
                End if
        End if
    End for
End if
ON ERR CALL("")
```

このプロジェクトメソッドがアプリケーションに追加されると、例えば、以下のように記述することができます。

```
$vsVolume:= Find volume for space(100*1024*1024)
If($vsVolume#"")
	` Continue
Else
	ALERT("少なくとも100 MBの空き空間を持つボリュームが必要です!")
End if
```

参照

VOLUME LIST

FOLDER LIST

FOLDER LIST(パス名 ; ディレクトリ)

引数	タイプ		説明
パス名	文字列	→	ボリューム、ディレクトリまたはフォルダへのパス名
ディレクトリ	配列	←	この位置にあるディレクトリの名前

説明

FOLDER LISTコマンドは、テキスト配列または文字列配列の＜ディレクトリ＞に、引数＜パス名＞に渡すパス名の位置にあるフォルダの名前を代入します。

指定した位置にフォルダがない場合には、このコマンドは空の配列を返します。＜パス名＞に渡すパス名が無効な場合には、**FOLDER LIST**コマンドはファイルマネージャエラーを生成します。このエラーの生成は、**ON ERR CALL**メソッドを使用して検出することができます。

警告：引数＜パス名＞の最大長は255文字です。これより長いパス名を渡すと、パス名は切り捨てられ、ファイルマネージャエラーが発生します。

参照

DOCUMENT LIST、VOLUME LIST

DOCUMENT LIST

DOCUMENT LIST(パス名 ; ドキュメント)

引数	タイプ		説明
パス名	文字列	→	ボリューム、ディレクトリまたはフォルダへのパス名
ドキュメント	配列	←	この位置にあるドキュメントの名前

説明

DOCUMENT LISTコマンドは、テキスト配列または文字列配列の＜ディレクトリ＞に、引数＜パス名＞に渡すパス名の位置にあるドキュメントの名前を代入します。

指定した位置にドキュメントがない場合には、このコマンドは空の配列を戻します。＜パス名＞に渡すパス名が無効な場合には、**DOCUMENT LIST**コマンドはファイルマネージャエラーを生成します。このエラーの生成は、ON ERR CALLメソッドを使用して検出することができます。

警告：引数＜パス名＞の最大長は255文字です。これより長いパス名を渡すと、パス名は切り捨てられ、ファイルマネージャエラーが発生します。

参照

DIRECTORY LIST、FOLDER LIST、VOLUME LIST

MAP FILE TYPES

MAP FILE TYPES(Macintosh ; Windows ; コンテキスト)

引数	タイプ		説明
Macintosh	文字列	→	Macintosh OS のファイルタイプ（4文字）
Windows	文字列	→	Windowsのファイル拡張子（1～3文字）
コンテキスト	文字列	→	Windowsの「ファイルを開く」ダイアログの「ファイルの種類」ドロップダウンリストに表示される文字列

説明

MAP FILE TYPESコマンドによって、Windowsのファイル拡張子とMacintoshのファイルタイプを関連づけることができます。

このルーチンを1回呼び出すだけで、データベースによる作業セッション全体についてマッピングを設定することができます。コールが行われると、Windows上で**Append document**、**Create document**、**Create resource file**、**Open resource file**、**Open resource file**コマンドを実行すると、実際に引数としてルーチンに渡されたMacintoshのファイルタイプは、自動的にWindowsの拡張子で置き換えられます。

引数＜Macintosh＞には、4文字のMacintoshのファイルタイプを渡します。4文字の文字列を渡さなければ、コマンドは動作せず、エラーが発生します。

引数＜Windows＞には、1から3文字のWindowsのファイル拡張子を渡します。1から3文字の文字列を渡さなければ、コマンドは動作せず、エラーが発生します。

引数＜コンテキスト＞には、Windowsの「ファイルを開く」ダイアログの「ファイルの種類」ドロップダウンリストに表示される文字列を渡します。コンテキストの文字列は 32 文字以内に制限されており、それ以上の文字は無視されます。

重要：Windowsのファイル拡張子とMacintoshのファイルタイプとのマッピングを実行すると、その作業セッション中はマッピングを変更、または削除できません。4Dアプリケーションの開発とデバッグの際に変更する必要がある場合は、データベースを開き直して、再度ファイル拡張子のマッピングを行います。

例題

▼次に示す4th Dimensionの簡単なコード（「On startup」データベースメソッド等で使用します）は、MacintoshのMS-Wordファイルタイプ“WDBN”をWindowsのファイル拡張子“DOC”にマッピングします。

```
MAP FILE TYPES("WDBN" ; "DOC" ; "Word documents")
```

前記のコールが実行された後、次に示すコードを実行すると、MacintoshとWindowsの「ファイルを開く」ダイアログにはWordのドキュメントが表示されます。

```
$DocRef:=Open document("" ; "WDBN")
If(OK=1)
        ` ...
End if
```

参照

Append document、Create document、Create resource file、Open resource file、Open resource file

GET DOCUMENT PROPERTIES

GET DOCUMENT PROPERTIES(ドキュメント ; ロック ; 非表示 ; 作成日 ; 作成時間 ; 更新日 ; 更新時間)

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前、または ドキュメントの完全なパス名
ロック	ブール	←	ロックの場合はTrue、アンロックの場合はFalse
非表示	ブール	←	非表示の場合はTrue、表示の場合はFalse
作成日	日付	←	作成日
作成時間	時間	←	作成時間
更新日	日付	←	更新日
更新時間	時間	←	更新時間

説明

GET DOCUMENT PROPERTIESコマンドは、引数＜ドキュメント＞に渡した名前またはパス名を持つドキュメントファイルに関する情報を返します。

呼び出し後、以下の情報が返されます。

- 引数＜ロック＞には、ドキュメントがロックされていればTrueが返されます。ロックされているドキュメントを開いたり削除することはできません。
- 引数＜非表示＞には、ドキュメントが隠されていればTrueが返されます。
- 引数＜作成日＞および＜作成時間＞には、ドキュメントが作成された日付と時間が返されます。
- 引数＜更新日＞および＜更新時間＞には、ドキュメントが更新された日付と時間が返されます。

例題

ドキュメントデータベースを作成し、そのデータベースに作成したすべてのレコードをディスク上のドキュメントにデータを書き出す場合を想定します。データベースは定期的にアップデートされたため、ドキュメントが存在しなかったり、ドキュメントが最後に保存された後でレコードが更新されていた場合には、各ドキュメントを作成、または再作成するような、データ書き出しのアルゴリズムを記述しようとしています。このため、ドキュメント（存在する場合には）の更新日付と時間を対応するレコードと比較する必要があります。この例を図解するために、以下のような表を使用して定義を表しています。



各レコードに日付と時間の両方を保存するのではなく、任意の以前の日付および時間と、レコードが保存された日付および時間との間の経過秒数を示す「タイムスタンプ」の値を保存することができます（この例では、Sep, 1st 1997 at 00:00:00、つまり1997年9月1日午前0時を使用しています）。

この例では、[ドキュメント]作成日フィールドにレコードが最初に作成された時のタイムスタンプがあり、[ドキュメント]更新日フィールドにレコードが最後に更新された時のタイムスタンプがあります。

この後に示されている「Time stamp」プロジェクトメソッドでは、パラメータが渡されない場合には、現在の日付と時間を特定の日付と時間としてタイムスタンプを計算します。

```
`「Time stamp」プロジェクトメソッド
` Time stamp {(日付 ; 時間) } → 倍長整数
` Time stamp {(日付 ; 時間) } → 1997年9月1日から経過した秒数
C_DATE($1 ; $vdDate)
C_TIME($2 ; $vhTime)
C_LONGINT($0)
If(Count parameters=0)
	$vdDate:=Current date
	$vhTime:=Current time
Else
	$vdDate:=$1
	$vhTime:=$2
End if
$0:=(($vdDate-!97/09/01!)*86400)+$vhTime
```

注：このメソッドを使用すると、日付と時間を97/09/01の00:00:00から2063/01/19の03:14:07の間でコード化することができるため、倍長整数の0から2^31-1の範囲に対応できます。

一方、次に示されている「Time stamp to date」および「Time stamp to time」のプロジェクトメソッドでは、タイムスタンプに保存されている日付および時間を抽出することができます。

```
`「Time stamp to date」プロジェクトメソッド
` Time stamp to date(倍長整数) → 日付
` Time stamp to date(Time stamp) → 抽出する日付
C_DATE($0)
C_LONGINT($1)
$0:=!97/09/01!+($1//86400)
`「Time stamp to time」プロジェクトメソッド
` Time stamp to time(倍長整数) → 日付
` Time stamp to time(Time stamp ) → 抽出する日付
C_TIME($0)
C_LONGINT($1)
$0:=Time(Time string(!!00:00:00!!+($1%86400)))
```

レコードの作成や更新の方法に関係なく、レコードのタイムスタンプが、正しく更新されるようにするには、[ドキュメント]テーブルのトリガを使用して、その規則を強制します。

```
`[ドキュメント]テーブルのトリガ
Case of
    ¥(Database event=Save New Record Event)
        [ドキュメント]作成日:= Time stamp
        [ドキュメント]更新日:= Time stamp
    ¥(Database event=Save Existing Record Event)
        [ドキュメント]更新日:= Time stamp
End case
```

このトリガをデータベースに実装すると、以下の「CREATE DOCUMENTATION」プロジェクトメソッドの作成に必要となるすべての準備が整います。ドキュメントの作成および更新の日付および時間の処理には、**GET DOCUMENT PROPERTIES**コマンドおよび**SET DOCUMENT PROPERTIES** コマンドを使用します。

```
`「CREATE DOCUMENTATION 」プロジェクトメソッド
C_STRING(255 ; $vsPath ; $vsDocPathName ; $vsDocName)
C_LONGINT($vlDoc)
C_BOOLEAN($vbOnWindows ; $vbDoIt ; $vbLocked ; $vbInvisible)
C_TIME($vhDocRef ; $vhCreatedAt ; $vhModifiedAt)
C_DATE($vdCreatedOn ; $vdModifiedOn)
If(Application type=4D Client)
    ` 4D Clientを実行している場合には、4D Clientが存在する クライアントマシンに
    `ドキュメントをローカルに保存します。
    $vsPath:= Long name to path name(Application file)
Else
    `それ以外の場合には、データファイルが存在する位置にドキュメントを保存する
```

```
        $vsPath:= Long name to path name(Data file)
End if
        `任意で "Documentation" と命名したディレクトリにあるドキュメントを保存する
$vsPath:=$vsPath+"Documentation"+Char(Directory symbol )
        `このディレクトリが存在しない場合には、作成する
If(Test path name($vsPath)   #   Is a directory)
        CREATE FOLDER($vsPath)
End if
        `古いドキュメントつまり対応するレコードが削除されているドキュメントは
        `削除しなければならないため、既にあるドキュメントのリストを作成する
ARRAY STRING(255 ; $asDocument ; 0)
DOCUMENT LIST($vsPath ; $asDocument)
        ` 部門テーブルのレコードを選択する
ALL RECORDS([ドキュメント])
        ` 各レコードに対して
$vlNbRecords:=Records in selection([ドキュメント])
$vlNbDocs:=0
$vbOnWindows:= On Windows
For($vlDoc ; 1 ; $vlNbRecords)
        `ディスクにドキュメントを(再) 作成しなければならないと想定する
        $vbDoIt:=True
        `ドキュメントの名前およびパス名を求める
        $vsDocName:="DOC"+String([ドキュメント]番号 ; "00000")
        $vsDocPathName:=$vsPath+$vsDocName
        `このドキュメントは既に存在するか ?
        If(Test path name($vsDocPathName+".HTM")=Is a document)
        ` その場合には、ドキュメントのリストからドキュメントを除去します。
        ` このようにすると、ドキュメントが削除される場合があります。
                $vlElem:=Find in array($asDocument ; $vsDocName+".HTM")
                If($vlElem>0)
                        DELETE ELEMENT($asDocument ; $vlElem)
                End if
                ` レコードが最後に更新された後で、ドキュメントが保存されたか?
                GET DOCUMENT PROPERTIES($vsDocPathName+".HTM" ;
                        $vbLocked ; $vbInvisible ; $vdCreatedOn ; $vhCreatedAt ;
                                $vdModifiedOn ;$vhModifiedAt)
                If(Time stamp($vdModifiedOn ; $vhModifiedAt) >= [ドキュメント]
                                更新日)
                        `その場合には、ドキュメントを再作成する必要はない
                        $vbDoIt:=False
                End if
```

```
Else
        `ドキュメントは存在しないため、これら2つの変数をリセットし、
        `ドキュメントの最終的なプロパティを設定する前にこれらを計算す
        `る必要があることを確認できるようにする
        $vdModifiedOn:=!00/00/00!
        $vhModifiedAt:=!!00:00:00!!
End if
` ドキュメントを(再) 作成する必要があるか?
If($vbDoIt)
        `その場合には、更新されたドキュメントの数を増分する
        $vlNbDocs:=$vlNbDocs+1
        `ドキュメントが既に存在する場合には、削除する
        DELETE DOCUMENT($vsDocPathName+".HTM")
        `再度、作成する
        If($vbOnWindows)
                $vhDocRef:=Create document($vsDocPathName ; "HTM")
        Else
                $vhDocRef:=Create document($vsDocPathName+".HTM")
        End if
        If(OK=1)
                ` ...
                ` ここで、ドキュメントの内容を記述する
                ` ...
                CLOSE DOCUMENT($vhDocRef)
                If($vdModifiedOn=!00/00/00!)
                        `ドキュメントは存在しなかったため、
                        `更新の日付および時間を正しい値に設定する
                        $vdModifiedOn:=Current date
                        $vhModifiedAt:=Current time
                End if
                `ドキュメントのプロパティを変更し、作成の日付および時間
                `が対応するレコードと等しくなるようにする
                SET DOCUMENT PROPERTIES($vsDocPathName+".HTM" ;
                                                        $vbLocked
                 ; $vbInvisible ; Time stamp to date([ドキュメント]作成日) ;
                Time stamp to time([ドキュメント]作成日) ; $vdModifiedOn ;
                                                        $vhModifiedAt)
        End if
End if
` 現在実行中の処理を確認するために
SET WINDOW TITLE("ドキュメント処理中 : "+String($vlDoc)+"/"+String
```

```
            ($vlNbRecords))
        NEXT RECORD([ドキュメント])
End for
        ` 古いドキュメント、つまりまだ$asDocument配列内にあるドキュメント
                                                    を削除する
For($vlDoc ; 1 ; Size of array($asDocument))
        DELETE DOCUMENT($vsPath+$asDocument{$vlDoc})
        SET WINDOW TITLE("ドキュメント削除中 : "+Char(34)+$asDocument
        {$vlDoc}+Char(34))
End for
        ` We're done
ALERT("処理されたドキュメントの数："+String($vlNbRecords)+Char(13)+
        "更新されたドキュメントの数： "+String($vlNbDocs)+Char(13)+
        "削除されたドキュメントの数： "+String(Size of array($asDocument)))
```

参照

Document creator、Document type、SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES(ドキュメント ; ロック ; 非表示 ; 作成日 ;
作成時間 ; 更新日 ; 更新時間)

引数	タイプ		説明
ドキュメント	文字列	→	ドキュメントの名前、または ドキュメントの完全なパス名
ロック	ブール	→	ロックの場合はTrue、アンロックの場合はFalse
非表示	ブール	→	非表示の場合はTrue、表示の場合はFalse
作成日	日付	→	作成日
作成時間	時間	→	作成時間
更新日	日付	→	更新日
更新時間	時間	→	更新時間

説明

SET DOCUMENT PROPERTIESコマンドは、引数＜ドキュメント＞に渡した名前またはパス名を持つドキュメントファイルについての情報を変更します。

呼び出しの前に以下の情報を渡します。

- ドキュメントをロックするには、引数＜ロック＞にTrueを渡します。ロックされたドキュメントを、開いたり削除することはできません。ドキュメントのロックを解除するには＜ロック＞にFalseを渡します。
- ドキュメントを隠すには、引数＜非表示＞にTrueを渡します。デスクトップウインドウでドキュメントが表示されるようにするには、＜非表示＞にFalseを渡します。
- 引数＜作成日＞および＜作成時間＞に、ドキュメントの作成日および作成時間を渡します。
- 引数＜更新日＞および＜更新時間＞に、最新のドキュメント更新日および更新時間を渡します。

作成および最新の更新の日付および時間は、ドキュメントを作成、またはこれにアクセスするたびに、システムのファイルマネージャによって管理されます。このコマンドを使用すると、特別な用途のためにこれらのプロパティを変更することができます。**GET DOCUMENT PROPERTIES**コマンドの例を参照してください。

参照

GET DOCUMENT PROPERTIES、SET DOCUMENT CREATOR、SET DOCUMENT TYPE

GET DOCUMENT ICON

GET DOCUMENT ICON(ドキュメントパス; アイコン{; サイズ})

引数	タイプ		説明
ドキュメントパス	文字列	→	アイコンを取得するドキュメントのショートネームまたはロングネーム、または空の文字列で標準の「ファイルを開く」ダイアログボックスを表示
アイコン	ピクチャ	→	ピクチャ変数またはフィールド
		←	ドキュメントアイコン
サイズ	倍長整数	→	返されたピクチャのサイズ（ピクセル単位）

説明

GET DOCUMENT ICONコマンドは、＜アイコン＞に指定された4Dのピクチャ変数またはピクチャフィールドに、＜ドキュメントパス＞に渡された名前を持つドキュメントファイルのアイコンを返します。ファイルはいずれのタイプでも構いません（実行形式ファイル、ドキュメント、ショートカットやエイリアス等）。ただし、このコマンドはフォルダアイコンを返しません。

＜ドキュメントパス＞にはファイルのロングネーム（フルパス名）を指定します。また、ショートファイルネームだけを渡すこともできますが、その場合必ずファイルはデータベースの現在作業しているディレクトリ（通常は、データベースストラクチャファイルのあるフォルダ）に配置しておく必要があります。

＜ドキュメントパス＞に空の文字列を渡すと、標準の「ファイルを開く」ダイアログボックスが表示されます。ここでユーザは、読み込むファイルを選択することができます。ダイアログボックが有効になると、システム変数Documentには選択したファイルへのフルパス名が納められます。

＜アイコン＞には4Dのピクチャ変数またはピクチャフィールドを渡します。コマンドの実行後、この引数にはファイルのアイコンが納められます（PICTフォーマット）。

オプションの引数＜サイズ＞を指定すると、返されるアイコンの寸法をピクセル単位で設定することができます。実際には、この値はアイコンの入っている正方形の横のサイズを表わします。通常、アイコンは32×32ピクセル（“ラージアイコン”）または16×16ピクセル（“スモールアイコン”）の大きさで定義されます。この引数に0を渡すか、または省略した場合、使用できる最大のアイコンが返されます。

Get document size

Get document size(ドキュメント {; *}) → 数値

引数	タイプ		説明
ドキュメント	時間	→	ドキュメント参照番号
	または、文字列		または、ドキュメントの名前
*	*		Macintoshのみ： 省略した場合、データフォークのサイズ 指定した場合、リソースフォークのサイズ
戻り値	数値	←	ドキュメントのサイズ（バイト単位）

説明

Get document size関数は、ドキュメントのサイズをバイト単位で表示して返します。

ドキュメントが開かれている場合には、引数＜ドキュメント＞にドキュメント参照番号を渡します。ドキュメントが開かれていない場合には、＜ドキュメント＞にドキュメントの名前またはパス名を渡します。

Macintoshでは、オプション引数＜*＞を渡さないと、データフォークのサイズが返されます。＜*＞を渡すと、リソースフォークのサイズが返されます。

参照

Get document position、SET DOCUMENT POSITION、SET DOCUMENT SIZE

SET DOCUMENT SIZE

SET DOCUMENT SIZE(ドキュメント ; サイズ)

引数	タイプ		説明
ドキュメント	時間	→	ドキュメント参照番号
サイズ	数値	→	新しいサイズ（バイト単位）

説明

SET DOCUMENT SIZEコマンドは、ドキュメントのサイズを引数＜サイズ＞に渡したバイト数に設定します。

ドキュメントが開かれている場合には引数＜ドキュメント＞にドキュメント参照番号を渡します。Macintoshでは、ドキュメントデータフォークのサイズが変更されます。

参照

Get document position、Get document size、SET DOCUMENT POSITION

Get document position

Get document position(ドキュメント参照番号) → 数値

引数	タイプ		説明
ドキュメント参照番号	時間	→	ドキュメント参照番号
戻り値	数値	←	ドキュメント開始位置からのファイル位置（バイト単位）

説明

この関数は、引数＜ドキュメント参照番号＞に指定したドキュメント参照番号を持つ、現在開かれているドキュメントに対してのみ有効です。

Get document position関数は、ドキュメントの最初から見て、以下の読み込み（**RECEIVE PACKET**コマンド）または書き込み（**SEND PACKET**コマンド）が発生する位置を返します。

参照

RECEIVE PACKET、SEND PACKET、SET DOCUMENT POSTION

SET DOCUMENT POSITION

SET DOCUMENT POSITION(ドキュメント参照番号 ; オフセット {; アンカー})

引数	タイプ		説明
ドキュメント参照番号	時間	→	ドキュメント参照番号
オフセット	数値	→	ファイルの位置（バイト単位）
アンカー	整数	→	1=ファイルの先頭からの相対位置 2=ファイルの最後からの相対位置 3=現在の位置からの相対位置

説明

このコマンドは、引数＜ドキュメント参照番号＞に指定したドキュメント参照番号を持つ、現在開かれているドキュメント対してのみ有効です。

SET DOCUMENT POSITIONコマンドは、引数＜オフセット＞に渡す、以下の読み込み（**RECEIVE PACKET**コマンド）または書き込み（**SEND PACKET**コマンド）が発生する位置を設定します。

オプション引数＜アンカー＞を省略すると、位置はドキュメントの最めから相対的に表わされます。＜アンカー＞を指定すると、ここにリストされている値のうちいずれかを渡します。

アンカーによっては、引数＜オフセット＞に正の値または負の値を渡すことができます。

参照

Get document position、RECEIVE PACKET、SEND PACKET

Screen height

Screen height {(*)} → 数値

引数	タイプ		説明
*	数値	→	Windows：アプリケーションウインドウの高さ <*>が指定されている場合、画面の高さ Macintosh：メイン画面の高さ
戻り値	数値	←	ピクセル数で表される高さ

説明

Windowsの場合、**Screen height**関数は、4Dアプリケーションウインドウ（MDIウインドウ）の高さを返します。オプション引数<*>を指定した場合、画面の高さを返します。

Macintoshの場合、**Screen height**関数はメイン画面の高さを返します。メイン画面とは、メニューバーのある画面のことです。

参照

SCREEN COORDINATES、Screen width

Screen width

Screen width {(*)} → 数値

引数	タイプ		説明
*	数値	→	Windows：アプリケーションウインドウの幅 <*>が指定されている場合、画面の幅 Macintosh：メイン画面の幅
戻り値	数値	←	ピクセル数で表される幅

説明

Windowsの場合、**Screen width**関数は、4Dアプリケーションウインドウ（MDIウインドウ）の幅を返します。オプション引数<*>を指定した場合、画面の幅を返します。

Macintoshの場合、**Screen width**関数はメイン画面の幅を返します。メイン画面とは、メニューバーのある画面のことです。

参照

SCREEN COORDINATES、Screen height

Count screens

Count screens → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	画面の数

説明

Count screens関数は、マシンに接続されている画面モニタの数を返します。

Windowsでの注意：Windowsでは、Count screens関数は常に1を返します。

参照

Menu bar screen、SCREEN COORDINATES、SCREEN DEPTH、Screen height、Screen width

SCREEN COORDINATES

SCREEN COORDINATES(左 ; 上 ; 右 ; 下 { ; 画面})

引数	タイプ		説明
左	数値	←	画面エリアの左端のグローバル座標
上	数値	←	画面エリアの上端のグローバル座標
右	数値	←	画面エリアの右端のグローバル座標
下	数値	←	画面エリアの下端のグローバル座標
画面	数値	→	画面番号、または省略した場合には主画面（メインスクリーン）

説明

SCREEN COORDINATESコマンドは、＜画面＞に指定した画面のグローバル座標を引数＜左＞、＜上＞、＜右＞、および＜下＞に返します。

Windowsの場合

通常、引数＜画面＞は指定しません。

Macintoshの場合

引数＜画面＞を省略した場合、このコマンドは主画面（メインスクリーン）、つまりメニューバーが表示されている画面の座標を返します。

参照

Count screens、Menu bar screen、SCREEN DEPTH

SCREEN DEPTH

SCREEN DEPTH(深さ ; カラー {; 画面})

引数	タイプ		説明
深度	数値	←	画面の深度（カラーの数=2^深度）
カラー	数値	←	1=カラー画面 0=白黒またはグレイスケール
画面	数値	→	画面番号、または省略した場合には 主画面（メインスクリーン）

説明

SCREEN DEPTHコマンドは、モニタについての情報を引数＜深度＞と＜カラー＞に返します。

呼び出しの後

■ 画面の深度が引数＜深度＞に返されます。画面の深度は、モニタ上で表示されるカラーの数を表す2のべき乗の指数です。例えば、モニタが256色（2^8）に設定されている場合、画面の深度は8になります。

4th Dimensionでは、以下の表のように前もって定義された定数が用意されています。

定数	タイプ	値
Black and white	倍長整数	0
Four colors	倍長整数	2
Sixteen colors	倍長整数	4
Two fifth six colors	倍長整数	8
Thousands of colors	倍長整数	16
Millions of colors 24 bit	倍長整数	24
Millions of colors 32 bit	倍長整数	32

モニタがカラーを表示するよう設定されている場合、＜カラー＞には1が返されます。モニタがグレイスケールを表示するよう設定されている場合、＜カラー＞には0（ゼロ）が返されます。この値は、Macintoshプラットフォーム上で重要であることに注意してください。

4th Dimensionでは、以下の表のように前もって定義された定数が用意されています。

定数	タイプ	値
Is gray scale	倍長整数	0
Is color	倍長整数	1

■ オプション引数＜画面＞には、情報を得たいモニタを指定します。Windowsでは、通常、引数＜画面＞は使用しません。Macintoshでは、引数＜画面＞を省略すると、このコマンドは、主画面（メインスクリーン）、つまり、メニューバーが表示されている画面の深度を返します。

例題

アプリケーションが多くのカラーグラフィックスを表示するとします。その場合には、データベースのどこかに以下のように記述することができます。

```
SCREEN DEPTH($vlDepth ; $vlColor)
If($vlDepth<8)
    ALERT("モニタのカラー表示を256色以上に設定すれば、
                                フォームの表示がよくなります。")
End if
```

参照

Count screens、SET SCREEN DEPTH

SET SCREEN DEPTH

SET SCREEN DEPTH(深さ ; カラー { ; 画面})

引数	タイプ		説明
深度	数値	←	画面の深度（カラーの数=2^深度）
カラー	数値	←	1=カラー画面 0=グレイスケール
画面	数値	→	画面番号、または省略した場合には 主画面（メインスクリーン）

説明

このコマンドは、Windows上では何も行いません。

Macintosh上では、**SET SCREEN DEPTH**コマンドは引数＜画面＞に渡した番号を持つ画面の深度とカラー／グレイスケールの設定を変更します。引数＜画面＞を省略した場合には、このコマンドは主画面（メインスクリーン）に対して適用されます。

引数＜カラー＞と＜深度＞に渡す値についての詳細は、**SCREEN DEPTH**コマンドの説明を参照してください。

参照

SCREEN DEPTH

Menu bar screen

Menu bar screen → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	メニューバーが表示されている画面の番号

説明

Menu bar screen関数は、メニューバーが表示されている画面の番号を返します。

Windowsでの注意：Windowsでは、Menu bar screen関数は常に1を返します。

参照

Count screens、Menu bar height

Menu bar height

Menu bar height → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	メニューバーの高さ（ピクセル単位）（メニューバーが表示されていない場合には0）

説明

Menu bar height関数は、メニューバーの高さをピクセル数で返します。

参照

HIDE MENU BAR、Menu bar height、SHOW MENU BAR

FONT LIST

FONT LIST(フォント)

引数	タイプ		説明
フォント	配列	→	フォント名の配列

説明

FONT LISTコマンドは、文字列またはテキスト配列の引数＜フォント＞に、システム上で使用可能なフォントの名前を格納します。

例題

フォーム上に、システム上で使用可能なフォントリストを表示するドロップダウンリストを作成したいとします。その場合、以下のようなドロップダウンリストのメソッドを記述します。

```
Case of
    ¥(Form event=On Load)
        ARRAY STRING(63 ; asFont ; 0)
        FONT LIST(asFont)
        ...
End case
```

参照

Font name、Font number

Font name

Font name(フォント番号) → 文字列

引数	タイプ		説明
フォント番号	数値	→	フォント名を返すフォント番号
戻り値	文字列	←	フォント名

説明

Font name関数は、＜フォント番号＞に指定した番号を持つフォントの名前を返します。指定された番号を持つ使用可能なフォントがない場合には、このコマンドは空の文字列を返します。

例題

1. デフォルト（省略時）のシステムフォントを使用してフォームオブジェクトを表示したい場合には、以下のように記述します。

```
FONT(myObject ; Font name(0)) `0はデフォルトのシステムフォントの番号です。
```

2. デフォルトのアプリケーションフォントを使用してフォームオブジェクトを表示したい場合には、以下のように記述します。

```
FONT(myObject; Font name(1)) `1はデフォルトのアプリケーションフォントの
                                番号です。
```

参照

FONT LIST、Font number

Font number

Font number(フォント名) → 数値

引数	タイプ		説明
フォント名	文字列	→	フォント番号を返すフォント名
戻り値	数値	←	フォント番号

説明

Font number関数は、＜フォント名＞に指定したフォント名を持つフォント番号を返します。指定された名前を持つフォントがない場合には、このコマンドは0を返します。

例題

データベースのフォームのいくつかが「特殊」という名前のフォントを使用するとします。その場合、データベースのどこかに、以下のように記述することができます。

```
If(Font number("特殊")=0)
    ALERT("特殊フォントをインストールすれば、このフォームの表示が
                                                よくなります。")
End if
```

参照

FONT LIST、Font name

PLATFORM PROPERTIES

PLATFORM PROPERTIES(プラットフォーム{; システム{; マシン}})

引数	タイプ		説明
プラットフォーム	数値	←	1　68KベースのMacintosh
			2　Power Macintosh
			3　Windows
システム	数値	←	動作している種類により異なる
マシン	数値	←	動作している種類により異なる

説明

PLATFORM PROPERTIESコマンドは、動作しているプラットフォームのタイプ、オペレーションシステム（OS）のバージョン、使用しているマシンに搭載されたプロセッサに関する情報を返します。

PLATFORM PROPERTIESコマンドは、引数＜プラットフォーム＞、＜システム＞、＜マシン＞に4D環境の情報を返します。

＜プラットフォーム＞は、ユーザが実行している4th Dimensionのバージョンが68KのMacintosh、Power Macintosh、Windowsのいずれであるかを示します。この引数は、次の定義済定数のいずれかを返します。

定数	タイプ	値
Macintosh 68K	倍長整数	1
Power Macintosh	倍長整数	2
Windows	倍長整数	3

＜システム＞と＜マシン＞に返される情報は、動作している4th Dimensionの種類によって異なります。

Macintosh版

Mac OS版の4th Dimensionを実行している場合、引数＜システム＞には32ビット（倍長整数）値が返されます。このうち上位16ビットは未使用、下位16ビットは次のような構造になります。

■ 上位バイトには、主要なバージョン番号が納められます。

■ 下位バイトは、2つの部分（各4ビット）に分かれています。上位部分はアップデートの主要なバージョン番号で、下位部分はアップデート番号の枝番です。例えばSystem9.0.4は$0904としてコード化され、10進数では2308になります。

注：4Dでは、%（モジューロ）と//（整数値を返す除算）数値演算子、または「ビットワイズ」演算子を使用して、これらの値を求めることができます。

Mac OSのメインバージョン番号を調べるには、次の式を使用します。

```
PLATFORM PROPERTIES($vlPlatform;$vlSystem)
$vlResult:=$vlSystem//256
    `$vlResult= 8の場合--> Mac OS 8.xで動作しています
    `$vlResult= 9の場合--> Mac OS 9.xで動作しています
    `$vlResult= 16の場合--> Mac OS 10.xで動作しています
```

Windows版

Windows版の4th Dimensionを実行している場合、引数＜システム＞には32ビット（倍長整数）値が返され、そのビットとバイトは次のような構造になります。

上位レベルのビットが0になっている場合、Windows NT4またはWindows 2000が動作していることを示します。ビットが1になっている場合は、Windows 95またはWindows 98が動作していることを意味します。

注：上位レベルのビットは倍長整数値の符号を決定します。したがって、4Dにおいて、ユーザはその値が正か負かを調べる必要があるだけです。正であればWindows NT、Windows 2000、またはWindows XPが動作しています。また、「ビットワイズ」演算子を使用することもできます。

下位バイトには、Windowsの主要なバージョン番号が納められています。4が返された場合、Windows 95、98もしくはWindows NTが動作中であることを示します。5が返された場合、Windows 2000またはWindows XPが動作しています。いずれの場合も、この値の符号により、NTまたは2000が動作しているかどうかを知ることができます。

次の下位バイトには、Windowsのバージョン番号の枝番が納められます。Windows 95が動作中である場合、この値は0になります。

注：4Dでは、%（モジューロ）と//（整数値を返す除算）数値演算子、または「ビットワイズ」演算子を使用して、これらの値を求めることができます。

■ 引数＜マシン＞には、次の定義済定義のいずれかと比較できる値が返されます。

定数	タイプ	数値
IINTEL 386	倍長整数	386
INTEL 486	倍長整数	486
Pentium	倍長整数	586
PowerPC 601	倍長整数	601
PowerPC 603	倍長整数	603
PowerPC 604	倍長整数	604

PowerPC G3	倍長整数	510
Other G3 and above	倍長整数	406

注：Macintoshの数値に関するアップデート一覧は、Apple Computer,Inc.よりDeveloperおよびTechnicalドキュメント上で公開されます。Apple社やその他のメーカーが新しいMacintoshモデルを発表した場合には、新しい値が追加される可能性があります。

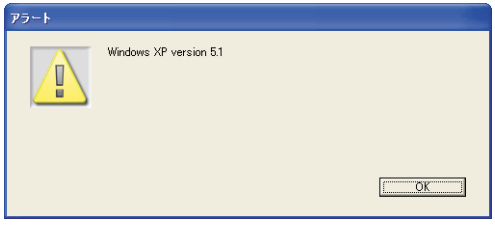
例題

次のプロジェクトメソッドは、使用しているOSを示した警告ボックスを表示します。

```
PLATFORM PROPERTIES($vlPlatform;$vlSystem;$vlMachine)
If(($vlPlatform<1) ¦($vlPlatform>3))
	$vsPlatformOS:=""
Else
	If($vlPlatform=3)
		$vsPlatformOS:=""
		If($vlSystem<0)
			$winMajVers:=((2^31)+$vlSystem)%256
			$winMinVers:=(((2^31)+$vlSystem)//256)%256
			If($winMinVers=0)
				$vsPlatformOS:="Windows 95"
			Else
				$vsPlatformOS:="Windows 98"
			End if
		Else
			$winMajVers:=$vlSystem%256
			$winMinVers:=($vlSystem//256)%256
			Case of
				¥($winMajVers=4)
					$vsPlatformOS:="Windows NT"
				¥($winMajVers=5)
					If($winMinVers=0)
						$vsPlatformOS:="Windows 2000"
					Else
						$vsPlatformOS:="Windows XP"
					End if
			End case
		End if
		$vsPlatformOS:=$vsPlatformOS+" version "
				+String($winMajVers)+"."+String($winMinVers)
	Else
		$vsPlatformOS:="Mac OS version "
```

```
                If(($vlSystem//256)=16)
                        $vsPlatformOS:=$vsPlatformOS+"10"
                Else
                        $vsPlatformOS:=$vsPlatformOS+String($vlSystem//256)
                End if
                        $vsPlatformOS:=$vsPlatformOS+"."
                          +String(($vlSystem//16)%16)+(("."+String($vlSystem%16))
                                                        *Num(($vlSystem%16)#0))
        End if
End if
ALERT($vsPlatformOS)
```

Windows上では、次の図のような「警告」ボックスが表示されます。



Macintosh上では、次の図のような「警告」ボックスが表示されます。



参照

ビットワイズ演算子

System folder

System folder→文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	稼働しているシステムディレクトリまたはフォルダへのパス名

説明

System folder関数は、アクティブなWindowsまたはMacintoshシステムフォルダにあるシステムフォルダへのパス名、あるいはアクティブなWindowsまたはMacintoshシステムフォルダ自体へのパス名を返します。

オプションの引数＜タイプ＞には、システムフォルダのタイプを示す値を指定します。以下の定義済み定数が、定数テーマ「システムフォルダ」として4Dより提供されています。

定数	タイプ	値
System	倍長整数	0
Fonts	倍長整数	1
Preferences or Profiles(All Users)	倍長整数	2
Preferences or Profiles(Current User)	倍長整数	3
Startup Items(All Users)	倍長整数	4
Startup Items(Current User)	倍長整数	5
Mac Shutdown Items(All Users)	倍長整数	6
Mac Shutdown Items(Current User)	倍長整数	7
Apple or Start Menu(All Users)	倍長整数	8
Apple or Start Menu(Current User)	倍長整数	9
Mac Extensions	倍長整数	10
Mac Control Panels	倍長整数	11
System Win	倍長整数	12
System32 Win	倍長整数	13
Favorites Win	倍長整数	14
Desktop Win	倍長整数	15
Program Files Win	倍長整数	16

システムフォルダのなかには、そのパス名がカレントユーザに特定されているものがあります。定数の2から9を使用すると、取得しようとするパス名が、すべてのユーザで共有のフォルダのものか、カレントユーザ用にカスタマイズされたフォルダへのものかを選択できます。

注：定数「Mac Shutdown Items」、「Mac Extensions」、「Mac Control Panels」は、Mac OSでのみ使用できます。これらの定数をWindowsで使用すると、System folder関数は空の文字列を返します。

また逆に、定数「System Win」、「System32 Win」、「Favorites Win」、「Desktop Win」、「Program Files Win」はWindowsでのみ使用できます。これらの定数をMac OSで使用すると、**System folder**関数は空の文字列を返します。

引数<タイプ>を省略すると、この関数はアクティブなシステムフォルダへのパス名を返します（= System定数）。

Temporary folder

Temporary folder→ 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	テンポラリ（中間）フォルダへのパス名

説明

関数は、システムによって設定される現在（カレント）のテンポラリ（中間）フォルダへのパス名を返します。

例題

▼ コマンドの例を参照してください。

参照

System folder

Current machine

Current machine → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	マシンのネットワーク名

説明

Current machine関数は、「ネットワーク」コントロールパネルで設定されたマシンのネットワーク名を返します。

例題

クライアント／サーバ版の4D環境でアプリケーションを実行していない場合でも、アプリケーションに含まれているいくつかのネットワークサービスが、マシンが正しく構成されていることを必要とする可能性があります。アプリケーションのOn Startupデータベースメソッドの中で、以下のように記述します。

```
If((Current machine="") |(Current machine owner=""))
    `マシンのネットワークIDを設定するようユーザに要求するダイアログ
                                        ボックスを表示する。
End if
```

参照

Current machine owner

Current machine owner

Current machine owner→ 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	マシンオーナーのネットワーク名

説明

Current machine owner関数は、「ネットワーク」コントロールパネルで設定されたマシンのオーナー名を返します。

例題

Current machine関数の例を参照してください。

参照

Current machine

Gestalt

Gestalt(セレクタ ; 値)→ 文字列

引数	タイプ		説明
セレクタ	文字列	→	4文字のGestaltセレクタ
値	数値	←	gestalt値
戻り値	数値	←	エラーコード

説明

Gestalt関数は、ユーザが＜セレクタ＞に渡すセレクタに基づいて、システムハードウェアおよびソフトウェアの特性を示す数値を＜値＞に返します。

必要な情報が得られると、**Gestalt**関数の結果として0を、取得できなかった場合には、エラーコード-5550を返します。セレクタがわからなければ、**Gestalt**関数はエラーコード-5551を返します。

重要：GestaltマネージャはMacintoshの一部です。セレクタのいくつかはWindows上でも実現されていますが、このコマンドの有効性はWindows上では限られています。

Gestalt関数に渡すことができるセレクタについての詳細は、Gestaltマネージャに関するApple社の開発者向け（Developer）ドキュメントを参照してください。

オンラインでのアドレスは、以下の通りです。

http://developer.apple.com/techpubs/macosx/Carbon/oss/GestaltManager/gestaltmanager.html

例題

Machintosh上で、Mac OSのバージョン7.6を使用している場合、以下のコードは、「システムバージョン0x0760を実行しています。」という警告を表示します。

```
$vlErrCode:=Gestalt("sysv" ; $vlInfo)
If($vlErrCode=0)
        ALERT("システムバージョン："+String($vlInfo ; "&x")
                                                + "を実行しています。")
End if
```

LOG EVENT

LOG EVENT(メッセージ {; 重要度 })

引数	タイプ		説明
メッセージ	文字列	→	メッセージの内容
重要度	整数	←	メッセージの重要度レベル

説明

注：この機能はWindowsでのみ可能です。

このコマンドは、Windowsの"Log events"を利用する事ができるようになりました。このログファイルは、実行されているアプリケーションから来るメッセージを受け取り保存します。したがって、ワークセッションの動向を管理することができるようになります。より詳しい情報は、『4th Dimensionデザインリファレンス』マニュアルを参照してください。

注：この機能を使用するには、Windowsの"Log events"サービスを開始する必要があります。

LOG EVENTコマンドは、WindowsNTの"Log events"に表示されるカスタムメッセージを追加することができるようにします。

メッセージにログイベント内に書くメッセージを渡します。

メッセージに重要度を付けることができ、ログイベントを読んだり解析する助けになります。重要度には、「情報」、「警告」および「エラー」の3つのレベルがあります。重要度引数には、メッセージの重要度レベルを設定します。

4th Dimensionには、前もって定義された以下のような定数があります。これらは"Windows NT Log Events"カテゴリーに置かれています。

定数	タイプ	値
Information Message(デフォルト値)	整数	0
Warning Message	整数	1
Sixteen colors	整数	2

重要度が省略されたり正しくない値を指定すると、デフォルト値（0）が使用されます。

例題

▼ データベースが開かれた時の情報をログしたい場合は、下記のコードをwStartupデータベースメソッド内に書きます。

```
LOG EVENT("請求書 DB オープン ")
```

データベースが開かれるたびに、この情報はWindowsのログイベントに重要度レベルが0として書き込まれます。

SET ENVIRONMENT VARIABLE

SET ENVIRONMENT VARIABLE(変数名; 変数値)

引数	タイプ		説明
変数名	文字列	→	設定する変数名
変数値	文字列	→	変数の値、または デフォルト値にリセットする場合は""

説明

このコマンドを使用し、Mac OS XやWindowsにおける環境変数の値を設定することができます。このコマンドは、**SET CGI EXECUTABLE**コマンドまたは**LAUNCH EXTERNAL PROCESS**コマンドとともに使用する必要があります。

＜変数名＞には定義する変数の名前を、＜変数値＞にはその値を渡します。

■ 環境変数とその値の総一覧を取得する方法については、お使いのOSのテクニカルドキュメントを参照してください。

■ **SET CGI EXECUTABLE**コマンドで使用できる環境変数の一覧を調べるには、CGIに関する4Dのドキュメントを参照してください。

■ **LAUNCH EXTERNAL PROCESS**コマンドで使用するために、2つの新しい環境変数、_4D_OPTION_CURRENT_DIRECTORYと_4D_OPTION_HIDE_CONSOLEを利用できるようになりました。カレントプロセスにおいて次に**LAUNCH EXTERNAL PROCESS**コマンドを呼び出す際に、これらの変数が有効になります。

- _4D_OPTION_CURRENT_DIRECTORY：起動する外部プロセスのカレントディレクトリを設定するために使用します。＜変数値＞には、このディレクトリのパス名を渡さなければなりません（Mac OSではHFSタイプのシンタックス、WindowsではDOSタイプ）。

- _4D_OPTION_HIDE_CONSOLE（Windowsのみ）：DOSコンソールのウインドウを隠すために使用します。＜変数値＞には、コンソールを隠す場合は“true”を、表示する場合は“false”を渡してください。

例題

LAUNCH EXTERNAL PROCESSコマンドの例題7、8を参照してください。

参照

LAUNCH EXTERNAL PROCESS、SET CGI EXECUTABLE、CGIの使用

LAUNCH EXTERNAL PROCESS

LAUNCH EXTERNAL PROCESS(ファイル名{; 入力ストリーム{; 出力ストリーム{;エラーストリーム}}})

引数	タイプ		説明
ファイル名	文字列	→	起動するファイルのファイルパスと引数
入力ストリーム	文字列	→	入力ストリーム（stdin）
出力ストリーム	文字列	←	出力ストリーム（stdout）
エラーストリーム	文字列	←	エラーストリーム（stderr）

注：このコマンドは、上級ユーザを対象に作成されています。

説明

LAUNCH EXTERNAL PROCESSコマンドを使用して、Mac OS XやWindows上の4th Dimensionから外部プロセスを起動することができます。

Mac OS Xにおいて、このコマンドは「ターミナル」から起動できるあらゆるアプリケーションへのアクセスを提供します。

注：4D Packユーザの場合、このコマンドはAP_Sublaunchコマンドと同じ機能を持ちます。

引数＜ファイル名＞には、実行するアプリケーションの固定ファイルパス、ならびに必要となる引数（必要な場合）を渡します。

Mac OS X上では、アプリケーション名だけを渡すこともできます。4th DimensionはPATH環境変数を使用して、実行形式ファイルの場所を探し出します。

警告：このコマンドは実行可能なアプリケーションのみを開始します。シェル（コマンドインタプリタ）の指示の一部を実行することはできません。例えば、Mac OS上でこのコマンドを使いエコーや間接参照を実行することはできません。

引数＜入力ストリーム＞（任意）には、外部プロセスのstdinを納めます。コマンドが実行されると、引数＜出力ストリーム＞と＜エラーストリーム＞（渡した場合）には、外部プロセスのstdoutとstderrがそれぞれ返されます。テキストデータのサイズが32KBより大きい場合、もしくはバイナリデータ（例えば画像）の場合、文字列の代わりにBLOBパラメータを使うことができます。

Mac OS Xの例題

次の例題は、「アプリケーション/ユーティリティ」フォルダにあるMac OS Xターミナルを使用します。

(1)ファイルに対するパーミッションを変更するには、次のように記述します（chmodはMac OS Xのコマンドであり、ファイルアクセスを変更する際に使用します）。

```
LAUNCH EXTERNAL PROCESS("chmod+x/folder/myfile.txt")
```

(2)テキストファイルを編集するには、次のように記述します（catはMac OS Xのコマンドであり、ファイルの編集を行う）。この例では、コマンドへのフルアクセスパスを渡しています。

```
C_TEXT(input;output)
input :=""
LAUNCH EXTERNAL PROCESS("/bin/cat/folder/myfile.txt"; input;output)
```

(3)「Users」フォルダの内容を取得するには、次のように記述します（ls-lはMac OS Xのコマンドであり、DOSのdirコマンドに相当）。

```
C_TEXT($in;$out)
LAUNCH EXTERNAL PROCESS("/bin/ls-l/Users";$in;$out)
```

Windowsの例題

(4)ノートパッドを開くには、次のように記述します。

```
LAUNCH EXTERNAL PROCESS("C:¥¥WINDOWS¥¥notepad.exe")
```

(5)ノートパッドを開いて特定のドキュメントをオープンするには、次のように記述します。

```
LAUNCH EXTERNAL PROCESS("C:¥¥WINDOWS¥¥notepad.exe
                                C:¥¥Docs¥¥new folder¥¥res.txt")
```

(6)Microsoft®Word®アプリケーションを起動し、特定のドキュメントをオープンするには、次のように記述します（""が2回使用されている点に注意）。

```
$mydoc:="C:¥¥Program Files¥¥Microsoft Office¥¥Office10
                ¥¥WINWORD.EXE¥"C:¥¥Documents and Settings¥¥Bob¥¥Desktop
                                ¥¥My Documents¥¥New folder¥¥test.xml¥""
LAUNCH EXTERNAL PROCESS($mydoc;$tIn;$tOut)
```

(7)Perlスクリプトを実行するには、次のように記述します（ActivePerlが必要）。

```
C_TEXT($input;$output)
SET ENVIRONMENT VARIABLE("myvariable";"value")
LAUNCH EXTERNAL PROCESS("D:¥¥Perl¥¥bin¥¥perl.exe D:¥¥Perl¥¥eg
                                ¥¥cgi¥¥env.pl";$input;$output)
```

(8)コンソールを開かずに特定のディレクトリでコマンドを実行するには、次のように記述します。

```
SET ENVIRONMENT VARIABLE("_4D_OPTION_CURRENT_DIRECTORY";
                                                    "C:¥¥4D_VCS")
SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
LAUNCH EXTERNAL PROCESS("mycommand")
```

参照

SET ENVIRONMENT VARIABLE

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は（ファイルが見つからない、メモリが不十分など）0が代入されます。

DEFAULT TABLE

DEFAULT TABLE(テーブル)

引数	タイプ		説明
テーブル	テーブル	→	デフォルトとして設定するテーブル

説明

DEFAULT TABLEコマンドは、＜テーブル＞をカレントプロセスのデフォルトテーブルとして設定します。

DEFAULT TABLEコマンドが実行されるまで、デフォルトテーブルは存在しません。デフォルトテーブルを設定した後で、＜テーブル＞を省略したコマンドはデフォルトテーブルに対して実行されます。例えば、以下のコマンドを見てください。

```
INPUT FORM([テーブル] ; "フォーム")
```

デフォルトテーブルで[テーブル]を設定した場合に、以下のような同じコマンドの別の記述が可能です。

```
INPUT FORM("フォーム")
```

デフォルトテーブルの設定のもう1つの目的は、テーブルに特定されないステートメントを作成することです。これによって、同じステートメントで異なるテーブルを操作することができます。また、テーブルへのポインタを使用して、テーブルに特定されないコードを作成することもできます。この手法に関する詳細は、**Table name**コマンドの説明を参照してください。

DEFAULT TABLEコマンドではフィールドを参照する場合に、テーブル名を省略することはできません。例えば、以下のように記述します。

```
[MyTable]MyField:="設定"    `正しい記述
```

以下のように記述することはできません。

```
DEFAULT TABLE([MyTable])
MyField:="設定" `誤った記述
```

これは、単にデフォルトテーブルが設定されるだけです。しかし、フォームメソッド、オブジェクトメソッドにおいてそれに属するテーブルのフィールドを参照する場合は、テーブル名を省略することができます。

4th Dimensionでは、すべてのテーブルは“開かれて”おり、使用する準備ができています。しかし、**DEFAULT TABLE**コマンドはテーブルを“開いたり”、カレントテーブルを設定、あるいは入出力のためにテーブルを準備することはありません。**DEFAULT TABLE**コマンドはプログラミングの労力の節約とステートメントを読みやすくするための便宜を図るだけです。

TIP：DEFAULT TABLEコマンドの使用やテーブル名の省略により、ステートメントを読みやすくすることができるかもしれませんが、多くのプログラマーはこのコマンドが実際の価値よりは多くの問題と混乱の原因となるとみなしています。

例題

▼ 以下の例は、最初に**DEFAULT TABLE**コマンドを使用しないステートメントを示しています。この後で、**DEFAULT TABLE**コマンドを使用した同じステートメントを示します。このステートメントは、新しいレコードをデータベースに追加するのによく使用されるループです。**INPUT FORM**コマンドと**ADD RECORD**コマンドは、1番目の引数としてテーブルを必要とします。

```
INPUT FORM([顧客] ; "住所")
Repeat
      ADD RECORD([顧客])
Until(OK=0)
```

デフォルトテーブルの指定により、以下のメソッドが導かれます。

```
DEFAULT TABLE([顧客])
INPUT FORM("住所")
Repeat
      ADD RECORD
Until(OK=0)
```

参照

Current default table

Current default table

Current default table → ポインタ

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ポインタ	←	デフォルトテーブルへのポインタ

説明

Current default table関数は、カレントプロセスに対して**DEFAULT TABLE**コマンドで最後に指定されたテーブルのポインタを返します。

例題

デフォルトテーブルが設定されているものとして、以下のコードはカレントデフォルトテーブルの名前をウインドウタイトルにセットします。

```
SET WINDOW TITLE(Table name(Current default table))
```

参照

DEFAULT TABLE、Table、Table name

Current form table

Current form table → ポインタ

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ポインタ	←	現在表示されているフォームが属すテーブルへのポインタ

説明

Current form table関数は、カレントプロセスで表示または印刷されているフォームが属するテーブルのポインタを返します。

カレントプロセスに表示または印刷されているフォームがない場合には、このコマンドはNilを返します。

カレントプロセスで複数のウインドウが開いている（最後に開かれたウインドウがカレントアクティブウインドウになる）場合には、このコマンドは、アクティブウインドウにフォームが表示されているテーブルへのポインタを返します。

現在表示されているフォームがサブフォームエリア用の「詳細」フォームである場合には、ユーザがデータ入力中であり、ダブルクリック可能なサブフォームエリアのレコードまたはサブレコードをダブルクリックしたことを意味します。この場合には、コマンドは以下を返します。

- サブフォームがテーブルを表示している場合には、サブフォームエリアに表示されたそのテーブルへのポインタ
- サブフォームエリアがサブテーブルを表示している場合には、重要な意味を持たないポインタ

例題

アプリケーション全体を通して、レコードを表示する際には、以下の表示方法に従います。フォーム内に変数「vsCurrentRecord」がある場合、ユーザが新しいレコードを処理していれば、「新規レコード」と表示します。ユーザが5200レコードから成るセレクションの56番目のレコードを処理していれば、「56／5200」と表示します。

そのためには、オブジェクトメソッドを使用して変数「vsCurrentRecord」を作成し、その後、このオブジェクトメソッドをコピーして、すべてのフォームに貼り付けます。

`「vsCurrentRecord」入力不可変数のオブジェクトメソッド

```
Case of
    ¥(Form event =On Load)
        C_STRING(31 ; vsCurrentRecord)
        C_POINTER($vpParentTable)
        C_LONGINT($vlRecordNum)
        $vpParentTable:=Current form table
        $vlRecordNum:=Record number($vpParentTable->)
    Case of
        ¥($vlRecordNum=-3)
            vsCurrentRecord:="新規レコード"
        ¥($vlRecordNum=-1)
            vsCurrentRecord:="レコードなし"
        ¥($vlRecordNum>=0)
            vsCurrentRecord:=String(Selected record number
                ($vpParentTable->))+" of "+String(Records in selection
                                                ($vpParentTable->))
    End case
End case
```

参照

DIALOG、INPUT FORM、OUTPUT FORM、PRINT SELECTION

HIDE TOOL BAR

HIDE TOOL BAR

引数	タイプ	説明

このコマンドには、引数はありません。

説明

HIDE TOOL BARコマンドは、ツールバーを隠します。ツールバーがすでに隠れている場合は、**HIDE TOOL BAR**コマンドは何も行いません。

参照

HIDE MENU BAR、SHOW MENU BAR、SHOW TOOL BAR

SHOW TOOL BAR

SHOW TOOL BAR

引数	タイプ	説明

このコマンドには、引数はありません。

説明

SHOW TOOL BARコマンドは、ツールバーを表示します。ツールバーがすでに表示さている場合は、**SHOW TOOL BAR**コマンドは何も行いません。

参照

HIDE MENU BAR、HIDE TOOL BAR、SHOW TOOL BAR

BUILD APPLICATION

BUILD APPLICATION{(プロジェクト名)}

引数	タイプ		説明
プロジェクト名	文字列	→	使用するプロジェクトのフルアクセスパス

説明

BUILD APPLICATIONコマンドは、アプリケーション生成プロセスを開始します。現行のアプリケーションプロジェクト、または引数＜プロジェクト名＞で指定されたアプリケーションプロジェクト内に設定されたパラメータが考慮されます。

アプリケーションプロジェクトはXMLファイルであり、アプリケーション生成に使用されるすべてのパラメータが格納されます。パラメータの大半は、「アプリケーションビルド...」ダイアログボックス上に表示されます（詳細は、前述の「アプリケーションビルダ」の節を参照）。

デフォルトとして、4th Dimensionは各データベースに対し“buildapp.xml”という名前のアプリケーションプロジェクトを作成し、それをデータベースの「Preferences」フォルダ内の「BuildApp」サブフォルダに配置します。

データベースがコンパイルされていない場合や、コンパイルされたコードが古い場合、コマンドはまず最初にコンパイラプロセスを開始します。その際、コンパイラウインドウは表示されず（エラー発生時を除く）、進捗バーだけが表示されます。

任意の引数＜プロジェクト名＞を渡さない場合、コマンドは現行のアプリケーションプロジェクトに指定されたパラメータを使用します。ダイアログボックスは表示されません。ダイアログボックスが有効な場合、システム変数＜Document＞には開いているプロジェクトファイルのフルパス名が含まれます。

有効なアプリケーションプロジェクト用のXMLファイル（拡張子“.xml”）のアクセスパスと名前を渡した場合、コマンドはこのファイルに指定されたパラメータを使用します。アプリケーションプロジェクトのXMLファイルにおいて使用可能な構造とキーに関する詳細は、4th Dimensionの『XMLキー』マニュアル（HTML形式で提供）を参照してください。

例題

▼ この例題は、1つのメソッド内で2つのアプリケーションを構築します。

```
BUILD APPLICATION("c:¥¥folder¥¥projects¥¥myproject1.xml")
If(OK=1)
    BUILD APPLICATION("c:¥¥folder¥¥projects¥¥myproject2.xml")
End if
```

システム変数またはセット

コマンドが正常に実行されると、システム変数OKに1が代入され、それ以外の場合は0が代入されます。後者の場合はエラーが生成され、このエラーは**ON ERR CALL**コマンドを用いて遮ることができます。

ENCODE

ENCODE(blob)

引数	タイプ		説明
blob	BLOB	→	Base64フォーマットにエンコードするBLOB
		←	Base64フォーマットにエンコードされたBLOB

説明

ENCODEコマンドは、BLOBをBase64フォーマットにエンコードします。コマンドは、引数として渡された＜blob＞を直接修正します。

Base64エンコーディングは、7桁を超える有効ビットを保持しないように、8ビットにコード化されたデータを修正します。例えば、XMLを用いたBLOBの処理にはこのエンコードが必要になります。

参照

DECODE

DECODE

DECODE(blob)

引数	タイプ		説明
blob	BLOB	→	Base64フォーマットにエンコードされたBLOB
		←	デコードされたBLOB

説明

DECODEコマンドを使用すると、引数＜blob＞に渡されたBase64フォーマットにエンコード済みのBLOBをデコードすることができます。コマンドは、引数として渡された＜blob＞を直接修正します。

コマンドは＜blob＞の内容を検証しません。渡したデータが実際にBase64フォーマットでエンコードされているかどうかを確認しなくてはなりません。これを行わない場合は結果が正しくなくなります。

参照

ENCODE

SPELL CHECKING

SPELL CHECKING

引数	タイプ	説明

このコマンドに引数は必要ありません。

説明

SPELL CHECKINGコマンドをは、現在表示中のフォーム上でフォーカスを取得しているフィールドや変数のスペルチェックを開始します。チェックされるオブジェクトは、文字列またはテキストタイプでなくてはなりません。

スペルチェックは、フィールドや変数の最初の語句から開始します。未知の語句が見つかると、スペルチェックダイアログボックスが表示されます（詳細については、『デザインリファレンス』マニュアルを参照）。**SET DICTIONARY**コマンドを使用しない限り、4th Dimensionはカレント辞書（アプリケーションの言語に対応）を使用します。

参照

SET DICTIONARY

SET DICTIONARY

SET DICTIONARY(辞書)

引数	タイプ		説明
辞書	倍長整数	→	スペルチェックに使用する辞書

説明

SET DICTIONARYコマンドは、カレント辞書を引数<辞書>で指定された辞書で置き換えます。カレント辞書は、4th Dimensionの統合スペルチェック機能で使用されます（詳細は、前述の「スペルチェック」の節を参照）。

デフォルトとして、4th Dimensionはアプリケーション言語に対応する辞書を使用します。フランス語、英語、ドイツ語、スペイン語の4つの辞書が使用できます。

<辞書>には、使用する辞書の番号を渡します。「Dictionaries」テーマ内にある次の定義済み定数のいずれかを使用することができます。

定数	タイプ	値
English Dictionary	倍長整数	69632
German Dictionary	倍長整数	131584
Spanish Dictionary	倍長整数	196608
French Dictionary	倍長整数	262144

また、それぞれ4つの主言語について様々な変数を利用することができます。コマンドによりサポートされているすべての変数は以下のとおりです。変数を読み込むには引数<辞書>に直接その値を渡します。

辞書	値
English(United Kingdom)：英語（英国）	65536
English Irish(Ireland)：アイルランド英語（アイルランド）	65600
English Australian(Australia)：オーストラリア英語（オーストラリア）	65664
English of New Zealand：ニュージーランド英語	65680
English American(USA)：アメリカ英語（米国）	65792
English Canadian(Canada)：カナダ英語（カナダ）	65920
English South African(South Africa)：南アフリカ英語（南アフリカ）	66048
English West Indian(Caribbean)：西インド諸島英語（カリブ）	66176
English Jamaican(Jamaica)：ジャマイカ英語（ジャマイカ）	66192
English(United Kingdom+ America)：英語（英国+米国）	69632
German standard(Germany, old spelling)：標準独語（ドイツ、旧スペル）	131072
German of Luxembourg：ルクセンブルグ独語	131073
German of Austria：オーストリア独語	131088

German of Liechtenstein：リヒテンシュタイン独語	131089
German of Switzerland(old spelling)：スイス独語（旧スペル）	131104
German of South Tyrol：南チロル独語	131120
German New spelling：独語新スペル	131328
German of Switzerland New spelling：スイス独語新スペル	131360
German Old and New spelling：独語新旧スペル	131584
German of Switzerland Old and New spelling：スイス独語新旧スペル	131616
Spanish standard(Spain)：標準スペイン語	196608
Spanish of Latin America standard：中南米標準スペイン語	196864
Spanish Argentinean(Argentina)：アルゼンチンスペイン語（アルゼンチン）	196865
Spanish Bolivian(Bolivia)：ボリビアスペイン語（ボリビア）	196866
Spanish Chilean(Chile)：チリスペイン語（チリ）	196867
Spanish Columbian(Columbia)：コロンビアスペイン語（コロンビア）	196868
Spanish Cuban(Cuba)：キューバスペイン語（キューバ）	196869
Spanish Costa Rican(Costa Rica)：コスタリカスペイン語（コスタリカ）	196870
Spanish Dominican(Dominican Rep.)：ドミニカスペイン語（ドミニカ共和国）	196871
Spanish Ecuadorian(Ecuador)：エクアドルスペイン語（エクアドル）	196872
Spanish Guatemalan(Guatemala)：グアテマラスペイン語：（グアテマラ）	196873
Spanish Honduran(Honduras)：ホンジュラススペイン語（ホンジュラス）	196874
Spanish Mexican(Mexico)：メキシコスペイン語（メキシコ）	196875
Spanish Nicaraguan(Nicaragua)：ニカラグアスペイン語（ニカラグア）	196876
Spanish Panamanian(Panama)：パナマスペイン語（パナマ）	196877
Spanish Paraguayan(Paraguay)：パラグアイスペイン語（パラグアイ）	196878
Spanish Peruvian(Peru)：ペルースペイン語（ペルー）	196879
Spanish Puerto Rican(Puerto Rico)：プエルトリコスペイン語（プエルトリコ）	196880
Spanish Salvadorian(El Salvador)：エルサルバドルスペイン語（エルサルバドル）	196881
Spanish Uruguayan(Uruguay)：ウルグアイスペイン語（ウルグアイ）	196882
Spanish Venezuelan(Venezuela)：ベネズエラスペイン語（ベネズエラ）	196883
Spanish Guinean(Equatorial Guinea)：ギニアスペイン語（ギニア）	197121
France, Monaco, Valle d'Aosta：フランス、モナコ、ヴァッレ・ダオスタ州	262144
Canada：カナダ	262160
Louisiana：ルイジアナ	262161
Belgium：ベルギー	262176
Luxembourg：ルクセンブルグ	262177
Switzerland：スイス	262192
Martinique, Guadeloupe, Haiti, Guyana： マルチニク、グアドループ、ハイチ、ギアナ	262208
Reunion, Seychelles, Comoro, Mauritius： レユニオン、セイシェル、コモロ、モーリシャス	262224
Tahiti, New Caledonia, Vanuatu, etc.：タヒチ、ニューカレドニア、バヌアツ等	262240
Morocco, Algeria, Tunisia：モロッコ、アルジェリア、チュニジア	262256

French African standard：アフリカ標準仏語	262272
Benin：ベニン	262273
Burkina Faso：ブルキナファソ	262274
Burundi：ブルンジ	262275
Cameroon：カメルーン	262276
Central African Republics：中央アフリカ共和国	262277
Congo(Brazzaville)：コンゴ（ブラザビル）	262278
Democratic Republic of Congo(ex-Zaire)：コンゴ民主共和国（前ザイール）	262279
Ivory Coast：コートジボワール	262280
Djibouti：ジブチ	262281
Gabon：ガボン	262282
Guinea：ギニア	262283
Mauritania：モーリタニア	262284
Niger：ニジェール共和国	262285
Rwanda：ルワンダ	262286
Senegal：セネガル	262287
Chad：チャド	262288
Togo：トーゴ	262289

イタリックの名称表記は定数使用時にインストールされる標準辞書

参照

SPELL CHECKING

システム変数またはセット

辞書が正常にロードされると、システム変数OKに1が代入され、それ以外の場合は0が代入されてエラーが返されます。

トランザクションを使用する

トランザクションは、あるプロセスにおいてデータベースに対して行われる一連の関連したデータ更新です。トランザクションは、そのトランザクションが受け入れられるまで、データベースに恒久的には保存されません。キャンセルされたり、他の外部的な原因でトランザクションが完了できなかった場合には、更新処理の結果は保存されません。

トランザクション処理中に、プロセス内でデータベースのデータに対して行った更新はすべて、一時的なバッファにローカルに保存されます。トランザクションが**VALIDATE TRANSACTION**コマンドで受け入れられた時点で、更新されたデータが永久に保存されます。トランザクションが**CANCEL TRANSACTION**コマンドでキャンセルされた場合には、更新されたデータは保存されません。

トランザクションは一時的なレコードアドレスを処理するため、トランザクションが受け入れられたりキャンセルされると、カレントプロセスの各テーブルのセレクションは空になります。これと同じ理由から、トランザクション内での命名セレクションの使用には十分な注意が必要です。トランザクションが受け入れられたりキャンセルされると、トランザクションの処理前、または処理中に作成された命名セレクションは、正しくないレコードアドレスを含んでいる可能性があります。例えば、命名セレクションが削除されたレコードのアドレスを含んでいたり、トランザクション中に追加されたレコードの一時的アドレスを含んでいる可能性があります。また、上記の注意点はセットにも適用されます。これは、セットがレコードアドレスを使ったビットテーブルをもとに作成されるためです。

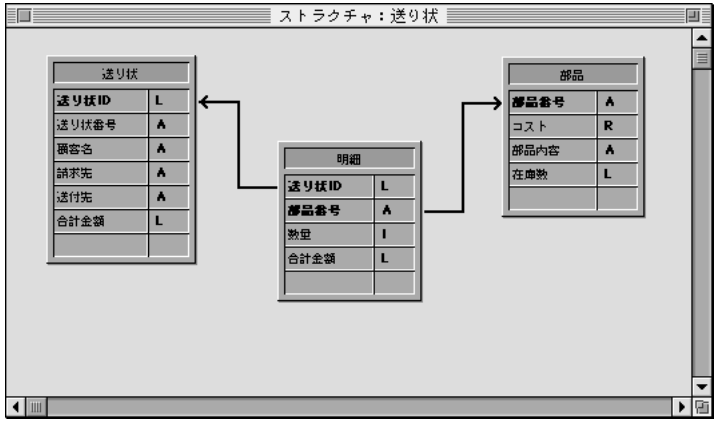
下記のコマンドは、レコード番号を使用します。トランザクションの中で使用することはできません。

GOTO RECORD
RELATE ONE SELECTION
RELATE MANY SELECTION

トランザクションの例

この例題は、簡単な送り状システムです。送り状の明細は、[明細]テーブルに格納されています。このテーブルは、[明細]テーブルの“送り状ID”フィールドと[送り状]テーブルの“送り状ID”フィールドで[送り状]テーブルにリレートしています。送り状が追加されると、重複不可のIDは**Sequence number**関数を使用して計算されます。[送り状]テーブルと[明細]テーブルのリレートは1対nの自動リレートになっており、「リレート値自動代入」チェックボックスがチェックされています。

[明細]テーブルと[部品]テーブルのリレートは、マニュアルリレートです。



ユーザは送り状を入力する場合に、以下のような動作を実行しなければなりません。

■ [送り状]テーブルにレコードを追加する

■ [明細]テーブルにレコードをいくつか追加する

■ 送り状にリストされた各部品の[部品]在庫数フィールドを更新する

この例題は、トランザクションの使用が必要になる典型的な場面の1つです。処理中に必ずこれらのレコードをすべて保存できるということや、またはレコードが追加されない場合やレコードが更新されない場合は、トランザクションをキャンセルできるということを確認する必要があります。つまり、リレートされたレコードを保存しなくてはなりません。

トランザクションを使用しない場合には、データベースの論理的なデータの整合性を保証することはできません。例えば、[部品]テーブルのレコードがロックされていると、[部品]在庫数フィールドに格納されている数量を更新することはできません。したがって、このフィールドは論理上、正しいものではなくなります。つまり、販売した部品の合計と倉庫内に残っている在庫数が、レコードに入力したオリジナルの数量と等しくならなくなります。こういう状況を避けるために、トランザクションを使用します。

トランザクションを使って、データ入力を実行する方法にはいくつかあります。

1. 「デザイン」モードの「データベースプロパティ」ダイアログボックス内にある「データ入力時に自動トランザクションを使う」チェックボックスを選択して4th Dimensionにトランザクション処理を任せることができます。この場合、4th Dimensionはトランザクションを開始して、ユーザがデータ入力を受け付けたかどうかに応じてそのトランザクションを有効にする、または取り消します。サブフォームにリレートしたテーブルを持つフォームでデータ入力作業を実行する場合は、トランザクションが必要です。このオプションはデータベース全体に適用されます。

   トランザクションを独自に処理したい場合は、**START TRANSACTION**、**VALIDATE TRANSACTION**、**CANCEL TRANSACTION**のトランザクションコマンドを使用する必要があります。

2. 以下のように記述することができます。

```
READ WRITE([明細])          ` [明細]テーブルをリードライト状態にセットする
READ WRITE([部品])          ` [部品]テーブルをリードライト状態にセットする
INPUT FORM([送り状] ; "入力")     ` 入力レイアウトを選択する
Repeat
        START TRANSACTION
        ADD RECORD([送り状])     ` 必要なだけレコードを追加する
        If(OK=1)
                VALIDATE TRANSACTION
        Else
                CANCEL TRANSACTION
        End if
Until(OK=0)
READ ONLY(*) ` [明細]、[部品]テーブルをリードオンリー状態にセットする
```

3. データ入力実行中のレコードロックを減らすには、フォームメソッド内からトランザクションを管理し、必要になった時にだけテーブルをリードライト状態にしてアクセスすることができます。

   サブフォームにリレートテーブルの[明細]を持つ[送り状]テーブルの入力フォームを使ってデータ入力を行います。このフォームには、「動作なし」ボタン属性を持つ「bキャンセル」と「b登録」の2つのボタンがあります。

   メソッドは以下のようになります。

```
READ WRITE([明細])          ` [明細]テーブルをリードライト状態にセットする
READ ONLY([部品]) ` [部品]テーブルをリードライト状態にセットする
INPUT FORM([送り状] ; "入力")     ` 入力レイアウトを選択する
Repeat
```

```
        ADD RECORD([送り状])    ` 必要なだけレコードを追加する
    Until(b登録=0)
    READ ONLY([明細]) ` [明細]テーブルをリードオンリー状態にセットする
```

[部品]テーブルがデータ入力中はリードオンリー状態になっていることに注目してください。リードライト状態は、データ入力が有効な場合にのみ利用できます。

[送り状]入力フォームから開始されるトランザクションを下記に示します。

```
    Case of
        ¥(Form event=On Load)
            START TRANSACTION
            [送り状]送り状ID := Sequence number([送り状]送り状ID)
        Else
            [送り状]合計金額 := Sum([明細]合計金額)
    End case
```

「bキャンセル」ボタンをクリックすると、トランザクションはもちろんのことデータ入力も取り消す必要があります。下記に「bキャンセル」ボタンのオブジェクトメソッドを示します。

```
    Case of
        ¥(Form event=On Clicked)
            CANCEL TRANSACTION
            CANCEL
    End case
```

「b登録」ボタンをクリックすると、トランザクションはもちろんのことデータ入力も受け付ける必要があります。下記に「b登録」ボタンのオブジェクトメソッドを示します。

```
    Case of
        ¥(Form event=On Clicked)  ` 登録ボタンをクリックした場合
            $明細数:=Records in selection([明細])
            READ WRITE([部品])          ` 部品を更新する
            FIRST RECORD([明細])        ` 最初の明細で開始する
            $受け入れ:=True             ` すべてにOKであることように仮定する
            For($Line ; 1 ; $明細数)    ` 全明細をループする
                RELATE ONE([明細]部品番号)
                OK:=1 ` 続行したいと仮定する
                While(Locked([部品]) &(OK=1))
                    CONFIRM("部品番号："+[明細]部品番号
                                +" が使用されています。待ちますか？")
                    If(OK=1)
                        DELAY PROCESS(Current process ; 60)
                        LOAD RECORD([部品])
```

```
                                                    ` レコードをロードしてくる
                        End if
                        End while
                        If(OK=1)
                            [部品]在庫数:=[部品]在庫数 - [明細]数量
                            ` 倉庫の数量を更新する
                            SAVE RECORD([部品])
                                                    ` レコードを保存する
                        Else
                            $Line:=$明細数 + 1 ` ループを抜ける
                            $受け入れ:=False
                        End if
                    NEXT RECORD([明細]) ` 以下の明細に移動する
                End for
                READ ONLY([部品]) ` 部品テーブルをリードオンリー状態にする
                If($受け入れ)
                    SAVE RECORD([送り状]) ` 送り状レコードを保存する
                    VALIDATE TRANSACTION ` すべてのデータベースの修正
                                                    を受け入れる
                Else
                    CANCEL TRANSACTION  `すべてをキャンセルする
                End if
                CANCEL ` フォームを抜ける
            End case
```

このコードでは、ボタンのクリックに関係なく、**CANCEL**コマンドを実行します。新しいレコードは**ACCEPT**コマンドを呼び出しても受け入れられず、**SAVE RECORD**コマンドで受け入れられます。さらに、**SAVE RECORD**コマンドが**VALIDATE TRANSACTION**コマンドの直前に呼び出されている点に注意してください。したがって、[送り状]テーブルのレコードを保存するということは、実際にはトランザクションの一部であるということです。**ACCEPT**コマンドを呼び出してレコードを受け入れることもできますが、その場合、送り状レコードが保存される前にトランザクションが受け入れられてしまいます。つまり、レコードはトランザクションの外で保存されてしまいます。

必要に応じ、前述の例題のように、データ入力中のトランザクション処理を4Dに任せたり、またはデータベースを独自にカスタマイズすることができます。最後の例題では、[部品]テーブルのロックレコードの処理をさらに開発することも可能です。

参照

CANCEL TRANSACTION、In transaction、START TRANSACTION、VALIDATE TRANSACTION

START TRANSACTION

START TRANSACTION

説明

START TRANSACTIONコマンドは、カレントプロセスでトランザクションを開始します。データベースに対するすべての更新は、そのトランザクションを受け入れる（**VALIDATE TRANSACTION**コマンドを実行）か、キャンセルされる（**CANCEL TRANSACTION**コマンドを実行）までは、一時的なバッファにローカルに保存されます。

グローバルプロセスが複数存在する場合、複数のトランザクションを使用することができます。しかし、トランザクションをネスト（入れ子）することはできません。別のトランザクション内でトランザクションを開始した場合、4Dは2番目のトランザクションを無視します。

参照

CANCEL TRANSACTION、In transaction、Using Transactions、VALIDATE TRANSACTION

VALIDATE TRANSACTION

VALIDATE TRANSACTION

説明

VALIDATE TRANSACTIONコマンドは、**START TRANSACTION**コマンドで開始したカレントプロセスのトランザクションを受け入れます。**VALIDATE TRANSACTION**コマンドは、トランザクション中に行われたデータベースへの更新を保存します。

参照

CANCEL TRANSACTION、In transaction、START TRANSACTION、Using Transactions

CANCEL TRANSACTION

CANCEL TRANSACTION

説明

CANCEL TRANSACTIONコマンドは、**START TRANSACTION**コマンドで開始したカレントプロセスのトランザクションをキャンセルします。**CANCEL TRANSACTION**コマンドは、トランザクション中に実行された処理をキャンセルして、データベースをトランザクション開始前の状態に戻します。

この節の前述の例を参照してください。

参照

In transaction、START TRANSACTION、Using Transactions、VALIDATE TRANSACTION

In transaction

In transaction → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	現在のプロセスがトランザクション内の場合、Trueを返す

説明

In transaction関数は、現在のプロセスがトランザクション内にあればTRUE（真）、なければFALSE（偽）を返します。

例題

複数のレコードに対する操作（レコードの追加、修正、または削除）を実行すると、ロックされたレコードに出くわす可能性があります。このような場合、データの整合性を維持するためには、失敗したときに操作全体をロールバックして、データベースを元の状態に戻せるように、トランザクションの中で操作を行わなければなりません。

トリガ内から、あるいはサブルーチン（トランザクションの中でも、外でも呼び出せる）から操作を実行する場合には、**In transaction**関数を使用して、現在のプロセスのメソッドまたは呼び出し側のメソッドがトランザクションを開始したかどうかをチェックすることができます。トランザクションが開始されていなければ、操作を開始してはいけません。失敗した場合に、操作をロールバックすることができなくなってしまいます。

参照

CANCEL TRANSACTION、START TRANSACTION、VALIDATE TRANSACTION

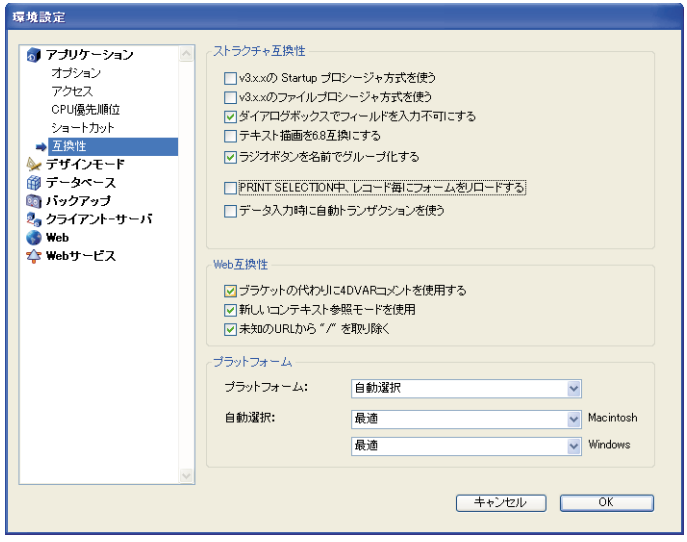
トリガについて

トリガは、テーブルに付属するメソッドであり、テーブルのプロパティです。トリガを呼び出す必要はありません。ユーザーインタフェースを使用した操作およびメソッドによるレコード操作（追加、削除、修正、ロード）をするたびに、4Dデータベースエンジンによって、自動的に起動されます。まず、簡単なトリガを記述し、その後でより洗練されたものにすることができます。

トリガを使用すれば、データベースのレコードに対して「不正な」操作が行われるのを防ぐことができます。トリガは、偶発的にデータが失われたり、変更されたりするのを防ぐだけでなく、テーブルに対する操作を制限するための非常に強力なツールです。例えば、送り状のシステムにおいて、誰かが、送り状の送付先である顧客を指定せずに、送り状を追加するのを防止することができます。

4Dの旧バージョンとの互換性

トリガは、バージョン6で導入された新しいタイプのメソッドです。4th Dimensionの旧バージョンでは、テーブルメソッド（ファイルプロシージャと呼ばれていました）は、データの入力、表示、あるいは印刷のために、テーブル用のフォームが使用される場合にのみ4Dによって実行されていましたが、あまり使われていませんでした。トリガは旧バージョンのファイルプロシージャよりもずっと低いレベルで実行されることに注意していください。ユーザアクションによって（データ入力等）、あるいはプログラムによって （**SAVE RECORD**コマンドへの呼び出し等）、レコードに対してどのような操作を行った場合でも、4Dによってテーブルのトリガが起動されます。トリガは旧バージョンのファイルプロシージャとはまったく異なるものです。バージョン3のデータベースからバージョン6に変換して、この新しいトリガの機能を利用したい場合には、以下に示した「環境設定」ダイアログボックスにおいて、「v3.x.xのファイルプロシージャ方式を使う」プロパティの選択を解除する必要があります。



トリガのアクティブ化と作成

デフォルトでは、「デザイン」モードでテーブルを作成したときには、テーブルにはトリガはありません。

テーブルのトリガを使用するには、以下を実行する必要があります。

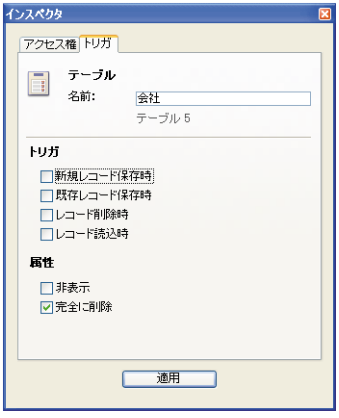
■ トリガをアクティブにし、4Dに対してトリガをいつ起動すべきかを知らせる。

■ トリガ用のコードを記述する。

まだ記述されていないトリガをアクティブにする、あるいはトリガをアクティブにしないで記述しても、テーブルに対して実行される操作に影響を与えることはありません。

1.トリガをアクティブにする

テーブルのトリガをアクティブにするには、「インスペクタ」ウインドウでテーブルの「トリガ」オプション（データベースイベント）のいずれかを選択しなければなりません。



新規レコード保存時：

このオプションを選択すると、レコードがテーブルに追加されるたびに、トリガが起動されます。

以下のような場合に、トリガが起動されます。

■ データ入力でレコードを追加する（「ユーザ」モードまたは**ADD RECORD**コマンドを使って）。

■ **CREATE RECORD**コマンドと**SAVE RECORD**コマンドでレコードを作成し、保存する。トリガは、レコードの作成時ではなく、**SAVE RECORD**コマンドを呼び出した時に起動されることに注意してください。

■ レコードを読み込む（「ユーザ」モードまたは「データ読み込み」コマンドを使用）。

■ 新規レコードを作成または保存するコマンドを使用する（**ARRAY TO SELECTION**コマンド、**SAVE RELATED ONE**コマンド等）。

■ **CREATE RECORD**コマンドと**SAVE RECORD**コマンドを呼び出すプラグインを使用する。

既存レコード保存時：

このオプションを選択すると、テーブルのレコードが修正されるたびに、トリガが起動されます。

以下のような場合に、トリガが起動されます。

■ データ入力でレコードを修正する（「ユーザ」モードまたは**MODIFY RECORD**コマンドを使用）。

■ **SAVE RECORD**コマンドで既に存在しているレコードを保存する。

■ 既存レコードを保存するコマンドを使用する（**ARRAY TO SELECTION**コマンド、**APPLY TO SELECTION**コマンド、**MODIFY SELECTION**コマンド等）。

■ **SAVE RECORD**コマンドを呼び出すプラグインを使用する。

レコード削除時：

このオプションを選択すると、テーブルのレコードが削除されるたびに、トリガが起動されます。

以下のような場合に、トリガが起動されます。

■ レコードを削除する（「ユーザ」モードまたは**DELETE RECORD**コマンド、あるいは**DELETE SELECTION**コマンドを使用）。

■ リレートの削除制御オプションによって、リレート先レコードの削除を引き起こす何らかの操作を実行する。

■ **DELETE RECORD**コマンドを呼び出すプラグインを使用する。

レコード読込時：

このオプションを選択すると、テーブルのレコードがロードされるたびに、トリガが起動されます。これは、カレントレコードをデータファイルからロードするあらゆる状況を含みます。このオプションは、前述の3つのオプションほど頻繁には使用しません。

注：このオプションは、データファイルからカレントレコードがロードされるようなあらゆる状況を含みますが、以下の機能を使用する場合は除きます。

■ クエリ：標準のクエリエディタで用意されたユーザクエリ、または**QUERY**や**QUERY SELECTION**コマンドによるクエリ

■ 並び替え：標準の並び替えエディタで用意された並び替え、または**ORDER BY**コマンドを使用した並び替え

■ 一連の値に関する計算：**Sum**、**Average**、**Min**、**Max**、**Std deviation**、**Variance**、**Sum square**

■ コマンド：**RELATE ONE SELECTION**、**RELATE MANY SELECTION**

4D側の処理を最適化するため、「レコード読込時」オプションでは、インデックスを利用する可能性のあるコマンドの使用時には、トリガの呼び出しを行いません。実際のところ、インデックスが使用される際、レコードはロードされません。これとは逆に、インデックスが使用されない場合（つまり、処理されるフィールドがインデックス付きではない場合）、レコードがロードされます。このトリガの呼び出しが不確実であるため、信頼できる方法でこれを利用することができません。

重要：複数のレコードに影響する操作を実行する、またはコマンドを呼び出すと、トリガは各レコードに対して一度ずつ呼び出されます。例えば、100個のレコードから構成されるカレントセレクションを持ったテーブルに対してAPPLY TO SELECTIONコマンドを呼び出すと、トリガは100回起動されます。

2.トリガを作成する

テーブルのトリガを作成するには、「エクスプローラ」ウインドウを使用するか、Alt（Windowsの場合）またはoption（Macintosh）キーを押して、「ストラクチャ」ウインドウのテーブルタイトルをダブルクリックしてください。詳細については、『4th Dimension デザインリファレンス』を参照してください。

データベースイベント

トリガは、前述の4つのデータベースイベントのいずれかに対して起動することができます。トリガ内で、**Database event**関数を呼び出すことによって、どのイベントが発生しているかを検出します。この関数は、データベースイベントを示す数値を返します。

一般には、**Database event**関数によって返される結果に関して、**Case of**文を用いて、トリガを記述します。

```
` 任意テーブルのトリガ
    C_LONGINT($0)
    $0:=0 `データベースリクエストが許可されると仮定する
    Case of
     ¥(Database event=On Saving New Record Event)
          `新規に作成されたレコードの保存のために適切な動作(アクション)
                                                を実行する
     ¥(Database event=On Saving Existing Record Event)
          `既存のレコードの保存のために適切な動作を実行する
     ¥(Database event=On Deleting Record Event)
          `レコードの削除のために適切な動作を実行する
     ¥(Database event=On Loading Record Event)
          `レコードのメモリへのロードのために適切な動作を実行する
    End case
```

トリガと関数

トリガには、2つの目的があります。

■ レコードが保存、削除される前に、あるいはロードされた直後に、レコードに対して動作（アクション）を実行する。

■ データベース操作を許可または拒絶する。

1.動作を実行する

[ドキュメント]テーブルにレコードが保存（追加または修正）されるたびに、作成時を示すタイムスタンプと最新の修正時を示す「タイムスタンプ」メソッドでレコードを「マーク」したいとします。この場合、以下のようなトリガを記述できます。

```
`[ドキュメント]テーブルのトリガ
Case of
    ¥(Database event=Save New Record Event)
        [ドキュメント]作成日:= Time stamp
        [ドキュメント]修正日:= Time stamp
    ¥(Database event=Save Existing Record Event)
        [ドキュメント]修正日:= Time stamp
End case
```

注：この例で使用されている「Time stamp」ユーザ定義関数は、固定日付が任意に選択された時点からの経過秒数を返す小さなプロジェクトメソッドです。

いったんこのトリガを記述し、アクティブにすると、ユーザがどのような方法で[ドキュメント]テーブルにレコードを追加または修正しても（データ入力、データ読込み、プロジェクトメソッド、4Dプラグイン）、レコードが最終的にディスクに書き込まれる前に、トリガによって、[ドキュメント]作成日と[ドキュメント]修正日のフィールドに自動的に日付が割り当てられます。

注：この例の詳細については、GET DOCUMENT PROPERTIESコマンドの例を参照してください。

2.データベース操作を許可または拒絶する

データベース操作を許可または拒絶するには、トリガは、$0関数結果にトリガエラーコードを返さなければなりません。

例題

[従業員]テーブルの場合を取り上げてみましょう。データ入力に際して、[従業員]保険証番号に関する規則を強制します。「確認」ボタンをクリックする際に、ボタンのオブジェクトメソッドを使用して、そのフィールドをチェックします。

```
`「b確認」ボタンのオブジェクトメソッド
If(Good SS number([従業員]保険証番号))
        ACCEPT
Else
        BEEP
        ALERT("正しい保険証番号を入力してください。")
End if
```

フィールド値が妥当であれば、データ入力を受け入れます。フィールド値が無効であれば、警告（アラート）を表示して、データ入力の状態になります。

またプログラムによって[従業員]レコードを作成した場合、以下のコードはプログラムとしては正当ですが、前述のオブジェクトメソッドで示された規則に違反します。

```
` 任意プロジェクトメソッドの抜粋
` ...
CREATE RECORD([従業員])
[従業員]名前 :="DOE"
SAVE RECORD([従業員]) `←DB規則に違反している。保険証番号は保存されない
```

[従業員]テーブルのトリガを使用すれば、データベースのあらゆるレベルで[従業員]保険証番号の規則を強制できます。トリガは以下のようになります。

```
`[従業員]テーブルのトリガ
$0:=0
$dbEvent:=Get database event
Case of
        ¥(($dbEvent=Save New Record Event) |($dbEvent=
                                        Save Existing Record Event))
        If(Not(Good SS number([従業員]保険証番号)))
                $0:=-15050
        Else
        ` ...
        End if
End case
```

いったんトリガを記述し、アクティブにすると、「**SAVE RECORD**([従業員])」はデータベースエンジンエラーコード-15050を生じ、レコードは保存されません。

同様に、4Dプラグインが不正な保険証番号を持つ[従業員]レコードを保存しようとしても、トリガが同様のエラーを生じ、レコードは保存されません。

トリガを使用すれば、意図的であれ、偶発的であれ、誰も（ユーザ、データベース設計者、プラグイン、4D Serverを使用する4D Openクライアント）保険証番号の規則に違反できないことが保証されます。

テーブルのトリガを使用しない場合でも、レコードを保存または削除しようとした際に、データベースエンジンエラーが生じる場合があることに注意してください。例えば、重複不可属性を持つインデックス（索引）フィールドに重複する値のレコードを保存しようとすると、エラー-9998が返されます。

したがって、エラーを返すトリガは、アプリケーションに新しいデータベースエンジンエラーを追加することになります。

- 4Dは“通常”のエラーを管理する。重複不可のインデックス、リレーショナルデータ制御等。
- トリガを使用して、アプリケーションに固有のカスタムエラーを管理する。

重要：エラーコード値は、任意のものを返すことができます。ただし、4Dデータベースエンジンによって既に予約されているエラーコードは使用しないでください。-32000から-15000までの範囲のエラーコードを使用することを強く推奨します。-15000より大きなエラーコードは、データベースエンジン用に予約されています。

プロセスレベルでは、データベースエンジンエラーを処理するのと同様に、トリガエラーを処理します。

- 4Dに標準のエラーダイアログボックスを表示させ、その後メソッドが停止されるようにすることができる。
- **ON ERR CALL**コマンドでインストールされるエラー処理メソッドを使用して、適切な方法でエラーから回復することができる。

注：データ入力中に、レコードの正当性確認または削除を行おうとして、トリガエラーが返されると、エラーは重複不可のインデックスエラーと同様に処理されます。エラーダイアログが表示され、データ入力の状態になります。「ユーザ」モードでしかデータベースを使用しない場合でも（「カスタム」モードではなく）、トリガの使用には利点があります。

トリガがエラーを返さない（$0:=0）場合でも、これは、必ずしもデータベース操作が成功であることを意味するわけではありません。重複不可のインデックス違反が生じることもあります。また、操作がレコードの更新である場合には、レコードがロックされている可能性があり、I／Oエラーが生じることもあります。そうしたチェックはトリガの実行後に行われます。しかし、実行プロセスのより高度なレベルでは、データベースエンジンによって返されるエラーもトリガによって返されるエラーも同じです。

トリガと4Dアーキテクチャ

トリガはデータベースエンジンレベルで実行されます。以下の図にその様子をまとめています。



トリガは、データベースエンジンが実際に配置されたマシン上で実行されます。これは、シングルユーザ版の4Dでは明白です。4D Serverでは、トリガはクライアントマシンではなく、サーバマシン上の動作プロセス内で実行されます。

トリガが起動される場合、トリガはデータベース操作を実行しようとするプロセスのコンテキスト内で実行されます。トリガの実行を引き起こすこのプロセスは、起動プロセスと呼ばれます。

トリガは特に、起動プロセスのカレントセレクション、カレントレコード、テーブルの読み／書き（read／write）状態、およびレコードロック操作を用いて動作します。

警告：トリガは、そのトリガが属しているテーブルのカレントレコードを変更することはできませんし、また、してはいけません。トリガ内で、複数のフィールドに関して重複する値をチェックする必要がある場合には、SET QUERY DESTINATIONコマンドを使用してください。このコマンドを使用すれば、テーブルのカレントセレクションやカレントレコードを変更することなく、テーブルを照会することができます。

4D環境の他のデータベースオブジェクトや言語オブジェクトは注意して使用してください。これは、トリガが起動プロセスのマシンとは別のマシン上で実行される可能性があるためです。4D Serverは、これに当てはまります。

- **インタープロセス変数：**トリガは、トリガが実行されるマシンのインタープロセス変数にアクセスします。4D Serverでは、トリガが起動プロセスのマシンとは別のマシンにアクセスする可能性があります。

- **プロセス変数：**独立したプロセス変数テーブルがすべてのトリガによって共有されます。トリガは、起動プロセスのプロセス変数にはアクセスしません。

- **ローカル変数：**トリガ内でローカル変数を使用できます。その有効範囲は、トリガの実行中です。ローカル変数は、トリガの実行のたびに作成され、削除されます。

- **セマフォ：**トリガは、（トリガが実行されるマシン上の）ローカルセマフォだけでなく、グローバルセマフォもテスト、または設定できます。ただし、トリガは即座に実行されなければならないため、トリガ内からセマフォをテスト、または設定する場合には、十分な注意が必要です。

- **セットと命名セレクション：**トリガ内からセットや命名セレクションを使用する場合、トリガが実行されるマシン上で作業することになります。

- **ユーザインタフェース：**トリガ内でユーザインタフェース要素を使用しないでください(警告（アラート)、メッセージ、ダイアログボックスを使用しない)。したがって、トリガのトレースはデバッガウインドウに限定する必要があります。クライアント／サーバでは、トリガは4D Server上で実行されることを覚えておいてください。サーバマシン上で警告（アラート）メッセージを表示しても、クライアント上のユーザの助けにはなりません。起動プロセスにユーザインタフェースの処理を行わせるようにしてください。

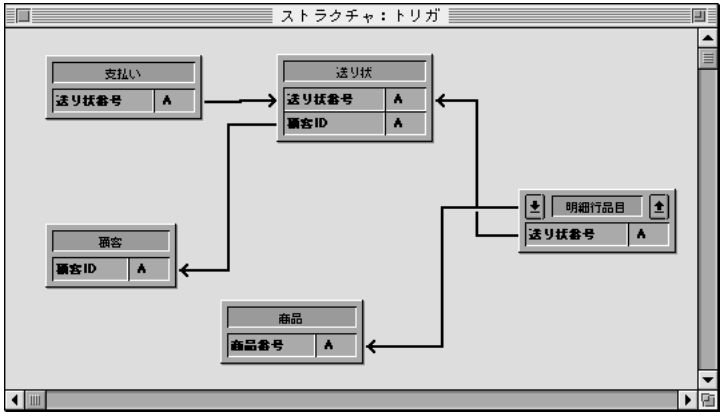
トリガとトランザクション

トランザクションは、起動プロセスレベルで処理されなければなりません。トリガレベルでトランザクションを管理してはいけません。1つのトリガの実行中に、複数のレコードを追加、修正、あるいは削除しなければならない場合（下記の例を参照）には、最初にトリガ内から**In transaction**コマンドを使用して、起動プロセスが現在トランザクション内にあるかをテストしなければなりません。そうでない場合には、トリガがロックされたレコードに出くわす可能性があります。したがって、起動プロセスがトランザクション内になければ、レコードに対する操作を開始しないでください。起動プロセスに、そのプロセスが実行しようとしているデータベース操作はトランザクション内で実行されなければならないことを知らせるために、$0にエラーを返すだけにしてください。そうしないと、ロックされたレコードに出くわした場合、起動プロセスにはトリガの動作（アクション）をロールバックする方法がなくなります。

注：トリガとトランザクションを組み合わせた処理を最適化するため、4Dでは VALIDATE TRANSACTIONの実行後、トリガは呼び出されません。これにより、トリガが2度実行されることを避けられます。

トリガのカスケード

以下の例のようなストラクチャがあるとします。



注：上記のテーブルは、簡略化されています。実際には、テーブルには、ここに示したよりも多くのフィールドがあります。

データベースが、ある送り状の削除を“許可”するとしましょう。そのような操作がトリガレベルでどのように処理されるかを検討してみます（プロセスレベルで削除を実行することも可能です）。

データのリレートに関する整合性を維持するために、送り状を削除する場合、[送り状]のトリガで、以下の動作が実行される必要があります。

■ [顧客]レコードにおいて、送り状の金額分だけ、総売上げフィールドの額を減らす。

■ その送り状に関連したすべての[明細行品目]レコードを削除する。

■ これはまた、[明細行品目]トリガが、削除された明細品目に関連した[商品]レコードの売上げ数量フィールドの数量を減らすことを意味する。

■ 削除された送り状に関連したすべての[支払い]レコードを削除する。

第1に、[送り状]のトリガは上記の動作を、起動プロセスがトランザクション内にある場合にだけ実行しなければなりません。そうすれば、ロックされたレコードに出くわした場合にロールバックを実行することができます。

第2に、[明細行品目]のトリガは[送り状]のトリガから波及しています。明細品目の削除は、[送り状]のトリガ内から**DELETE SELECTION**コマンドを呼び出した結果であるため、[明細行品目]のトリガは[送り状]のトリガの実行の“範囲内で”実行されます。

この例のすべてのテーブルに、すべてのデータベースイベントに対してアクティブなトリガが設定されていると考えてください。このとき、トリガは以下のように波及します。

- 起動プロセスが送り状を削除するため、[送り状]のトリガが起動される。
- [送り状]のトリガが総売上げフィールドを更新するため、[顧客]のトリガが起動される。
- [送り状]のトリガが明細品目を削除するため（繰り返し）、[明細行品目]のトリガが起動される。
- [明細行品目]のトリガが売上げ数量フィールドを更新するため、[商品]のトリガが起動される。
- [送り状]のトリガが支払いを削除するため（繰り返し）、[支払い]のトリガが起動される。

この波及関係において、[送り状]のトリガはレベル1で、[顧客]、[明細行品目]、および[支払い]のトリガはレベル2で、そして[商品]のトリガはレベル3で実行されていると言えます。

トリガ内から、**Trigger level**関数を使用して、トリガが実行されるレベルを検出することができます。さらに、**TRIGGER PROPERTIES**コマンドを使用して、他のレベルについての情報を得ることができます。

例えば、プロセスレベルで[商品]レコードが削除されている場合、[商品]のトリガはレベル3ではなく、レベル1で実行されます。

Trigger level関数と**TRIGGER PROPERTIES**コマンドを使用すれば、動作（アクション）の原因を検出できます。前述の例では、送り状はプロセスレベルで削除されます。仮に、プロセスレベルで[顧客]レコードを削除すると、その結果、[顧客]のトリガはその顧客に関連したすべての送り状を削除しようとします。これにより、前述の例と同じように、[送り状]のトリガが起動されることになりますが、起動される理由は異なります。[送り状]のトリガ内から、そのトリガがレベル1で実行されたか、レベル2で実行されたかを、検出することができます。トリガがレベル2で実行された場合には、次に、それが[顧客]レコードが削除されたためであるかどうかをチェックできます。そうであれば、総売上げフィールドの更新にわずらわされる必要はありません。

トリガ内での通し（シーケンス）番号の使用

「On saving new record」データベースイベントを処理する際に、テーブルのレコードに対して一意なID番号を維持するために、**Sequence number**関数を呼び出すことができます。

例題

```
`[送り状]テーブルのトリガ
Case of
    ¥(Database event=On Saving new record)
    ` ...
        [送り状]送り状番号:=Sequence number([送り状])
    ` ...
End case
```

ただし、トランザクション内でこれを実行する場合には、誤って通し番号を増やすのを避けるために、フィールドに関するテストで、この呼び出しを囲まなければなりません。

典型的な例としては、データ入力トランザクション中にサブフォームにレコードを入れる場合が挙げられます。

例題

```
` [明細行品目]テーブルのトリガ
Case of
    ¥(Database event=On Saving new record)
        ` ...
        If([明細行品目]商品番号=0)
            [明細行品目]商品番号:=Sequence number([送り状])
        End if
        ` ...
End case
```

参照

Database event、Record numbers、Trigger level、TRIGGER PROPERTIES

Database event

Database event → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	0=トリガの実行サイクル外 1=新規レコード保存時 2=既存レコード保存時 3=レコード削除時 4=レコード読込時

説明

トリガ内から呼び出されると、**Database event**関数はデータベースイベントの種類、つまりそのトリガが起動された理由を示す数値を返します。

データベースイベントには、以下のようなあらかじめ定義された定数があります。

定数	タイプ	値
Save New Record Event	倍長整数	1
Save Existing Record Event	倍長整数	2
Delete Record Event	倍長整数	3
Load Record Event	倍長整数	4

データベースイベントが既存のレコードの保存イベント（**Save Existing Record Event**）である場合、**Modified**関数を使用して、レコードの特定のフィールドが修正されたかどうかを検出することができます。修正された場合には、**Old**関数を使用して、現在ディスクに格納されている修正前のフィールドの値を検索することもできます。

注：この2つの関数は、英数字や実数といった単純なフィールドに適用される場合にのみ重要です。ピクチャやBOLB、サブテーブルフィールドでは、これらのコマンドの結果は重要ではありません。

トリガ内で、複数のレコードに対するデータベース操作を実行すると、実行するべき操作をトリガが正確に実行されるのを妨げる条件 （通常はロックされたレコード）に出くわす可能性があります。このような状況の例としては、[送り状]テーブルにレコードを追加中に、[商品]テーブルの複数のレコードを更新する場合が挙げられます。この時点で、データベース操作を試みるのを中止し、データベースエラーを返して、そのデータベースリクエストが実行できないことを起動プロセスに知らせなければなりません。このとき、起動プロセスは、トランザクションの中で、トリガによって実行された不完全なデータベース操作をキャンセルできなければなりません。この種の状況が発生した場合には、トリガ内から、何らかの操作を試みる前からそのプロセスがトランザクション内にあることを確かめる必要があります。これを実行するには、**In transaction**関数を使用します。

トリガのカスケードを呼び出す場合には、4th Dimensionには、使用可能なメモリの容量以外に制限はありません。トリガの実行を最適化するために、データベースイベントだけでなく、トリガがカスケードされて起動される際の呼び出しのレベルに基づいて、トリガのコードを記述することもできます。例えば、[送り状]テーブルに対する削除データベースイベントの中で、[送り状]レコードの削除が、削除された[顧客]レコードに関連したすべての送り状の削除にともなうものである場合には、[顧客]総売上げフィールドの更新をスキップすることもできます。これを実行するには、**Trigger level**関数と**TRIGGER PROPERTIES**コマンドを使用します。

例題

Database event関数を使用して、以下のようにトリガを作成します。

```
` 任意テーブル用のトリガ
C_LONGINT($0)
$0:=0 `データベースリクエストが許可されると仮定する
Case of
    ¥(Database event=Save New Record Event)
        `新規に作成されたレコードの保存のために適切な動作を実行する
    ¥(Database event=Save Existing Record Event)
        `既存のレコードの保存のために適切な動作を実行する
    ¥(Database event=Delete Record Event)
        `レコードの削除のために適切な動作を実行する
    ¥(Database event=Load Record Event)
        `レコードのメモリへのロードのために適切な動作を実行する
End case
```

参照

In transaction、Modified、Old、Trigger level、TRIGGER PROPERTIES

Trigger level

Trigger level → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	トリガの実行レベル トリガの実行サイクル外であれば、0

説明

Trigger level関数は、トリガの実行レベルを返します。

実行レベルについての詳細は、前述の「トリガのカスケード」の節を参照してください。

参照

Database event、TRIGGER PROPERTIES

TRIGGER PROPERTIES

TRIGGER PROPERTIES(トリガレベル ; イベント ; テーブル番号 ; レコード番号)

引数	タイプ		説明
トリガレベル	数値	→	トリガ実行レベル
イベント	数値	←	データベースイベント
テーブル番号	数値	←	影響を受けるテーブル番号
フィールド番号	数値	←	影響を受けるフィールド番号

説明

TRIGGER PROPERTIESコマンドは、＜トリガレベル＞に渡すトリガの実行レベルについての情報を返します。トリガ実行レベルの波及に基づいて異なる動作を実行するには、**TRIGGER PROPERTIES**コマンドを**Trigger level**関数と組み合わせて使用します。詳細については、前述の「トリガのカスケード」の節を参照してください。

存在しないトリガ実行レベルを渡すと、コマンドはすべての引数に0を返します。

渡されたトリガ実行レベルのデータベースイベントの種類が、引数＜イベント＞に返されます。

データベースイベントには、以下のようなあらかじめ定義された定数があります。

定数	タイプ	値
Save New Record Event	倍長整数	1
Save Existing Record Event	倍長整数	2
Delete Record Event	倍長整数	3
Load Record Event	倍長整数	4

渡されたトリガ実行レベルのデータベースイベントに関係するレコードのテーブル番号とレコード番号が引数＜テーブル番号＞と＜レコード番号＞に返されます。

注：トランザクション中、新規に作成されたレコードには一時的なレコード番号が与えられることを覚えておいてください。

参照

Database event、Trigger level

ユーザフォームの概要

4th Dimension 2004において、開発者はユーザに対して独自のフォームの作成や変更を実行する機会を与えることができます。その結果、“ユーザフォーム”を他の4th Dimensionフォームと同じように使用することができます。

ユーザフォーム入門

ユーザフォームは開発者が「デザイン」モードで作成した4th Dimension標準のフォーム(“ソース”フォームまたは“ディベロッパ”フォーム）をベースにしており、そのフォームには「ユーザ編集可」プロパティが適用されています。簡易版の「フォーム」エディタ（**EDIT FORM**コマンドで呼び出される）を使用して、ユーザはフォームの外観やグラフィックオブジェクト、非表示項目などを修正することができます。また開発者は、認可する動作を制御することができます。

ユーザフォームは、次の2通りの方法で使用することができます。

- ユーザは**EDIT FORM**コマンドを使用して“ソース”フォームを修正し、独自のニーズに合わせます。ユーザフォームはローカル上に保持され、オリジナルフォームの代わりとして自動的に使用されます。

  この動作は、例えば、フォームに会社ロゴを追加したり、不要なフィールドを隠すなど、開発者がダイアログボックスのパラメータを現場で設定する必要性に対処するものです。

- “ソース”ファイルは基本テンプレートとして機能し、ユーザは**CREATE USER FORM**コマンドを使用して、いくつでも自由にコピーを作成することができます。**EDIT FORM**コマンドを使用すると、それぞれのコピーの設定を自由に行えます（内容、名前など)。ただし、各ユーザフォームの名前はユニークでなくてはなりません。これらのコピーを利用するには、**INPUT FORM**コマンドと**OUTPUT FORM**コマンドを使用します。これらのコマンドは引数としてユーザフォームを受け入れるようになりました。

  この対応方法により、例えば開発者はユーザ用にカスタマイズしたレポートを作成することができます。

ユーザフォームの保存と管理

ユーザフォームのメカニズムは、4th Dimension、4D Server、4D Runtime、または4D Engineを使用したコンパイル済みデータベースおよびインタプリタ版のデータベースの双方で機能します。クライアント／サーバモードでは、ユーザが修正したフォームをすべてのマシンで利用することができます。

フォームの変更は自動的に処理されます。ユーザが「ユーザ編集可」としてフォームを設定している場合、「デザイン」モードにおいてそのフォームがロックされます。フォームオブジェクトへアクセスできるようにするには、開発者がアイコンをクリックして、そのフォームのロックを明示的に解除しなければなりません。この操作により、関連するユーザフォームが古くなりますが、その場合は再作成しなくてはなりません。“ソース”フォームが削除されると、関連するユーザフォームも一緒に削除されます。

ユーザフォームは、拡張子“.4DA”が付けられた別ファイルに保存され、メインのストラクチャファイル（.4DB/.4DC）と同じ階層に配置されます。このファイルは“ユーザストラクチャファイル”と呼ばれます。このファイルの機能は透過的です。つまり、4th Dimensionはユーザフォームが存在すれば、それを使用します（**LIST USER FORMS**コマンドを使用し、有効なユーザフォームをいつでも探すことができます）。**INPUT FORM**コマンドと**OUTPUT FORM**コマンドがユーザフォームを探すのもこのファイルの中です。ユーザフォームが古ければ、それが削除され、4th Dimensionはデフォルトとしてソースフォームを使用します。

クライアント／サーバでは、メインのストラクチャファイルと同じ規則に従って、「.4DA」ファイルが各クライアントマシンへ送信されます。

また、この原理により、開発者がストラクチャファイルの更新を行った場合でも、ユーザフォームの陳腐化を避けることができます。

エラーコード

ユーザフォーム管理のコマンドを使用したとき、特定のエラーコードが返されることがあります。-9750から-9759までのこれらのコードは「データベースエンジンエラー」の節で後述します。

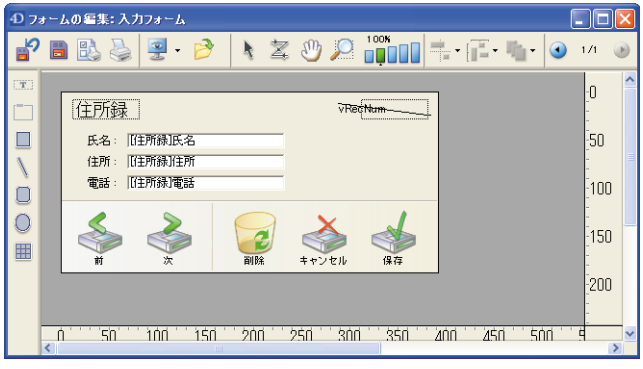
EDIT FORM

EDIT FORM (テーブル; フォーム{; ユーザフォーム{; ライブラリ}})

引数	タイプ		説明
テーブル	テーブル	→	変更するフォームの属すテーブル
フォーム	文字列	→	変更するフォームの名前
ユーザフォーム	文字列	→	修正するユーザフォームの名前
ライブラリ	文字列	→	利用可能なオブジェクトライブラリのパス名

説明

EDIT FORMコマンドは、引数<テーブル>、<フォーム>および任意の引数である<ユーザフォーム>を用いて指定されたフォームを「ユーザフォーム」エディタ上に開きます。



注：そのプロセスの最初のウインドウである場合にのみ、エディタウインドウが開かれます。言い換えれば、エディタを表示するには常に新規プロセスをオープンする必要があるということです。

引数<ユーザフォーム>を渡さず、しかも<フォーム>に関連付けられたユーザフォームがもともと存在していない場合、エディタ上にはソースフォームが表示されます。この後、変更後のフォームがユーザストラクチャファイル（.4DA）にコピーされ、<フォーム>の代わりに使用されます。

このコマンドを用いて、ユーザフォームが<フォーム>を元にして既に生成されている場合、エディタ上にユーザフォームが表示されます。ソースフォームから始めたい場合は、まず最初に**DELETE USER FORM**コマンドを使用してユーザフォームを削除しなければなりません。

引数＜ユーザフォーム＞を使用すると、変更するユーザフォーム（**CREATE USER FORM**コマンドを用いて作成済み）を設定することができます。この場合、そのフォームがエディタ上に表示されます。

オプションの引数＜ライブラリ＞には、オブジェクトライブラリのフルアクセスパスを渡し、ユーザがフォームをカスタマイズするための権限を与えます。オブジェクトライブラリを＜ユーザフォーム＞と一緒に使用する場合、グラフィックプロパティおよび自動動作でオブジェクトをペーストすることができます。メソッド付きのオブジェクト？はライブラリに表示されません。追加されたライブラリオブジェクトの名前、変数およびタイプがユーザフォーム（およびそのオブジェクト）に対応しているかどうかについては開発者の確認範囲になるので注意してください。

クライアント／サーバモードでは、このライブラリがすべてのクライアントマシンで使用できるように、「Plug-ins」フォルダと同じ階層にあるデータベースの「Extra」フォルダ内に配置しなくてはなりません。そのライブラリが有効であれば、それがフォームウインドウで開かれます。オブジェクトライブラリに関する詳細は、『デザインリファレンス』マニュアルを参照してください。

例題

▼次の例題は、ダイアログのフォームの変更をユーザに許可するプロジェクトメソッドです。

```
EDIT FORM([Dialogs];"Welcome";"Lib_Logos.4il")
```

参照

CREATE USER FORM, DELETE USER FORM, LIST USER FORMS, Overview of user forms

システム変数またはセット

処理が正常に実行されると、システム変数OKに1が返されます。それ以外は0が返されます。

エラー処理

以下の場合、エラーが生成されます。

■ ユーザが「デザイン」モードでフォームの変更を行えない場合や、フォームが存在しない場合

■ フォームが既に開かれ、別プロセスで変更されている場合

■ 適切なアクセス権を持たないため、ユーザがそのフォームへアクセスできない場合

これらのエラーは**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて遮ることができます。

CREATE USER FORM

CREATE USER FORM (テーブル; フォーム; ユーザフォーム)

引数	タイプ		説明
テーブル	テーブル	→	ソースフォームのテーブル
フォーム	文字列	→	ソースフォーム名
ユーザフォーム	文字列	→	新規ユーザフォームの名前

説明

CREATE USER FORMコマンドはそのテーブルと名前を引数として渡した4th Dimensionの＜フォーム＞を複製し、＜ユーザフォーム＞で指定した名前のユーザフォームを新たに作成します。

作成が完了すると、**EDIT FORM**コマンドを使用して、フォーム＜ユーザフォーム＞を変更することができます。このコマンドにより、1つのソースフォームを元にしてX個のユーザフォームを作成することができます（例えば、各種レポートフォーム）。

参照

DELETE USER FORM, EDIT FORM, INPUT FORM, LIST USER FORMS, OUTPUT FORM,Overview of user forms

システム変数またはセット

処理が正常に実行されると、システム変数OKに1が返されます。それ以外は0が返されます。

エラー処理

以下の場合、エラーが生成されます。

■ ＜フォーム＞が既にユーザフォームである場合。

■ ＜ユーザフォーム＞に指定した名前が、ソースフォームまたは既存のユーザフォームと同じである場合。

■ 適切なアクセス権を持たないため、ユーザがそのフォームへアクセスできない場合。

このエラーは**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて遮ることができます。

LIST USER FORMS

LIST USER FORMS (テーブル; フォーム; ユーザフォーム配列)

引数	タイプ		説明
テーブル	テーブル	→	ソースフォームのテーブル
フォーム	文字列	→	ソースフォームの名前
ユーザフォーム配列	文字列配列	←	ソースフォームを元にしたユーザフォームの名前

説明

このコマンドは、引数<テーブル>と<フォーム>で指定されたディベロッパフォームを元に作成されたユーザフォームの名前を配列<ユーザフォーム配列>に代入します。

ユーザフォームが**EDIT FORM**コマンドを使用して直接作成された場合、<ユーザフォーム配列>の項目には空の文字列（""）だけが代入されます。

指定したディベロッパフォームに対するユーザフォームが存在しない場合、この配列は空になります。

参照

CREATE USER FORM, EDIT FORM, Overview of user forms

DELETE USER FORM

DELETE USER FORM (テーブル; フォーム; ユーザフォーム)

引数	タイプ		説明
テーブル	テーブル	→	ユーザフォームのテーブル
フォーム	文字列	→	ソースフォーム名
ユーザフォーム	文字列	→	ユーザフォーム名

説明

このコマンドを使用し、引数<テーブル>、<フォーム>、<ユーザフォーム>で指定したユーザフォームを削除することができます。

ユーザフォームが**EDIT FORM**コマンドを使用して直接作成された場合、<ユーザフォーム>に空の文字列（""）を渡してください。

参照

CREATE USER FORM, LIST USER FORMS, Overview of user forms

システム変数またはセット

処理が正常に実行されると、システム変数OKに1が返されます。それ以外は0が返されます。

エラー処理

以下の場合、エラーが生成されます。

■ ユーザフォームが存在しない場合

■ ユーザが十分なアクセス権を持たない場合

このエラーは**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて遮ることができます。

BEEP

BEEP

説明

BEEPコマンドは、PCまたはMacintoshでビープ音を発生します。発生するビープ音は、「サウンド」コントロールパネルで変更することができます。

---

*警告：Web接続プロセス内から**BEEP**コマンドをコールしないでください。これは、クライアントである*Web*ブラウザマシンではなく、*Web*サーバマシン上の*4th Dimension*でビープ音が発生するためです。*

例題

▼ 以下の例は、クエリでレコードが見つからなかった場合に、ビープ音を発生し、警告が表示されます。

```
QUERY([顧客];[顧客]名字=$vsNameToLookFor)
If(Records in selection([顧客])=0)
    BEEP
    ALERT("該当する名前の顧客は存在しません。")
End if
```

参照

PLAY

PLAY

PLAY(サウンド名 {; チャネル})

引数	タイプ		説明
サウンド名	文字列	→	発生する音の名前 Windows：.WAV、MID、AVIファイル 任意のプラットフォーム：Macintosh ベースの“snd”リソース、または非同期 指定を中断する場合、空の文字列
チャネル	数値	→	指定した場合、シンセサイザチャネル 省略した場合、同期指定

説明

Windows上での説明

Windowsにおいて、**PLAY**コマンドはサウンド（.WAVファイル）、MIDI（.MIDファイル）、ビデオ（.AVIファイル）の各Windowsファイルを再生します。＜サウンド名＞には再生したいファイルへの完全なパス名を指定します。

注：非同期モードでは、マルチメディアファイルやオブジェクトは再生できません。これを実現するには、OLEサービスを使用してください。

Macintosh、またはWindows（いくらかの制限があります。後述の“重要”を参照してください）において、**PLAY**コマンドはMacintosh上で＜サウンド名＞に指定された名前を持つサウンドリソースを再生します。

引数＜チャネル＞は、Macintoshのシンセサイザチャネルを指定します。＜チャネル＞を指定しない場合には、単純デジタル音の同期指定とみなします。同期指定の場合には、発生した音が終了するまで以降の処理をすべて中断します。＜チャネル＞に0を指定した場合には、単純デジタル音の非同期指定とみなします。非同期指定の場合には、音をバックグラウンドで発生し、処理を通常どおり実行します。

非同期指定の音の発生を終了するためには、次のような方法でします。

```
PLAY("" : 0)
```

MacintoshとWindowsで同時にデータベースを使って作業を実行する場合、Windowsプラットフォーム上でMacintoshのサウンドを再生することも可能です。

■ Macintoshでは、ResEditやResorcerer等のリソースエディタを利用して、目的の“snd”リソースをストラクチャファイルのリソースフォークにコピーしてください。

■ 4D Transporterを使い、データベースをMacintoshからWindowsへトランスポートします

重要：Windows版の4th Dimensionでは、MACEで圧縮されたMacintoshのサウンドは再生されません。Macintoshの“snd”リソースがWindows上で再生されない場合は、以下の要件に準拠しているかどうかを調べてください。

“snd”リソースフィールド	値（16進数表記）
Version	0x0001
NbSynth	0x0001
SynthResID	0x0005
SynthInitOptions	0x000000A0
NbSoundCommand	0x0001
FirstCommand	0x8051

Resorcererを使用して、“snd”リソースの内部データをチェックすることができます。

例題

1. 以下の例題はWindows上でビデオファイルを再生する方法を示しています。

```
$DocRef := Open document( ""; "AVI")
If(OK=1)
        CLOSE DOCUMENT($DocRef)
        PLAY(Document)
End if
```

2. 以下の例は、On Startupデータベースメソッドに含まれるものです。Macintoshでは“ウエルカム”の音でユーザを歓迎します。

```
PLAY("Welcome Sound")    ` "Welcome Sound"の音を再生する
```

参照

BEEP

Get platform interface

Get platform interface → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	数値	←	現在使用中のプラットフォームインタフェース

説明

Get platform interface関数は、フォームの表示用に現在使用しているプラットフォームインタフェースを表す数値を返します。

Get platform interface関数は、下記の値の1つを返します。

定数	タイプ	値
Automatic Platform	倍長整数	-1
Mac OS 7	倍長整数	0
Windows NT3.51	倍長整数	1
Windows 9x	倍長整数	2
Platinum	倍長整数	3
Mac Theme	倍長整数	4

SET PLATFORM INTERFACEコマンドまたは「デザイン」モードの「データベースプロパティ」ダイアログボックスで、プラットフォームインタフェースを変更することができます。

参照

SET PLATFORM INTERFACE

SET PLATFORM INTERFACE

SET PLATFORM INTERFACE(インタフェース)

引数	タイプ		説明
インタフェース	数値	→	新しいプラットフォームインタフェース設定
		-1	自動
		0	Macintosh（漢字Talk7）
		1	Windows NT3.51
		2	Windows 95
		3	Platinum
		4	Mac Theme

説明

SET PLATFORM INTERFACEコマンドは、フォームの表示用にプラットフォームインタフェースを設定します。

4th Dimensionでは、以下の表のように前もって定義された定数の1つを引数＜インタフェース＞に渡します。

定数	タイプ	値
Automatic Platform	倍長整数	-1
Mac OS 7	倍長整数	0
Windows NT3.51	倍長整数	1
Windows 9x	倍長整数	2
Platinum	倍長整数	3
Mac Theme	倍長整数	4

注意：現在のMac OSでは「アピアランス」の「テーマ」によりユーザーインタフェイスを定義することができます。これはMac OSでのみ有効です。Mac OSの「テーマ」でインタフェイスを定義したデータベースをWindows上で表示するときは、Windows 9xのインタフェイスが適用されます。コマンドに渡した値が現在のものと同じなら、何も起こりません。

注意：プラットフォームインタフェースの設定は、「デザイン」モードの「データベースプロパティ」ダイアログボックスで変更することもできます。

例題

4Dのクライアント／サーバアーキテクチャでは、MacintoshとWindowsステーションが同時に異なるプラットフォームインタフェースを使用することが可能です。データベースの「On startup」データベースメソッドで、**SET PLATFORM INTERFACE**コマンドをこの目的のために利用することができます。

```
` この例題では、ユーザの環境設定は[環境設定]テーブルに納められています。
` Current Userに対応するレコードを探します
QUERY([環境設定] ; [環境設定]ユーザ名=Current User)
If(Records in selection([環境設定]) = 0)
   ` 見つからない場合、デフォルトの設定を探します
   QUERY([環境設定] ; [環境設定]ユーザ名 = "デフォルト")
End if
   ` ユーザの環境設定に対応するプラットフォームインタフェースを設定します
SET PLATFORM INTERFACE([環境設定]プラットフォーム)
```

参照

Get platform interface

SET TABLE TITLES

SET TABLE TITLES(テーブルタイトル ; テーブル番号)

引数	タイプ		説明
テーブルタイトル	文字列配列	→	ダイアログボックスに表示されるテーブル名
テーブル番号	数値配列	→	実際のテーブル番号

説明

SET TABLE TITLESコマンドを使用すれば、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4thDimensionダイアログボックスにデータベースのテーブルを表示する際に、それらのテーブルのマスク、名前の変更、並べ替えが行えます。

ダイナミック名を使用した場合、このコマンドの使用により、フライ上でフォームのテーブルラベルの名前を変えることもできます。ダイナミックフィールドの挿入やフォーム中のテーブル名についての詳細は、『4th Dimensionデザインリファレンス』を参照してください。

引数＜テーブルタイトル＞と＜テーブル番号＞の配列は同期させる必要があります。配列＜テーブルタイトル＞には、テーブルの名前を、表示したい名前で渡します。ある特定のテーブルを表示したくない場合には、そのテーブル名または新しく付けたタイトルを、配列に指定しないでください。テーブルは、この配列に指定した順序で表示されます。配列＜テーブル番号＞の各要素には、配列＜テーブルタイトル＞の同じ番号の要素に渡されるテーブル名または新規タイトルに対応する実際のテーブル番号を渡します。

例えば、テーブルA、B、Cで構成されるデータベースがあり、テーブルはこの順序で作成されたとします。表示の際にはこれらのテーブルをX、Y、Zという名前にし、さらにテーブルBは表示したくないとします。最終的に、ZとXを、この順序で表示することにします。この場合、配列＜テーブルタイトル＞2つの要素としてZとXを渡し、配列＜テーブル番号＞の2つの要素として3と1を渡します。

SET TABLE TITLESコマンドはデータベースの実際の構造(ストラクチャ) を変更するわけではありません。これは、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4th Dimensionダイアログボックスやダイナミック名を使ったフォームをを次回使用する際にのみ影響します。**SET TABLE TITLES**コマンドの有効範囲は、そのワークセッションです。クライアント／サーバにおける利点の1つは、複数の4Dクライアントステーションがそれぞれ異なる見方でサーバのデータベースを同時に「見る」ことができる点です。**SET TABLE TITLES**コマンドは、何度でも呼び出すことができます。

SET TABLE TITLESコマンドは、以下のよう場合に使用します。

■ データベースを動的にローカライズする。

■ データベースの実際の定義に関係なく、自由にテーブルを表示する。

■ 固有のユーザまたは任意ユーザのカスタム特権に応じてテーブルの表示を変える。

警告：SET TABLE TITLESコマンドはテーブルの非表示属性を上書きしません。データベースの設計（デザイン）レベルでテーブルが非表示として設定されている場合、SET TABLE TITLESコマンドへの呼び出しにそのテーブルを指定しても、テーブルは表示されません。

例題

■ 各国で販売する予定の4Dアプリケーションを構築しているとします。この場合、ローカライズの問題を慎重に考慮する必要があります。「ユーザ」モードまたは「カスタム」モードで表示される標準の4th Dimensionダイアログボックスとダイナミック名を使用したフォームに注意すれば、[翻訳]テーブルといくつかのプロジェクトメソッドを使用して、必要なだけ各国向けにローカライズされたフィールドを作成し、使用することによって、ローカライズのニーズに対応できます。

■ データベースに以下のテーブルを追加します。

翻訳	
実際の名前	A
言語	A
翻訳された名前	A

■ 次に、以下に示した「TRANSLATE TABLES AND FIELDS」プロジェクトメソッドを作成します。このメソッドはデータベースの実際のストラクチャをブラウズし、引数として渡される言語に対応するローカライズ版の作成に必要なすべての[翻訳]レコードを作成します。

```
`「TRANSLATE TABLES AND FIELDS」プロジェクトメソッド
` TRANSLATE TABLES AND FIELDS(文字列)
` TRANSLATE TABLES AND FIELDS(言語 )
C_STRING(31 ; $1)
C_LONGINT($vlTable ; $vlField)
For($vlTable ; 1 ; Count tables) `テーブルをループする
    `指定された言語用のテーブル名の翻訳があるかどうかをチェックする
    QUERY([翻訳];[翻訳]実際の名前=Table name($vlTable) ; *)
    QUERY([翻訳]  &  [翻訳]言語=$1)
    If(Records in selection([翻訳])=0)
```

```
            `なければ、レコードを作成する
            CREATE RECORD([翻訳])
            [翻訳]実際の名前:=Table name($vlTable)
            [翻訳]言語:=$1
            `翻訳したテーブル名を入力する必要がある
            SAVE RECORD([翻訳])
        End if
        For($vlField ; 1 ; Count fields($vlTable))
            `指定された言語用のフィールド名の翻訳があるかどうかをチェックする
            QUERY([翻訳];[翻訳]実際の名前=Field name($vlTable ; $vlField) ; *)
            QUERY([翻訳]  &  [翻訳]言語=$1)
            If(Records in selection([翻訳])=0)
                `なければ、レコードを作成する
                CREATE RECORD([翻訳])
                [翻訳]実際の名前:=Field name($vlTable ; $vlField)
                [翻訳]言語:=$1
                `翻訳したフィールド名を入力する必要がある
                SAVE RECORD([翻訳])
            End if
        End for
    End for
```

■ この時点で以下の行を実行すれば、テーブルタイトルとフィールドタイトルの日本語のローカライズ版に必要なすべてのレコードが作成されます。

```
TRANSLATE TABLES AND FIELDS("Japanese")
```

■ この呼び出しの実行後、新しく作成されたレコードのそれぞれに対して“[翻訳]翻訳された名前”を入力できます。

■ 最後に、日本語のローカライズ版を使用してデータベースの標準の4Dダイアログボックスやダイナミック名を使用したフォームを表示したい場合はいつでも、「*LOCALIZED TABLES AND FIELDS*」プロジェクトメソッドを使ってに以下の行を実行します。

```
LOCALIZED TABLES AND FIELDS("Japanese")
```

「LOCALIZED TABLES AND FIELDS」プロジェクトメソッドを次に示します。

```
`「LOCALIZED TABLES AND FIELDS 」プロジェクトメソッド
` LOCALIZED TABLES AND FIELDS(文字列)
` LOCALIZED TABLES AND FIELDS(言語)
C_STRING(63 ; $1)
C_LONGINT($vlTable ; $vlNbTable ; $vlField ; $vlNbField)
$vlNbTable:=Count tables `データベースに存在するテーブル数を数える
```

```
ARRAY STRING(31 ; $asTableName ; $vlNbField) ` SET TABLE TITLESコマンドに
ARRAY INTEGER($aiTableNumber ; $vlNbField)  `渡される配列を初期化する
For($vlTable ; 1 ; $vlNbTable) ` テーブルをループする
    $asTableName{$vlTable}:=Table name($vlTable) `テーブル名を取得する
    $aiTableNumber{$vlTable}:=$vlTable `実際のテーブル番号を格納する
    QUERY([翻訳];[翻訳]実際の名前=$asTableName{$vlTable}; *)
                                                `翻訳テーブルを検索する
    QUERY([翻訳]  &  [翻訳]言語=$1)
    If(Records in table([翻訳])>0)
        `可能なら、ローカライズされたテーブル名を使用する
        $asTableName{$vlTable}:=[翻訳]翻訳された名前
    End if
    $vlNbField:=Count fields($vlTable) `そのテーブルのフィールド数を取得する
    ARRAY STRING(31 ; $asFieldName ; $vlNbTable)
    ` SET TABLE TITLESコマンドに渡される配列を初期化する
    ARRAY INTEGER($aiFieldNumber ; $vlNbTable)
    For($vlField;1;$vlNbField) `フィールドをループする
        $asFieldName{$vlField}:=Field name($vlTable; $vlField)
                                                `フィールド名を取得する
        $aiFieldNumber{$vlField}:=$vlField `実際のフィールド番号を格納する
        QUERY([翻訳];[翻訳]実際の名前=$asFieldName{$vlField} ; *)
        `翻訳テーブルを検索する
        QUERY([翻訳]  &  [翻訳]言語=$1)
        If(Records in table([翻訳])>0)
            `可能なら、ローカライズされたフィールド名を使用する
            $asFieldName{$vlField}:=[翻訳]翻訳された名前 `
        End if
    End for
    SORT ARRAY($asFieldName ; $aiFieldNumber ; >)
    SET FIELD TITLES(Table($vlTable)-> ; $asFieldName ; $aiFieldNumber)
End for
SORT ARRAY($asTableName ; $aiTableNumber ; >)
SET TABLE TITLES($asTableName ; $aiTableNumber)
```

■ 新しいローカライズ版は、コードの修正や再コンパイルを実行することなく、データベースに追加できることに注意してください。

参照

Count tables、SET FIELD TITLES、Table name

GET TABLE TITLES

GET TABLE TITLES（テーブル名;テーブル番号）

引数	タイプ		説明
テーブル名	テキスト配列	←	カレントテーブル名
テーブル番号	倍長整数配列	←	テーブル番号

説明

GET TABLE TITLESコマンドはテーブル名とテーブル番号の配列にストラクチャウインドウもしくは**SET TABLE TITLES**コマンドを使用して定義されたデータベースのテーブル名と番号を代入します。この二つの配列の中身は同期化されます。

セッション中に**SET TABLE TITLES**コマンドが呼び出された場合、**GET TABLE TITLES**コマンドはこのコマンドで定義されて”修正された”名前とテーブル番号だけを返します。

それ以外の場合、**GET TABLE TITLES**コマンドはストラクチャウインドウで定義されたすべてのデータベーステーブル名を返します。いずれの場合もコマンドは目に見えないテーブル（invisible table）は返しません。

参照

GET FIELD TITLES、SET TABLE TITLES

SET FIELD TITLES

SET FIELD TITLES(テーブル | サブテーブル ; フィールドタイトル ; フィールド番号)

引数	タイプ		説明
テーブル¦サブテーブル	テーブル、または サブテーブル	→	フィールドタイトルを設定するテーブルまたはサブテーブル
フィールドタイトル	文字列配列	→	ダイアログボックスに表示されるフィールド名
フィールド番号	数値配列	→	実際のフィールド番号

説明

SET FIELD TITLESコマンドを使用すれば、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」のような標準の4th Dimensionダイアログボックスに表示する際に、テーブルやサブテーブルに渡される、そのテーブルやサブテーブルのフィールドのマスク、名前の変更、並べ替えが行えます。

引数＜フィールドタイトル＞と＜フィールド番号＞配列は同期させる必要があります。配列＜フィールドタイトル＞には、フィールドの名前を、表示したい名前で渡します。ある特定のフィールドを表示したくない場合には、そのフィールド名または新しく付けたタイトルを、配列に指定しないでください。フィールドは、この配列に指定した順序で表示されます。配列＜フィールド番号＞の各要素には、配列＜フィールドタイトル＞の同じ番号の要素で渡されるフィールド名または新規タイトルに対応する実際のフィールド番号を渡します。

例えば、フィールドF、G、Hで構成されるテーブルまたはサブテーブルがあり、フィールドはこの順序で作成されたとします。表示の際には、これらのフィールドをM、N、Oという名前にし、さらにフィールドNは表示したくないとします。最終的に、OとMを、この順序で表示することにします。この場合、配列＜フィールドタイトル＞の2つの要素としてOとMを渡し、配列＜フィールド番号＞の2つの要素として3と1を渡します。

SET FIELD TITLESコマンドは、テーブルの実際の構造（ストラクチャ）を変更するわけではありません。これは、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4th Dimensionダイアログボックスを次回使用する際にのみ影響します。**SET FIELD TITLES**コマンドの有効範囲は、そのワークセッションです。クライアント／サーバにおける利点の1つは、複数の4Dクライアントステーションがそれぞれ異なる見方でサーバのテーブルを同時に「見る」ことができる点です。**SET FIELD TITLES**コマンドは、何度でも呼び出せます。ただし、これは標準の4th Dimensionダイアログボックスが次回に表示された時にしか影響しないことに注意してください。

SET FIELD TITLESコマンドは、以下のような場合に使用します。

■ データベースを動的にローカライズする。

■ データベースの実際の定義に関係なく、自由にフィールドを表示する。

■ 固有のユーザまたは任意ユーザのカスタム特権に応じてフィールドの表示を変える。

警告：SET FIELD TITLESコマンドはフィールドの非表示属性を上書きしません。データベースの設計（デザイン）レベルでフィールドが非表示として設定されている場合、SET FIELD TITLESコマンドへの呼び出しにそのフィールドを指定しても、フィールドは表示されません。

SET FIELD TITLESが呼び出されるたびに、必ずその前後で**SET TABLE TITLES**が呼び出されます。テーブルのタイトルを変更したくないかそうでないかにかかわらず、コマンドはタイトルに影響しません。

例題

▼ **SET TABLE TITLES**コマンドの例を参照してください。

参照

Count fields、Field name、SET TABLE TITLES

GET FIELD TITLES

GET FIELD TITLES（テーブル;フィールド名;フィールド番号）

引数	タイプ		説明
テーブル	テーブル	←	フィールド名を代入する目的のテーブル
フィールド名	テキスト配列	←	カレントフィールド名
フィールド番号	倍長整数配列	←	フィールド番号

説明

GET FIELD TITLESコマンドは、目的テーブルのデータベースのフィールド名および番号の配列にフィールド名とフィールド番号を代入します。これら二つの配列の中身は同期化されます。

セッション中に**SET FIELD TITLES**コマンドが呼び出された場合、**GET FIELD TITLES**コマンドはこのコマンドで定義されて"修正された（modified）"名前とフィールド番号だけを返します。それ以外の場合、**GET FIELD TITLES**コマンドはストラクチャウインドウで定義されたすべてのデータベースのフィールド名を返します。いずれの場合もコマンドは見えないフィールド（invisible filed）は返しません。

参照

GET TABLE TITLES、SET FIELD TITLES

Shift down

Shift down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	shiftキーのステータス

説明

Shift down関数は、shiftキーが押されているとTrue（真）を返します。

例題

ボタン「bAnyButton」用の以下のオブジェクトメソッドは、ボタンがクリックされたときにどのモディファイアキーが押されているかによって、異なる動作（アクション）を実行します。

```
`「bAnyButton」ボタンのオブジェクトメソッド
Case of
        `このほかの複数のキーの組み合わせをここでテストすることも可能
        ...
    ¥(Shift down & Windows Ctrl down)
        `ShiftおよびWindows Ctrl(またはMacintosh command)
                                                キーが押されてた場合
        DO ACTION1
        ...
    ¥(Shift down)
        `Shiftキーだけが押されてた場合
        DO ACTION2
        ...
    ¥(Windows Ctrl down)
        `Windows Ctrl（またはMacintosh command）キーだけが押された場合
        DO ACTION3
        ...
        `このほかの個々のキーをここでテストすることも可能
        ...
End case
```

参照

Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Windows Alt down、Windows Ctrl down

Caps lock down

Caps lock down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	Caps Lockキーのステータス

説明

Caps lock down関数は、Caps Lockキーが押されているとTrue（真）を返します。

例題

Shift down関数の例を参照してください。

参照

Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

Windows Ctrl down

Windows Ctrl down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	WindowsのCtrlキーのステータス （Macintoshの場合は、commandキー）

説明

Windows Ctrl down関数は、WindowsのCtrlキーが押されているとTrue（真）を返します。

注：Macintoshプラットフォーム上で呼び出された場合は、Windows Ctrl down関数はMacintoshのcommandキーが押されているとTrue（真）を返します。

例題

Shift down関数の例を参照してください。

参照

Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down

Windows Alt down

Windows Alt down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	WindowsのAltキーのステータス（Macintoshの場合は、optionキー）

説明

Windows Alt down関数は、WindowsのAltキーが押されているとTrue（真）を返します。

注：Macintoshプラットフォーム上で呼び出された場合は、Windows Alt down関数はMacintoshのoptionキーが押されているとTrue（真）を返します。

例題

Shift down関数の例を参照してください。

参照

Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Ctrl down

Macintosh command down

Macintosh command down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	Macintoshのcommandキーのステータス（Windowsの場合は、Ctrlキー）

説明

Macintosh command downコマンドは、Macintoshのcommandキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh command down関数はWindowsのCtrlキーが押されているとTrue（真）を返します。

例題

Shift down関数の例を参照してください。

参照

Caps lock down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

Macintosh option down

Macintosh option down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	Macintoshのoptionキーのステータス（Windowsの場合は、Altキー）

説明

Macintosh command downコマンドは、Macintoshのoptionキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh option down関数はWindowsのAltキーが押されているとTrue（真）を返します。

例題

Shift downコマンドの例を参照してください。

参照

Caps lock down、Macintosh command down、Macintosh control down、Shift down、Windows Alt down、Windows Ctrl down

Macintosh control down

Macintosh control down → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	Macintoshのcontrolキーのステータス

説明

Macintosh control downコマンドは、Macintoshのcontrolキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh control down関数は常にFalse（偽）を返します。このMacintoshと同等のキーは、Windows上にはありません。

例題

Shift down関数の例を参照してください。

参照

Caps lock down、Macintosh command down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

GET MOUSE

GET MOUSE(水平 ; 垂直 : マウスボタン { ; *})

引数	タイプ		説明
水平	数値	←	マウスの水平座標
垂直	数値	←	マウスの垂直座標
マウスボタン	数値	←	マウスボタンの状態 0= ボタンは押されていない 1= ボタンの押下 2= 右マウスボタンの押下 3= 両方のボタンの押下
*		→	指定した場合、グローバル座標システムが使用される。 省略した場合、ローカル座標システムが使用される。

説明

GET MOUSEコマンドは、マウスの現在の状態を返します。

水平座標と垂直座標が、引数＜水平＞と＜垂直＞に返されます。オプション引数＜*＞を渡すと、これらの座標は画面（スクリーン）に対して相対的に表されます。引数＜*＞を省略すると、座標はカレントプロセスの最前面のウインドウに対して相対的に表されます。

引数＜マウスボタン＞は、上記のようにボタンの状態を返します。

注：＜マウスボタン＞の値2と3は、バージョン10.2.5以降のMac OS X上でのみ返されます。

例題

Pop up menu関数の例題を参照してください。

参照

Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、ON EVENT CALL、Shift down、Windows Alt down、Windows Ctrl down

Pop up menu

Pop up menu(項目テキスト{; デフォルト{; X座標; Y座標}}) → 数値

引数	タイプ		説明
項目テキスト	テキスト	→	メニューテキストの定義
デフォルト	数値	→	デフォルトとして選択されるメニュー項目の番号
X座標	数値	→	左上端のX座標
Y座標	数値	→	左上端のY座標
戻り値	数値	←	選択されたメニュー項目の番号

説明

Pop up menu関数は、マウスの現在の位置にポップアップメニューを表示します。

ユーザインタフェースの規則に従うために、通常このコマンドは、マウスクリックに応じて、またマウスボタンが押されたままの状態である場合に呼び出します。

ポップアップメニューの項目（アイテム）は、以下のように引数＜項目テキスト＞を使用して、以下のように定義します。

■ 各項目の間をセミコロン「;」で区切ります。例えば、“項目テキスト1 ; 項目テキスト2 ; 項目テキスト3”。

■ 項目を選択不可にするには、項目テキスト内に開いたカッコ「(」を指定します。

■ 区切り線を指定するには、項目テキストとして「(-」を渡します。

■ 行のフォントスタイルを指定するには、項目テキスト内に記号「<」に続けて以下の文字のいずれかを指定します。

<B　太字（ボールド）
<I　斜体（イタリック）
<U　アンダーライン
<O　アウトライン（Macintoshのみ）
<S　シャドウ（Macintoshのみ）

■ 項目にチェックマークを付けるには、項目テキスト中に疑問符「?」に続けてチェックマークとして表示したい文字を指定します。Macintoshの場合、その文字が表示されます。Windowsの場合は、どのような文字を渡したかにかかわらず、チェックマークが表示されます。

■ 項目にアイコンを追加するには、項目テキスト内にアクセント（^）に続けて、1バイトの文字を指定します。指定する文字のASCIIコードから48を引いた数がMacintoshベースのアイコンリソースのリソースIDになります。つまり、Ascii("1バイトの文字") - 48 + 256 = “cicn”リソースのリソースID、ということです。

■ 項目にショートカットを追加するには、項目テキスト中にスラッシュ「／」に続けてその項目のためのショートカット文字を指定します。ただし、この最後のオプションは純粋に情報提供のためのものであることに注意してください。ショートカットがポップアップメニューをアクティブにすることはありません。ただし、アプリケーションのメインメニューバーにそのポップアップメニュー項目に相当するものがある場合には、ショートカットを含めることも可能です。

オプション引数の＜デフォルト＞によって、ポップアップメニューが表示される際に選択される省略時（デフォルト）のメニュー項目を指定することができます。1からメニュー項目の数までの値を渡してください。この引数を省略すると、コマンドは最初のメニュー項目を省略時の設定として選択します。

メニュー項目を選択した場合には、コマンドはその番号を返します。選択しなければ、ゼロ（0）を返します。

注：適当な項目数のポップアップメニューを使用してください。50以上もの項目を表示したい場合は、ポップアップメニューではなく、フォーム内のスクロール可能エリアの使用を検討するほうが賢明です。

例題

プロジェクトメソッド「MYコンテキストメニュー」は、テーブル名選択コンテキストメニューをプルダウンします。

```
`「MYコンテキストメニュー」プロジェクトメソッド
GET MOUSE($vlMouseX ; $vlMouseY ; $vlButton)
If(($vlButton=2) | (Macintosh control down))
	$vtItems:="About this database...<I;(-;!-Other Options;(-"
	For($vlTable ; 1 ; Count tables)
		$vtItems:=$vtItems+";"+Table name($vlTable)
	End for
	$vlUserChoice:=Pop up menu($vtItems)
	Case of
		¥($vlUserChoice=1)
			`情報を表示する
		¥($vlUserChoice=2)
			`オプションを表示する
	Else
		If($vlUserChoice>0)
```

```
                    ` 番号が$vlUserChoice-4のテーブルに移動する。
                End if
            End case
        End if
```

このプロジェクトメソッドは、以下から呼び出せます。

■ マウスボタンがリリースされるのを待たずにマウスクリックに反応するフォームオブジェクトのメソッド（透明ボタン）

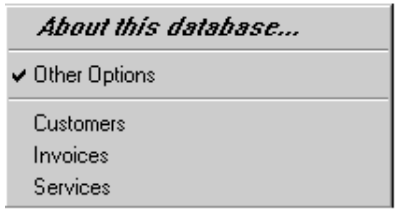
■ イベントを「スパイ」し、他のプロセスと通信するプロセス

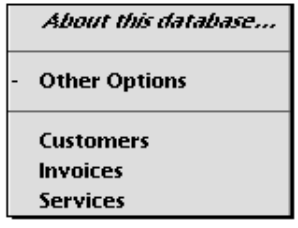
■ **ON EVENT CALL**コマンドを使用してインストールされるイベント処理メソッド

最後の2つのケースでは、フォームオブジェクトにおいてクリックが行われる必要はありません。これは、**Pop up menu**関数の利点の1つです。一般的に、ポップアップメニューを表示するにはフォームオブジェクトを使用します。ポップアップメニューを使えば、どこにでもメニューを表示できます。

Windowsでは、ポップアップメニューはマウスの右ボタンを押すことによって表示されます。Macintoshでは、「control+クリック」を押すことによって表示されます。ただし、このメソッドは、実際にはマウスがクリックされたかどうかをチェックしないことに注意してください。このテストは呼び出し側のメソッドが行います。

以下の図は、Windows（左）上とMacintosh（右）上で表示されるポップアップメニューの例です。Windows版では、標準のチェックマークが使われていることに注意してください。





参照

GET MOUSE

POST KEY

POST KEY(コード {; モディファイア {; プロセス}})

引数	タイプ		説明
コード	数値	→	文字のASCIIコード、 またはファンクションキーコード
モディファイア	数値	→	モディファイアキーのステータス
プロセス	数値	→	送信先のプロセス参照番号、 省略または0の場合、アプリケーション イベントキュー

説明

POST KEYコマンドは、キーストロークをシミュレート（模倣）します。これは、ユーザが実際にキーボード上で文字を入力した場合と同様の結果を生じます。

引数＜コード＞には、その文字のASCIIコードを渡します。

オプション引数＜モディファイア＞を渡す場合には、イベント（モディファイア）定数を1つあるいは組み合わせて渡します。例えば、shiftキーをシミュレートするには、shiftキーマスクを渡します。引数＜モディファイア＞を渡さなかった場合には、モディファイアはシミュレートされません。

オプション引数＜プロセス＞を指定すると、キーストロークは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、キーストロークはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

例題

▼ **Process number**コマンドの例を参照してください。

参照

POST CLICK、POST EVENT

POST CLICK

POST CLICK(マウスX ; マウスY {; プロセス}{; *})

引数	タイプ		説明
マウスX	数値	→	水平座標
マウスY	数値	→	垂直座標
プロセス	数値	→	送信先のプロセス参照番号、 省略または0の場合、アプリケーション イベントキュー
*		→	指定された場合はグローバルな座標 システムが使用される 省略された場合はローカルな座標 システムが使用される

説明

POST CLICKコマンドはマウスクリックをシミュレート（模倣）します。これは、ユーザが実際にマウスボタンをクリックした場合と同様の結果を生じます。

引数＜マウスX＞と＜マウスY＞には、クリックの水平座標と垂直座標を渡します。オプション引数＜*＞を省略すると、これらの座標は、＜プロセス＞に渡したプロセス番号を持つプロセスの最前面のウインドウに対して相対的に表されます。引数＜*＞を渡した場合、これらの座標は画面（スクリーン）に対して相対的に表されます。

オプション引数＜プロセス＞を指定すると、クリックは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、クリックはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

参照

POST EVENT、POST KEY

POST EVENT

POST EVENT(タイプ ; メッセージ ; 時間 ; マウスX ; マウスY ; モディファイア {; プロセス})

引数	タイプ		説明
タイプ	数値	→	イベントの種類
メッセージ	数値	→	イベントメッセージ
時間	数値	→	イベント時間（Tick単位）
マウスX	数値	→	マウスの水平座標
マウスY	数値	→	マウスの垂直座標
モディファイア	数値	→	モディファイアキーのステータス
プロセス	数値	→	送信先のプロセス参照番号、省略または0の場合、アプリケーションイベントキュー

説明

POST EVENTコマンドは、キーボードまたはマウスイベントをシミュレート（模倣）します。これは、ユーザが実際にキーボードやマウス上で動作を行った場合と同様の結果を生じます。

引数＜タイプ＞には、以下のようなあらかじめ定義された定数のいずれかを渡します。

定数	タイプ	値
Mouse down event	倍長整数	1
Mouse up event	倍長整数	2
Key down event	倍長整数	3
Key up event	倍長整数	4
Auto key event	倍長整数	5

イベントがマウス関連のイベントであれば、引数＜メッセージ＞に0（ゼロ）を渡します。イベントがキーボード関連のイベントであれば、シミュレートされる文字のASCIIコードを＜メッセージ＞に渡します。

引数＜時間＞には、通常、**Tickcount**関数によって返される値を渡します。

イベントがマウス関連のイベントであれば、クリックの水平座標と垂直座標を＜マウスX＞と＜マウスY＞に渡します。

引数＜モディファイア＞には、イベント（モディファイア）定数を1つあるいは組み合わせて渡します。例えば、shiftキーをシミュレートするには、shiftキービットを渡します。

オプション引数＜プロセス＞を指定すると、イベントは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、イベントはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

参照

POST CLICK、POST KEY

SCROLL LINES

SCROLL LINES({*; }オブジェクト{; 位置}{;* })

引数	タイプ		説明
*		→	指定した場合、<オブジェクト>はオブジェクト名（文字列） 省略した場合、<オブジェクト>はテーブル
オブジェクト	フォームオブジェクト	→	オブジェクト名（“*”を指定した場合）、またはテーブル（“*”を省略した場合）
位置	倍長整数	→	セレクション中のレコードの位置
*		→	スクロール後に行を最初の位置に表示する

説明

SCROLL LINESコマンドを使用すると、最初に選択したレコードや特定のレコードが表示されるように、リストフォーム（**MODIFY SELECTION**、**DISPLAY SELECTION**コマンドを用いて表示される）またはサブフォームの行をスクロールすることができます。

最初の任意の引数“*”を渡すと、引数<オブジェクト>がサブフォームオブジェクトの名前であることを示します（この場合、<オブジェクト>には文字列を渡す）。この引数を渡さない場合、引数<オブジェクト>がテーブルであることを示します（リストフォームテーブルまたはサブフォームテーブル）。

引数<位置>を渡さない場合、リスト内で最初に選択したレコード（強調表示される）が表示されるように、リスト行を縦方向にスクロールします。

選択されたレコードが存在しない場合、コマンドは何も行いません。少なくとも1つの選択レコードが表示されている場合も、コマンドは何も行いません。

引数<位置>を使用すると、現在の選択項目中のレコード位置（レコード選択番号）を指示することができます（その位置）。この引数を渡すと、指定したレコード（強調表示されているかどうかに関わらず）が表示されるように、リスト行が縦方向にスクロールされます。レコードが既に表示されている場合、コマンドは何も行いません。

2番目の任意の引数“*”を渡すと、このコマンドを用いて表示される行はリストの一番目の位置に配置されます。

注：HIGHLIGHT RECORDSコマンドには引数“*”（任意）が導入され、これを使用するとスクロール管理をSCROLL LINESコマンドに委ねることができます。

参照

HIGHLIGHT RECORDS

GET HIGHLIGHT

GET HIGHLIGHT(エリア ; 先頭 ; 最終)

引数	タイプ		説明
エリア	フィールドまたは変数	→	入力可能のフィールドまたは変数
先頭	数値	←	反転表示された先頭位置
最終	数値	←	反転表示された最終位置

説明

GET HIGHLIGHTコマンドは、現在反転表示されているテキストを検出するために使用します。

警告：GET HIGHLIGHTコマンドに対して、フィールドや変数による入力可能エリアの名前を渡した場合でも、このコマンドは現在編集中のエリアに適用されたときにだけ、選択位置を返します。

注：このコマンドはサブフォームのリストフォームにあるフィールドに対して使用することはできません。

テキストの反転は、ユーザによる指定や**HIGHLIGHT TEXT**コマンドの実行で実行することができます。

変数<先頭>に、反転表示された文字の先頭位置を代入します。変数<最終>に、反転表示された最終の文字位置に1を加えた値を代入します。

<先頭>と<最終>が同じ場合は、テキストは選択されていない状態で、反転されている文字はありません。この場合の挿入ポインタ（カーソル）は、<先頭>の1文字前にあります。

例題

1. 以下の例では"[Products]Comments"というフィールドの反転表示された部分を検出します。

```
GET HIGHLIGHT([Products]Comments;vFirst;vLast)
If(vFirst<vLast)
        ALERT("The selected text is:"+Substring([Products]Comments;vFirst;
                                                vLast-vFirst))
```

```
End if
```

2. **FILTER KEYSTROKE**コマンドの例を参照してください。

参照

FILTER KEYSTROKE、HIGHLIGHT TEXT、Keystroke

HIGHLIGHT TEXT

HIGHLIGHT TEXT(エリア; 先頭 ; 最終)

引数	タイプ		説明
エリア	フィールドまたは変数	→	入力可能なフィールドまたは変数
先頭	数値	→	反転表示の先頭位置
最終	数値	→	反転表示の最終位置

説明

HIGHLIGHT TEXTコマンドは、＜エリア＞の中のテキストの一部を反転表示します。

＜エリア＞が現在編集中のオブジェクトの場合、フォーカスはこのエリアにセットされます。

注：HIGHLIGHT TEXTコマンドをサブフォームのリストフォーム内にあるフィールドに対して使用することはできません。

＜先頭＞は、反転表示する先頭文字の位置です。＜最終＞は、反転表示する最終文字の位置に1を加えた値です。＜先頭＞と＜最終＞が同じ場合は、挿入ポインタ（カーソル）が＜先頭＞で指定された文字の前に置かれ、文字は全く反転表示されません。

＜最終＞が＜エリア＞の文字数より大きい場合は、＜先頭＞からテキストの最終までのすべての文字を反転表示します。

例題

1. 以下の例は、フィールド“コメント”のテキストを反転表示します。結果を以下の図に示します。

```
HIGHLIGHT TEXT(コメント ; 9 ; 13)` テキストの反転表示
```

2. 以下の例は、フィールド“コメント”上に挿入ポインタ(カーソル) を表示します。結果を以下の図に示します。

```
HIGHLIGHT TEXT(コメント ; 11 ; 11)` コメントフィールドの挿入ポイントの位置
```

3. 以下の例は挿入ポイントを入力可フィールド[製品]コメントの終わりに移動します。

```
$vLen:=Length([製品]コメント)+1
HIGHLIGHT TEXT([製品]コメント;$vLen;$vLen)
```

4. **FILTER KEYSTROKE**コマンドの例を参照してください。

参照

GET HIGHLIGHT

SET CURSOR

SET CURSOR {(カーソル)}

引数	タイプ		説明
カーソル	数値	→	Macintoshベースのカーソルリソース番号

説明

SET CURSORコマンドは、マウスカーソルを引数＜カーソル＞に渡したID番号を持つMacintoshベースの“CURS”リソースに格納されたカーソルに変更します。

引数を省略すると、マウスカーソルは標準の矢印に設定されます。

使用可能なカーソルのリストを得るには、**RESOURCE LIST**コマンドを使用してください。

参照

RESOURCE LIST

Focus object

Focus object → ポインタ

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ポインタ	←	フォーカスを取得しているオブジェクトへのポインタ

説明

Focus object関数は、カレントフォーム上でフォーカスを取得しているオブジェクトへのポインタを返します。フォーカスを取得したオブジェクトが存在しない場合、コマンドはNilを返します。**Focus object**関数を使用すれば、現在選択されているオブジェクトを知らなくても、そのフォームエリアに対して動作を実行することができます。オブジェクトに対して動作を実行する前に**Type**関数を使用して、オブジェクトが正しいデータタイプかどうかを確認してください。**Focus object**関数は、サブフォーム内のフィールドで使用することはできません。

注：リストボックスタイプのオブジェクトと一緒に使用すると、Focus object関数はフォーカスを取得しているリストボックスの列（配列）へのポインタを返します。

例題

▼以下の例は、ボタンのオブジェクトメソッドです。このメソッドは、カレントオブジェクトのデータを大文字に変換します。この場合、オブジェクトのデータタイプは、テキストまたは文字列でなければなりません（データタイプが0または24）。

```
$p:= Focus object ` ポインタを保存する
` データタイプが文字列またはテキストエリアの場合
If((Type($p->)=0) |(Type($p->)=24))
        $p->:=Uppercase($p->) ` エリアのテキストを大文字に変換する
End if
```

REDRAW

REDRAW(オブジェクト)

引数	タイプ		説明
オブジェクト	オブジェクト	→	サブフォームを再描画するサブテーブル、またはサブフォームを再描画するテーブル、またはエリアを再描画するフィールド、またはエリアを再描画する変数

説明

メソッドを使用して、サブフォームで表示されるフィールドやサブフィールドの値を変更する場合に、フォームを確実に変更するために**REDRAW**コマンドを使用します。

Webサーバ：ON Timerフォームイベントの後で実行した場合、REDRAW関数はWebブラウザに送った4Dフォームは定期的に更新するために呼びだせます。詳しい説明はSET TIMER関数を参照してください。

参照

SET TIMER

INVERT BACKGROUND

INVERT BACKGROUND({*; } テキスト変数 | テキストフィールド)

引数	タイプ		説明
*		→	変数またはオブジェクト名を指定
テキスト変数 ¦ テキストフィールド	変数または フィールド	→	反転表示される変数またはフィールド

説明

INVERT BACKGROUNDコマンドは、フォーム中の＜テキスト変数＞または＜テキストフィールド＞を反転表示するために使用します。このコマンドの有効範囲は使用中のフォームです。

このコマンドは、画面への表示やドットマトリックスプリンタへの印刷を実行する際に使用します。ポストスクリプトプリンタは反転させた印刷をすることはできません。

出力フォーム内の変数を反転することはできません。**INVERT BACKGROUND**コマンドは、入力可能な変数に対して使用しないようにしてください。フォーカスを取得している変数やオブジェクトに対し、このコマンドは何も行いません。複数行に渡る入力可能なテキストエリアにこのコマンドを実行して反転表示させた後、文字を入力すると、文字を含む行だけが反転表示されなくなります。

例題

▼以下の例は、フィールドの値を判定する入力フォーム内の変数のオブジェクトメソッドです。このオブジェクトメソッドは、フィールドの値が正の場合は何も実行せず、負の場合はフォーム内にある変数の表示を反転します。

```
vAmount:=[Accounts]Amount        ` 変数にフィールドの値を代入する
If(vAmount <0)                   ` 変数の値が負の場合
      NVERT BACKGROUND(vAmount)` 変数を反転表示にする
End if
```

注：このコマンドは本来、白黒のユーザインタフェースを作成するためのものでしたが、現在はほとんど使われません。一般的に、フィールドや変数を反転するにはカラーを使用します。

参照

SET COLOR、SET RGB COLOR

EDIT ACCESS

EDIT ACCESS

説明

EDIT ACCESSコマンドは、ユーザにパスワードシステムの編集環境を提供します。このコマンドは、「デザイン」モードの「パスワード」エディタを表示します。

グループは、デザイナ、管理者、グループオーナーによって編集されます。デザイナと管理者は、すべてのグループを編集することができます。グループオーナーは、所有するグループだけを編集することができます。また、グループオーナーはユーザをグループに割り当てたり、削除することができます。**EDIT ACCESS**コマンドは、グループが定義されていない場合には効果がありません。

デザイナと管理者は、ユーザをグループに割り当てたり、削除するだけでなく、新しいユーザを登録することもできます。

例題

▼ 以下の例は、ユーザに対して「パスワード」エディタを表示します。

```
EDIT ACCESS
```

参照

CHANGE ACCESS、CHANGE PASSWORD

CHANGE CURRENT USER

CHANGE CURRENT USER({ユーザ名}{;パスワード})

引数	タイプ		説明
ユーザ	文字列¦数値	→	ユーザ名またはユニークなユーザID
パスワード	文字列	→	パスワード（符号化されていない）

説明

CHANGE CURRENT USERコマンドを使用すると、データベースを終了しなくてもデータベースのカレントユーザの識別情報を変更することができます。ユーザは、データベース接続ダイアログボックスを使用するか（引数を指定せずにコマンドを呼び出した場合）、あるいは直接コマンドを用いて、自分自身の識別情報を変更することができます。ユーザが自分の識別情報を変更すると、選ばれたユーザに属するアクセス権を優先して、以前のアクセス権はすべて破棄されます。

引数を指定せずに**CHANGE CURRENT USER**コマンドを実行すると、データベース接続ダイアログボックスが表示されます。次にユーザはデータベースへ入力を行うため、有効なユーザ名とパスワードを入力するか選択しなければなりません。接続ダイアログボックスの内容は、データベース「環境設定」の「アプリケーション/アクセス」ページに設定したオプションに応じて変わります。

また、2つの任意の引数と＜パスワード＞を渡して、使用する新しいアカウントをプログラムから指定することができます。

引数＜ユーザ名＞には、使用するアカウントのユーザ名またはユニークなユーザID（ユーザ参照）を渡します。ユーザの名前およびIDは、**GET USER LIST**コマンドを用いて取得することができます。

ID	ユーザの説明
1	Designer（デザイナ）
2	Administrator（管理者）
1～15000	Designerが作成したユーザ（ユーザ番号3はDesignerが作成した1番目のユーザ、ユーザ番号 4は2番目のユーザ、等）
-11～-15010	Administratorが作成したユーザ（ユーザ番号-11はAdministratorが作成した1番目のユーザ、ユーザ番号-12は2番目のユーザ、等）

ユーザアカウントが存在しない、または削除されている場合は、エラー-9979が返されます。このエラーは**ON ERR CALL**コマンドにより設定されたエラー処理用メソッドを用いて遮ることができます。エラー処理メソッドを使用しない場合は、このコマンドを呼び出す前に**Is user deleted**関数を使用して、ユーザアカウントを検証することができます。

引数<パスワード>には、符号化されていないユーザアカウントパスワードを渡します。このパスワードがユーザと一致しない場合、コマンドはエラーメッセージ-9978を返し、処理は行いません。

フラッディング（総当たり攻撃）つまり、複数のユーザ名とパスワードの組み合わせによる攻撃を避けるため、コマンドの実行は遅延されます。この結果、このコマンドは4度目に呼び出された後、10秒間待機した後ではじめて実行されます。この遅延はワークステーション全体にわたるものです。

独自のアクセス管理ダイアログボックスを提供する

CHANGE CURRENT USERコマンドを使用すると、4th Dimensionアクセス制御システムと同じ利点を活用した、ユーザ名やパスワード入力を行う独自のダイアログボックス（入力ルールや期限切れルールを伴う）を実装することが可能です。

このダイアログボックスの機能は次の通りです。

1. データベースへの入力は、ダイアログボックスを用いずに、“デフォルトユーザ”モードで直接行われます。

2. 「On Startup」データベースメソッドにより、ユーザ名とパスワード入力のための独自のダイアログボックスが表示されます。このダイアログボックスでは、あらゆるタイプの処理を予測することができます。

   ■ **GET USER LIST**コマンドを使用し、4th Dimension標準のアクセスダイアログボックスと同様に、データベースユーザの一覧を表示することができます。

   ■ パスワード入力フィールドに各種制御機能を盛り込み、入力された文字が有効であるかどうかをチェックすることができます（最小文字数、パスワードの一意性、等）。

   ■ ダイアログボックスが確定された際、期限に関するルールを適用することができます。つまり、有効期限、初回接続への強制変更、不正なログインが繰り返された場合にアカウントをロック、使用済みのパスワードを記憶、等のルールです。

3. 入力が確定されると、ユーザアカウントのアクセス権を用いてデータベースを開くため、必要な情報（ユーザ名とパスワード）が**CHANGE CURRENT ACCESS**コマンドに渡されます。

例題

▼ 次の例題は、接続ダイアログボックスを表示します。

```
CHANGE CURRENT USER
```

Validate password

Validate password(ユーザID ; パスワード) → ブール

引数	タイプ		説明
ユーザID	数値	→	ユニークなユーザID番号
パスワード	文字列	→	暗号化されていないパスワード
戻り値	ブール	←	True=ユーザアカウント用のパスワードの場合 False=ユーザアカウントのパスワードでない場合

説明

Validate password関数は、引数＜パスワード＞に渡した文字列が引数＜ユーザID＞に渡されたID番号を持つユーザアカウント用のパスワードの場合、True（真）を返します。

フラッディング（総当たり攻撃）つまり、複数のユーザ名・パスワードの組み合わせによる攻撃を避けるため、コマンドの実行は遅延されます。この結果、このコマンドは4度目に呼び出された後、10秒間待機した後ではじめて実行されます。この遅延はワークステーション全体にわたるものです。

例題

▼ 以下の例は、ユーザ“山田”のパスワードが“Laurel”であるかどうかを調べます。

```
GET USER LIST(atUserName ; alUserID)
$vlElem:=Find in array(atUserName ; "山田")
If($vlElem>0)
    If(Validate password(alUserID{$vlElem} ; "Laurel"))
        ALERT("正解です。")
    Else
        ALERT("間違っています！")
    End if
Else
    ALERT("ユーザ名が見つかりません。")
End if
```

参照

GET USER PROPERTIES、SET USER PROPERTIES

CHANGE PASSWORD

CHANGE PASSWORD(パスワード)

引数	タイプ		説明
パスワード	文字列	→	新しいパスワード

説明

CHANGE PASSWORDコマンドは、カレントユーザのパスワードを変更します。このコマンドは、カレントパスワードを新しいパスワード、＜パスワード＞に置き換えます。

警告：パスワードでは大文字小文字が区別されます。

例題

▼以下の例では、ユーザがパスワードの変更を行います。

```
CHANGE ACCESS ` ユーザに「パスワード」ダイアログを表示
If(OK=1)
    $pw1:=Request("新しいパスワード："+Current user)
        ` パスワードは少なくとも6文字以上にする
    If(((OK=1) &($pw1#"")) &(Length($pw1)>5))
        ` パスワードが正しく入力されたことを確認する
        $pw2:=Request("もう一度パスワードを入力してください：")
        If((OK=1) &($pw1=$pw2))
            CHANGE PASSWORD($pw2) ` パスワードの変更
        End if
    End if
End if
```

参照

CHANGE ACCESS

Current user

Current user → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	文字列	←	カレントユーザのユーザ名

説明

Current userコマンドは、カレントユーザのユーザ名を返します。

例題

▼ **User in group**コマンドの例題を参照してください。

参照

CHANGE ACCESS、CHANGE PASSWORD、User in group

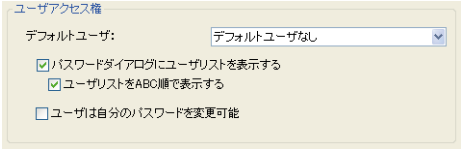
Get default user

Get default user→ 数値

引数	タイプ		説明
戻り値	数値	←	ユニークなユーザID番号

説明

このコマンドは、データベースの「環境設定」ダイアログボックスで「デフォルトユーザ」として設定されたユーザのユニークなユーザIDを返します。



デフォルトユーザ

ユーザIDとして、次の数値が使用されます。

ID	ユーザの説明
1	Designer（デザイナ）
2	Administrator（管理者）
1～15000	Designerが作成したユーザ（ユーザ番号3はDesignerが作成した1番目のユーザ、ユーザ番号4は2番目のユーザなど）
-11～-15010	Administratorが作成したユーザ（ユーザ番号-11はAdministratorが作成した1番目のユーザ、ユーザ番号-12は2番目のユーザなど）

デフォルトユーザが指定されていない場合、コマンドは0を返します。

User in group

User in group(ユーザ ; グループ) → ブール

引数	タイプ		説明
ユーザ	文字列	→	ユーザ名
グループ	文字列	→	グループ名
戻り値	ブール	←	True=ユーザはグループに属する False=ユーザはグループに属さない

説明

User in groupコマンドは、＜ユーザ＞が＜グループ＞に属していれば“True（真）”を返します。

例題

▼以下の例は、特定の送り状を探しています。カレントユーザが“Manager”グループに属していれば、そのユーザは機密情報を表示するフォームをアクセスすることができます。そのユーザが“Manager”グループに属さなければ、別のフォームを表示します。

```
QUERY([送り状] ; [送り状]小売り > 10000)
If(User in Group(Current user ; "Manager"))
    OUTPUT FORM([送り状] ; "機密出力")
    INPUT FORM([送り状] ; "機密入力")
Else
    OUTPUT FORM([送り状] ; "出力")
    INPUT FORM([送り状] ; "入力")
End if
MODIFY SELECTION([送り状] ; *)
```

参照

Current user

DELETE USER

DELETE USER(ユーザID)

引数	タイプ		説明
ユーザID	数値	→	削除するユーザのID番号

説明

DELETE USERコマンドは、引数<ユーザID>に渡したユーザID番号を持つユーザを削除します。この場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、または既に削除されている場合は、エラーコード-9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

削除されたユーザ名は、データベースが開かれたり、**CHANGE ACCESS**コマンドを呼び出した際に表示される「パスワード」ウインドウにはもはや表示されません。しかし、一意のユーザID番号を管理するために、そのユーザアカウントはパスワードシステム内に保持されています。削除されたユーザ名は、「デザイン」モードの「パスワード」ウインドウにおいて緑（グリーン）で表示されます。

参照

GET USER LIST、GET USER PROPERTIES、Is user deleted、SET USER PROPERTIES

エラー処理

DELETE USERコマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Is user deleted

Is user deleted(ユーザID) → ブール

引数	タイプ		説明
ユーザID	数値	→	ユーザID番号
戻り値	ブール	←	True=ユーザアカウントが削除されたり、存在しない場合 False=ユーザアカウントがアクティブの場合

説明

Is user deleted関数は、引数＜ユーザID＞に渡された一意のユーザID番号を持つユーザアカウントを調べます。

ユーザアカウントが存在しなかったり、既に削除されてしまっている場合、**Is user deleted**関数はTrueを返します。ユーザアカウントがアクティブの場合はFalseを返します。

参照

DELETE USER、GET USER PROPERTIES、SET USER PROPERTIES

エラー処理

Is user deleted関数を呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET USER LIST

GET USER LIST(ユーザ名 ; ユーザID)

引数	タイプ		説明
ユーザ名	文字列配列	←	「パスワードエディタ」ウインドウ内に表示されるユーザの名前
ユーザID	数値配列	←	対応する一意のユーザID番号

説明

GET USER LISTコマンドは、引数<ユーザ名>と<ユーザID>の配列に「パスワード」ウインドウ内に表示されるユーザの名前と一意のID番号割り当てます。

引数<ユーザ名>配列には、「パスワード」ウインドウ内に表示されるユーザ名が代入されます。これらのユーザ名には、アカウントが無効になったユーザの名前（「パスワード」ウインドウに緑（グリーン）で表示されるユーザ名）も含まれます。

注：削除されたユーザを見つけるには、Is user deleted関数を使用します。

<ユーザ名>と同期される引数<ユーザID>配列には、対応する一意のユーザID番号が代入されます。このID番号は、下記の値または範囲を持っています。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3～15000	デザイナによって作成されたユーザ （つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザ）
-11～-15000	管理者によって作成されたユーザ （つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザ）

参照

GET GROUP LIST、GET USER PROPERTIES、SET USER PROPERTIES

エラー処理

GET USER LISTコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET USER PROPERTIES

GET USER PROPERTIES(ユーザID; ユーザ名; 起動メソッド; パスワード;
ログイン回数; 前回ログイン{; メンバー{; グループオーナ})

引数	タイプ		説明
ユーザID数値	→		ユニークなユーザ ID番号
ユーザ名	文字列	←	ユーザの名前
起動メソッド	文字列	←	Startupメソッドの名前
パスワード	文字列	←	常に空の文字列
ログイン回数	数値	←	データベースにログインした回数
前回ログイン	日付	←	データベースに最後にログインした日付
メンバー	数値配列	←	ユーザが属すグループのID番号
グループオーナ	数値	←	ユーザグループオーナーのID番号

説明

GET USER PROPERTIESコマンドは、引数＜ユーザID＞に渡された一意のユーザID番号を持つユーザに関する情報を返します。この場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、または既に削除されている場合は、エラーコード-9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。それ以外は、**Is user deleted**関数を呼び出して、**GET USER PROPERTIES**コマンドを呼び出す前にユーザアカウントを調べることができます。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3〜15000	デザイナによって作成されたユーザ （つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザ）
-11〜-15000	管理者によって作成されたユーザ （つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザ）

このコマンドを呼び出すと、引数＜ユーザ名＞、＜起動メソッド＞、＜パスワード＞、＜ログイン回数＞、＜最終日付＞にユーザの名前、Startupメソッド、空の文字列、およびデータベースにログインした回数および最後にログインした日付を返します。

オプション引数＜グループID＞を指定すると、ユーザが属すグループの一意のID番号を返します。グループID番号は、以下の範囲の番号を持つことができます。

任意の引数＜グループオーナ＞を渡すと、ユーザグループの“オーナー”、つまりこのユーザが作成したオブジェクトのデフォルトオーナーグループのID番号を取得することができます。

グループのID番号は、次の通りです。

グループID番号	グループの種類
15000～32767	デザイナまたは関連したグループオーナーによって作成されたグループ（つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ）
-150011～-32768	管理者または関連したグループオーナーによって作成されたグループ（つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ）

参照

GET GROUP LIST、GET USER LIST、SET USER PROPERTIES

エラー処理

GET USER PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

USERS TO BLOB

USERS TO BLOB (ユーザ)

引数	タイプ		説明
ユーザ	BLOB	→	ユーザを格納するBLOB
		←	ユーザアカウント（符号化済）

説明

USERS TO BLOBコマンドは、すべてのユーザアカウントのリストと、Administrator（管理者）が作成したデータベースグループをBLOBタイプの引数<ユーザ>に格納します。

このコマンドを実行できるのは、データベース管理者だけです。他のユーザがこのコマンドを実行しようとすると、コマンドは何も行わず、権限エラー（-9949）が生成されます。

生成されたBLOBは自動的に符号化され、**BLOB TO USERS**コマンドを使用した場合にのみ読み取ることができます。このBLOBは、ハードディスク上のファイルやフィールドに保存可能です。

このコマンドは、ツールバーからグループやユーザを保存する場合と同等の働きをします。唯一異なる点は、このコマンドではファイルだけではなく、BLOBフィールドにもユーザアカウントを保存できるところです。

この概念を使用すると、各ユーザのバックアップをデータベースデータに保持することができます。したがって、バックアップメカニズムや、データベースストラクチャファイルの更新時にユーザを自動的にロードするシステムを設置することができます（ユーザアカウントに関連する情報は、データベースストラクチャファイルに保存されます）。

参照

BLOB TO USERS

BLOB TO USERS

BLOB TO USERS (ユーザ)

引数	タイプ		説明
ユーザ	BLOB	→	データベース管理者が作成し保存したデータベースのユーザアカウントを格納するBLOB（符号化済）

説明

BLOB TO USERSコマンドは、BLOBタイプの引数＜ユーザ＞に格納されたユーザアカウントをデータベースへ追加します。BLOB＜ユーザ＞は符号化されており、**USERS TO BLOB**コマンドを用いて作成されていなければなりません。

このコマンドを実行できるのは、データベース管理者（Administrator）だけです。他のユーザがこのコマンドを実行しようとすると、コマンドは何も行わず、権限エラー（-9949）が生成されます。

ユーザアカウントを追加する際、次の規則が適用されます。

- ユーザIDは参照番号として使用されます。ユーザはそれぞれのユーザIDに従って順番に処理されます。
- 番号がストラクチャファイル内に既に存在する場合、BLOBに格納された情報に従い、必要に応じてアカウント情報が変更されます。
- 番号がストラクチャファイル内に存在しない場合、BLOBに格納された情報に従い、ユーザが作成されます。
- その番号が、ストラクチャファイルで削除されたユーザアカウントと一致する場合、BLOBに格納された情報に応じてそのアカウントが変更されます。
- BLOBに格納された情報により、ユーザアカウントが削除されていることが判明した場合、そのアカウントはストラクチャファイルから削除されます。
- BLOBの情報に従って、更新後のユーザがグループに関連付けられます。
- グループが存在しなければ、作成されます。

互換性に関する注意：4th Dimension 2004以前のバージョンの「パスワード」メニューにより作成されたユーザ＆グループファイル（.4UG拡張子）は、次の手順を用いて4th Dimension 2004へロードすることができます。

DOCUMENT TO BLOB(mydoc; blob)
BLOB TO USERS(blob)

しかし、4th Dimension 2004で生成されたユーザ＆グループファイルを以前のバージョンの4th Dimensionで開くことはできません。

参照

USERS TO BLOB

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

Set user properties

Set user properties(ユーザID; ユーザ名; 起動メソッド; パスワード;
ログイン回数; 前回ログイン{; メンバー{; グループオーナ}}) →数値

引数	タイプ		説明
ユーザID	数値	→	ユーザアカウントのユニークなID番号、または-1：Designerに属するユーザを追加する、または-2：Administratorに属するユーザを追加する
ユーザ名	文字列	→	新しいユーザの名前
起動メソッド	文字列	→	新しいユーザstartupメソッドの名前
パスワード	文字列	→	新しい（符号化されていない）パスワード、またはパスワードを変更しない場合は“*”
ログイン回数	数値	→	データベースへの新しいログイン回数
前回ログイン	日付	→	データベースに前回ログインした新しい日付
メンバー	数値配列	→	ユーザが属すグループのID番号
グループオーナ	数値	←	ユーザグループオーナーのID番号
戻り値	数値	←	新しいユーザのユニークなID番号

説明

Set user properties関数は、引数<ユーザID>に渡された一意のユーザID番号を持つユーザアカウントのプロパティを変更および更新することができます。また、デザイナまたは管理者に関連したユーザを追加することができます。

既存のユーザアカウントのプロパティを修正している場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、または既に削除されている場合は、エラーコード-9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。それ以外は、**Is user deleted**関数を呼び出して、**Set user properties**関数を呼び出す前にユーザアカウントを調べることができます。

ユーザID番号は、下記の値または範囲の番号を持っています。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3～15000	デザイナによって作成されたユーザ （つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザ）
-11～-15000	管理者によって作成されたユーザ （つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザ）

デザイナに関連した新しいユーザを追加するには、引数＜ユーザID＞に-1を渡します。管理者に関連した新しいユーザを追加するには、引数＜ユーザID＞に-2を渡します。この関数を呼び出すと、そのユーザが正常に追加された場合は、その追加されたユーザのID番号が＜ユーザID＞に返されます。

＜ユーザID＞に-1、-2または有効なユーザID番号が渡されない場合、**Set user properties**関数は何も行いません。

この関数を呼び出す前に、引数＜ユーザ名＞、＜起動メソッド＞、＜パスワード＞、＜ログイン回数＞、＜最終日付＞に新しいユーザの名前、Startupメソッド、暗号化されたパスワード、およびデータベースにログインした回数および最後にログインした日付を設定します。引数＜パスワード＞には暗号化されていないパスワードを渡します。渡されたパスワードは、ユーザアカウントに保存される前に、4Dにより暗号化されます。

新しいユーザ名はユニークでない名前でも通過します。(既に同じ名前のユーザがあった時）関数はなにもせず、エラー-9979が戻ります。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。そのユーザのすべてのプロパティを変更したくない場合は（グループ以外。後述)、まず**GET USER PROPERTIES**コマンドを呼び出して、それから変更したくないプロパティとして返された値をこのコマンドに渡します。

注：アカウントのパスワードは変更しないように、引数＜パスワード＞の値として“*”記号を受け付けます。この値を設定すると、このアカウントのパスワードを変更せずにこのユーザアカウントの他のプロパティを変更することができます。

オプション引数＜グループID＞を指定しなかった場合、そのユーザのカレントグループは変更されません。ユーザを追加する際に＜グループID＞を指定しなかった場合は、そのユーザはグループに属しません。

オプション引数＜グループID＞を指定すると、そのユーザのすべてのグループを変更します。この関数を呼び出す前に、引数＜グループID＞配列にそのユーザが属すグループの一意のID番号を割り当てなければなりません。グループID番号は、以下の範囲の番号を持つことができます。

任意の引数＜グループオーナ＞を渡すと、ユーザグループの“オーナー”、つまりこのユーザが作成したオブジェクトのデフォルトオーナーグループの ID番号を指定することができます。

グループのID番号は、次の通りです。

グループID番号	グループの種類
15000～32767	デザイナまたは関連したグループオーナーによって作成されたグループ（つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ)
-150011～-32768	管理者または関連したグループオーナーによって作成されたグループ（つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループ)

あるユーザのすべてのグループを無効にするには、＜グループID＞に空の文字列を渡します。

参照

DELETE USER、GET GROUP LIST、GET USER LIST、GET USER PROPERTIES、Is user deleted

エラー処理

Set user properties関数を呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET GROUP LIST

GET GROUP LIST(グループ名 ; グループID)

引数	タイプ		説明
グループ名	文字列配列	←	「パスワードエディタ」ウインドウ内に表示されるグループの名前
グループID	数値配列	←	対応する一意のグループID番号

説明

GET GROUP LISTコマンドは、引数＜グループ名＞と＜グループID＞の配列に「パスワード」ウインドウ内に表示されるグループの名前と一意のID番号を割り当てます。

＜グループ名＞と同期される引数＜グループID＞配列には、対応する一意のグループID番号が代入されます。グループID番号は、以下の範囲の値を持つことができます。

グループID番号	グループの種類
15000～32767	デザイナまたは関連したグループオーナーによって作成されたグループ（つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ)
-150011～-32768	管理者または関連したグループオーナーによって作成されたグループ（つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ)

参照

GET GROUP PROPERTIES、GET USER LIST、SET USER PROPERTIES

エラー処理

GET GROUP LISTコマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET GROUP PROPERTIES

GET GROUP PROPERTIES(グループID ; グループ名 ; オーナー {; メンバーID})

引数	タイプ		説明
グループID	数値	→	一意のグループID番号
グループ名	文字列	←	グループの名前
オーナー	数値	←	グループオーナーのユーザID番号
メンバーID	数値配列	←	グループメンバーのID番号

説明

GET GROUP PROPERTIESコマンドは、引数<グループID>に渡された一意のグループID番号を持つグループのプロパティを返します。この場合、**GET USER LIST**コマンドで返された有効なグループID番号を渡さなければなりません。グループID番号は、以下の範囲の番号を持つことができます。

グループID番号　グループの種類

15000～32767　デザイナまたは関連したグループオーナーによって作成されたグループ

（つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ)

-150011～-32768 管理者または関連したグループオーナーによって作成されたグループ

（つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ)

有効なグループID番号を指定しなかった場合、**GET GROUP PROPERTIES**コマンドは空の引数を返します。

このコマンドを呼び出すと、引数<グループ名>と<オーナー>にそのグループの名前とオーナー名が返されます。

オプション引数<メンバーID>を指定すると、グループに属すユーザとグループの一意のID番号が返されます。メンバーID番号は、以下の範囲の値を持つことができます。

メンバーID番号	メンバーの種類
1	デザイナ
2	管理者
3〜15000	デザイナによって作成されたユーザ （つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザ）
-11〜-15000	管理者によって作成されたユーザ （つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザ）
15000〜32767	デザイナまたは関連したグループオーナーによって作成されたグループ （つまりグループID番号15001がデザイナによって最初に作成された グループ、グループID番号15002が2番目に作成されたグループ）
-150011〜-32768	管理者または関連したグループオーナーによって作成されたグループ （つまりグループID番号-15001が管理者によって最初に作成された グループ、グループID番号15002が2番目に作成されたグループ）

参照

GET GROUP LIST、GET USER LIST、SET GROUP PROPERTIES

エラー処理

GET GROUP PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Set group properties

Set group properties(グループID ; グループ名 ; オーナー {; メンバーID})

→ 数値

引数	タイプ		説明
グループID	数値	→	グループの一意のID番号、または -1=デザイナに関連したグループの追加 -2=管理者に関連したグループの追加
グループ名	文字列	→	新しいグループの名前
オーナー	文字列	→	新しいグループオーナーのユーザID番号
メンバーID	数値配列	→	新しいグループメンバーのID番号
戻り値	数値	←	新しいグループのユニークなID番号

説明

Set group properties関数は、引数<グループID>に渡された一意のグループID番号を持つ既存グループのプロパティを変更および更新することができます。また、デザイナまたは管理者に関連したグループを追加することができます。

既存グループのプロパティを修正している場合、**GET USER LIST**コマンドで返された有効なグループID番号を渡さなければなりません。グループID番号は、以下の範囲の番号を持つことができます。

グループID番号	グループの種類
15000～32767	デザイナまたは関連したグループオーナーによって作成されたグループ（つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループ）
-150011～-32768	管理者または関連したグループオーナーによって作成されたグループ（つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループ）

デザイナに関連した新しいユーザを追加するには、引数<グループID>に-1を渡します。管理者に関連した新しいユーザを追加するには、引数<グループID>に-2を渡します。

<グループID>に-1、-2または有効なグループID番号が渡されない場合、**Set group properties**関数は何も行いません。

この関数を呼び出す前に、引数<グループ名>と<オーナー>に新しいグループの名前とオーナーを設定します。そのグループのすべてのプロパティを変更したくない場合は（メンバー以外。後述）、まず**GET GROUP PROPERTIES**コマンドを呼び出し、変更したくないプロパティに対して返された値をこの関数に渡します。

オプション引数＜メンバーID＞を指定しなかった場合、そのグループのカレントメンバーリストは変更されません。グループを追加する際に＜メンバー＞を指定しなかった場合は、そのグループはメンバーを持ちません。

注：グループオーナーは、オーナーが所有するグループのメンバーとして自動的に設定されるわけではありません。引数＜メンバー＞を使って、グループオーナーをそのグループに含めるかどうかはユーザ次第です。

オプション引数＜メンバーID＞を指定すると、そのグループのメンバーリスト全体を変更します。この関数を呼び出す前に、引数＜メンバーID＞配列にそのグループがメンバーとして取得するユーザとグループの一意のID番号を割り当てなければなりません。メンバーID番号は、以下の範囲の値を持つことができます。

メンバーID番号	メンバーの種類
1	デザイナ
2	管理者
3～15000	デザイナによって作成されたユーザ （つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザ）
-11～-15000	管理者によって作成されたユーザ （つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザ）
15000～32767	デザイナまたは関連したグループオーナーによって作成されたグループ （つまりグループID番号15001がデザイナによって最初に作成された グループ、グループID番号15002が2番目に作成されたグループ）
-150011～-32768	管理者または関連したグループオーナーによって作成されたグループ （つまりグループID番号-15001が管理者によって最初に作成された グループ、グループID番号15002が2番目に作成されたグループ）

グループからすべてのメンバーを削除するには、＜メンバーID＞に空の文字列を渡します。

参照

GET GROUP LIST、GET USER LIST、SET GROUP PROPERTIES

エラー処理

Set group properties関数を呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

CHANGE LICENSES

CHANGE LICENSES

説明

CHANGE LICENSESコマンドは、4Dの「ライセンス更新...」ダイアログボックスを表示します。このダイアログボックスを使用し、ユーザはプラグインやWebサーバのアクティブ化を行ったり、あるいは4D Serverの場合は、エクスパンション番号を追加してデータベースとそのプラグインを同時に使用できるクライアント数を増やすことができます。

注：このダイアログボックスについての詳しい説明は『4D Product Lineインストールガイド』を参照してください。

「デザイン」モードでこのデータベースプロパティダイアログボックス内にある「ライセンス」ボタンをクリックすると、ライセンスダイアログボックスを表示することができます。**CHANGE LICENSES**コマンドを使ってユーザ、カスタムメニューモードからライセンスダイアログボックスを表示することができます。

CHANGE LICENSESコマンドは、コンパイル済4Dアプリケーションにライセンスを追加する便利な方法です。4Dのディベロッパや情報システム管理者が、4Dアプリケーションを配付するためにこのコマンドを使用できます。そして、アプリケーションのアップデート版を毎回送付することなく、ユーザにライセンスを入力させることができます。

例題

独自のシステム定義、または環境設定用のダイアログボックス内に、以下のメソッドを持つボタンを設定します。

```
`「bライセンス」ボタンのオブジェクトメソッド
CHANGE LICENSES
```

SET PLUGIN ACCESS

SET PLUGIN ACCESS (プラグイン; グループ)

引数	タイプ		説明
プラグイン	倍長整数	→	プラグイン番号
グループ	文字列	→	プラグインに関連付けるグループ名

説明

SET PLUGIN ACCESSコマンドにより、データベースに組み込まれた“シリアル番号を持つ”各プラグインの使用を許可するユーザグループを、プログラムから設定することができます。これにより、ユーザはプラグインのライセンスがどのように使用されているかを管理することができます。

注：この操作は「デザイン」モードの「グループ」エディタを使用して実行することもできます。

引数＜プラグイン＞には、ユーザグループを関連付けるプラグインの番号を渡します。プラグインライセンスには、4D Client WebやSOAPのライセンスが含まれます。「Is license available」テーマ内にあるいずれかの定数を渡すことができます。

定数	タイプ	値
4D Draw License	倍長整数	808464694
4D For OCI License	倍長整数	808465208
4D View License	倍長整数	808465207
4D Write License	倍長整数	808464697
4D Client Web License	倍長整数	808465209
4D Client SOAP License	倍長整数	808465465
4D ODBC Pro License	倍長整数	808464946
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715

＜グループ＞には、そのプラグインの使用を許可するユーザが属すグループ名を渡します。

注：プラグインを使用することができるのは一度に1ユーザグループのみです。このコマンドが実行された場合、もし別のグループがプラグインのアクセス権を持っていても、その権利は失われます。

参照

Get plugin access、GET PLUGIN LIST

Get plugin access

Get plugin access(プラグイン) → 文字列

引数	タイプ		説明
プラグイン	倍長整数	→	プラグイン番号
戻り値	文字列	←	プラグインに関連付けられたグループ名

説明

Get plugin access関数は、引数<プラグイン>に渡した番号のプラグインの使用が許可されているユーザグループ名を返します。そのプラグインに関連付けられたグループが存在しない場合、コマンドは空の文字列（""）を返します。

引数<プラグイン>には、ユーザグループが関連付けられたプラグインの番号を渡します。プラグインライセンスには、WebやWebクライアントのライセンスが含まれます。「Is license available」テーマ内にあるいずれかの定数を渡すことができます。

定数	タイプ	値
4D Draw License	倍長整数	808464694
4D For OCI License	倍長整数	808465208
4D View License	倍長整数	808465207
4D Write License	倍長整数	808464697
4D Client Web License	倍長整数	808465209
4D Client SOAP License	倍長整数	808465465
4D ODBC Pro License	倍長整数	808464946
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715

参照

GET PLUGIN LIST、SET PLUGIN ACCESS

GET PLUGIN LIST

GET PLUGIN LIST(番号配列：名前配列)

引数	タイプ		説明
配列番号	倍長整数の配列	←	プラグイン番号
配列名	文字列の配列	←	プラグインの名称

説明

GET PLUGIN LISTコマンドは配列番号および配列名に4th Dimensionで読み込まれたプラグインの番号および名前を代入します。これらの配列はコマンドにより自動的にサイズが調整され、同期化が行われます。

注：引数＜番号配列＞に返された値は"Is License available"テーマ内の定数と比較することができます。

GET PLUGIN LISTコマンドは統合されたプラグイン（例えば4D Chart）や、サードパーティー製のプラグインなどを含むすべてのプラグインに使用することができます。

参照

Get plugin access、SET PLUGIN ACCESS

SAVE VARIABLES

SAVE VARIABLES (ドキュメント ; 変数1{ ;...; 変数N})

引数	タイプ		説明
ドキュメント	文字列	→	変数を保存するドキュメントファイル
変数	変数	→	保存する変数

説明

SAVE VARIABLESコマンドは、ディスク上の＜ドキュメント＞ファイルに＜変数＞を保存します。

変数は同じタイプでなくてもかまいませんが、文字列、テキスト、実数、整数、倍長整数、日付、時間、ブール、またはピクチャのいずれかである必要があります。

＜ドキュメント＞に空の文字列（""）を指定した場合には標準の「ファイル作成」ダイアログボックスを表示します。ここで、ユーザは作成するドキュメントファイルの名前を入力することができます。この場合、ドキュメントファイルが作成されると、4Dのシステム変数Documentにドキュメント名が入ります。

変数を正常に保存した場合は、システム変数OKに1が代入され、それ以外の場合には、システム変数OKに0が代入されます。

注：SAVE VARIABLESコマンドで変数をドキュメントファイルに保存する場合は、4D専用の内部フォーマットで保存します。したがって、保存した変数はLOAD VARIABLESコマンド以外では読み込むことができません。SAVE VARIABLESコマンドで作成したドキュメントは、RECEIVE VARIABLE またはRECEIVE PACKETコマンドで読み込まないでください。

警告：このコマンドは、配列をサポートしません。新しく導入されたBLOBコマンドを使用してください。

例題

▼ 以下の例は、ドキュメントファイル“保存”に3つの変数を保存します。

```
SAVE VARIABLES ("保存" ; v文字列 ; v数値 ; vピクチャ)
```

システム変数とセット

変数が正常に保存されると、システム変数OKに1が代入され、それ以外の場合には0が代入されます。

参照

BLOB TO DOCUMENT、BLOB TO VARIABLE、DOCUMENT TO BLOB、LOAD VARIABLES、VARIABLE TO BLOB

LOAD VARIABLES

LOAD VARIABLES (ドキュメント ; 変数1{ ;...; 変数N})

引数	タイプ		説明
ドキュメント	文字列	→	4D変数を保存した ドキュメントファイル
変数	変数	→	値を受け取る変数

説明

LOAD VARIABLESコマンドは、**SAVE VARIABLES**コマンドで作成したディスク上の＜ドキュメント＞ファイルから＜変数＞を読み込みます。

＜変数1＞～＜変数N＞が作成されます。これらの変数が既に存在する場合には上書きします。

＜ドキュメント＞に空の文字列（""）を指定した場合には、標準の「ファイルを開く」ダイアログボックスを表示します。ここで、ユーザは読み込むドキュメントファイルを選択することができます。ドキュメントファイルが選択されると、4Dシステム変数Documentにドキュメント名が入ります。

コンパイルされたデータベースでは、各変数はディスクからロードされた時と同じタイプでなくてはなりません。

警告：このコマンドは、配列をサポートしません。新しく導入されたBLOBコマンドを使用してください。

例題

▼ 以下の例は、ドキュメントファイル“保存”から3つの変数を読み込みます。

```
LOAD VARIABLES ("保存" ; v文字列 ; v数値 ; vピクチャ)
```

システム変数とセット

変数を正常に読み込んだ場合は、システム変数OKに1が代入され、それ以外の場合には、システム変数OKに0が代入されます。

参照

BLOB TO DOCUMENT、BLOB TO VARIABLE、DOCUMENT TO BLOB、RECEIVE VARIABLE、VARIABLE TO BLOB

CLEAR VARIABLE

CLEAR VARIABLE (変数)

引数	タイプ		説明
変数	変数	→	消去する変数

説明

CLEAR VARIABLEコマンドは、インタプリタモードとコンパイルモードでは異なる動作をします。

インタプリタモード

CLEAR VARIABLEコマンドはメモリから＜変数＞を消去します。その結果、変数は未定義になり、その値を読み込もうとすると構文エラーが発生します。再度この変数に対して値を割り当てると、4Dは即座に変数を再作成します。変数が消去された後、この変数に対して**Undefined**関数を適用すると、“True（真）”が返されます。

コンパイルモード

CLEAR VARIABLEコマンドは＜変数＞をデフォルトタイプの値に再設定するだけです(つまり、文字列およびテキスト変数は空の文字列に、数値変数は0に、配列は要素を空にします)。ですが、変数は依然として存在します。つまり、コンパイルモードでは変数は決して未定義にはなりません。

＜変数＞はプロセス変数、またはインタープロセス変数でなくてはなりません。

注：プロセス終了時にプロセス変数を消去する必要はありません。4Dにより自動的に消去されるためです。

ローカル変数、つまり名前の先頭にドル記号（$）のついた変数は、**CLEAR VARIABLE**コマンドでは消去できません。ローカル変数は、その変数の属するメソッドの実行が終了した時点で、自動的に消去されます。

▼フォーム上で、ユーザインタフェースの目的だけで、ドロップダウンリスト“asMyDropDown”を使用するものとします。つまり、データ入力中には配列を使用しますが、フォームから抜けた後はその配列が不要になります。したがって、On Loadイベントでこの配列を消去します。

```
` asMyDropDown ドロップダウンリストのオブジェクトメソッド
Case of
	¥ (Form event=On Load)
		` 配列を初期化
		ARRAY STRING (63;asMyDropDown;...)
		` ...
	¥ (Form event=On Unload)
		` 配列は不要
		CLEAR VARIABLE (asMyDropDown)
		` ...
End case
```

参照

Undefined

Undefined

Undefined (変数) → ブール

引数	タイプ		説明
変数	変数	→	判定する変数
戻り値	ブール	←	True：変数は現在未定義である False：変数は現在定義されている

説明

Undefined関数は、＜変数＞が定義されていない場合は“True（真）”を返し、＜変数＞が定義されている場合には“False（偽）”を返します。変数が定義されるのは、変数に値を代入した時点です。まだ値を代入していない変数や**CLEAR VARIABLE**コマンドで消去した変数は、未定義になります。

4D Compilerを使用してコンパイルされたデータベースの場合、**Undefined**関数はすべての変数に対して“False（真）”を返します。

例題

1. バージョン6まで、実行しているのがインタプリタ版かコンパイル版かを判断する1つの方法として、以下のように記述していました。

```
変数:="こんにちは"
CLEAR VARIABLE (変数)
If (Undefined (変数))
      ` インタプリタ版です
Else
      ` コンパイル版です
End if
```

   バージョン6以降は、**Compiled application**コマンドを使用すると便利です。

2. 以下の例は、アプリケーションの特定モジュール用のメニュー項目が選択されたときに、プロセスの作成を管理します。プロセスが既に存在する場合、そのプロセスを前面に配置し、存在しない場合には、プロセスを開始します。このため、アプリケーションのモジュールごとにインタープロセス変数“<>PID_...”を保守します。このインタープロセス変数は、**On Startup**データベースメソッドで初期化されます。

   データベースの開発を実行する際、モジュールを新しく追加します。**On Startup**データベースメソッドを修正する（対応する“<>PID_...”の初期化を追加するため）代りに、データベースを再度開いてモジュールを追加するたびにすべてを再度初期化します。新規モジュールの追加を管理するには、**Undefined**関数を使用します。

```
` 「M_ADD_CUSTOMERS」グローバルメソッド
` このコードは開発の中盤で管理を実行する
If (Undefined(<>PID_ADD_CUSTOMERS))
        C_LONGINT (<>PID_ADD_CUSTOMERS)
        <>PID_ADD_CUSTOMERS:=0
End if
If (<>PID_ADD_CUSTOMERS=0)
                <>PID_ADD_CUSTOMERS:=New process ("P_ADD_CUSTOMERS";
                                                64*1024;"P_ADD_CUSTOMERS")
Else
        SHOW PROCESS (<>PID_ADD_CUSTOMERS)
        BRING TO FRONT (<>PID_ADD_CUSTOMERS)
End if
```

注：:P_ADD_CUSTOMERS、プロセスマスターメソッド、は<>PID_ADD_CUSTOMERS を終了時に0にセットします。

参照

CLEAR VARIABLE

概要

4th Dimension、4D Server、ならびに4D Clientには、4Dデータベースやあらゆるタイプの HTMLページをWeb上に公開できるWebサーバエンジンが組み込まれています。4D Web サーバエンジンの主な機能は次の通りです。

■ やさしい公開方法

Web上へのデータベース公開は、いつでも開始したり中止することができます。これを行うには、メニューコマンドを選択するか、またはランゲージコマンドを実行するだけです。

■ コンテキストモードと非コンテキストモード

4D Webサーバは、コンテキストモードと非コンテキストモードという2つのモードで動作することができます。4D Webサーバは、これらのうち一方のモードで使用するか、あるいは必要に応じてあるモードからもう一方のモードへオンザフライで切り替えることもできます。

■ コンテキストモード（4th Dimensionならびに4D ServerのWebサーバでのみ利用可能）では、ユニークで独創的な機能が提供されます。このモードにおいて、4DはWebブラウザを標準的なデータベースクライアントとして管理します。作成したデータベースは直接Web上に公開されます。データベースやWebサイト、ならびにこの2つの間の橋渡しをするCGIの開発は必要ありません。データベースがWebサイトになります。

データベースのストラクチャやデータに変更が行なわれると、そのデータベースに接続しているすべてのブラウザへ即座に反映されます。4Dはデータベースのメニューバーやフォーム、メソッドをHTMLに変換します。したがって、HTMLを知らなくてもWeb上に4Dデータベースを公開することができます。4Dは各Webブラウザのコンテキスト（セレクション、変数等）を用いてデータを自動的に管理します。

この代わりに、コンテキストモードにおけるWebナビゲーションには特有の制約があります。詳細については、「コンテキストモードの使用」の節を参照してください。

■ 非コンテキストモード（標準モード）で使用する場合、4D Webサーバは完全に標準的なHTTPサーバになります。つまり、Webページはコンテキストを保持する必要なく送信されます。ユーザは、Webブラウザにページを送信する前に4Dデータベースのデータにアクセスし、スタティックデータとデータベースのデータを共に含む“セミダイナミック”HTMLページを作成することができます。Webサーバによる処理を全く必要としないスタティックなWebページを送信することも可能です。

■ 専用のデータベースメソッド

「**On Web Authentication**」と「**On Web Connection Database**」データベースメソッドは、Webサーバにおけるリクエストの入り口点となります。これらのメソッドを使用して、あらゆるタイプのリクエストの評価やルート付けを行います。

■ 特別なタグとURLの使用

4D Webサーバでは、ユーザとのやりとりを可能にするさまざまなメカニズムが提供されています。特に次のような事柄です。

■ ブラウザ側に送信されるとWebサーバの処理が開始する特別なタグをWebページに組み込むことができます。

■ 任意のアクションを実行するために4Dを呼び出す特別なURLを使用することができます。

■ これらのURLをフォームアクションとして使用し、ユーザによるHTMLフォームの送信時に処理を開始するすることもできます。

■ アクセスセキュリティ

自動設定オプションを使用し、Webブラウザに対する特定のアクセス権を認可したり、4th Dimensionに統合されたパスワードシステムを使用することができます。また、データベース内でのアクセス管理を簡略化するために、“一般Webユーザ”を定義することができます。

「**On Web Authentication**」データベースメソッドを使用すると、Webサーバがリクエストを処理する前に、任意のリクエストの評価を行うことができます。さらに、デフォルトのHTMLルートフォルダを指定することにより、ディスク上のファイルへのアクセスを制限することができます。

最後に、Web経由で実行されるプロジェクトメソッドを個別に指定しなくてはなりません。

■ SSL接続

4D Web ServerはセキュアドモードでSSL(Secured Socket Layer)プロトコルを通じてブラウザに情報を送ることができます。このプロトコルは多くのブラウザと互換性があり、送信、受信について信頼性と変換された情報の完全性を実現しています。

■ インターネットフォーマットの拡張サポート
4D Webサーバは、HTTP/1.1と互換性があり、XML文書とWML（Wireless Markup Language）テクノロジーをサポートします。

■ CGIサポート
4D Web Serverは他のCGIを通じたHTTPサーバにより呼び出されるのと同様に、CGIを非常に容易に呼び出すことができます。

■ データベースの同時処理

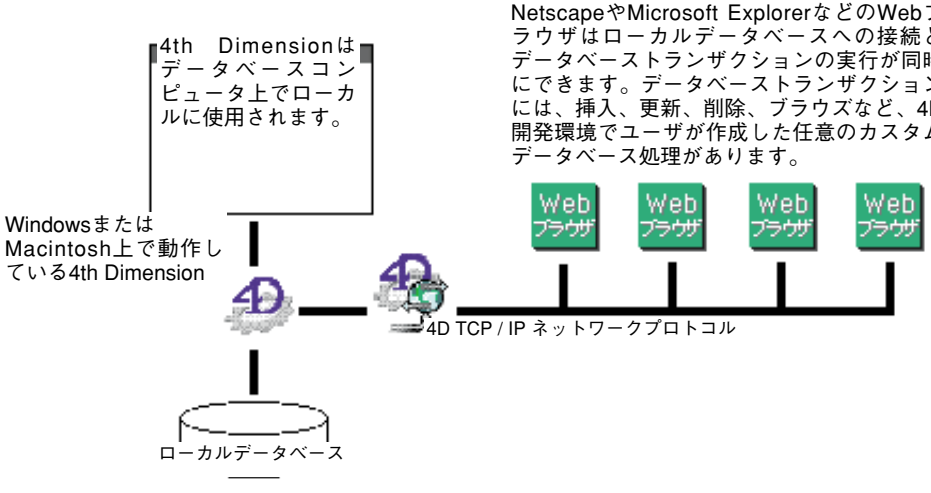
4th DimensionとWeb

4th Dimensionを使用してWeb上に4Dデータベースを公開する場合、同時に以下のことができます。

■ 4Dでローカルにデータベースを使用する

■ Webブラウザを使用してデータベースに接続する

これを以下の図にまとめます。



4D ServerとWeb

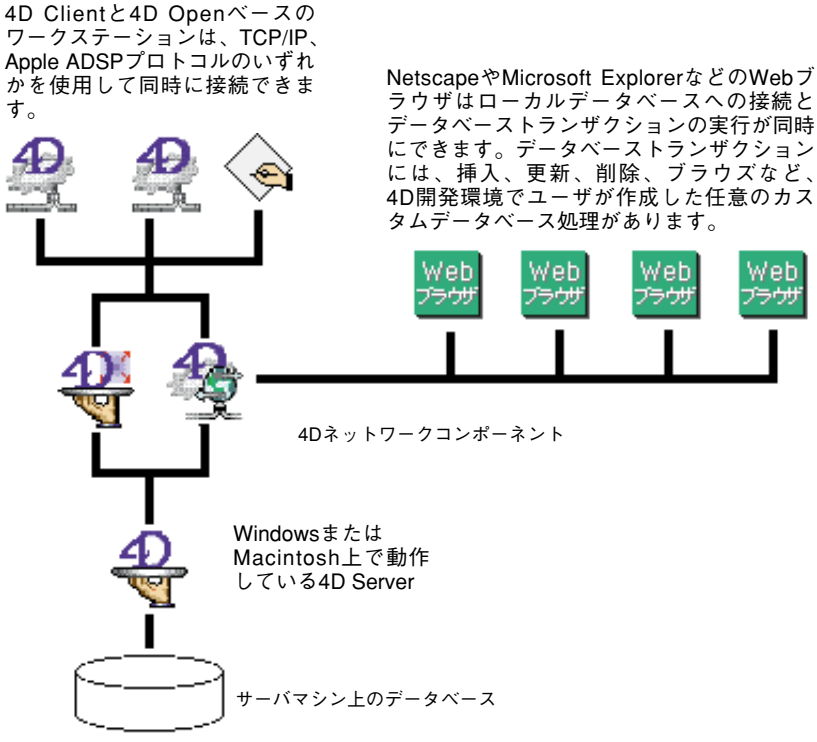
4D Serverを使用してWeb上に4Dデータベースを公開する場合には、以下を使用して4Dデータベースへの接続とその処理を同時に行えます。

■ 4D Clientワークステーション

■ 4D Openをベースにしたアプリケーション

■ Webブラウザ

これを以下の図にまとめます。



4D ClientとWeb

4D Clientを使用して4Dデータベースがweb上に公開されると、次の方法で4Dデータベースへ接続し、同時に使用することができます。

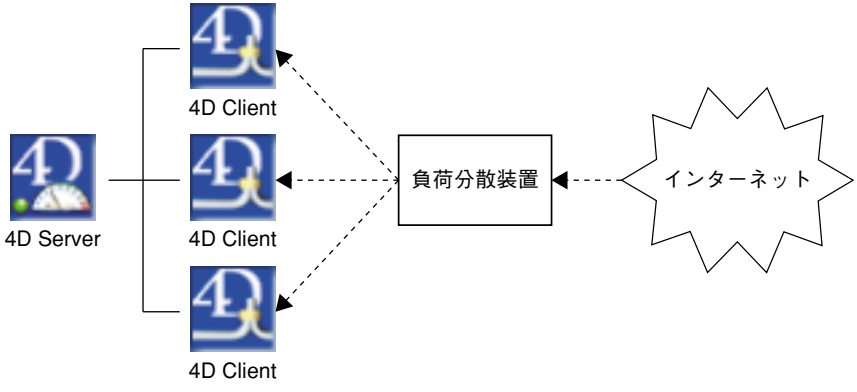
- 4D Clientマシン経由
- 4D Openを使用したアプリケーション経由
- Webブラウザ経由。このケースでは、データが4D Serverからも公開されている場合に、Webブラウザは4D Clientまたは4D Server経由で、公開されたデータベースに接続することができます。さらに、これにより異なるデータアクセスモードに対処できるようになります（パブリック、管理、等）。

コンテキストモードの場合を除き、4D Clientでは4D Webサーバの基本的な仕組みが同じ方法で使用されます。ただし、4D ClientのWebサーバではコンテキストモードが使用できません（このモードに関する詳細は、「コンテキストモードの使用」の節を参照してください）。

同様に、ランゲージコマンドの動作は、4th Dimension、4D Server、4D Clientのいずれでコマンドが実行されても通常は同じです。重要なのは、コマンドはそれが実行されるマシンのWebサイトに対して適用されるという点です。したがって、**Execute on server**関数や**EXECUTE ON CLIENT**コマンドを使用して、これを管理しなくてはなりません。

■ 4D Clientを使用したロードバランシング（負荷分散）システム
任意の4D ClientマシンをWebサーバとして使用できるため、負荷分散装置（ロードバランサー）を備えたダイナミックなWebサーバシステムを構築することができます。このシステムにより大規模な開発が可能となり、次の事柄が実現します。

■ 4D Webサーバのパフォーマンスを最適化するためのロードバランシングシステムの構築：各4D Client Webサーバにインストールされる Webサイトミラーを使用し、現時点のそれぞれの負荷に基づいて負荷分散装置（ハードウェアまたはソフトウェア）が各クライアントマシンにリクエストを送信します。



■ 障害対策済のWebサービスの構築：4D Webサイトは、2台以上の4D Clientマシン上にミラー処理されます。ある4D Client Webサーバがダウンすると、もう一方が処理を引き継ぎます。

■ 例えば、リクエストの発信元に応じて、同一データからさまざまなビューを作成できます。社内ネットワークにおいて、プロテクトされた4D Client Webサーバがイントラネット上のリクエストを処理し、ファイヤウォール上にある別の4D Client Webサーバがインターネット上のリクエストを処理することができます。

■ 異なる4D Client Webサーバ間でのタスク分配：ある4D Client WebサーバがSOAPリクエストを処理し、別のサーバが標準的なリクエストを処理する、など。

参照

接続セキュリティ、SEND HTML FILE、SET HOME PAGE、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER、CGIの使用、SSLプロトコルの使用、コンテキストモードの使用、Webサーバセッティング

Webサーバー設定と接続管理

4th Dimension、4D Server、ならびに4D ClientにはWebサーバ機能が組み込まれ、透過的かつダイナミックにデータベースのデータをWeb上に公開することができます。

この節では、4Dデータベースの公開とブラウザへの接続、および接続管理プロセスに必要となる手順について説明します。

Web上への4Dデータベース公開条件

4th Dimension、4D Server、または4D Clientを使用してWeb上に4Dデータベースを公開するには、次に述べる項目が必要になります。

- 必要とする4D Web接続ライセンス、4D Server Web接続ライセンス、または4D Client Web接続ライセンスがアプリケーションにインストールされていなくてはなりません。詳細については『4D Product Line インストールガイド』を参照してください。
- Web接続は、TCP／IPプロトコルを使用してネットワーク上に構築されます。
  したがって、
  - マシンにTCP／IPをインストールし、正しく設定しておく必要があります。詳細についてはコンピュータまたはオペレーティングシステムのマニュアルを参照してください。
  - SSLをネットワーク接続に使う場合には、要求されたコンポーネントが確実にインストールされているか確認してください（後述のWeb Service、SSLプロトコルの使用を参照）。
- 上記の条件をすべてチェックし、対処した後は、4DでWebサービスを開始する必要があります。この点に関しては、この節でさらに詳しく説明します。

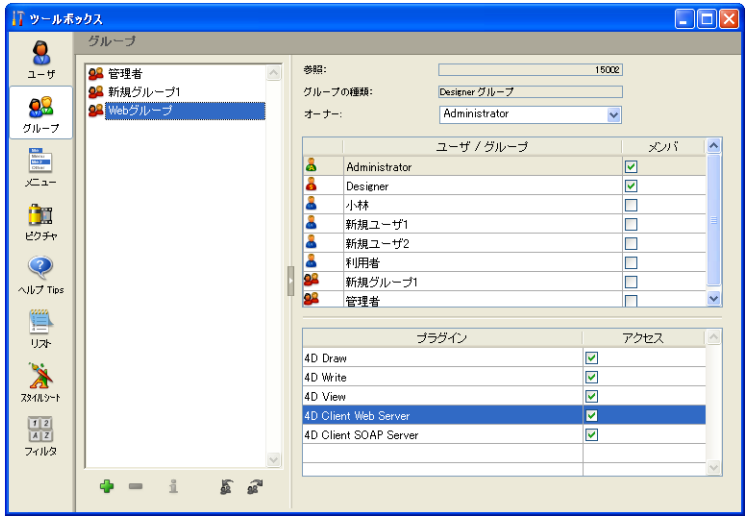
公開の認可（4D Client）

デフォルトとして、すべての4D Clientマシンは接続しているデータベースをWeb上に公開することができます。ただし、4Dのパスワードシステムを使用すると、各4D ClientのWeb公開を制御することができます。

実際のところ、4D ClientのWebライセンスは、4D Serverからはプラグインランセンスとみなされます。したがって、プラグインと同じ方法で、Webサーバライセンスの使用権を特定グループのユーザに限定しなくてはなりません。

これを行うには、4D Clientを使用して「ツールボックス」の「グループ」ページを表示します（これらのパラメータを変更するための適切なアクセス権が必要です）。

左側のリストからグループを選択した後、「プラグイン」配布エリアの「Web Server」行の隣にある「アクセス」オプションをチェックします。



上図：“Web”グループに属するユーザだけが、それぞれの4D ClientマシンをWebサーバとして公開する権限を与えられます。

Mac OS XにおけるWebサーバの設定

Mac OS Xにおいて、Webパブリッシング用に予約されているTCP／IPを使用するには、特定のアクセス権が必要となります。つまり、そのマシンの“ルート”ユーザだけが、これらのポートを使用してアプリケーションを起動することができます。

これらのポート番号は0から1023までです。デフォルトとして4Dデータベースの公開には、標準モードではTCPポート80、SSLモードではポート443が使用されます。

“ルート”ユーザとして接続せずに、デフォルトのTCPポートを使用して4Dデータベースを公開すると、警告ダイアログボックスが表示されます。



Mac OS X上でWebサーバを使用するには、4通りの方法があります。

■ 4D Webサーバが使用するTCPポート番号を変更する

1023より大きな番号のポート番号を使用しなければなりません。例えば、標準モードの場合はポート8080、SSLモードの場合はポート8043を使用します。

番号の変更を行うには、「環境設定」ダイアログボックス（「Webサーバ設定」の節を参照）、または**SET DATABASE PARAMETER**コマンドを使用します。この場合、それぞれのデータベース接続用URLに続けてポート番号を指定する必要があります（例えば、http://www.mydatabase.com/pages/mypage.html:8080とhttps://www.mydatabase.com/pages/payment.html:8043）。

■ “ルート”ユーザとして接続する

デフォルトとして、Mac OS Xが動作するマシンでは“ルート”ユーザが有効ではありません。まず“ルート”ユーザを有効にしたあと、そのユーザ名を使用してログインしなければなりません。“ルート”ユーザを有効にするには、Apple社より提供され、「Applications:Utilities」フォルダにインストールされているNetInfo Managerユーティリティを使用します。

ユーティリティの起動後、「ドメイン」メニューから「セキュリティ」コマンドを選択し、さらに「ルートユーザを有効」オプションを選択します。まず最初に同じメニューにある「認証...」コマンドを使い、マシン管理者を指定しなければなりません（短い名前と管理者のパスワードを入力する）。



この操作に関する詳細は、Mac OS Xのドキュメントを参照してください。

“ルート”ユーザを作成したら、このセッションをクローズし（Appleメニュー）、“ルート”ユーザ名を使用してログインします。これで、ポート番号80でWebサーバを起動したり、あるいは暗号化接続を使用して4D Webサーバを起動することができます。

■ ポート転送

この3番目のソリューションにより、"ルート"ユーザではなくても、サーバに接続されたそれぞれのポート番号を指定することなくMac OS X上で4D Web データベースを公開することができます。これはポート転送に基づくものです。原則として、標準のTCPポート番号（80）で受け取ったリクエストは4Dデータベースで指定されたポート（1023より大きい番号であること）にシステムレベルで転送します。SSL接続の場合には、この設定が適用されない点に注意してください（TCPポート番号443は変更できません）。この設定を行うには、"ルート"ユーザとして接続して端末を起動し、Unixコマンドを使用する必要があります。

Mac OS X上でポートの転送を設定する方法は以下のとおりです（ユーザのIPアドレスが192.168.93.45と仮定した場合）。

1. ルートユーザとしてセッションを開く（上記参照）。
2. ターミナルプログラムを起動する（このプログラムはアプリケーション内の「ユーティリティ」フォルダにあります）。
3. ルートユーザのパスワードに続いて"su"と入力する（"substitute user（代理ユーザ）"の特別なアカウント）。
4. 以下のコマンドを入力する。

```
ipfw add 400 fwd 192.168.93.45,8080 tcp from any to 192.168.93.45 80
```

ただし、192.168.93.45の代わりにユーザ自身のIPアドレスを使用することも可能です。400という数は、この操作の参照番号です。

5. ターミナルプログラムを終了する。
6. 標準ユーザとして4Dアプリケーションを起動する。
7. データベースの環境設定でWeb公開用のTCPポートを8080に設定する。これにより、Mac OS Xはユーザには透過的な方法で、ポート番号80で受け取ったリクエストを即座に8080のポートに転送することができます。

この動作モードを削除する方法は以下のとおりです。

1. ターミナルプログラムを起動し、以下のコマンドを入力する。

```
ipfw delete 400
```

これにより、ポート80で受け取ったリクエストはポート8080に転送されなくなります。

■ 一時的ルートセッションのオープン

このソリューションの機能は次の通りです。まず初めに、4th Dimension Webサーバは、この目的のために開かれた“ルート”セッションで開始されますが、このセッションは開始直後にクローズされます。この新しい機能は、4th Dimension、4D Server、4D Client、および4D Runtime実行形式アプリケーションで使用することができます。

運用手順を次に説明します。

1. 4th Dimensionアプリケーションが従来のユーザセッションを用いて実行される。
2. Webサーバが標準ポート（ポート80）で開始されると、警告ダイアログボックスが表示され、その操作が実行不可能であることをユーザに通知する。

このダイアログボックスを使用して、ユーザはWebサーバを開始可能にするためにアクセス権を変更することができます。アクセス権を変更するには、そのマシンの管理者名とパスワードを入力しなくてはなりません。

3. ユーザはマシンの管理者名とパスワードを入力する。

   4th Dimensionはこの情報を使用してアプリケーションのアクセス権を変更し、ユーザセッションを“ルート”として設定することができます。

4. ユーザはアプリケーションを終了して再起動するよう要求される。

5. 起動時に、4th Dimensionアプリケーションはルートセッションで開始する。

6. アプリケーションの「環境設定」において、「起動時にデータベースを公開する」オプションが選択されている場合、Webサーバはポート80で開始される。

   「WebサーバにSSLを許可する」オプションが選択されている場合、SSLポート（デフォルトでは443）もオープンされます。

7. しばらくすると、ルートセッションは自動的に閉じられ、カレントユーザセッションに置き換わります。

   Webサーバは公開されたままであり、ユーザセッションは正常に継続されます。

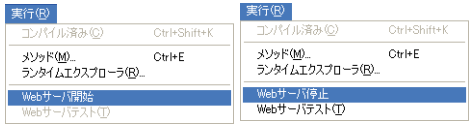
   このメカニズムの難点は、実行開始後、セッション中はWebサーバの中止、開始、再開を任意に行うことができないところです。また、アプリケーションを再起動しない限り、標準ユーザセッションから“ルート”セッション（より上位のアクセス権）へ移ることもできません。このメカニズムは、アプリケーションの起動時にのみ有効です。

4D Webサーバの開始

4D Webサーバは、次に示す3つの方法で開始することができます。

■ 4th Dimensionや4D Clientの「実行」メニュー、または4D Serverの「Web」メニューを使用します。「Webサーバ」メニューにより、いつでもWebサービスの開始や終了を行えます。

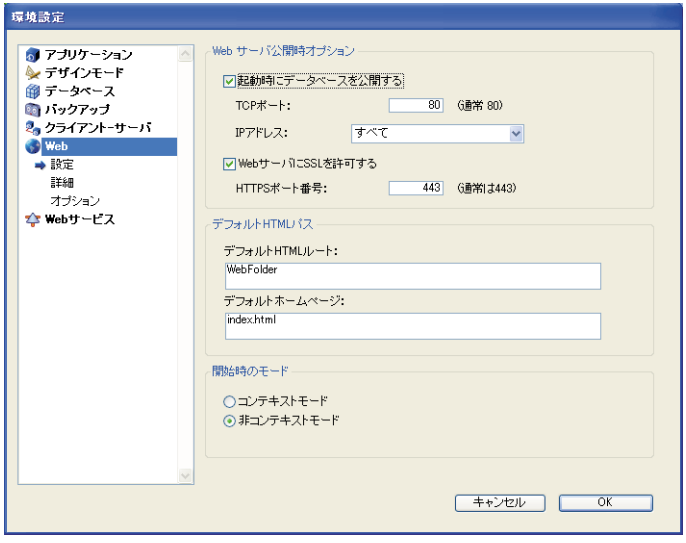
4th Dimensionおよび4D Client



4D Server



■ データベースが開かれるたびにデータベースを自動的に公開します。データベースをWeb上へ自動的に公開するには、4D Serverのメインメニューバー、または4th Dimensionや4D Clientの「デザイン」モードで「編集」メニューから「環境設定...」を選択します。すると、「環境設定」ウインドウが表示されます。ここで、「Web」テーマの「設定」ページをクリックしてください。



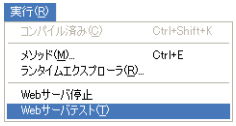
「Webサーバ公開時オプション」エリアで、「起動時にデータベースを公開する」チェックボックスを選択してから「OK」ボタンをクリックします。これを一度行えば、ユーザが4th Dimensionや4D Server、または4D Clientでデータベースを開くたびに、Web上へ自動的に公開されます。

■ プログラムで**START WEB SERVER**コマンドを呼び出す方法。

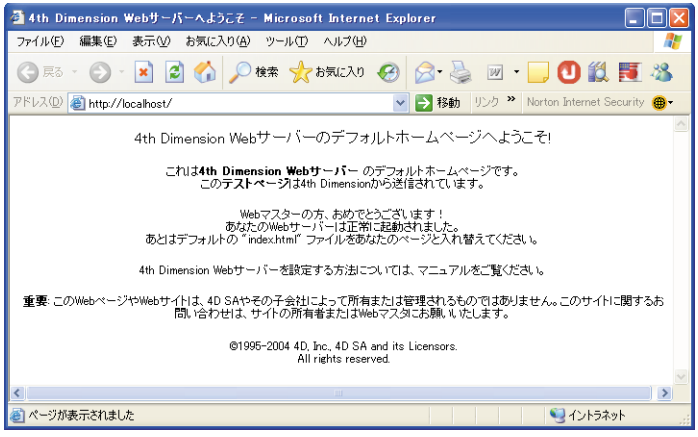
Tips：Web上へのデータベース公開を開始または終了する際に、4Dを停止させて、データベースを再オープンする必要はありません。Webサーバの終了や再開は、「Webサーバ」メニューを使用するか、またはSTART WEB SERVERコマンドとSTOP WEB SERVERコマンドを呼び出すことにより、必要なだけ何回でも行えます。

Webサーバの検証

「Webサーバテスト」コマンドを使用して、内蔵のWebサーバが正常に機能しているかどうかを確認することができます（4th Dimensionと4D Clientのみ）。このコマンドは、Webサーバが開始している時に、「実行」メニューから利用することができます。



このコマンドを選択すると、使用しているデフォルトWebブラウザ上に、4Dアプリケーションにより公開されたWebサイトのホームページが表示されます。



このコマンドを使用し、Webサーバやホームページの表示等が正常に機能しているかどうかを検証することができます。このページは、URL Localhostを用いて呼び出されます。これはWebブラウザが実行されるマシンのIPアドレスを指定するための標準ショートカットです。このコマンドは、アプリケーションの「環境設定」で指定されたTCP公開ポート番号を考慮します。

Web上に公開された4Dデータベースへの接続

Webへの4Dデータベースの公開を開始したら、ユーザはWebブラウザを使用して、そのデータベースに接続することができます。これを実行するには以下のように行います。

■ Webサイトに登録名（例："www.flowersforever.com"）がある場合には、その名前をブラウザのOpen、アドレス、場所エリアのいずれかに指定します。その後、enterキーを押して接続します。

■ Webサイトに登録名がない場合には、マシンのIPアドレス（例：192.168.0.99）をブラウザのOpen、アドレス、場所エリアのいずれかに指定します。その後、enterキーを押して接続します。

現時点で、お使いのブラウザにはWebサイトのホームページが表示されているはずです。標準設定のままでデータベースを公開した場合には、4th DimensionのWebサーバのデフォルトホームページが現われます。このページを使用して、接続やサーバ操作のテストを行うことができます。

また、次に示すいずれかの状況に直面する可能性があります。

1. 接続に失敗し、"...the server may not be accepting connections or may be busy..."（サーバは接続を受け付けないか、またはビジー状態です）といったメッセージを受け取った場合

   この場合、以下を確認します。

   ■ 入力した名前またはIPアドレスが正確なことを確認します。

   ■ 4th Dimensionや4D Server、または4D Clientが稼働しており、そのWebサーバが開始していることを確認します。

   ■ デフォルトのWeb TCPポート以外のTCPポートでサービスを受けるようにデータベースが設定されているかどうかを確認します（下記の4を参照）。

   ■ サーバマシンとブラウザマシンの両方でTCP/IPが正しく設定されているかどうかを確認します。両方のマシンは同一のネットとサブネット上にあるか、あるいはルータが正しく設定されている必要があります。

   ■ ハードウェアが正しく接続されていることを確認します。

   ■ サイトのテストをローカルでしているのではなく、インターネットやイントラネット上で他者がサービスを提供しているWebデータベースに接続しようとしているのであれば、表示されたメッセージが正しいこともあります。つまり、サーバがオフになっているかビジー状態であるということです。この場合には、ログオンできるまで少し待って再試行するか、Webデータベースの公開先に連絡してください。

2. 接続に成功したが、HTTPエラーの404 “File not found（ファイルが見つかりません）”というメッセージが表示される場合。これは、そのサイトのホームページを提供できないことを表わしています。この場合、データベースの「環境設定」（「Webサーバ設定」の節を参照）やSET HOME PAGEコマンドを用いて指定した場所に、ホームページが実際に存在しているかどうかを確認してください。

3. 接続に成功したが、“メニューバー/このデータベースはWeb上に公開する準備が整っていません。まず始めにメニューバーを作成してください。”というメッセージを表示するページが現われた場合

この場合は、コンテキストモードで公開されたデータベースに正しく接続したものの、ホームページとメニューバーが定義されていないことを表わします（コンテキストモードでは、HTMLページが指定されていない場合に4Dはメニューバー番号1をデフォルトのホームページとして公開します）。詳細については、「Webサーバ：入門編」の節を参照してください。

4. 接続はしたが、予想していたようなWebページを取得できない場合

これは、1つのマシン上で同時に複数のWebサーバが稼働している場合に発生する可能性があります。以下の例を参照してください。

■ 独自のWebサーバが既に稼働しているWindowsシステム上で、4DのWebデータベースが1つだけ動作している場合。

■ 1つのマシン上で複数の4D Webデータベースを稼働している場合。

このような状況の元では、4D Webデータベースが発行されるTCPポート番号を変更する必要があります。これを実行するには、「Webサーバ設定」の節を参照してください。

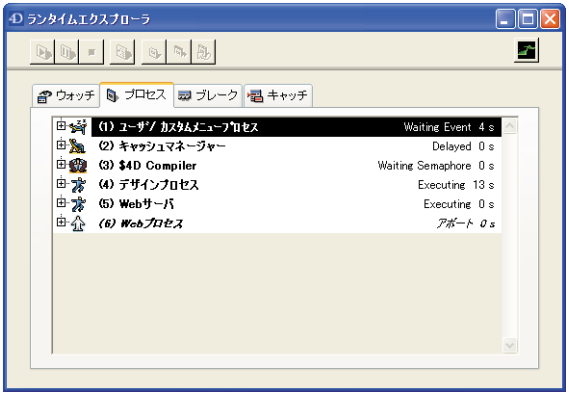
注：データベースがパスワードシステムで守られていれば、正当なユーザ名とパスワードを入力する必要があります（詳しい説明は「接続セキュリティ」を参照してください）。

Webプロセスの管理

データベースのWeb公開やブラウザへの接続は、各種4Dプロセスがサポートします。この節では、これらのプロセスならびにその特性について説明します。

「Webサーバ」プロセス

Webサーバプロセスは、データベースがWebサイトとして公開されている時に稼働し、実行します。次に示すランタイムエクスプローラの「プロセスページ」では、「Webサーバ」プロセスは稼働中かつ実行中である5番目のプロセスです。



これは、4Dカーネルプロセスです。したがって、アボートボタンを使用してこのプロセスをアボートすることはできません。また、**CALL PROCESS**等のコマンドを使用してプロセス間通信を実行することもできません。「Webサーバ」プロセスはユーザインタフェースコンポーネント（ウインドウ、メニュー等）を持たないことに注意してください。

「Webサーバ」プロセスは、以下の方法で起動できます。

■ 4D Serverの「Webサーバ」メニュー、または4th Dimensionや4D Clientの「実行」メニューから「Webサーバ開始」を選択する。

■ 4Dコマンドの**START WEB SERVER**コマンドを呼び出す。

■ 「環境設定」の「起動時にデータベースを公開する」オプションが選択されているデータベースをオープンする。

「Webサーバ」プロセスは、以下の方法で停止できます。

■ 4D Serverの「Webサーバ」メニュー、または4th Dimensionや4D Clientの「実行」メニューから「Webサーバ停止」を選択する。

■ 4Dコマンドの**STOP WEB SERVER**コマンドを呼び出す。

■ 現在公開されているデータベースを停止する。

「Webサーバ」プロセスの目的は、Web接続を処理するだけです。「Webサーバ」プロセスを起動することによって、実際にWeb接続がオープンされるのではなく、Webユーザに対してWeb接続の開始を許可するだけです。また、「Webサーバ」プロセスの停止によって、現在稼働中の「Web接続」プロセスがあればそれをクローズするというのではなく、Webユーザに対して新しいWeb接続の開始を許可しなくなるということです。

「Webサーバ」プロセスを停止する際に、実行中の「Web接続」プロセスが存在する場合、これらのプロセスは通常通り実行を継続します。

結果として、Webサーバプロセスの終了は時間の延長を必要とします。

「Web接続」プロセス

Webブラウザがデータベースへ接続しようとするたびに、接続リクエストは「Webサーバ」プロセスによって処理されます。「Webサーバ」プロセスは、以下の手順を実行します。

■ まず初めに、「Webプロセス」と呼ばれる一時的なローカル4Dプロセスをいくつか作成し、Webブラウザとの接続を評価し、管理します。

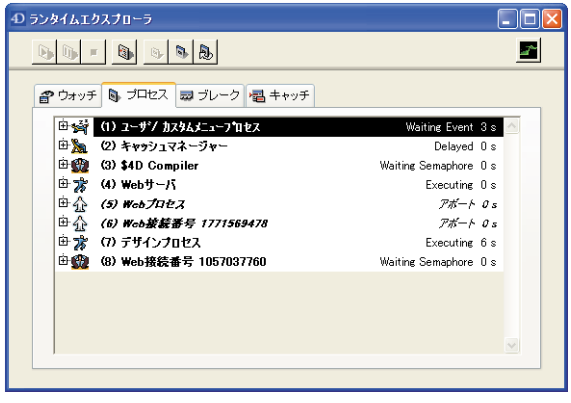
注：これらの一時的なプロセスは各HTTPリクエストを管理します。即座に実行した後アボートされるか遅延されます。非コンテキストモードに関連するWebサーバでは、4Dは5秒間Webプロセスのプールを凍結し、新たなHTTPクエリを実行する時に再利用します。SET DATABASE PARAMETERコマンドを使ってこの動作をカスタマイズできます。

■ リクエストにはコンテキストの作成が不要である場合、Webプロセスがリクエストの処理を行い、応答（必要な場合）をブラウザへ送信します。この後で、一時的なプロセスはアボートされるか、または遅延されます（上記を参照）。

■ リクエストにはコンテキストの作成が必要である場合、新規接続で利用できるリソースか存在するかどうかを確認します。そうでない場合には、Webブラウザに以下のメッセージを送信します。“This database has not been setup for the Web yet”（このデータベースはWeb用に設定されていません）。

Web接続が正常に開始された場合には、「Web接続」プロセスが起動します。このプロセスが、この接続の全Webセッションを扱います。

「プロセスリスト」ウインドウは、Webブラウザ接続が開始された後に起動された「Web接続」プロセス“Web接続番号1057037760”を表示しています。



開始後にアボートされた6番目のプロセスがWeb接続の初期化を行っている点に注意してください。

注：コンテキストの管理についての詳しい説明は、次節の「コンテキストモード」を参照してください。

■ セッション中に、コンテキストモードから非コンテキストモードに接続をスイッチすると、（そのIDを持つ）Web接続プロセスはアボートされます。

逆に言えば、セッション中に、接続を非コンテキストモードからコンテキストモードへスイッチすると、番号付きのWeb接続プロセスが作成されます。

参照

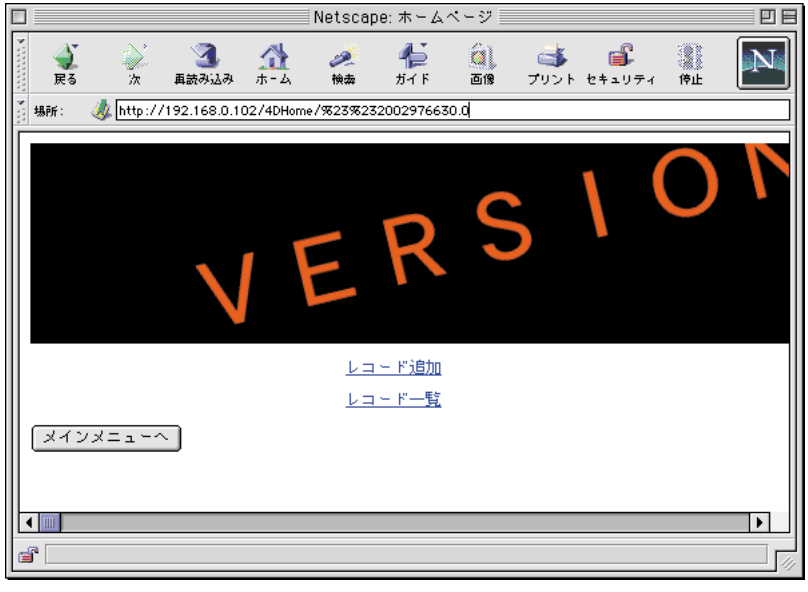
SEND HTML FILE、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER、SSLプロトコルの使用

コンテキストモードの例題

標準的コンテキストモードによるWebに自動的に公開された4Dデータベースの例題を示します。データベースのストラクチャ（このセクションの終りで説明されます）は簡単です。1つのテーブル、入力フォーム、出力フォーム、メニューバーからなるデータベースでホームページは作られます。データベースはWebサーバで公開されます。

1. Webサーバデータベースへの接続

Webサーバに接続するには、Webブラウザ上でデータベースを開きます。ホームWebページが表示されます。この画面はMacintosh上で稼働するNetscapeのものです。

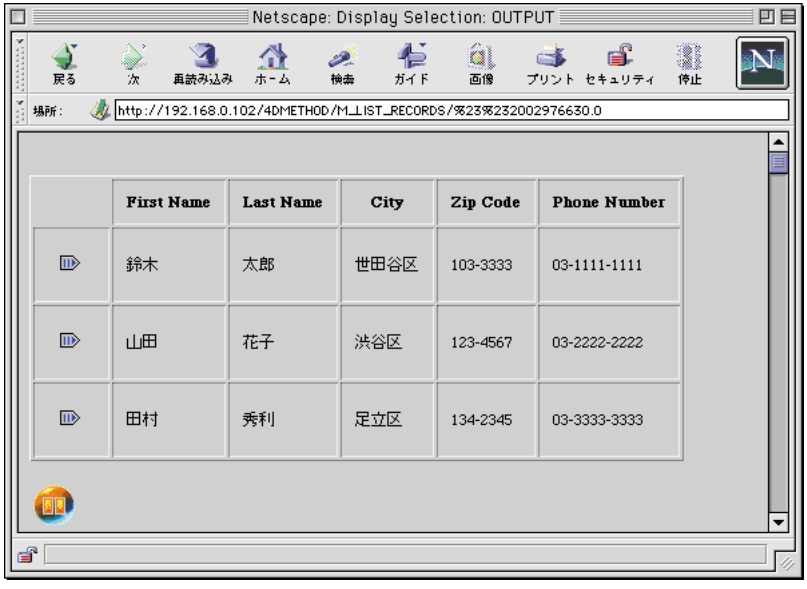


注：ブラウザ上で日本語を表示する際には、必ず「データベースプロパティ」ウインドウの「接続設定」タグにある「文字セット」を「Shift-JIS（Japanese）」に設定してください。この設定を行わないと、ブラウザ側で日本語が正しく表示されないことがあります。

2. レコードの表示とブラウズ

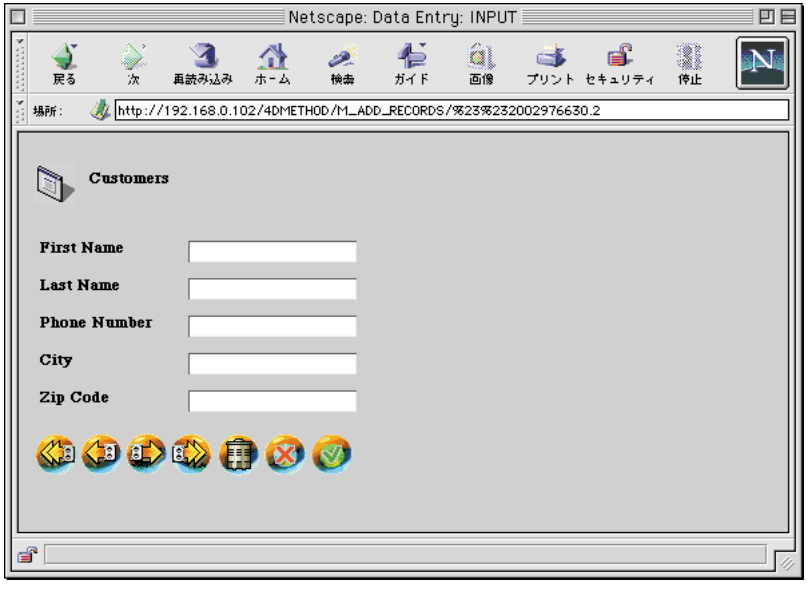
リンクされている「レコード一覧」テキストをクリックします。これにより4Dの**DISPLAY SELECTION**コマンドで表示した画面と同等のWebの画面が、以下のように表示されます。

ここでは、用途に応じてレコードをブラウズできます。「終了」ボタン（画面左下のボタン）をクリックすると、Webサイトのホームページに戻ります。



3. レコードの追加

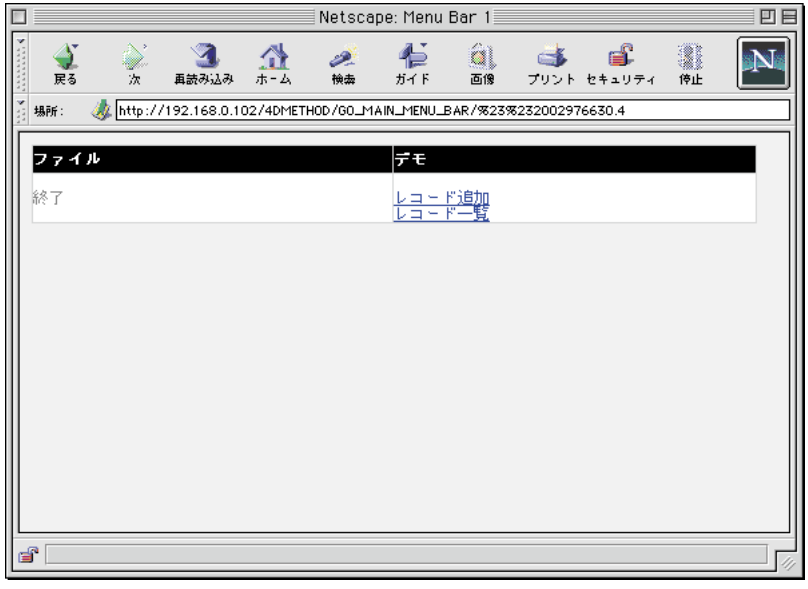
Webサイトのホームページで、リンクされた「レコード追加」テキストをクリックすると4D の**ADD RECORD**コマンドで表示した画面と同等のWebの画面が、以下のように表示されます。



ユーザは必要なだけの数のレコードを追加できます。追加が終わったら、「キャンセル」ボタン（Xが付いているボタン）をクリックして、Webサイトのホームページに戻ります。

4. メインメニューでのレコードのリストまたは追加

ホームページで「メインメニューへ」ボタンをクリックします。これでホームページを終了し、4Dの「カスタム」メニューバーと同等のWebの画面が、以下のように表示されます。



ここで、各メニュー項目をクリックすると、レコードの追加と一覧表示を実行することができます。ホームページから使用したのと同じ4Dメソッドがメニュー項目に対応しています。

5. 接続の終了

作業を終了したら、ブラウザを終了してください。タイムアウト遅延時間が経過すると、「Web接続プロセス」を終了します。

Web接続の開始

WebブラウザがWebサーバとして公開された4Dデータベースに接続する場合には、いつでも以下のアクションを実行します。

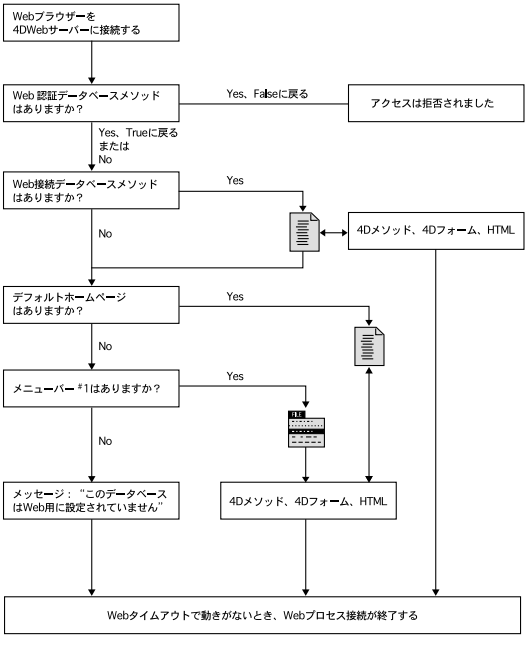
■ **On Web Authentication**データベースメソッドが存在する場合は、それを実行します。

■ このデータベースメソッドがTrueを返すか存在しない場合、**On Web Connection**データベースメソッドが存在すれば、それを実行します。

■ **On Web Authentication**データベースメソッドが存在しないか、最後まで終了した場合、4Dはデータベースプロパティで定義されたデフォルトのホームページを表示します。デフォルトホームページが定義されていない場合、4th Dimensionはカレントメニューバー（デフォルトでメニューバー#1）を表示します。

■ デフォルトホームページもメニューバーもない場合、4th Dimensionは「このデータベースはWeb用に設定されていません」と表示します。

これらのアクションを下図にまとめます。



On Web Connectionデータベースメソッドは、HTMLページと同様、データベース内に定義された任意のプロジェクトメソッドやフォームを呼び出すことができます。このデータベースメソッドは、実際にセッション全体を処理することができます。

4Dや4D ServerへのWeb接続は、クライアント／サーバ接続と同じではありません。HTMLとWebをサポートするHTTPプロトコルは、「セッションをベースにした」プロトコルではありません。HTTPプロトコルはむしろ「リクエストをベースにした」プロトコルです。クライアント／サーバでは、ユーザはサーバに接続し、セッションで作業を行ってからサーバからの接続を解除します。HTTPでは、Webサーバの割り込み（アテンション）またはWebサーバからのアクションを必要とするアクションをユーザが実行するたびに、リクエストはサーバに送信されます。つまり、HTTPリクエストは「接続+リクエスト+応答の待機+接続解除」のシーケンスとして理解されます。

HTTP経由でクライアント／サーバセッションを実行するために、デフォルトで4Dは、URLの透過的な符号化（エンコード）によって、Web接続を一意に特定するコンテキストを保守し、それと同時に接続を処理する4Dプロセスに接続を対応させます。これがコンテキストモードです。

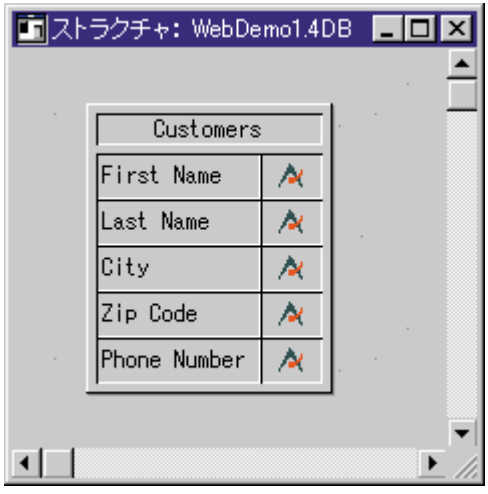
ただし、このモードで4Dにはクライアント／サーバの接続解除アクションと同等のセッションを終了させるものを提供する方法がありません。このため、クライアント／サーバセッションの終了はタイムアウト方式によって処理されます。Web接続を処理する4Dプロセスは、データベースのWebタイムアウト設定と同じだけ遅延時間が経過しても何の動作も検出されないと終了します。

1つになったデータベースとWebサーバ

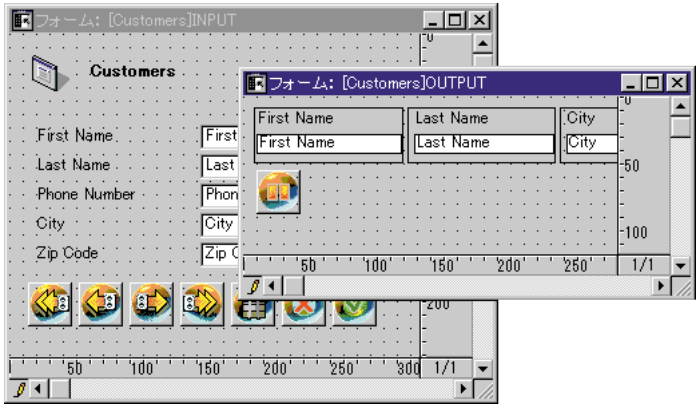
4Dメニューバー、フォーム、メソッドを使用すると、4D Webサーバセッションを完全に管理することができます。前述の例では、レコードの一覧表示と追加処理が簡単な4Dメソッドとフォームで実行されました。HTMLホームページがなかった場合には、Webブラウザには接続上でメニューバー#1が表示されていました。

HTMLホームページを省略した場合、データベースのクライアント／サーバトランザクションをサポートするWebサーバの構築は、1ユーザまたは複数のユーザに対してWindowsまたはMacintosh上に4Dデータベースを構築することで構成されます。以下の手順は、この方法でサンプルデータベースを作成する工程を説明しています。

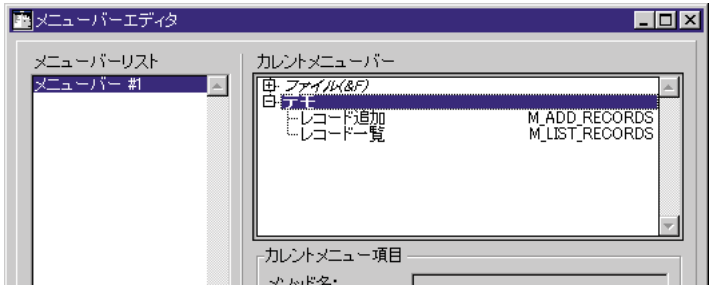
1. 以下の図はサンプルデータベースのストラクチャを示します。



2. レコードの操作ができるように入力フォームと出力フォームを追加する。



3. 「カスタム」メニューで作業を行い、Web接続をサポートするようにメニューバー番号1を追加する。

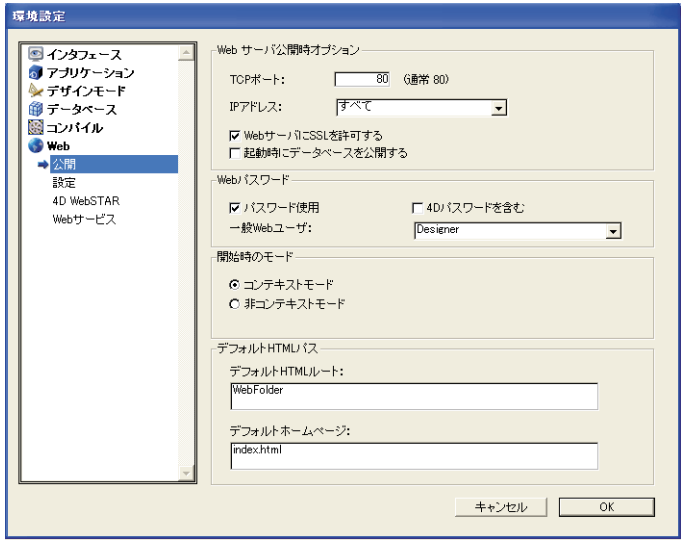


4. 2つのプロジェクトメソッドを作成する。

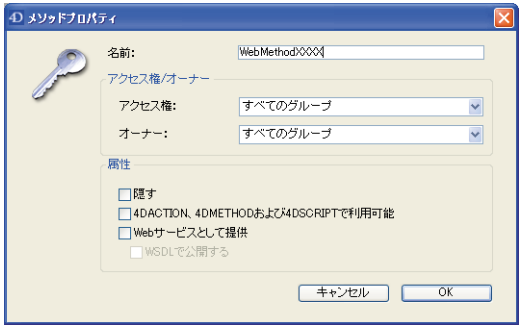
```
        `M_ADD_RECORDSプロジェクトメソッド
C_TEXT($1) `このパラメータは明示的に宣言されなければならない
Repeat
        ADD RECORD([Clients])
Until(OK=0)

        `M_LIST_RECORDSプロジェクトメソッド
C_TEXT($1) `このパラメータは明示的に宣言されなければならない
ALL RECORDS([Clients])
MODIFY SELECTION([Clients])
```

各メソッドには、「新規プロセス開始」属性が設定されます。

■ Webサーバがコンテキストモードで開始し、デフォルトのホームページは「環境設定」で定義されます。



■ ホームページには“レコードの追加”と“レコード一覧”という2つのリンク先が納められ、そのURLを介して「レコードの追加」と「レコード一覧」という4Dプロジェクトメソッドの実行をそれぞれ起動します。その表記方法は非常にシンプルで、“/4DMETHOD/メソッド名”というURLを用いることにより、任意のHTMLオブジェクトをデータベースのプロジェクトメソッドにリンクさせることができます。また、4DMETHODを使用して呼び出される各メソッドには、「4DACTION、4D METHODおよび4D SCRIPTで利用可能」属性を指定しなければなりません。



これらのリンクを定義すると、WebブラウザからURLが送り返された際に、4Dはキーワードである“/4DMETHOD/”の後ろに指定されたプロジェクトメソッドを実行します。このプロジェクトメソッドが終了した後、メソッドが起動されたHTMLページへ戻ります。このプロジェクトメソッド自体が、4Dフォームや他のHTMLページ等を表示できる点に注意してください。

作成する4DベースのWebサイトは、すべて4Dを基にしたシステムであっても、4DフォームとHTMLページを組み合わせたものでも構いません。4DデータベースからHTMLページを利用するにあたり興味深いところは、4D環境とHTML開発環境の両方を活用できるということです。もちろん、希望しない場合にはHTMLページを使用する必要はありません。

この例題のHTMLページには、レコードをサブミットするために使用するボタンがあります。HTMLボタンには、ノーマル、サブミット、リセットという3つのタイプがあります。

■ **ノーマル**：ノーマルボタンは、「/4DMETHOD/」キーワードを使用して4Dメソッドを参照するURLとみなされます。ノーマルボタンはナビゲーションのために使用されます。

■ **サブミット**：サブミットボタンは、ユーザが入力した値があればその値を用いてフォームをWebサーバにサブミットします。このボタンは、通常の4Dフォームよりも HTMLページを使用してデータ入力を実行したい場合に役立ちます。

■ **リセット**：リセットボタンは、4D開発環境においてはあまり使い道がありません。つまり、リセットボタンはユーザがフォームに入力した値があればそれをクリアし、サーバへはリクエストを一切送信しません。

4DにHTMLページを組み込む際、通常は「ノーマル」ボタンと「サブミット」ボタンを使用します。このホームページのボタンのコードは次のようになります。

```
INPUT TYPE="SUBMIT" NAME="/4DMETHOD/GO_MAIN_MENUBAR"
```

4D側からHTMLフォームをサブミットするには、フォームのサブミット後に4Dで実行されるPOSTアクション4Dメソッドを指定する必要があります。

これを行うには、「FORM ACTION="/4DMETHOD/GO_MAIN_MENU_BAR" METHOD ="POST"」という行を指定しなければなりません。

■ GO_MAIN_MENU_BARプロジェクトメソッドは次の通りです。

```
SET HOME PAGE("")
```

本例題において、このメソッドの目的は1つだけです。つまり、Webブラウザに現在表示されているデフォルトホームページから抜け、カレントのメニューバーへ移動することです。すると、4Dはデータベースのメニューバー#1に切り替えます。

これで完成です。

5分間で、ローカル上で動作するデータベースとイントラネットやインターネット上で公開可能なWebサーバの両方を備える4Dデータベースを作成しました。

参照

SEND HTML FILE、SET WEB DISPLAY LIMITS、SET WEB TIMEOUT、START WEB SERVER、STOP WEB SERVER.

接続セキュリティ

4D Webサーバの接続セキュリティは、下記の要素に基づいています。

■ Webパスワード管理システムと「**On Web Authentication**」データベースメソッドとの組み合わせ。

■ "一般Webユーザ"の定義

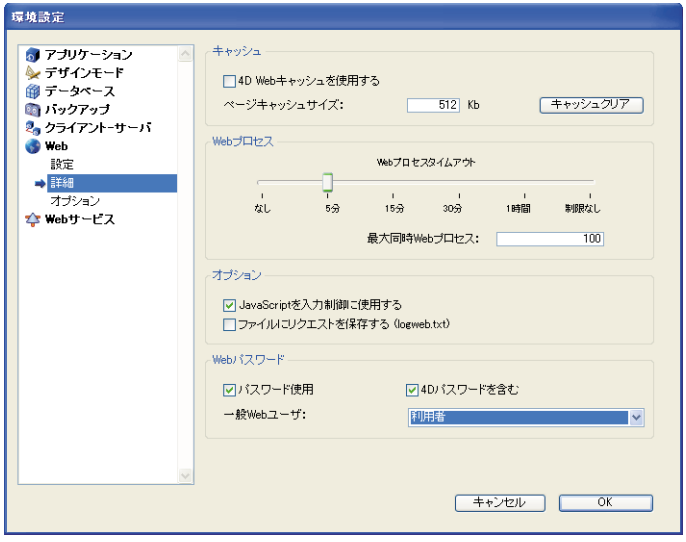
■ デフォルトのHTML、ルートフォルダの設定

■ データベースの各プロジェクトメソッドに対する「4DACTIONで利用可能」プロパティの設定。

注意：接続の安全性それ自体はSSLプロトコルを通して管理することができます。詳細は「SSLプロトコルの利用」を参照してください。

Webアクセス用のパスワードマネージメントシステム

「環境設定」ダイアログボックスにおいて、Webサーバに適用するアクセス管理システムを指定することができます。これを行うには、「環境設定」ダイアログボックスで「Web」テーマの「詳細」ページを選択します。



パスワードエリアでは、"パスワード使用"と"4Dパスワードを含む"の2つのオプションが選択できます。"4Dパスワードを含む"のチェックボックスは"パスワード使用"が選択された場合にのみ指定できます。

■ **パスワード使用**：Webサーバのパスワードシステムを起動させます。接続時にブラウザ上にダイアログボックスが表示され、名前とパスワードが入力できます。名前とパスワードならびに接続パラメータ（IPアドレスおよびポート、URL、...）は、**On Web Authentication**データベースメソッドに渡され、独自のパスワードシステムの構築等必要な処理ができるようになります。

注：この場合、On Web Authenticationデータベースメソッドが存在しなければ接続は拒絶されます。

■ **4Dパスワードを含む**：独自のパスワードシステムの代り、またはそれに付加するものとして、4Dで定義されているデータベースパスワードシステムを使用することができます。

注：

・4D Client Webサーバの場合、4D Clientマシンにより公開されたすべてのサイトは、同じユーザテーブルを共有するということを覚えておいてください。ユーザやパスワードの検証は、4D Serverアプリケーションにより実行されます。

・HTTPリクエストでは、ユーザが入力したパスワードは暗号化されません（基本モード）。

4DWebサーバのアクセスシステムのオーバービュー

4D Webサーバへの接続を判断するシステムは、2つのパラメータの組合せによります。

■ 「環境設定」ダイアログボックスの「Webパスワード」オプション

■ **On Web Authentication**データベースメソッドの存在

以下の場合があります。

オプションが何も指定されてない場合

注：デフォルトとして、新規データベースにはこれらのパラメータが設定されます。

■ **On Web Authentication**データベースメソッドが存在している場合、$1と$2の他は、$3と$4にブラウザとサーバのIPアドレスが渡され、$5と$6のユーザ名とパスワードは空白のままデータベースメソッドが実行されます。この場合、ブラウザのIPアドレスまたはサーバのIPアドレスを使用して接続を判断することができます。

■ **On Web Authentication**データベースメソッドが存在しない場合、接続は自動的に受け入れられます。
"パスワード使用"オプションが指定されて、"4Dパスワードを含む"オプションは指定されていない場合

■ **On Web Authentication**データベースメソッドが存在している場合、すべての引数が渡されます。したがって、ユーザ名、パスワード、ブラウザまたはサーバのIPアドレスに応じて、接続をより細かく判断することができます。

■ **On Web Authentication**データベースメソッドが存在しない場合、接続は自動的に拒否されて、認承メソッドが存在しないことを示すメッセージがブラウザに送られます。

注：ブラウザによって送られたユーザ名が空白で、On Web Authenticationデータベースメソッドが存在しない場合、「パスワード」ダイアログがブラウザに送られます。

"パスワード使用"および"4Dパスワードを含む"オプションが指定されている場合

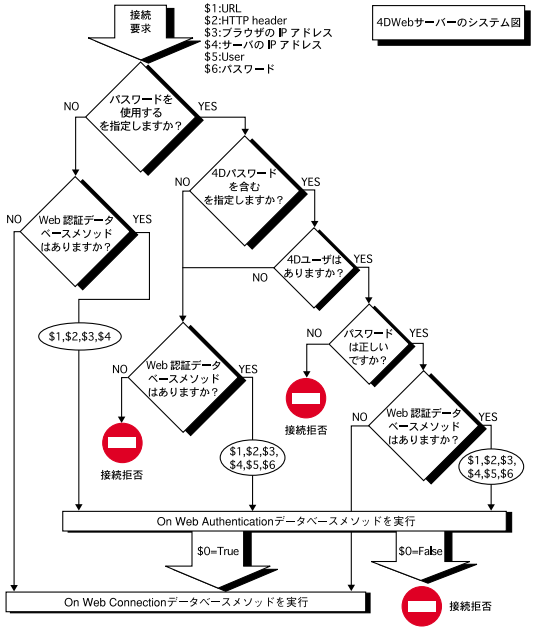
■ ブラウザによって送られたユーザ名が4Dのユーザとして登録されており、パスワードが正しければ、接続は受け入れられます。パスワードが誤っていれば接続は拒否されます。

■ ブラウザによって送られたユーザ名が4Dのユーザとして登録されてない場合、2つの可能性があります。

On Web Authenticationデータベースメソッドが存在している場合、引数$1、$2、$3、$4、$5、$6が渡されるので、ユーザ名、パスワード、ブラウザまたはWebサーバのIPアドレスを使用して接続を判断することができます。

On Web Authenticationデータベースメソッドが存在しない場合、接続は拒否されます。

4D Webサーバのアクセスシステムは、以下の図のようにまとめられます。



ロボットに関するセキュリティ注意事項

特定のロボット（query engines, spiders...）は、Webサーバやスタティックホームページを閲覧していきます。ロボットに、すべてのサイトをアクセスできるようにさせたい場合、どのURLへのアクセスを許さないのかを定義できます。

これを実行するには、ROBOTS.TXTファイルをサーバのルートに置きます。このファイルは下記の書式で構成されていなければなりません。

▼ 例：

```
User-Agent:*
Disallow:/4D
Disallow:/%23%23
Disallow:/GIFS/
```

"User-Agent:*"は、すべてのロボットに影響することを意味します。

"Disallow:/4D"は、/4Dで始まるURLにロボットがアクセスできないことを意味します。

"Disallow:/%23%23"は、/%23%23で始まるURLにロボットがアクセスできないことを意味します。

"Disallow:/GIFS/"は、/GIFS/フォルダまたはそのサブフォルダにロボットがアクセスできないことを意味します。

▼ 他の例として、

```
User-Agent:*
Disallow:/
```

この場合、ロボットはサイト全体へのアクセスが許されません。

一般Webユーザ

4Dパスワードテーブルに定義されたユーザを「一般Webユーザ」として指定することができます。この場合、データベースに接続するWebユーザは、この一般ユーザに設定されたアクセス権と制限が適応されます。したがって、データベースの異なる部分への閲覧を簡単に管理できます。

注：Webユーザがデータベースの異なる部分（テーブルやメニュー等）へのアクセス制限を受けるこのオプションを、パスワードシステムで管理されるWebサーバの接続管理システムと混同しないようにしてください。

▼ 一般Webユーザを定義するには、

1. 「デザイン」モードにおいて、パスワードエディタで少なくとも1つのユーザを作成する。

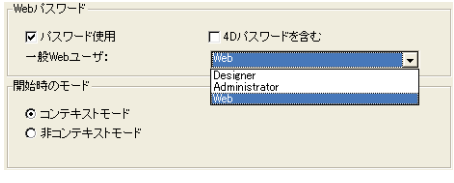
   必要であれば、ユーザにパスワードを設定できます。

2. 各エディタで、このユーザのアクセス（可能または禁止）を設定する。

3. 「環境設定」ダイアログボックスで、「Web」テーマの「詳細」ページを選択する。

   「Webパスワード」エリアには、「一般Webユーザ」のドロップダウンリストがあります。デフォルトでは、一般ユーザはデザイナであり、Webユーザはデータベース全体へのアクセス権を持ちます。

4. ドロップダウンリストからユーザを選択し、ダイアログボックスを有効にする。



データベースへの接続を許されたすべてのWebユーザは、この一般Webユーザに関連したアクセス権およびアクセスの制限が適応されます（"4Dパスワードを含む"オプションが選択され、接続するユーザが4Dパスワードテーブル内に存在しない場合を除きます。下記を参照してください）。

Webパスワードシステムとの相互作用

パスワード使用オプションは、一般のWebユーザがどのように操作するかには影響を与えません。このオプションの状態がどうであれ「一般Webユーザ」に関連したアクセス権と制限は、データベースに接続することを許されたすべてのWebユーザに適用されます。

しかし、4Dパスワードを含むオプションが選択されると、下記の2つの可能性があります。

- ユーザの名前とパスワードが4Dのユーザとして登録されていない場合、**On Web Authentication**データベースメソッドによって接続が受け入れられると、一般Webユーザのアクセス権が適用されます。
- ユーザの名前とパスワードが4Dのユーザとして登録されている場合には、「一般Webユーザ」のパラメータは無視され、ユーザ自身のアクセス権で接続します。

デフォルトHTMLルートフォルダを定義する

「環境設定」のこのオプションを使用し、4Dがブラウザに送信するスタティックHTMLページやセミダイナミックHTMLページ、ピクチャ等が含まれるフォルダを指定することができます。

さらに、Webサーバドライブ上のHTMLルートフォルダより上位の階層へのアクセスができないように定義したことになります。

このアクセス制限は、Webブラウザに送られるURLならびに"**SEND HTML FILE**"のような4DのWebサーバコマンドに適用されます。ブラウザによって送られたURLまたは4Dコマンドが、HTMLルートフォルダの上位にあるファイルにアクセスしようとした場合に、ファイルが発見できなかったことを示すエラーが返されます。

デフォルトとして、4Dは「WebFolder」という名前のHTMLルートフォルダを定義します。このフォルダが存在しない場合は、Webサーバを初めて開始した際、ディスク上にHTMLルートフォルダが物理的に作成されます。デフォルトの場所をそのまま使用した場合、ルートフォルダは次の場所に作成されます。

- 4th Dimensionおよび4D Serverでは、データベースのストラクチャファイルと同じ階層内。
- 4D Clientでは、「4D Client.exe」ファイル（Windows）またはソフトウェアパッケージ（Mac OS）と同じ階層内。

デフォルトのHTMLルートフォルダの名前や場所は、「環境設定」ダイアログボックスで変更することができます（「Web」テーマの「設定」ページ）。



"デフォルトHTMLルート"入力エリアに、定義したいフォルダの新しいアクセスパスを入力します。

このダイアログボックスに入力するアクセスパスは相対的です。つまり、データベース（4th Dimensionまたは4D Server）のストラクチャ、または4D Clientアプリケーションやソフトウェアパッケージ（4D Client）が含まれるフォルダを起点に指定します。

4D Webサーバはデータベースのマルチプラットフォームの互換性を保つため、アクセスパスを記述するための特別な書き込み規約を使用します。構文のルールは下記の通りです。

■ フォルダはスラッシュ（"/"）で分離する

■ アクセスパスはスラッシュ（"/"）で終了してはいけない

■ フォルダ階層の中で1レベル「上がる」には、".."（ピリオドを2つ）を、フォルダ名の前に入力する

■ アクセスパスはスラッシュ（"/"）で始めてはならない（HTMLルートフォルダをデータベースや4D Clientのフォルダに設定したい場合を除きます。下記を参照してください）。

HTMLルートフォルダをデータベースや4D Clientのフォルダに設定したいが、上位のフォルダへのアクセスを禁止したい場合は、"4Ddatabase/Web"を入力します。HTMLルートフォルダをデータベースフォルダにしたいが、上のフォルダへのアクセスを禁止したい場合は、"/"をエリア内に入力します。全ボリュームへアクセスを可能にするには、デフォルトHTMLルート"エリアを空にします。

警告：「環境設定」ダイアログボックスにおいて、デフォルトHTMLルートフォルダを指定しない場合、データベースのストラクチャファイルまたは4D Clientアプリケーションが含まれるフォルダが使用されます。

注：「環境設定」ダイアログボックスにおいて、HTMLルートフォルダが変更された場合、アクセスが制限されているファイルが保存されないようにキャッシュがクリアされます。

データベースの環境設定とSET HTML ROOT（コンテキストモード）

いったんHTMLルートフォルダが「データベースプロパティ」ダイアログで指定されると、**SET HTML ROOT**コマンド（コンテキストモードのみ）を使って変更することができますが、変更は現在のWebプロセスのみに適用され、HTMLページのキャッシュはクリアされます。

しかし、**SET HTML ROOT**コマンドは、「環境設定」で定義されたデフォルトHTMLルートフォルダを考慮します。「環境設定」ダイアログボックスで定義されたフォルダが“WebPages/”であり、SET HTML ROOT（"Folder"）をという命令を渡した場合、デフォルトのHTMLルートフォルダは“WebPages/Folder/”になります。また、この場合でも、接続制限は“WebPages”フォルダの上位階層のフォルダに対して適用されます。

注：Webサーバが非コンテキストモードの時は、SET HTML ROOTコマンドは何も行ないません。

4DACTION、4D METHOD、4D SCRIPTを利用する

4DACTION（非コンテキストモード）ならびに4DMETHOD（コンテキストモード）という特別なURLを使用し、Web上に公開された4Dデータベースのプロジェクトメソッドの実行を開始することができます。例えば、http://www.server.com/4DACTION/Erase_Allというリクエストの場合、Erase_Allプロジェクトメソッドが存在すれば、それが実行されます。

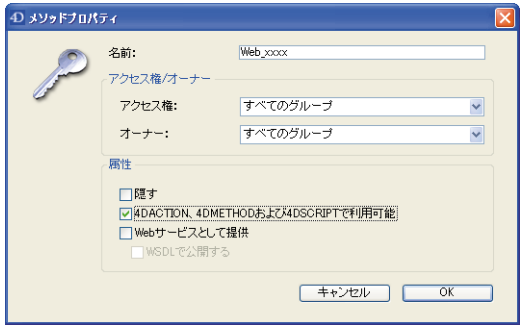
この結果、この仕組みにはデータベースに関するセキュリティ上のリスクが存在します。特に、インターネットユーザが故意に（または無意識に）Webを介して意図せずにメソッドの実行を起動してしまう危険性があります。

次の3つの方法により、この危険を回避することができます。

■ 4Dパスワードシステムを用いて、プロジェクトメソッドへのアクセスを制限する。
欠点：このシステムでは4Dのパスワードを使用する必要があり、あらゆるタイプのメソッドの実行を禁止します（HTMLタグの使用も含む）。

■ 「On Web Authentication」データベースメソッドを用いて、URL経由でのメソッド呼び出しをフィルタリングする。
欠点：データベースに多数のメソッドが存在する場合、このシステムの運用が難しい可能性があります。

■ 「メソッドプロパティ」ダイアログの「4DACTION、4D METHODおよび4D SCRIPTで利用可能」オプションを使用する。



このオプションを使用して、4DACTIONならびに4DMETHOD（コンテキストモード）という特別なURLを用いて呼び出されるプロジェクトメソッドを個別に指定します。このオプションが選択されていない場合、特別なURLを含むHTTPリクエストを使用して、そのプロジェクトメソッドを実行することができません。その代わりに、他のタイプの呼び出し（4Dタグ、他のメソッド等）を用いて実行されます。

4th Dimension 2003以降で作成されたデータベースに対し、デフォルトではこのオプションが選択されていません。4DACTIONや4DMETHOD Web URLを使用して実行されるメソッドには、このオプションを必ず指定しなくてはなりません。

これとは逆に、互換性上の理由から、既存のデータベース（2003より以前のバージョンの4Dで作成）に対しては、このオプションが選択されています。デフォルトでは、特別な4DのURLを用いて、すべてのプロジェクトメソッドにアクセスすることができます。

「エクスプローラ」において、“4DACTION、4D METHODおよび4D SCRIPTで利用可能な”プロジェクトメソッドには、特定のアイコンが使用されます。

参照

On Web Authentication データベースメソッド、On Web Connection データベースメソッド、SSLプロトコルの使用

「On Web Authentication」データベースメソッド

「**On Web Authentication**」データベースメソッドは、Webサーバエンジンへのアクセスを管理します。Webブラウザのリクエストにおいてサーバ上での4Dメソッドの実行が必要となる場合に（4DACTION、4DCGI URL、または4DSCRIPTタグ等を用いて呼び出されたメソッド）、4th Dimension、4D Server、または4D Clientによりこのデータベースメソッドが呼び出されます。

このメソッドには6つの引数$1、$2、$3、$4、$5、$6が渡されます。またひとつのブール値が返されます。これらの引数の説明は次の通りです。

パラメータ	タイプ	内容
$1	テキスト	URL
$2	テキスト	HTTPヘッダ + HTTPボディ(上限32kb)
$3	テキスト	Webクライアント（ブラウザ）のIPアドレス
$4	テキスト	サーバのIPアドレス
$5	テキスト	ユーザ名
$6	テキスト	パスワード
$0	ブール	True= リクエストが受け入れられた False= リクエストが拒否された

これらの引数は以下のように定義しなければなりません。

```
` On Web Authentication データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)
` メソッドのコード
```

注：On Web Authenticationデータベースメソッドの全引数が有効とは限りません。データベースメソッドが受け取る情報は、「環境設定」ダイアログボックスで事前に選択したオプションよって異なります（「接続セキュリティ」の節を参照してください）。

URL

最初の引数（$1）は、ブラウザのロケーションエリアにユーザが入力したURLであり、ホストアドレスが削除されたものです。

イントラネット接続を例に説明しましょう。4D WebサーバマシンのIPアドレスが「192.168.0.99」であると仮定します。以下の表は、Webブラウザに入力されたURLに従った$1の値を示しています。

Webブラウザロケーションエリアに入力されたURL	引数$1の値
192.168.0.99	/
http://192.168.0.99	/
192.168.0.99/Customers	/Customers
http://192.168.0.99/Customers	/Customers
http://192.168.0.99/Customers/Add	/Customers/Add
192.168.0.99/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

特別なケース：非コンテキストモードにおいて、要求されたURLが既存のページか4Dの特別なURLのいずれでもない場合、引数 $1の値は“/”で始まりません。例えば、URLが“http://192.168.0.99/Clients/Add”であり、“Add”ページが「Clients」フォルダに存在しない場合、 “Clients/Add”を$1の値として「On Web Authentication」データベースメソッドが呼び出され、次に「On Web Connection」データベースメソッドが呼び出されます。

■ HTTPリクエストのヘッダとボディ

2番目の引数（$2）は、Webブラウザから送信されたHTTPリクエストのヘッダとボディです。このヘッダが**On Web Authentication**データベースメソッドにそのまま渡されていることに注意してください。そのコンテンツは接続を試行するWebブラウザの特徴によって変わります。

アプリケーションの中でこのヘッダとボディ情報を使用する場合、それを解析するのはプログラマの仕事です。

注：パラメータの詳しい説明はOn Web Connectionを参照してください。

■ WebクライアントのIPアドレス

$3には、ブラウザが動作しているマシンのIPアドレスが渡されます。この情報は、イントラネット接続とインターネット接続を区別できるようにするものです。

■ WebサーバのIPアドレス

$4には、4D Webサーバの呼び出しに利用するIPアドレスが渡されます。バージョン6.5以降では、複数のIPアドレスを持つマシンでマルチホーミングを可能にしています。より詳しい情報は、「Webサービス：Webサーバセッティング」を参照してください。

■ ユーザ名とパスワード

$5および$6には、ユーザによって入力されたユーザ名とパスワードが渡されます。「パスワード入力」ダイアログは、「環境設定」ダイアログボックスで「パスワード使用」オプションが選択された場合、各接続時に表示されます（より詳しい情報は、「Webサービス：接続セキュリティ」を参照してください)。

注：ユーザ名が4Dに存在するブラウザから送られた場合、$6パラメータ（ユーザのパスワード）はセキュリティの理由で返されません。

■ 引数$0

On Web Authenticationデータベースメソッドはブールタイプの戻り値を返します。

■ 接続を受け入れる場合は、$0にTrueを設定します。

■ 接続を拒否する場合は、$0にFalseを設定します。

On Web Connectionデータベースメソッドは、**On Web Authentication**によって接続が受け入れられた場合にのみ実行されます。

警告：$0に値が入っていない、または$0がOn Web Authenticationデータベースメソッドで定義されていない場合、接続は接続は受け付けられたとみなされ、On Web Connectionデータベースメソッドが実行されます。

注：

・On Web Authenticationデータベースメソッド内では、いかなるインタフェースエレメント（ALERTやDIALOG等）も呼び出してはなりません。呼び出した場合、操作は中断され、接続は拒絶されます。データベースメソッドが実行されている間にエラーが発生した場合も同様です。

・「メソッドプロパティ」ダイアログボックスの「4DACTION、4D METHODおよび4D SCRIPTで利用可能」オプションを使用すると、各プロジェクトメソッドに対して4DACTIONや 4DMETHODによる実行を禁止することができます。この件に関する詳細は、「接続セキュリティ」の節を参照してください。

「On Web Authentication」データベースメソッドの呼び出し

リクエストや処理において4Dメソッドの実行が必要になると、モードに関わらず「On Web Authentication」データベースメソッドが自動的に呼び出されます。また、Webサーバが無効なスタティックURL（例えば、リクエストされたスタティックページが存在しない場合）を受信した場合にも呼びだされます。

したがって「On Web Authentication」データベースメソッドは次の状況で呼び出されます。

■ 4Dが“4DACTION/”で始まるURLを受信した場合

■ 4Dが“4DMETHOD/”で始まるURLを受信した場合

■ 4Dが“4DCGI/”で始まるURLを受信した場合

■ 4Dが存在しないスタティックページを要求するURLを受信した場合

■ 4Dがセミダイナミックページの4DSCRIPTタグを処理する場合

■ 4Dがセミダイナミックページのメソッドに基づいた4DLOOPタグを処理する場合

注：有効なスタティックページを要求するURLをサーバが受信した場合、「On Web Authentication」データベースメソッドは呼び出されません。

例題

▼ **On Web Authentication**データベースメソッドの代表的な例を示します。

```
a`Web認証データベースメソッド
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:=$4
`セキュリティ上で,@（ワイルドカード）が含まれる名前を拒否します
If(WithWildcard($user) | WithWildcard($password))
	$0:=False
		`ワイルドカードメソッドを以下に記述します
Else
		`4D userかどうかチェックする
	GET USER LIST($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($4Duser))
		`4Dが定義したユーザではありません。Webユーザのテーブルを見ます。
		QUERY([WebUsers];[WebUsers]User=$user;*)
		QUERY([WebUsers]; & [WebUsers]Password=$password)
		$0:=(Records in selection([WebUsers]) = 1)
	Else
		$0:=True
	End if
End if
	`これはイントラネット接続ですか？
If(Substring($BrowserIP;1;7) # "192.100.")
	$0:=False
End if
```

▼ WithWildcardメソッド：

```
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
```

参照

On Web Connectionデータベースメソッド、Webサービス：接続セキュリティ、Webサービス：特別なURLとフォームアクション

「On Web Connection」データベースメソッド

「On Web Connection」データベースメソッドは、次の3つの状況で呼びだされます。

- Webサーバが「4DCGI」URLで始まるリクエストを受信した場合。
- Webサーバが無効なリクエストを受信した場合。
- また、Webブラウザがコンテキストモードでデータベースへの接続を開始するか、あるいはWebサーバがコンテキストの作成を必要とするリクエストを受信するたびに、4th Dimensionや4D Serverから呼び出されます（このケースは、4D Clientでは対処されません。4D Clientはコンテキストモードをサポートしません。）。

詳細については、「**On Web Connection**データベースメソッドの呼び出し」の節を参照してください。

リクエストは「**On Web Authentication**」データベースメソッド（存在する場合）によって事前に許可されていなければなりません。また、データベースはWebサーバとして公開されていなければなりません。

On Web Connectionデータベースメソッドは、4Dから渡される6つのテキスト引数を受け取ります。

これら引数の内容は以下の通りです。

パラメータ	タイプ	内容
$1	テキスト	URL
$2	テキスト	HTTPヘッダ+ HTTPボディ（上限32kb）
$3	テキスト	Webクライアント（ブラウザ）のIPアドレス
$4	テキスト	サーバのIPアドレス
$5	テキスト	ユーザ名
$6	テキスト	パスワード

これらの2つの引数は、以下のように宣言できます。

```
`「On Web Connection」データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
` メソッドのコード
```

URLエクストラデータ

最初の引数（$1）は、ブラウザのロケーションエリアでユーザが入力したURLであり、ホストアドレスが削除されたものです。

イントラネット接続を例に説明しましょう。4D Webサーバマシンの IPアドレスが「192.168.0.99」であると仮定します。以下の表は、Webブラウザに入力されたURLに従った$1の値を示しています。

Webブラウザロケーションエリアに入力されたURL	引数$1の値
192.168.0.99	/
http://192.168.0.99	/
192.168.0.99/Customers	/Customers
http://192.168.0.99/Customers	/Customers
http://192.168.0.99/Customers/Add	/Customers/Add
192.168.0.99/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

この引数は、必要に応じて自由に使用してください。4DはURLのホスト部分以外に渡される値は単に無視します。

例えば、値"/Customers/Add"で「[顧客]テーブルに新しいレコードを追加するために直接移動せよ」という意味の表記を設定することができます。データベースのWebユーザに対して利用可能な値のリストやデフォルトのブックマークを提示することで、アプリケーションのさまざまな部分へのショートカットを提供できます。この方法により、Webユーザは、データベースに新しく接続するたびにナビゲーションパス全体を通らずに、即座にWebサイトのリソースにアクセスできます。

警告：前回のセッションで作成されたブックマークでデータベースにユーザが再度入るのを避けるために、4Dは標準4D URLのいずれかに対応するURLがあればそれを途中で処理します。

■ HTTPリクエストのヘッダとボディ

2番目の引数（$2）は、Webブラウザから送信されたHTTPリクエストのヘッダとボディです。このヘッダが**On Web Connection**データベースメソッドにそのまま渡されていることに注意してください。そのコンテンツは接続を試行するWebブラウザの特徴によって変わります。

Mac OS上で動作するNetscape 4.5を使用している場合、次のようなヘッダを受け取ります。

```
GET/ HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.5(Macintosh; I; PPC)
Host: 192.168.0.99
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: us
Accept-Charset: iso-8
```

Windows上で動作するMicrosoft Internet Explorer 6を使用している場合、次のようなヘッダを受け取ります。

```
229=1406
GET/ HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.0(compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.0.99
Accept: image/gif, image/x-xbitmap, image/pjpeg, */*
Accept-Language: us
```

アプリケーションがこの情報を扱う場合には、ヘッダとボディの解析はプログラマの責任になります。

■ WebクライアントのIPアドレス

3番目の引数（$3）には、ブラウザが動作しているマシンのIPアドレスが渡されます。この情報は、イントラネット接続とインターネット接続を区別できるようにするものです。

■ WebサーバのIPアドレス

4番目の引数（$4）には、4D WebサーバのIPアドレスが渡されます。4Dは複数のIPアドレスを持つマシンでマルチホーミングを可能にしています（より詳しい情報は「Webサーバセッティング」を参照）。

■ ユーザ名とパスワード

5、6番目の引数$5および$6には、ユーザによって入力されたユーザ名とパスワードが渡されます。「パスワード入力」ダイアログは、「環境設定」ダイアログボックスで「パスワード使用」オプションが選択された場合、各接続時に表示されます（「接続セキュリティ」を参照）。

注：ブラウザから送信されたユーザ名が4Dに存在する場合は、セキュリティ上の理由から、$6パラメータ（ユーザのパスワード）が返されません。

On Web Connectionデータベースメソッドの呼び出し

特別な「4DCGI」URL、またはカスタマイズしたコマンドのURLのいずれかを使用する場合、「On Web Connection」データベースメソッドは、4D Webサーバの入り口点として使用されます。また、コンテキストモードにおいても入口点としての役割を果たします（4th Dimensionと4D Serverの使用時）。

警告：インタフェース要素（ALERT、DIALOG等）を表示する4Dコマンドを呼びだすと、メソッドの処理が終了します。

On Web connectionデータベースメソッドは以下の場合に呼ばれます。

■ ブラウザから非コンテキストモードで動作する4D Webサーバに接続する場合。データベースメソッドは「/アクション...」というURLを使用して呼び出されます。

■ 4Dが「/4DMETHOD」というURLを受け取った場合。Webサーバはコンテキストモードへ切り替わり、データベースメソッドは$1に代入された「/4DMETHOD/メソッド名」というURLを使用して呼び出されます。

■ 4Dが「/4DCGI」というURLを受け取った場合。データベースメソッドは$1に代入された「/4DCGI/アクション」というURLを使用して呼び出されます。

■ Webページが「<path>/<file>」タイプのURLを使用して呼び出され、指定されたファイルが見つからない場合。こうした特別の場合、$1に受け取られるURLは"/"で始められません[1]。

■ タイプ<file>のURLでWebページが呼ばれ、デフォルトのホームページが定義されていない場合、データベースメソッドはURLでコールされます。こうした特別の場合、$1に受け取られるURLは"/"で始められません。

(*) このような特殊な状況では、$1に受け取るURLが“/”で始まりません。

On Web Connectionデータベースメソッドがコンテキストから、または非コンテキストでの接続から呼び出されたかをを知るには、**Web Context**関数を使うことができます。コンテキストモードで呼ばれた場合はTrueが戻り、そうでなければFalseが戻ります。

したがって、**On Web Connection**データベースメソッドを以下の構造にすることをおすすめします。

```
` On Web connectionデータベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
If(Web Context) `もしコンテキストモードなら
	WithContext($1;$2;$3;$4;$5;$6)
		` WithContextは4D 6.0.xのOn Web connection
		`データベースメソッドのすべてを含む
Else
	NoContext($1;$2;$3;$4;$5;$6)
		`NoContextメソッドは非コンテキストモードで
		'リクエストの処理を実行する（通常は短い）
End if
```

例：コンテキストモードにおけるクライアントローカルホームページの実装

以下の例では、**On Web Connection**データベースメソッドに送られる引数$1は、ある組織の中でクライアントホームページを実装するために使用されます。イントラネットサーバはコンテキストモードで動作します。

データベースには、[顧客]と[商品]という2つのテーブルがあります。次に示す**On Startup**データベースメソッドは、後で**On Web Connection**データベースメソッドで使用されるインタープロセス配列を初期化します。

```
`「On Startup」データベースメソッド
` テーブルリスト
ARRAY STRING(31 ; <>asTables ; Count tables)
For($vlTable ; 1 ; Size of array(<>asTables))
	<>asTables{$vlTable}:=Table name($vlTable)
End for
	`ログイン時の標準Webアクション
ARRAY STRING(31 ; <>asActions ; 2)
<>asActions{1}:="Add"
<>asActions{2}:="List"
```

On Web Connectionデータベースメソッドの主な仕事は、アドレスのホスト部分の後ろにあるURLに渡されたエクストラデータを解読して、それに従ってアクションを実行することです。このメソッドは、以下の通りです。

```
`「On Web Connection」データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
C_TEXT($vtURL)
```

```
If(Web context) `もしコンテキストモードなら
    `万一に備えて$1が"/"または"/..."と同じかどうかをチェックする
    If($1="/@")
        `URLから最初の"/"を引いたものをローカル変数にコピーする
    $vtURL:=Substring($1 ; 2)
        `URLを解析し、URLのトークンでローカル配列を登録する
        `例えば、、URLのエクストラデータが"aaa/bbb/ccc"の場合には、
        `結果の配列は"aaa"、"bbb"、"ccc"の順の3つの要素になる
    $vlElem:=0
    ARRAY TEXT($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
            $vtURL:=""
        End if
    End while
    `URLのHOST部分の後にエクストラデータが渡された場合
    If($vlElem>0)
        `On Startupデータベースメソッドで初期化されたインタープロセス配列を
            使用して、1番目のトークンがテーブルの名前であるかどうかを
                                                チェックする
        $vlTableNumber:=Find in array(<>asTables ; $atTokens{1})
        If($vlTableNumber>0)
            `その場合には、このテーブルに対するポインタを取得する
            $vpTable:=Table($vlTableNumber)
            `入力フォームと出力フォームを設定する
            INPUT FORM($vpTable-> ; "Web入力")
            OUTPUT FORM($vpTable-> ; "Web出力")
            `On Startupデータベースメソッドで初期化されたインター
                    プロセス配列を使用して2番目のトークンが既知の
                        標準アクションであるかどうかをチェックする
            $vlAction:=Find in array(<>asActions ; $atTokens{2})
            Case of
                `レコードの追加
                ¥($vlAction=1)
```

```
                                    Repeat
                                        ADD RECORD($vpTable-> ; *)
                                    Until(OK=0)
                                    `レコードの一覧表示
                                ¥($vlAction=2)
                                    READ ONLY($vpTable->)
                                    ALL RECORDS($vpTable->)
                                    DISPLAY SELECTION($vpTable-> ; *)
                                    READ WRITE($vpTable->)
                            Else
                            `ここで追加の標準テーブルアクションを実装できる
                            End case
                    Else
                    `ここでその他の標準アクションを実装できる
                    End if
            End if
    End if
    `これまでのコードで何が起こっても、通常のLog Onプロセスで追跡する
    WWW NORMAL LOG ON
    Else
    ... ` ここは非コンテキストモードが管理するコードが
            `実行されます。
    End if
```

この時点で、この組織の人々は、データベースに接続し、リストされたメソッドで設定された表記に従ったURLを入力できます。また、ユーザは毎回URLを再入力したくない場合には、お気に入りを作成することもできます。最終的なソリューションは、組織のメンバ全員に対してデータベースをアクセスするためにローカルで使用するHTMLページを提供することです。このHTMLページは、以下のようになります。



つまり、HTMLページ「ACME_IS.HTM」は、組織内の4Dベースの情報システムに対するクライアントのローカルホームページなのです。ユーザが「新規顧客追加」リンクをクリックすると、WebブラウザはURLが「http://192.168.0.99/Products/Add」であるホストに接続します。データベースコンピュータのIPアドレスが「192.168.0.99」である場合、**On Web Connection**データベースメソッドは$1にURLのエクストラデータ"/Products/Add"を受け取り、このため[商品]テーブルへのレコード追加処理に進みます。

最後には、ユーザはそのページからデスクトップ上へリンクをドラッグ&ドロップして、次に示す「新規顧客追加」アイコンのようなInternet Shortcutアイコンを作成できます。これらのアイコンをダブルクリックするだけで、4D Webデータベースの任意の場所に直接移動できます。



次は、このHTMLページのソースコードです。

```
<HTML>
<HEAD>
 <TITLE>ACME Intranet Information System</TITLE>
</HEAD>
<BODY>
<H1><B>ACME Intranet Information System</B></H1>
<P ALIGN=CENTER><TABLE BORDER=1 CELLPADDING=1 WIDTH="100%">
 <TR>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Customers/Add">新規顧客追加</A>
 </TD>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Products/Add">新商品追加</A>
 </TD>
 </TR>
 <TR>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Customers/List">顧客一覧</A>
 </TD>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Products/List">商品一覧</A>
 </TD>
 </TR>
</TABLE></P>
<P><B><A HREF="Help.HTM">If you need Help click Here!</A></B></P>
</BODY>
</HTML>
```

参照

データベースメソッド、On Web Authenticationデータベースメソッド、URLとフォームアクション、コンテキストモードの使用

HTMLオブジェクトと4Dオブジェクトのバインド

この節では、Web経由での情報のやり取り、つまり動的に値を送受信するために4D Webサーバで利用できる方法について説明します。

次の事柄が処理されます。

■ 4D変数に保存された値を動的に送信。

■ Webフォームを介してダイナミックな値を受信。

■ COMPILER_WEBプロジェクトメソッドの使用。

■ サーバ側のイメージマッピングの管理。

■ JavaScriptの埋め込み。

注：ダイナミックな値の送受信は、変換された4Dフォームを使用し、コンテキストモードで自動的に実行されます。この件に関する詳細は、「コンテキストモードの使用」の節を参照してください。

ダイナミックな値の送信

HTMLページに4D変数への参照を挿入することができます。これらの参照を任意のタイプのHTMLオブジェクトにバインドすることができます。Webページがブラウザに送信されると、4Dはこれらの参照を変数の現在値で置き換えます。この結果、受信したページにはスタティック要素と4Dの値の両方が含まれます。このタイプのページはセミダイナミックページと呼ばれます。

注意：

・作業にはプロセス変数を使用します。

・HTMLは言語に関連づけられた単語処理をするので、普通テキスト変数を伴って使用するでしょう。しかし、BLOB変数を使うこともできます（これにより、テキスト変数の32000バイトの上限を回避することができます）。ただし、“text without length”フォーマットのBLOB変数を生成する必要があります。

まず最初に、HTMLオブジェクトは4D変数の値を使用してその値を初期化することができます。

次に、Webフォームがサブミットして返された後に、HTMLオブジェクトの値は4D変数に納めて返すことができます。これを実行するには、フォームのHTMLソースの中で、バインドしたい4Dプロセス変数の名前と同じ名前のHTMLオブジェクトを作成します。この詳細については、「ダイナミックな値の受信」の節で更に詳しく検討します。

注：非コンテキストモードでは、4Dピクチャ変数を参照することはできません。

HTMLオブジェクト値は4D変数値で初期化できるため、プログラムでHTMLオブジェクトのvalueフィールドに<!--#4DVAR 変数名-->を含めることにより、デフォルト値を提供できます。この“変数名”は、現在のWebプロセスで定義された4Dプロセス変数の名前です。この変数名を標準のHTML表記<!--#...-->で囲みます。

注意：いくつかのHTMLエディタはHTMLオブジェクトの変数フィールド中の<!--#4DVAR 変数名-->を受け付けないかもしれません。その場合、HTMLコード中に記述する必要があります。

また、<!--#4DVAR-->タグを使用し、送信するページに4D表記を挿入することができます（フィールド、配列要素等）。このタイプのデータを使用したこのタグの処理は、変数の場合と同じです。詳細については「4DのHTMLタグ」の節を参照してください。

実際、<!--#4DVAR 変数名-->構文を使用すると、HTMLページの任意の場所に4Dデータを挿入できます。例えば、以下のように書いた場合に、

Welcome to <!--#4DVAR vtSiteName-->!

4D変数「vtSiteName」の値が、HTMLページ内に挿入されます。

以下に例を示します。

```
` 以下の4Dコードは、"4D4D"をプロセス変数「vs4D」に割り当てます
vs4D:="4D4D"
` 次に、HTMLページ"AnyPage.HTM"を送信します
SEND HTML FILE("AnyPage.HTM")
```

HTMLページAnyPage.HTMのソースは、以下の通りです。

```
<HTML>
<HEAD>
   <TITLE>AnyPage</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

        function ls4DWebServer(){
           return (document.frm.vs4D.value=="4D4D")
        }

        function HandleButton(){
           if (ls4DWebServer()){
              alert("あなたは4D Webサーバに接続しています!")
           } else {
              alert("あなたは4D Webサーバに接続していません!")
           }
        }


   //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frm">

<P><INPUT TYPE="hidden" NAME="vs4D" VALUE="<!--4DVAR vs4D-->"></P>

<P><A HREF="JavaScript:HandleButton()"><IMG SRC="AnyGIF.GIF" BORDER=0
ALIGN=bottom></A></P>

<P><INPUT TYPE="submit" NAME="bOK" VALUE="OK"></P>

</FORM>
</BODY>
</HTML>
```

前述のHTMLソースコードのvs4Dという名前のhidden入力オブジェクトを見てください。このオブジェクトの値はテキスト値“<!--#4DVAR vs4D-->”に設定されています。HTMLファイルを送信しているプロジェクトメソッドには、以前に定義された4Dプロセス変数vs4Dがあるため、4DはこのHTMLオブジェクトの値を置き換えて4D変数の値である"4D4D"に設定します。

埋め込みJavaScript関数「Is4DWebServer」は、vs4D HTMLオブジェクトの値をテストします。以下にその手順を示します。HTMLページが4Dから提供される場合には、オブジェクトの値は"4D4D"に変更されます。ただし、HTMLページが別のアプリケーション（つまり、Macintosh上の4D WebSTAR）から提供される場合には、オブジェクトはHTMLページ内で定義された値“<!--#4DVAR vs4D-->”のままになります。つまり、Webブラウザ側にあるページ内からJavaScriptを使ってそのオブジェクトの値をテストすることで、ページが4Dから提供されたものかどうかが検出できるのです。

この最初の例は、4Dで提供される際に他のWebサーバとの互換性を維持しながら追加の機能を提供する「インテリジェントな」HTMLページを構築する方法を示しています。

重要：ユーザがバインドできるのはプロセス変数だけです。また、現行のバージョンでは、4D配列でHTML SELECTオブジェクトをバインドすることはできません。一方、SELECTオブジェクトの各要素は独立した4D変数を参照できます（最初の要素をV1へ、2番目の要素をV2へなど）。

4DからWebブラウザへ向けてのバインド設定は、任意のカプセル化メソッド（**SEND HTML FILE**、**SEND HTML BLOB**コマンド、ならびにコンテキストモードにおける4Dフォームのスタティックテキストやテキスト変数、またはBLOB変数）を使用して行われます。

サーバから送信されたページの解析

■ コンテキストモードでHTMLページ（HTMLドキュメントや変換された4Dフォーム）を送信する前に、常に4DはHTMLソースコードを解析し、4D変数を参照するオブジェクトを探します。

■ 非コンテキストモードでは最適化のため、HTMLページが“.HTML”や“.HTM”で終わるシンプルなURLを用いて呼び出される場合には、4D Webサーバによる HTMLソースコード解析が行われません。もちろん、4Dでは必要に応じてページの解析を“強制的に”行える仕組みが提供されています（「4DのHTMLタグ」の節を参照）。

4D変数へのHTMLコードの挿入

4Dの変数にはHTMLコードを挿入することができます。HTMLスタティックページがウェブサーバー上に表示されているとき、変数の値はHTMLコードに置き換えられ、ブラウザによって変換されます。

4D変数にHTMLコードを挿入するためには、２つの方法があります。

■ ４Dの変数を、ASCIIコードの1で始まるようにして(例えば、vtHTML:=Char(1)+"...HTMLコード...")、それを<!--#4DVAR vtHTML-->タグを使うHTMLページに加える。

■ 直接4D変数を(例えば、vtHTML:="...HTMLコード...")、<!--#4DHTMLVAR vtHTML--> タグを使うHTMLページに挿入する。

テキスト変数かBLOB変数(Text Without Lengthモードで生成されたもの)を使うことができます。

詳細は「4D HTMLタグ」の節を参照してください。

ダイナミックな値の受信

SEND HTML FILE、**SEND HTML BLOB**コマンドを使用してHTMLページを送信する場合、「Webブラウザから4Dへ」の方向で4D変数をHTMLオブジェクトにバインドすることもできます。このバインドは双方向に作動します。つまり、HTMLフォームがサブミットされると、4DはHTMLオブジェクトの値を4Dプロセス変数にコピーして戻します。データベースをコンパイルする場合を考慮して、これらの変数はCOMPILER_WEBメソッド内で必ず宣言してください（後述する節を参照）。

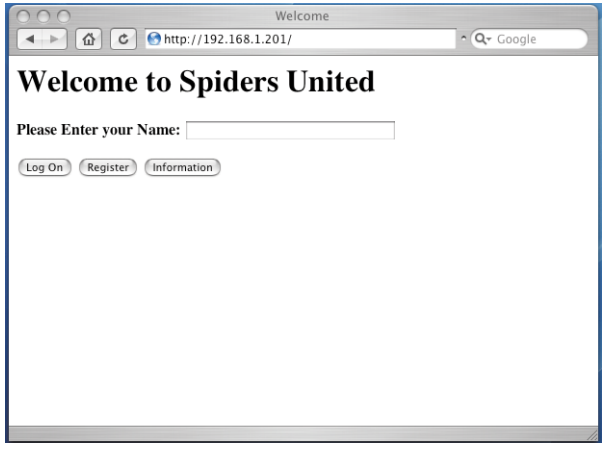
さらに、**GET WEB FORM VARIABLES**コマンドを使用すると、Webフォーム内に含まれているフィールドが事前に分からなくても、4Dへ送信されたWebフォームの値を取得することができます。

警告：値を4Dプロセス変数に取り戻すことは、HTMLページがSEND HTML FILEかSEND HTML BLOBコマンドを使用して送信されたときにのみ可能です。コンテキストモードにおいてHTMLを4Dフォームに組み込んだ場合、値の取得はフォームに実際に存在する4Dのオブジェクトに制限されます。

次のHTMLページソースコードを参照してください。

```
<HTML>
<HEAD>
   <TITLE>Welcome</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

         function GetBrowserInformation(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
            return true
         }

         function LogOn(formObj){
            if(formObj.vtUserName.value!=""){
               return true
            } else {
               alert("Enter your name, then try again.")
               return false
            }
         }
    //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frmWelcome"
                           onsubmit="return GetBrowserInformation(frmWelcome)">

<H1>Welcome to Spiders United</H1>

<P><B>Please Enter your Name:</B>
   <INPUT TYPE="text" NAME="vtUserName" VALUE="<!--4DVAR vtUserName-->" SIZE=30></P>
<P>
   <INPUT TYPE="submit" NAME="vsbLogOn" VALUE="Log On" onclick="return LogOn(frmWelcome)">
   <INPUT TYPE="submit" NAME="vsbRegister" VALUE="Register">
   <INPUT TYPE="submit" NAME="vsbInformation" VALUE="Information">
</P>

<P>
   <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>
</FORM>
</BODY>
</HTML>
```

4Dがこのページをwebブラウザに送ると、以下のように表示されます。



このページの主な機能は、以下のようになります。

■ vsbLogOn、vsbRegister、vsbInformationという3つのサブミットボタンがあります。

■ ユーザが「ログイン」ボタンをクリックすると、フォームのサブミットは最初にJavaScript関数「LogOn」で処理されます。名前が入力されていないと、フォームは4Dへのサブミットを行わないで、JavaScriptの警告を表示します。

■ フォームには4DメソッドをPostするSubmitスクリプト（GetBrowserInformation）があります。このスクリプトは、名前が「vtNav_App」で始まる4つの隠しオブジェクトに対してNavigatorプロパティをコピーします。

■ オブジェクト「vtUserName」の初期値は、<!--#4DVAR vtUserName-->です。

このHTMLページを**SEND HTML FILE**コマンドを使用して送信する4Dメソッド「WWW Welcome」の内容を見てみましょう。このメソッドは、「On Web Connection」データベースメソッドから呼び出されます。

```
`「WWW Welcome 」プロジェクトメソッド
` WWW Welcome → ブール
` WWW Welcome → Yes = セッションを開始できる
C_BOOLEAN($0)
$0:=False
`ブラウザ情報を返す隠しINPUT HTMLオブジェクト
C_TEXT(vtNav_appName ; vtNav_appVersion ; vtNav_appCodeName ;
                                                    vtNav_userAgent)
vtNav_appName:=""
vtNav_appVersion:=""
vtNav_appCodeName:=""
vtNav_userAgent:=""
`ユーザ名が入力されるテキストINPUT HTMLオブジェクト
C_TEXT(vtUserName)
vtUserName:=""
` HTMLサブミットボタンの値
C_STRING(31 ; vsbLogOn ; vsbRegister ; vsbInformation)
Repeat
    `サブミットボタンの値をリセットするのを忘れずに！
    vsbLogOn:=""
    vsbRegister:=""
    vsbInformation:=""
    `Webページを送信する
    SEND HTML FILE("Welcome.HTM")
    `どのサブミットボタンがクリックされたのかを検出するためにボタンの値
                                                    をテストする
    Case of
```

```
            `「Log On 」ボタンがクリックされた場合
            ¥(vsbLogOn  #  "")
                  QUERY([WWWユーザ];[WWWユーザ]ユーザ名
                                                =vtUserName)
                  $0:=(Records in selection([WWWユーザ])>0)
                  If($0)
                        WWW POST EVENT("Log On" ; WWWログ情報)
                        `WWW POST EVENTメソッドはデータベーステーブル
                                                に情報を保存する
                  Else
                        CONFIRM("このユーザ名は登録されていません。
                                                登録しますか?")
                        $0:=(OK=1)
                        If($0)
                              $0:= WWW Register
                              `WWW Registerメソッドで新しいWebユーザ
                                                の登録ができる
                        End if
                  End if
            `「Register」ボタンがクリックされた場合
            ¥(vsbRegister  #  "")
                  $0:= WWW Register
                  `「Information  」ボタンがクリックされた場合
            ¥(vsbInformation  #  "")
                  DIALOG([ユーザインタフェース] ; "WWW情報")
      End case
Until(Not(<>vbWebServicesOn)  |  $0)
```

このメソッドの機能は、以下の通りです。

■4D変数「vtNav_appName」、「vtNav_appVersion」、「vtNav_appCodeName」、「vtNav_userAgent」（同じ名前のHTMLオブジェクトにバインドされている）は、JavaScriptスクリプト「GetBrowserInformation」を使用してHTMLオブジェクトに割り当てられた値を戻します。単純、かつ直接的にメソッドは変数を文字列として初期化してから、Webページがサブミットされた後で値を戻します。

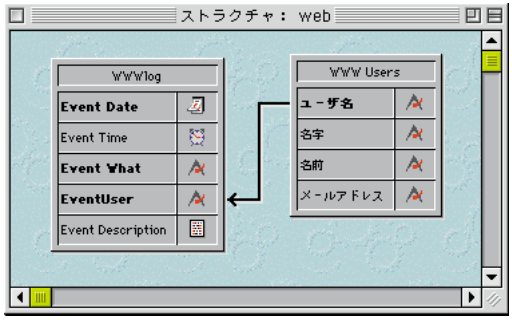
■ 4D変数「vsbLogOn」、「vsbRegister」、「vsbInformation」は、3つのサブミットボタンにバインドされます。これらの変数は、ページがブラウザに送られるたびにリセットされることに注意してください。これらのボタンのどれかでサブミットが実行されると、ブラウザはクリックされたボタンの値を4Dに返します。4D変数はそのたびにリセットされるため、空の文字列ではなくなった変数によって、どのボタンがクリックされたのかがわかります。それ以外の2つの変数は空の文字列であり、これはブラウザが空の文字列を返したためではなく、ブラウザがそれらの変数について何も「言わなかった」ためです。その結果、4Dは変数をそのままにしていたのです。このため、ページがブラウザに送信されるたびに、これらの変数を空の文字列でリセットする必要があるのです。

これが、複数のサブミットボタンがWebページに存在する時に、どのサブミットボタンがクリックされたのかを見分ける方法です。4Dフォーム内の4Dボタンは、数値変数です。ただし、HTMLでは、すべてのオブジェクトはテキストオブジェクトになります。

4D変数をSELECTオブジェクトでバインドする場合には、テキスト変数もバインドします。4Dでは、ドロップダウンリストのどの要素が選択されたのかをテストするには、4D配列の数値をテストします。HTMLでは、これはHTMLオブジェクトにバインドされた4D変数に返される選択された項目の値になります。

4D変数でどのオブジェクトをバインドするかに関係なく、返される値はテキストタイプであり、文字列またはテキストの4Dプロセス変数をバインドすることになります。

この例で注目すべき点は、ブラウザについての情報を取得した後、Webとデータベースの機能をもう一度組み合せることで、これらの値を4Dテーブルに格納できるということです。これが、(一覧されていない)「WWW POST EVENT」プロジェクトメソッドが実行する作業です。このメソッドは「イベントをポストする」わけではなく、Webセッション情報を次に示すテーブルに保存します。



情報をテーブルに保存した後、他のプロジェクトメソッドを使用してその情報をWebユーザに送り返すことができます。これを実行するには、単に**QUERY**コマンドを使用して適切な情報を検索し、**DISPLAY SELECTION**コマンドを使用して[WWW Log]レコードを表示します。以下の図は、Webサイトに登録されたユーザが利用できるログ情報を示しています。



この例で示すバインド機能を利用すると、HTMLダイアログや4Dフォームを使って提示する、またはユーザから収集した情報をすべて組み合わせ、データベースのWebサイトに非常に役立つ管理機能を追加することができます。

COMPILER_WEBプロジェクトメソッド

4D Webサーバはポストされたフォームを受け取ると、**COMPILER_WEB**という名前のプロジェクトメソッド（存在する場合）を自動的に呼び出します。このメソッドには、ポストされたフォーム内のフィールドと同じ名前を持つ全変数のタイプ指定や、変数の初期化指定が含まれていなければなりません。これはデータベースのコンパイル時に、コンパイラにより使用されます。

COMPILER_WEBメソッドはすべてのWebフォームに共通です。デフォルトでは**COMPILER_WEB**メソッドが存在していません。したがって、これを明示的に作成しなければなりません。

注：GET WEB FORM VARIABLESコマンドを使用することもできます。このコマンドは、サブミットされたHTMLページに含まれるすべての変数の値を取得します。

Webサービス：SOAPリクエストが受け入れられるたびに、COMPILER_WEBメソッドが呼びだされます（存在する場合）。Webサービスとして公開されるすべてのメソッドに対し、COMPILER_WEBメソッドを使用して、SOAP入力引数に関連付けた4D変数をすべて宣言しなければなりません。実際は、Webサービスメソッドでプロセス変数を使用する場合には、メソッド呼び出しの前にそれらの変数が宣言されている必要があります。この件に関する詳細は、SOAP DECLARATIONコマンドの説明を参照してください。

HTMLオブジェクトと4D変数とのバインド設定・イメージマッピング

「コンテキストモードの使用」の節で説明したように、Webページとして4Dフォームを使用すると、4Dは静止ピクチャに重なる非表示系のボタンを使って、サーバ側のイメージマッピングを提供します。

SEND HTML FILEまたは**SEND HTML BLOB**コマンドを使用してHTMLドキュメントを送信する場合には、4D変数をイメージマップのHTMLオブジェクト（INPUT TYPE="IMAGE"）でバインドし、情報を取り出すことができます。例えば、「bImageMap」という名前のイメージマップのHTMLオブジェクトを作成することができます(実際には任意の名前を付けることができます)。ブラウザ側にあるイメージをクリックするたびに、クリック位置でのサブミットが4D Webサーバに送り返されます。クリックの座標（イメージの左上端から相対的に表される）を取り出すには、4Dプロセス変数bImageMapと、bImageMap_X変数およびbImageMap_Y変数（倍長整数タイプ）をリードする必要があるだけです。これらの変数はクリックの縦方向の座標と横方向の座標を含んでいます。これらの変数はCOMPILER_WEBプロジェクトメソッドで定義する必要があります。

HTMLページでは、以下のようなコードを作成します。

```
<P><INPUT TYPE="image" SRC="MonImage.GIF" NAME="bImageMap"
                                                    BORDER=0></P>
```

HTMLページを送信する4Dメソッドは次の通りです。

```
SEND HTML FILE("ThisPage.HTM")
```

COMPILER_WEBプロジェクトメソッドにおいて、以下のようなコードを作成します。

```
C_LONGINT(bImageMap_X;bImageMap_Y)
bImageMap_X:=-1 `変数の初期化
bImageMap_Y:=-1 `変数の初期化
```

次に、POSTアクション4Dメソッドまたはカレントメソッドで、POSTアクションメソッドが**SEND HMTL FILE**("")呼び出しを行った後に、変数bImageMap_XとbImageMap_Y におけるクリックの座標を取り出します。

```
If(($bImageMap_X#-1)&($bImageMap_Y#-1))
      ` 座標に対応した処理を行う
End if
```

JavaScriptカプセル化

4Dは、HTMLドキュメントに埋め込まれたJavaScriptソースコードをサポートします。また、JavaScriptの「.js」ファイルのHTMLドキュメントの埋め込みをサポートします（例えば、<SCRIPT SRC="...">）。

標準モードで**SEND HTML FILE**または**SEND HTML BLOB**コマンドを使用すると、HTMLソースエディタで準備したページや、4Dを使用してプログラムで構築し、ディスク上に保存していたページを送信できます。どちらの場合でも、開発者がページを全面的に制御できます。FORMマークアップでスクリプトを使用するのと同様に、ドキュメントのHEADセクションにJavaScriptスクリプトを挿入できます。前の例では、開発者がフォームに名前を付けることができたため、スクリプトはフォーム"frm"を参照します。FORMマークアップレベルで、フォームのサブミッションのトリガ、受け付け、拒否もできます。

コンテキストモードにおいて、4DフォームにHTMLをカプセル化する場合には、HEADセクションやFORM宣言に対する制御は行えません。このため、スクリプトの有効範囲が異なります。例えば、HTMLフォームに対してはその名前でアクセスすることはできません。ただし、下記のものと上記の例とで、JavaScript関数「Is4DWebServer」を比べてみると、

```
function Is4DWebServer( ) {
        return(document.form[0].vs4D.value=="4D4D")
}
```

どちらの関数も同じことをしていますが、2番目の例はHTMLドキュメントオブジェクトのフォームプロパティを使用してforms[0]要素からオブジェクトにアクセスしています。結果として、4Dが変換されたHTMLページ（フォーム）に名前を設定してもしなくても、その名前を知らないときでも作動します。

注：4DはJavaアプレットのトランスポートをサポートしています。

参照

4D HTMLタグ、SEND HTML BLOB、SEND HTML FILE、URLとフォームアクション

URLとフォームアクション

4D Webサーバでは、各種URLとフォームアクションが提供されます。これによりコンテキストモードおよび非コンテキストモードの両方において、さまざまなアクションをデータベースに実装することができます。

これらのURLは以下の通りです。

■ 4DMETHOD/：コンテキストモードにおいて、任意のHTMLオブジェクトをデータベースのプロジェクトメソッドにリンクすることができます。

■ 4DACTION/：非コンテキストモードにおいて、任意のHTMLオブジェクトをデータベースのプロジェクトメソッドにリンクすることができます。

■ 4DCGI/：任意のHTMLオブジェクトから「**On Web Connection**」データベースメソッドを呼び出すことができます。

これに加えて、4D Webサーバは更にいくつかのURLを受け入れます。

■ /4DSTATS、/4DHTMLSTATS、/4DCACHECLEAR、/4DWEBTESTにより、4D Webサイトの動作状況に関する情報を取得することができます。

■ /4DWSDLを使用すると、サーバ上に公開されたWebサービスの定義ファイルにアクセスすることができます。詳細については、「Webサービス（サーバ）コマンド」の節、または『デザインリファレンス』マニュアルを参照してください。

URL 4D ACTION/

シンタックス：4DACTION/MyMethod{/Param}

モード：非コンテキストモード。コンテキストモードから呼び出されると、そのコンテキストのプロセスは中止され、非コンテキストモードに切り替わります。

使用方法：URLまたはフォームアクション

このURLを使用すると、コンテキストモードでHTMLオブジェクト（テキスト、ボタン等）を4Dプロジェクトメソッドにリンク付けることができます。リンクは、/4DACTION/メソッド名/引数と指定します。「メソッド名」は4Dプロジェクトメソッドの名前で、ユーザがそのリンクをクリックすると実行されます。

4Dが/4DACTION/MyMethod/Paramリクエストを受け取ると、「On Web Authentication」データベースメソッド（存在する場合）が呼び出されます。Trueが返された場合、メソッドMyMethodが実行され、オプションの/Param文字列は引数として$1に納められます（後述の「URLでコールした4Dメソッドに渡されたテキスト引数」の節を参照してください）。

スタティックなWebページにおいて、4DACTION/をURLに関連付けることができます。URLのシンタックスは、以下の形式でなければなりません。

```
<A HREF="/4DACTION/MyMeth/Param">処理を行う</A>
```

通常、プロジェクトメソッド MyMethは“応答”を返します（**SEND HTML FILE**や**SENT HTML BLOB**コマンドを使用してHTMLページを送信する等）。ブラウザをブロックしないように、必ずこの処理はできるだけ短時間で行ってください。

注：/4DACTIONで呼び出されたメソッドでは、インタフェースエレメント（DIALOG 、ALERT等）を呼び出してはいけません。

警告：「4DACTION/」URLを用いて4Dメソッドを実行できるように、「メソッド」プロパティで「4DACTION、4D METHODおよび4D SCRIPTで利用可能」オプション（デフォルトでは未選択）を指定しなくてはなりません。

例題

この例題は、「4DACTION/」URLをHTMLのピクチャオブジェクトに関連付け、ページ内のピクチャを動的に表示する方法について説明しています。以下の指示をスタティックなHTMLページに挿入します。

```
<IMG SRC="/4DACTION/PICTFROMLIB/1000">
```

PICTFROMLIBメソッドは以下の通りです。

```
C_TEXT($1)`常にこの引数は宣言されていなければなりません
C_PICTURE($PictVar)
C_BLOB($BlobVar)
C_LONGINT($Number)
`文字列$1からピクチャの番号を取得します
$Number:=Num(Substring($1;2;99))
GET PICTURE FROM LIBRARY($Number;$PictVar)
PICTURE TO GIF($PictVar;$BlobVar)
SEND HTML BLOB($BlobVar;"Pict/gif")
```

フォームをPOSTする4DACTION

“ポストされた”フォームを使用したい場合、4D Webサーバでは追加機能が提供されます。これらのフォームはスタティックなHTMLページであり、Webサーバにデータを送信します。これらのフォームには必ずPOSTタイプを関連付けなければならず、フォームのアクションは/4DACTION/メソッド名で始まることが必須条件です。

注：フォームは2種類のメソッドを使用してサブミットすることができます（ともに4Dで使用されます）。

POST：通常は、Webサーバにデータを追加するために使用されるーデータベースへ

GET：通常は、Webサーバへのリクエストに使用されるーデータベースから

この場合、WebサーバがPOSTされたフォームを受け取ると、COMPILER_WEBプロジェクトメソッド（存在する場合、下記参照）が呼び出され、この後に**On Web Authentication**データベースメソッド（存在する場合）が呼び出されます。メソッドよりTrueが返された場合、メソッド「メソッド名」が実行されます。4Dは、フォーム内に存在するHTMLフィールドを解析して値を取り出し、自動的にその内容を4D変数へ代入します。フォームのフィールドと4D変数は、同じ名前でなければなりません。

注：詳細については、「HTMLオブジェクトと4Dオブジェクトのバインド」の節を参照してください。

フォームで適用するHTML構文は、下記のタイプになります。

■ フォーム内のアクションを定義するには

```
<FORM ACTION="/4DACTION/メソッド名" METHOD=POST>
```

■ フォーム内のフィールドを定義するには

```
<INPUT TYPE=フィールドタイプ NAME=フィールド名 VALUE="デフォルト値">
```

フォームの各フィールドに対し、4Dはフィールド値を同じ名前を持つ変数の値にセットします。フォームのオプション（例えばチェックボックス）に対しては、4Dは関連する変数を、選択されていれば1に、そうでなければ0にセットします。

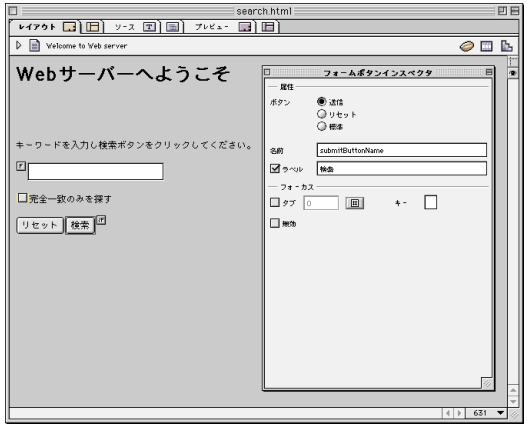
数値データの入力に関し、4Dはフィールドの値を文字から実数へと変換します。

注：フォームのフィールドにOKという名前が付けられている場合（例えばサブミットボタン）、フィールドの値が空でなければ、システム変数OKに1が代入されます。それ以外の場合は0が代入されます。

例題

非コンテキストモードで開始され、使用されている4D Webデータベースにおいて、ブラウザではスタティックなHTMLページを使用したレコードの検索を行いたいとします。このページを“search.htm”とします。さらに、このデータベースには他にもスタティックページがあり、例えば、検索結果を表示することができます（“results.htm”）。このページには、POSTタイプとともに/4DACTION/PROCESSFORMアクションが関連付けられています。

下図は、Adobe GoLiveのHTMLエディタに表示されるページです。



このページに対応するHTMLコードを以下に示します。

```
<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST>
<INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR>
<!-- 通常は、VALUEにボタン名を入れますが、インタプリタ上の理由により、
                                        VALUEには番号を入れなければなりません-->
<INPUT TYPE=CHECKBOX NAME=EXACT VALUE="1">完全に一致する語句を
                                                検索する<BR>
<!-- OK は特殊なケースです-->
<INPUT TYPE=SUBMIT NAME=OK VALUE="検索">
</FORM>
```

データ入力中に、データ入力エリアに“ABCD”と入力して、オプションをチェックし、検索ボタンをクリックして確定します。

次に、4Dは以下のようなCOMPILER_WEBプロジェクトメソッドを呼び出します。

```
C_TEXT(VNAME)
VNAME:=""
C_LONGINT(vEXACT)
vEXACT:=0
OK:=0 `特殊なケース
```

この例では、VNAMEには文字列“ABCD”が入り、vEXACTは1となり、OKは1になります（ボタン名がOKであるためです）。

4Dは「**On Web Authentication**」データベースメソッド（存在する場合）を呼び出した後、以下に示す**PROCESSFORM**プロジェクトメソッドを呼び出します。

```
If(OK=1)
        If(vEXACT=0) `オプションが選択されなかった場合
```

```
            vNAME:=VNAME+"@"
        End if
        QUERY([Jockeys];[Jockeys]Name=vNAME)
        vLIST:=Char(1) `リストをHTMLで返す
        FIRST RECORD([Jockeys])
        While(Not(End selection([Jockeys])))
            vLIST:=vLIST+[Jockeys]Name+" "+[Jockeys]Tel+ŧ<BR>ŧ
            NEXT RECORD([Jockeys])
        End while
        SEND HTML FILE("results.htm") `リストをresults.htmフォームに送信する
        `このフォームには変数vLISTへの参照が含まれる
                                                （つまり、 <!--#4DVAR vLIST-->）
        ...
End if
```

URL 4DMETHOD/

シンタックス：4DMETHOD/MyMethod{/Param}

モード：コンテキストモード。非コンテキストモードから呼び出されると、コンテキストモードに切り替わります。

使用方法：URLまたはフォームアクション

このURLを使用すると、コンテキストモードでHTMLオブジェクト（テキスト、ボタン等）を4Dプロジェクトメソッドにリンク付けることができます。リンクは、/4DMETHOD/メソッド名/引数と指定します。「メソッド名」は4Dプロジェクトメソッドの名前で、そのHTMLオブジェクトがクリックされると実行されます。また、オプションのテキスト引数である＜引数＞が$1に納められメソッドへと渡されます（後述の「URLでコールした4Dメソッドに渡されたテキスト引数」の節を参照してください）。リンクされたアイテムは、そのURLを介して4Dプロジェクトメソッドの実行を開始します。プロジェクトメソッドでは、4Dフォームや他のHTMLページ等を表示することができます。

4Dが/4DMETHODリクエストを受け取ると、「**On Web Authentication**」データベースメソッド（存在する場合）が呼び出されます。このメソッドから“True（真）”が返されると、「**On Web Connection**」データベースメソッド（存在する場合）が呼び出され、この後に文字列/Paramを引数として使用して（$1に代入される）「メソッド名」メソッドが実行されます。

/4DMETHOD/メソッド名をフォームアクションとしてHTMLスタティックページに割り当てた場合、このメソッドはフォームの「サブミット」ボタンがクリックされた時に実行されます。4D側でこのHTMLフォームをサブミットするには、フォームのサブミット後に4Dで実行されるPOSTアクション4Dメソッドを指定する必要があります。**SEND HTML FILE**コマンドの例題を参照してください。

HTMLページを4Dに統合する際に、最も多く使用するのは「ノーマル」と「サブミット」タイプのボタンです。

フォームで適用するHTMLのシンタックスは、次のタイプのものです。

```
<FORM ACTION="/4DMETHOD/メソッド名" METHOD=POST>
```

ポストされたフォームに関する詳細は、前述した節を参照してください。

警告：「4DMETHOD/」URLを使用して4Dメソッドを実行できるように、「メソッド」プロパティで「4DACTION、4D METHODおよび4D SCRIPTで利用可能」オプション（デフォルトでは未選択）を指定しなくてはなりません。この件に関する詳細は、「接続セキュリティ」の節を参照してください。

URL 4DCGI/<アクション>

シンタックス：4DCGI/<アクション>

モード：両モード

使用方法：URL

4D Webサーバが/4DCGI/<アクション>のURLを受け取ると、「**On Web Authentication**」データベースメソッド（存在する場合）が呼び出されます。このメソッドから“True（真）”が返されると、Webサーバは“現状のまま”このURLを$1に送信して「**On Web Connection**」データベースメソッドを呼び出します。

4DCGI/のURLは、いずれのファイルにも対応しません。その役割は4Dを呼び出すことです。このURLの役割は、「**On Web Connection**」データベースメソッドを使用して4Dを呼び出すことです。引数<アクション>には、あらゆるタイプの情報を納めることができます

このURLを使用して、あらゆるタイプのアクションを実行することができます。必要となる作業は、「**On Web Connection**」データベースメソッドまたはそのサブメソッドのひとつにおいて$1の値を検証し、4D に適切な動作を実行させるだけです。例えば、完全にカスタムなHTMLページを構築して、レコードの追加や検索、ソートを行ったり、GIFイメージをオンザフライで作成することができます。このURLの使用方法に関する例題は、**PICTURE TO GIF**コマンドおよび**SEND HTTP REDIRECT**コマンドの説明を参照してください。

アクションを発行する際は、データを送信するコマンド（**SEND HTML FILE**、**SEND HTML BLOB**等）を使用して、必ず“応答”を返さなければなりません。

警告：ブラウザを停止させないように、可能な限り短い動作を実行することを心掛けてください。

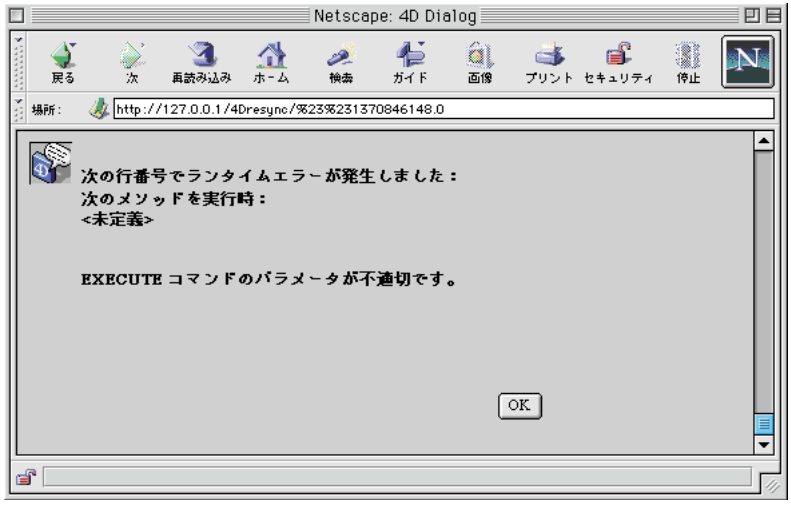
URLでコールした4Dメソッドに渡されたテキスト引数

4th Dimensionは、URLでコールされた任意の4Dメソッドに対してテキスト引数を送信します（4DMETHOD/, 4DACTION/...）。コンテキストでも非コンテキストモードでも、このテキスト引数に関する説明は以下の通りです。

- この引数を使用しない場合でも、「**C_TEXT**（$1）」というコマンドを使用して明示的に宣言しなければなりません。これを行わないと、Webからコンパイルモードで稼働しているデータベースへアクセスする際に、ランタイムエラーが発生します。
- この引数$1により、URLの終わりの部分にあるエクストラデータが返されます。このエキストラデータは、HTML環境から4D環境へ値を受け渡す際のプレースホルダとして利用することができます。

コンパイルモードにおけるランタイムエラー

以下の例題を見てみましょう。リンクを使用して、HTMLオブジェクトにバインドしたメソッドを実行し、Webブラウザ上に以下の画面を表示します。



このランタイムエラーは、4Dメソッドで$1テキスト引数が宣言されていないことに関連しています。このメソッドは、それを参照しているHTMLリンクがクリックされた際に呼び出されます。カレントHTMLページから実行されているため、このエラーは、実際にそのページをWebブラウザに送信したメソッドの行番号“0”を参照します。

次は、「Webサーバ入門」の例題です。の例題です。メソッドM_ADD_RECORDSおよびM_LIST_RECORDS内で$1テキスト引数を明示的に宣言し、問題を解消します。

```
`「M_ADD_RECORDS」プロジェクトメソッド
C_TEXT($1) ` この引数は必ず明示的に宣言する
Repeat
      ADD RECORD([Customers])
Until(OK=0)

`「M_LIST_RECORDS」プロジェクトメソッド
C_TEXT($1) ` この引数は必ず明示的に宣言する
ALL RECORDS([Customers])
MODIFY SELECTION([Customers])
```

上記の変更を行った後は、コンパイルモードでのランタイムエラーは発生しなくなります。

4Dメソッド内で明示的に宣言する引数

4Dメソッドを呼び出す起源と性質によって違った引数を宣言しなければなりません。

On Web Authenticationデータベースメソッドと**On Web Connection**データベースメソッド接続には6つの引数を宣言しなければなりません。

```
` On Web Connection データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
```

（4DMETHOD/）URLより呼び出されたメソッド引数$1を宣言しなければなりません。

```
` （4DMETHOD/）URLより呼び出されたメソッド
C_TEXT($1)
```

（4DACTION/）タグよりURLとして呼び出されたメソッドは、引数$1を宣言しなければなりません。

```
` 4DACTION/ URLで呼び出されたメソッド
C_TEXT($1)
```

ドキュメント中のHTMLコメントとして（4DACTION/）タグより呼び出されたメソッドは$0に値を返します。$0と$1引数を宣言しなければなりません。

```
` HTMLコメントとして（4DACTION/）タグより呼び出されたメソッド
C_TEXT($0; $1)
```

参照

HTMLオブジェクトと4Dオブジェクトのバインド、GET WEB FORM VARIABLES、コンテキストモードの使用、Webサービス：入門編

4D HTMLタグ

4D Webサーバでは、4D特定のさまざまなHTMLタグが提供されます。これらのタグにより、例えば**SEND HTML FILE**コマンドや**SEND HTML BLOB**コマンドを使用してWebサーバから送信されるスタティックなHTMLページに4D変数や4D表記、または各種処理への参照を挿入することができます。これらのページは、セミダイナミックページと呼ばれます。

使用できる4D HTMLタグは以下の通りです。

■ 4DVAR：4D変数や4D表記を挿入します。

■ 4DHTMLVAR：4DVARと似ていますが、HTMLコードを挿入します。

■ 4DSCRIPT：4Dメソッドを実行します。

■ 4DINCLUDE：ページを他のページに挿入します。

■ 4DIF、4DELSE、4DENDIF：HTMLコードに条件式を挿入します。

■ 4DLOOPと4DENDLOOP：HTMLコードにループを作成します。

互換性に関する注意：バージョン6.0.xの4Dでは、スタティックページへの4D変数挿入のための表記として、角括弧[VarName]が使用されていました。変換後のデータベースにおいて、標準のHTML構文（<!--#4DVAR MAVAR-->）を使用可能にするには、「環境設定」ダイアログボックスの「ブラケットの代りに4DVARコメントを使用する」オプションがチェックされていることを確認してください（「Webサーバセッティング」の節を参照）。

4D HTMLタグについて

4Dにより送られたセミダイナミックページの内容の解析は、**SEND HTML FILE**コマンド（.htm、.html、.shtm、.shtml）または**SEND HTML BLOB**コマンド（テキストまたはhtmlタイプのBLOB）のコール時、およびURLを使用して呼び出されたページの送信時に実行されます。

しかし、非コンテキストモードでは最適化のため、“.HTML”や“.HTM”で終わるHTMLページは解析されません。この場合、“強制的に”HTMLページの解析を行うには、“.shtm”や“.shtml”という接尾辞を必ず付加してください（例：http://www.server.com/dir/page.shtm）。

このタイプのページの使用例は、**WEB CACHE STATISTICS**コマンドの解説に示されています。

Webブラウザに送信したHTMLページ内にあるタグを4Dが解析するケースを以下に示します。

送信条件	送信ページの内容分析	
	コンテキストモード	非コンテキストモード
・ページの拡張子（一般的なケース）：		
.htm、.html、.shtm、.shtml（HTMLページ）	X	X
.xml、.xsl（XMLページ）	X	X
.wml（WMLページ）	X	X
・URL経由で呼び出されたページ	X	X
		拡張子が.htmまたは.htmlであるページを除く
・SEND HTML FILEコマンド呼び出し	X	X
・SEND HTML BLOBコマンド呼び出し（BLOBが“text/html”タイプの場合）	X	X
・<!--#4DINCLUDE-->タグによる包含	X	X
・{mypage.htm}タグによる包含	X	-

4Dが処理を行うには、HTMLコメントが<!--#4D...-->という形式でなくてはなりません。HTMLエディタの中には、コメントに他の情報を自動追加するものもあるため、注意が必要です。この場合、正しく解釈されなくなる可能性があります。

しかし<!--#リストの始まり-->のような、その他のHTMLコメントは問題ありません。

<!--#4D... というコメントが -->で終わっていない場合、“<!--#4D... : -->が必要です”というメッセージが挿入され、この段階で解析が中断されます（エラーを示すためにそのページが送信されます）。

複数タイプのコメントを混在させることができます。例えば、次のようなHTML構文を作成できます。

```
<HTML>
...
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->                (Methodの呼び出し)
<!--#4DIF(myvar=1)-->                       (If 条件)
        <!--#4DINCLUDE banner1.html-->      (サブページの挿入)
<!--#ENDIF-->(End if)
<!--#4DIF(myvar=2)-->
        <!--#4DINCLUDE banner2.html-->
<!--#ENDIF-->
<!--#4DLOOP [TABLE]-->                      (カレントセレクションをループ)
<!--#4DIF([TABLE]ValNum>10)-->              (If [TABLE]ValNum>10)
        <!--#4DINCLUDE subpage.html-->      (サブページの挿入)
<!--#4DELSE-->                              (Else)
        <B>Value: <!--#4DVAR [TABLE]ValNum--></B><BR>
                                            (フィールドの表示)
<!--#ENDLOOP-->(End for)
</BODY>
</HTML>
```

4DVAR

シンタックス：<!--#4DVAR VarName-->

<!--#4DVAR VarName-->タグを使用して、変数、配列要素、フィールドへの参照をHTMLページの任意の場所に挿入することができます。例えば、以下のように記述すると

<P>Welcome to <!--#4D VAR vtSiteName-->!</P>

4D変数であるvtSiteNameの値がHTMLページに挿入されます。

最初の桁がASCIIコードの1である場合（つまり、vtHTML:=Char(1)+"...HTMLコード..."）、4Dテキスト変数をHTMLコードに挿入することができます。また、4DHTMLVARタグを使用することもできます。

さらに、4DVARタグを使い、（変数だけではなく）4D言語表現を4D HTMLコメントに挿入することもできます。フィールド内容や（例：<!--#4DVAR [テーブル名]フィールド名-->）、配列項目の内容（例：<!--#4DVAR 配列{1}-->）を直接挿入することができます。この表記の変換には、変数の時と同じルールが使用されます。さらに、表記は4Dの構文ルールに従わなければなりません。

表記には4D関数のダイレクト呼び出しを含めることができますが、ローカライズの観点からすると、これはお勧めできません。例えば、<!--#4DVAR Current date-->の場合、英語版の4Dでは正しく解釈されますが、フランス語版では理解されません。同様の問題が実数に関しても存在します（言語によって、小数点の位置が異なります）。両ケースとも変数への割り当てはプログラムから行うよう強くお勧めします。

インタープリタ上のエラーが発生すると、挿入されたテキストは“<!-4DVAR myvar--> : ##エラー#エラーコード”と表示されます。

注：

・プロセス変数を使用した作業を行えます。

・ピクチャフィールドの内容を表示できます。さらに（コンテキストモードのみ）、ピクチャ変数の内容を表示することもできます。両モードとも、ピクチャ配列項目の内容は表示できません。

・HTMLは言語処理指向のアプリケーションなので、通常はテキスト変数を使用して作業を行います。しかし、BLOB変数を使用することも可能で（これにより、テキストタイプの変数に関する32,000バイトの制限を回避できます）、その方法は長さ属性なしテキスト（Text without length）モードでBLOBを生成するだけです。

・4DVARの使用例は、「HTMLオブジェクトと4Dオブジェクトのバインド」の節に示されています。

互換性に関する注意：バージョン6.0.xの4Dでは、スタティックページへの4D変数挿入のための表記として、角括弧 [VarName]が使用されていました。変換後のデータベースにおいて、標準のHTML構文（<!--#4DVAR MAVAR-->）を使用可能にするには、「環境設定」ダイアログボックスの「ブラケットの代りに4DVARコメントを使用する」オプションがチェックされていることを確認してください（「Webサーバセッティング」の節を参照）。

4DHTMLVAR

シンタックス：<!--#4DHTMLVAR VarName-->

このタグを使用して、変数や4D表記の評価、およびこれをHTML表記としてページ内に挿入することができます。実際、VarNameがASCIIコードの1で始まる場合、このタグは<!--#4DVAR VarName-->タグとまったく同じように作用します（前述の説明を参照）。

例として、使用可能なタグを用いた4Dテキスト変数myvarの挿入結果を以下に示します。

myvarの値	タグ	Web Pageへの挿入
myvar:="<B>"	<!--#4DVAR myvar-->	<B>
myvar:=Char(1)+"<B>"	<!--#4DVAR myvar-->	<B>
myvar:="<B>"	<!--#4DHTMLVAR myvar-->	<B>

インタプリタ上のエラーが発生すると、挿入されたテキストは“<!--#4DHTMLVAR myvar--> : ##エラー#エラーコード”と表示されます。

注：テキスト変数は、ISO Latin-1文字セットを使用して表わしてください（詳細は、Mac to ISO関数の説明を参照してください）。

4DSCRIPT/

シンタックス：<!--#4DSCRIPT/MethodName/MyParam-->

スタティックなHTMLページを送信する際に4DSCRIPTタグを使用して、4Dメソッドを実行することができます。<!--#4DSCRIPT/MyMethod/MyParam-->タグがHTMLコメントとしてスタティックページに存在すると、引数MyParamが文字列として$1にセットされて、メソッドMyMethodが強制的に実行されます。ホームページのロード時に、4Dは**On Web Authentication**データベースメソッド（存在する場合）を呼び出します。データベースメソッドよりTrueが返されると、4Dはメソッドを実行します。メソッドからは、$0にテキストが返されます。返された文字列がASCIIコードの1で始まる場合、HTMLであるとみなされます（変数に関しても同じ原則が適用されます）。

注：4DSCRIPTタグを用いたメソッドの実行は、メソッドプロパティで指定された「4DACTIONで利用可能」オプションの値に影響されません。

SEND HTML FILE（.htm、.html、.shtm、.shtml）または**SEND HTML BLOB**（テキスト／HTMLタイプのBLOB）が呼び出されると、ページコンテンツの解析が行われます。非コンテキストモードにおいては、URLの指すファイルの拡張子が“.shtm”または“.shtml”のいずれかである場合にも解析が行われる点に注意してください（例：http://www.server.com/dir/page.shtm）。

注：コンテキストモードでは、メソッドはコンテキスト内で実行されます。

例えば、“Today is <!--# 4DSCRIPT/MYMETH/MYPARAM-->”というコメントをスタティックページに挿入するとします。4Dはページをロードする時に、**On Web Authentication**データベースメソッド（存在する場合）を呼び出し、次にメソッドMYMETHを呼び出して、文字列“/MYPARAM”を引数$1として渡します。

このメソッドは$0 にテキストを返し（例：“12/31/99”）、したがって“Today is <!--# 4DSCRIPT/MYMETH/MYPARAM-->”という表記は、“Today is 12/31/99”になります。

メソッドMYMETHは次の通りです。

```
C_TEXT($0) `この引数は常に宣言しなければならない
C_TEXT($1) `この引数は常に宣言しなければならない
$0:=String(Current date)
```

警告：呼び出されるメソッドにおいて、引数$0 および$1は常に宣言しなければなりません。

注：4DSCRIPTによって呼び出されるメソッドでは、インタフェースエレメント（DIALOG、ALERT等）を呼び出してはいけません。

4Dはメソッドを出現順に実行するため、使用するモードに関わらず、数々の変数がドキュメント内で更に参照されている場合でも、これら変数の値をセットするメソッドを呼び出すことができます。

注：<!--#4DSCRIPT...-->コメントは、スタティックページにいくつでも挿入することができます。

注：以前のバージョンの4Dでは、同じタグである4DACTIONは、URL（例：http://myserver/4DACTION/meth）やスタティックページ内のHTMLコメント（<!--#4DACTION/meth-->）として使用できました。このようにすると間違いやすいため、バージョン6.7 の4Dでは、4DACTIONに代わり4DSCRIPTというタグが提供されます。このタグは、HTMLコメント（<!--#4DSCRIPT/meth-->）としてのみ使用され、<!--#4DACTION/meth-->と同じ結果になります。4DACTIONタグは、URLに対してのみ使用します。

4DINCLUDE

シンタックス：<!--#4DINCLUDE パス-->

このコメントを使用して、HTMLページに別のHTMLページ（引数＜パス＞で指定）のボディを挿入できます。HTMLページのボディは<BODY>タグと</BODY>タグの間に納められます（タグそのものは含まれません）。

<!--#4DINCLUDE -->コメントは、判定式（<!--#4DIF-->）やループ（<!--# 4DLOOP-->）を使用する際に大変役立ちます。条件に応じて、またはランダムにタグを含める場合、非常に便利です。

含める際に、モードやファイル名の拡張子に関わらず、4Dは呼び出されたページを解析し、その内容（変更された、またはされていない）を4DINCLUDE呼び出しを行なったページ内に挿入します。

<!--#4DINCLUDE -->コメントを使用して含められるページは、URLを介して呼び出されたページや**SEND HTML FILE**コマンドで送信されたページと同様に、Webサーバのキャッシュ内へロードされます。

＜パス＞には、含めようとするドキュメントへのパスを指定します。このパスは解析されるドキュメントへの相対位置です。フォルダ区切りにはスラッシュ記号（/）、1階層上がる（HTML構文）場合にはドット2つ（..）を使用してください。

1ページ内に使用できる<!--#4DINCLUDE パス-->の数には制限がありません。

しかし、<!--#4DINCLUDE パス-->の呼び出しは、1つのレベルでしか行うことができません。例えば、mydoc2.htmlがmydoc1.htmlに挿入されたタグ<!--#4DINCLUDE mydoc2-->で呼び出されている場合、mydoc2.htmlのボディページで<!--#4DINCLUDE mydoc3.html-->を挿入できないということです。

さらに、4Dはその包含式が再帰的ではないかを検証します。

エラーが発生すると、挿入されるテキストは“<!--#4DINCLUDE パス--> :ドキュメントは開かれませんでした”となります。

注：コンテキストモードにおいて、スタティックテキストエリアに挿入した{mypage.html}タグを介してページがフォームに挿入されると、4DINCLUDEコメント（存在する場合）は無視されます。

例題

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

4DIF、4DELSE、4DENDIF

シンタックス：<!--#4DIF 判定式--> <!--#4DELSE--> <!--#4DENDIF-->

<!--#4DIF 判定式-->コメントを、<!--#4DELSE-->（オプション）および<!--#4DENDIF-->コメントとともに使用すると、条件付きでHTMLコードを実行できるようになります。

引数＜判定式＞には、ブール値を返す有効な任意の4D表記を納めます。4D表記は括弧内に記載し、4Dの構文ルールに従わなければなりません。

<!--#4DIF 判定式-->から<!--#4DENDIF-->までのブロックは、複数のレベルで入れ子状態にして指定できます。4Dと同様に、それぞれの<!--#4DIF 判定式-->は<!--#4DENDIF-->と一対になっていなければなりません。

インタープリタ上のエラーが発生すると、<!--#4DIF-->と<!--#4DENDIF-->の間に入れられる内容の代わりに、“<!--#4DIF 判定式-->:ブール式が必要です”というテキストが挿入されます。

同様に、<!--#4DIF-->と<!--#4DENDIF-->の数が合わない場合、<!--#4DIF -->と<!--#4DENDIF-->の間に入れられる内容の代わりに、“<!--#4DIF 判定式-->:4DENDIF が必要です”というテキストが挿入されます。

例題

この例題コードは、スタティックなHTMLページに挿入されており、“vname#""”という判定式の結果に応じて異なるラベルを表示します。

```
<BODY>
...
<!--#4DIF(vname#"")-->
<!--#4DVAR vname-->で始まる名前
<!--#4DELSE-->
名前が見つかりません。
<!--#4DENDIF-->
...
</BODY>
```

4DLOOPと4DENDLOOP

シンタックス：<!--#4DLOOP 条件式--> <!--#4DENDLOOP-->

このコメントを使用して、条件を満たすかぎりHTMLコードの一部を繰り返すことができます。繰り返される部分は、<!--#4DLOOP-->と<!--#4DENDLOOP-->で指定されます。

<!--#4DLOOP 条件式-->から<!--#4DENDLOOP-->までのブロックは、入れ子状態にして指定できます。4Dと同様に、それぞれの<!--#4DLOOP 条件式-->は<!--#4DENDLOOP-->と一対になっていなければなりません。

条件式には3通りあります。

■ <!--#4DLOOP [テーブル]-->

このシンタックスでは、カレントプロセス内のテーブルのカレントセレクションの各レコードをループします。2つのコメントの間に挟まれたHTMLコード部分が、カレントセレクションのレコード毎に繰り返されます。

注：4DLOOPタグをテーブルと共に使用すると、レコードはリードオンリーモードでロードされます。

以下のHTMLコードは、

```
<!--#4DLOOP [People]-->
<!--#4DVAR [People]Name--> <!--#4DVAR [People]Surname--><BR>
```

```
<!--#4DENDLOOP-->
... 4D言語で記述すると以下のようになります。
FIRST RECORD([People])
While(Not(End selection([People])))
    ...
    NEXT RECORD([People])
End while
```

■ <!--#4DLOOP 配列-->

このシンタックスでは、各配列項目をループします。配列のカレントアイテム番号はHTMLコード部分が繰り返されるたびに増加します。

注：このシンタックスは、2次元配列には使用できません。2次元配列の場合には、ネストしたループをメソッドに組み合わせるとよいでしょう。

以下のHTMLコードの例は、

```
<!--#4DLOOP arr_names-->
<!--#4DVAR arr_names{arr_names}--><BR>
<!--#4DENDLOOP-->
```

... 4D言語で記述すると以下のようになります。

```
For($Elem;1;Size of array(arr_names))
    arr_names:=$Elem
    ...
End for
```

■ <!--#4DLOOP method-->

このシンタックスは、メソッドが“True（真）”を返す限りループを行います。メソッドは倍長整数タイプの引数を使用します。まず初期化できるように（必要な場合）値0を使用してメソッドが呼び出され、次に値1、そして2、3というように、メソッドが“True”を返す限り呼び出されます。

セキュリティ上の理由から、この初期化ステージ（引数として0を使用してメソッドを実行）の直前に、**On Web Authentication**データベースメソッドを1度呼び出すことができます。認証がOKであれば、初期化ステージが進められます。

注：4DLOOPタグを用いたメソッドの実行は、メソッドプロパティで指定された「4DACTIONで利用可能」オプションの値に影響されません。

警告：コンパイルする場合は、C_BOOLEAN($0)およびC_LONGINT($1)をメソッド内で必ず宣言してください。

例題

以下のHTMLコードの例は、

```
<!--#4DLOOP my_method-->
<!--#4DVAR var--> <BR>
<!--#4DENDLOOP-->
```

... 4D言語で記述すると以下のようになります。

```
If(AuthenticationWebOK)
    If(my_method(0))
        $counter:=1
        While(my_method($counter))
            ...
            $counter:=$counter+1
        End while
    End if
End if
```

メソッドmy_methodは次の通りです。

```
C_LONGINT($1)
C_BOOLEAN($0)
If($1=0)
    `初期化
    $0:=True
Else
    If($1<50)
        ...
        var:= ...
        $0:=True
    Else
        $0:=False `ループ中止
    End if
End if
```

インタープリタ上のエラーが発生すると、<!--#4DLOOP -->と<!--#4DENDLOOP-->の間に入れられる内容の代わりに、“<!--#4DLOOP 条件式-->:説明”というテキストが挿入されます。

説明として、以下のメッセージが表示されます。

■ 予期しない式のタイプです（標準エラー）。

■ テーブル名が正しくありません（テーブル名に関するエラー）。

■ 配列が必要です（変数が配列ではない、または2次元配列である）。

■ メソッドが存在しません。

■ シンタックスエラー（メソッドの実行中）。

■ アクセスエラー（テーブルやメソッドへアクセスするための十分なアクセス権がない）。

■ 4DENDLOOPが必要です（<!--#4DENDLOOP-->と<!--#4DLOOP-->の数が一致しない）。

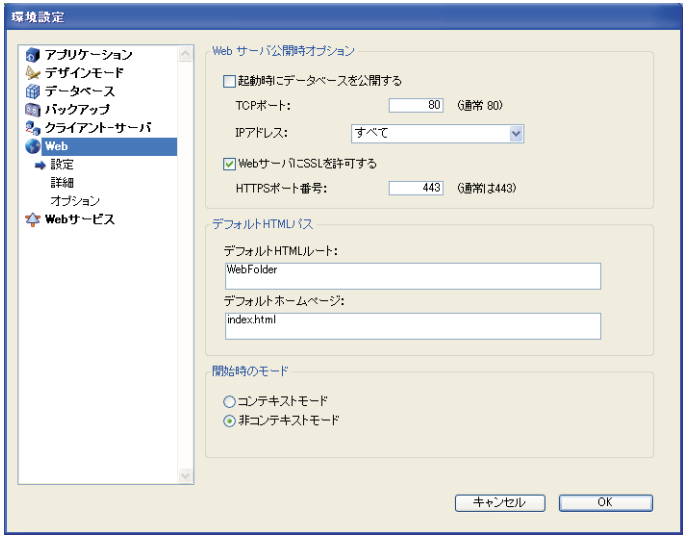
参照

HTMLオブジェクトと4Dオブジェクトのバインド、URLとフォームアクション、コンテキストモードの使用

Webサーバセッテイング

データベースの「環境設定」の「Web」テーマで指定したパラメータを使用して、4D Webサーバの運用設定を行うことができます。この節では、「Web」テーマの「設定」、「詳細」、「オプション」ページ、ならびに「アプリケーション/互換性」ページの「Web互換性」セクションの各パラメータについて説明します。

設定ページ



TCPポート番号

デフォルトでは、4Dは通常のWeb TCPポートにWebデータベースを公開します。このポートはポート80です。ポート80が他のWebサービスで既に使用されている場合には、データベース用に4Dが使用するTCPポートを変更する必要があります。TCPポートを変更すると、そのマシンのルートユーザでなくてもMac OS X上で4D Webサーバを開始することができます（「Webサーバのシステム設定と接続管理」の節を参照）。変更するには、「TCPポート」入力エリアに移動し、適切な値（同一のマシン上で稼働している別のTCP／IPサービスで使用されていないTCPポート番号）を指定します。

注：0を指定すると、4DはデフォルトのTCPポート番号80を使用します。

Webブラウザでは、デフォルト以外のTCPポート番号の場合、この番号をWebデータベースへの接続用に入力するアドレスに指定する必要があります。そのアドレスにはコロンとポート番号で構成される接尾辞を指定します。例えば、TCPポート番号8080を使用している場合には、「192.168.0.99:8080」と指定します。

---

*警告：デフォルトの番号（標準モードでは80、SLLモードでは443）以外のTCPポート番号を使用する場合は、同時に使用する予定の他のサービス用のデフォルトのポート番号は使用しないように注意してください。例えば、Webサーバマシンで*FTP*プロトコルも使用する予定の場合には（その影響がわからない限り）、*TCP*ポートの20と21は使用しないでください。これらのポートは*FTP*プロトコルのデフォルトポートです。標準的な*TCP*ポート番号の割り当て方法を調べるには、*4D Internet Commands*ドキュメントの「付録B：*TCP*ポート番号」の節を参照してください。256未満のポート番号は、既知のサービス用に予約されており、256から1024までのポート番号は*UNIX*プラットフォームから提供される特定のサービス用に予約されています。最大限の安全対策として、ポート番号には2000番台や3000番台など、上記の値を超える番号を指定してください。*

HTMLリクエストのIPアドレスを定義する

WebサーバがHTTPリクエストを受信するIPアドレスを定義することができます。

デフォルトで、サーバはすべてのIPアドレス（あらゆるIPアドレスオプション）に応答します。

ドロップダウンリストには、そのマシン上で利用できるすべてのIPアドレスが自動的に一覧表示されます。特定のアドレスを選択すると、サーバはこのアドレスに送信されるリクエストにのみ応答します

この設定は、複数のTCP/IPアドレスでマシンに置かれる4D Webサーバのためにあります。例えば、ほとんどのインターネットホストプロバイダの場合です。

マルチホーミングしている、そのようなシステムを実装することは、Webサーバマシンの上で特定の構成を必要とします。

セカンダリIPアドレスのインストール

マルチホーミングシステムの導入には、OSに応じた特別な設定が必要になります。

Macintosh上の設定

▼ Macintosh上でマルチホーミングの設定をするには

1. この機能を使用するには、バージョン1.3以降のOpen Transportを使用する。
2. 「コントロールパネル」の「TCP/IP」を開く。
3. 「設定方法：」のポップアップメニューから「手入力」を選択する。
4. テキストファイルを作成し、"IP Secondary Addresses"という名前を付け、システムフォルダ内の「初期設定」フォルダに入れる。

IP Secondary Addressesの各行にシステムで使用するセカンダリIPアドレス用のIPアドレス、サブネットマスクおよびルータアドレスを記述します。

WindowsNT、Windows2000上の設定

▼ WindowsNTまたはWindows2000上でマルチホーミングの設定をするには

1. 次のコマンドシーケンスを選択する。

■ Windows NT：スタートメニュー＞設定＞コントロールパネル＞ネットワーク＞プロトコルタブ＞TCP/IPプロトコル＞プロパティボタン＞詳細ボタン

■ Windows 2000：スタートメニュー>設定>ネットワークとダイヤルアップ接続>ローカルエリア接続>プロパティボタン > インターネットプロトコル（TCP/IP）>プロパティボタン>詳細設定...ボタン
「TCP/IP詳細設定」ダイアログが表示されます。

■ Windows XP：スタートメニュー > コントロールパネル>ネットワークとインターネット接続>ネットワーク接続>ローカルエリア接続（プロパティ）>インターネットプロトコル（TCP/IP）>プロパティボタン > 詳細設定...ボタン
「TCP/IP詳細設定」ダイアログが表示されます。

2. 「IPアドレス」エリアの「追加」ボタンをクリックし、追加するIPアドレスを入力する。

   最高で5種類のIPアドレスを指定することができます。この作業にはネットワーク管理者のサポートが必要になることがあります。より詳しい情報は、Windowsドキュメントをご覧ください。

WebサーバにＳＳＬを許可する

Webサーバで暗号化モードでの接続を許可するかどうかを表わします。このオプションについては、「SSLプロトコルの使用」の節で説明しています。

起動時にデータベースを公開する

4Dアプリケーションの起動時に、Webサーバを開始するかどうかを表わします。このオプションについては、「Webサービス：システム設定」の節で説明しています。

「Webパスワード」エリア

パスワードを使用して、Webサイトのアクセス保護に関する設定を行います。このオプションについては、「接続セキュリティ」の節で説明しています。

開始時のモード

Webサーバが開始するモードを指定することができます。このオプションについては、「コンテキストモードの使用」の節で説明しています。

一時的なコンテキストを再利用する（4D Clientでのみ表示される）

前回のWebリクエストの処理で作成されたWebプロセスを再利用することにより、4D ClientのWebサーバ操作を最適化することができます。実際のところ、4D ClientのWebサーバには、各Webリクエスト処理のために特定のWebプロセスが必要になります。つまり、必要に応じてこのプロセスが4D Serverマシンに接続し、データとデータベースエンジンにアクセスします。この後、独自の変数やセレクション等を使用して一時的なコンテキストが生成されます。リクエストの処理が終わると、このプロセスは終了します。

「一時的なコンテキストを再利用する」オプションが選択されると、4Dは4D Client上で作成された特定のWebプロセスを保持しておき、次回のリクエストでこのプロセスを再利用します。プロセス作成段階が省かれることにより、Webサーバのパフォーマンスが向上します。

その代わりこの場合は、誤った結果を取得しないように、4Dメソッドを使用して意識的に各変数を初期化しなくてはなりません。同様に、前回のリクエスト中に定義されたカレントセレクションやレコードを消去する必要があります。

HTTPSポート番号

SSL（HTTPSプロトコル）を用いた暗号化HTTP接続に対しWebサーバが使用するTCP/IPポート番号を変更することができます。デフォルトとして、HTTPSポート番号は443（標準値）に設定されています。

主に次の2つの理由から、このポート番号の変更を検討する場合があります。

- セキュリティ上の理由：ハッカーによるWebサーバへの攻撃は、おおむね標準のTCPポート（80と 443）に集中しています。
- Mac OS X上で、“一般”ユーザが暗号化モードでWebサーバを起動することを許可するため。Mac OS Xにおいて、Web公開用に予約されているTCP/IPポート（0から1023）を使用するには、特定のアクセス権が必要になります。つまり、ルートユーザだけが、これらのポートを使用してアプリケーションの起動を行えます。一般ユーザがWebサーバを開始できるようにするためのソリューションの一つが、TCP/IPポート番号の変更です（「Webサーバ設定と接続管理」の節を参照）。

任意の有効な値を渡すことができます（Mac OS X上でのアクセス制限を回避するため、1023より大きな値を渡してください）。TCPポート番号に関する詳細は、前述の「TCPポート番号」の節を参照してください。

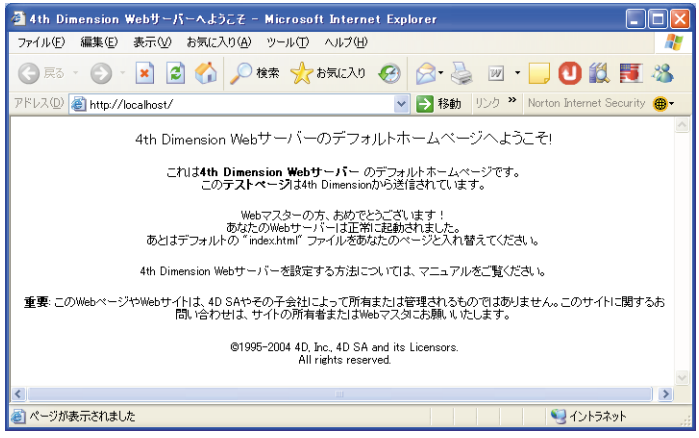
デフォルトHTMLルート

Webサイトの各ファイルのデフォルト位置を定義し、ディスク上のこれより上位の階層にあるファイルへはアクセスできないことを示します。このオプションについては、「接続セキュリティ」の節で説明しています。

デフォルトのホームページを定義する

そのWebセッションに対して指定されたモード（コンテキストまたは非コンテキスト）に関係なく、データベースに接続するすべてのブラウザに対してデフォルトのホームページを定義することができます。

デフォルトでは、Webサーバの初回起動時に、4Dは“index.html”という名前のホームページを作成し、それをHTMLルートフォルダに納めます。この設定を変更しない場合、Webサーバに接続している任意のブラウザは、次のようなページを取得します。



デフォルトホームページを変更するには、データベースのルートフォルダ内のホームページを独自の“index.html”ページで置き換えるか、あるいは「デフォルトホームページ」入力エリアに定義したいページの相対アクセスパスを入力します。

アクセスパスは、デフォルトのHTMLルートフォルダに対して相対的に設定しなくてはなりません。

ユーザのデータベースのマルチプラットフォームの互換性を確実にするために、4D Webサーバは、アクセスパスを定義するのに特定の表記法を使用します。

■ フォルダは"/"で区切る

■ アクセスパスの最後は"/"で終了しない

■ 上の階層のフォルダを定義する場合は、"..（2つのピリオド）"をフォルダの前に入力する

■ アクセスパスは、"/"で始めない

例えば、デフォルトホームページを"Web"フォルダ"（それ自体がそのデータベースのデフォルトHTMLルートフォルダ内に配置されている）の中の"MyHome.htm"にしたい場合、"Web/MyHome.htm"と入力します。

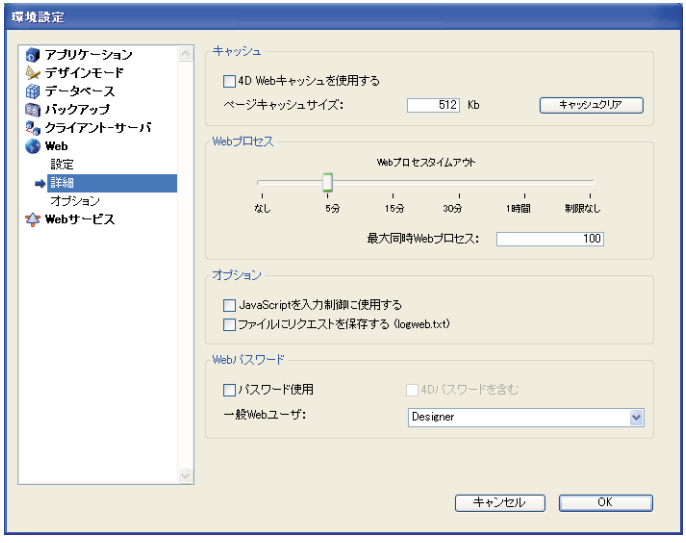
注：Webプロセスごとに、デフォルトのホームページを設定するには、「SET HOME PAGE」を使用することもできます。

デフォルトのカスタムホームページを指定しない場合、Webサーバの動作は開始時のモードによって異なります。

■ Webサーバがコンテキストモード（デフォルトで）で開始する場合、4Dの前バージョンの場合のように、カレントのメニューバー（デフォルトで、メニューバー#1）が送られます。

■ Webサーバが非コンテキストモード（標準モード）上で開始する場合、**On Web Connection**データベースメソッドがコールされます。メソッドを用いてリクエストを処理するかどうかはユーザ次第です。

詳細ページ



スタティックページのキャッシュ

4D Webサーバのキャッシュを使用し、リクエストに応じてスタティックページやGIFイメージ、JPEG画像（<128KB）およびスタイルシート（.cssファイル）をメモリにロードすることができます。スタティックなページを送る時、キャッシュを使用すると、かなりWebサーバパフォーマンスを向上させます。

キャッシュは、すべてのWebプロセスで共有されます。キャッシュのサイズは、「環境設定」で設定することができます。デフォルトとして、スタティックページにはキャッシュを使用できません（サイズは0です）。キャッシュを使用可能にするには、「4D Webキャッシュを使用する」オプションをチェックしてください。

キャッシュのサイズは、「ページキャッシュサイズ」エリアで変更することができます。セットした値はホストマシンの処理能力だけでなく、ユーザのWebサイトのスタティックなページの数とサイズに依存します。

注：ユーザのWebデータベースを使用する間、ユーザはWEB CACHE STATISTICSルーチンを使用することによってキャッシュのパフォーマンスをチェックすることができます。例えば、ユーザが使用するキャッシュの率が約100%であると気がついた場合、それに割り当てられたサイズを増やすことを考えることがあります。4DSTATS、そして、4DHTMLSTATS URLは、さらにキャッシュの状態に関する情報を得ることができます。詳しくは、「Webサイトに関する情報」を参照してください。

一度キャッシュが割込み可能であると、4D Webサーバはキャッシュの中で最初のブラウザによって要求されるページを探します。それがページを見つけた場合、すぐにそれを送ります。そうでない場合、4Dはディスクからのページをロードして、キャッシュでそれを設定します。キャッシュがいっぱいで、追加のスペースが必要な場合、最も少なく要求されたものの中で、4Dは最も古いページを「アンロードします」。

キャッシュのクリア

いつでも、ユーザはページのキャッシュとそれが含むイメージをクリアすることができます（例えば、ユーザがスタティックなページを変更して、キャッシュでそれを再ロードしたい場合）。そのために、「キャッシュクリア」ボタンをクリックしなければなりません。キャッシュは、その時すぐにクリアされます。

Webプロセスタイムアウト

Web接続プロセスのタイムアウトを指定することができます（コンテキストモードのみ）。このオプションについては、「コンテキストモードの使用」の節で説明しています。

Webプロセスの最大数を定義する

このオプションを使用して、サーバ上で同時に開始できる任意のタイプ（コンテキストモード、非コンテキストモード、またはプロセスの“再利用”に属するタイプ）の「最大同時Webプロセス」の上限数を正確に指定することができます。このパラメータを使用すると、リクエストが多すぎたり、コンテキスト作成要求が多すぎるために、4D Serverが限界に達してしまう危険性を回避することができます。

デフォルトでは、この値は32,000ですが、10から32,000までの値を自由に設定できます。

同時Webプロセスの最大数（マイナス1）に達すると、4Dは新しくプロセスを作成しなくなり、新しい各リクエストに対して「サーバは使用できません」（ステータスHTTP 503 - Service Unavailable）というメッセージを送信します。

適切な値の決定方法は？

理論上、Webプロセスの最大数は次の計算式の結果になります。

使用可能メモリ／Webプロセスのスタックサイズ

別の解決策は、ランタイムエクスプローラに表示されるWebプロセス情報を示す方法です。つまり現在のWebプロセス数およびWebサーバの開始以降に達した最大数が示されている情報です。

Webプロセスの再利用

Webプロセスの“再利用”により、非コンテキストモードにおけるWebサーバの反応度を上げることができます。この保存分のサイズは、再利用されるプロセスの最小数（デフォルトでは0）および最大数（デフォルトでは10）で決まります。これらのプロセスは、**SET DATABASE PARAMETER**コマンドを使用して変更することができます。Webプロセスの最大数が一度変更されると、この最大数が“再利用”の上限数より小さい場合、上限数はWebプロセスの最大数にまで減らされます。また、Webプロセスの最大数は、**SET DATABASE PARAMETER**コマンドを使用して定義することもできます。

Javascriptを入力制御に使用する

このオプションをチェックした場合、コンテキストモードにおけるデータ入力制御の一部は、自動Javascriptを用いてブラウザ側で行われます。

ブラウザ側において、データ入力制御およびデータに適用可能なデータタイプ（フィールドや変数）は次の通りです。

■ 最小値（数値）

■ 最大値（数値）

■ 必須入力

生成されるJavascriptのサイズは小さいものの、警告ダイアログボックスを表示しユーザのデータ入力受け入れを妨げることはありません（それはまだ4Dの責任です）。実際、データ入力エリアに不正な値が入力された場合には、ユーザがボタン（OK、Cancel等）をクリックすると、ブラウザに警告メッセージが表示されます。



この警告ダイアログボックスを確定すると、ユーザがボタンを再度クリックした場合には、そのボタンの動作が考慮されます。

「ユーザ」モードならびに「カスタム」モードにおいて、データ入力制御はすべてWebサーバ側で行われます。

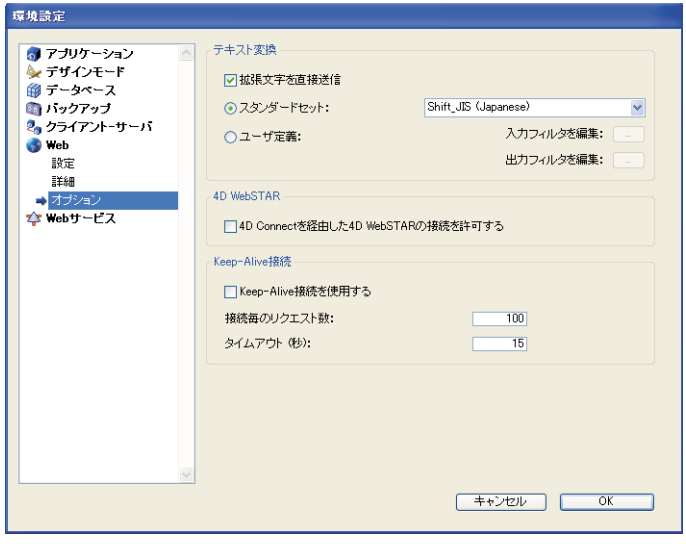
ファイルにリクエストを保存する（logweb.txt）

このオプションにより、Webサーバに送信したリクエストのログをCLFテキストファイル形式で生成することができます。このオプションについては、「Webサイトに関する情報」の節で説明しています。

パスワードエリア

パスワードを用いたWebサイトアクセス制限を設定します。このオプションについては「接続セキュリティ」の節で説明しています。

オプションページ



拡張ＡＳCII文字を直接送る

デフォルトとして、4D Webサーバは、ダイナミックWebページやスタティックWebページを送信する前に、HTML規格に従ってページ内にある拡張ASCII文字を変換します。この後、これらの文字はブラウザにより翻訳されます。

拡張ASCII文字をHTMLエンティティに変換せずに、“現状のまま”送信されるようにWebサーバを設定することができます。このオプションにより、外国語OS（特に日本のOS）上での処理速度が向上します。

これを行うには、「拡張文字を直接送信」オプションを選択します。

文字セット

「スタンダードセット」ドロップダウンリストを使用し、4D Webサーバで使用する一連の文字セットを指定することができます。

また、入力データと出力データ双方に対してASCII文字変換テーブル（Webフィルタ）を変更すると、カスタマイズした文字セットを指定することもできます。

これを行うには、「ユーザ定義」ラジオボタンを選択します。このパラメータは、“x-user-defined”文字セットを選択した場合に相当します。

すると、「入力フィルタを編集」および「出力フィルタを編集」項目に関連するボタンが有効になります。入力フィルタはブラウザから4D Webサーバへ送信される文字を解釈し、出力フィルタは4D Webサーバからブラウザへ送信される文字を解釈します。

ユーザが変更したいフィルタと一致するボタンをクリックします。

「入力フィルタ編集」は4D Webサーバにブラウザによって送られる文字を翻訳し、「出力フィルタ編集」はブラウザに4D Webサーバで送る文字を翻訳します。

以下のダイアログボックスが、表示されます。



スクロールエリアでは、フィルタしたいマックキャラクタを探して、クリックします。「ASCII Code」入力エリアでは、文字の新しいASCIIコードを入力します。フィルタしたいすべての文字のために、この操作を繰り返します。

フィルタを保存するには「保存...」ボタンをクリックします。「読み込み...」ボタンを使用し、続けてフィルタをロードすることができます。

Webの入力フィルタや出力フィルタを有効にするには、「テーブル更新」ボタンをクリックします。

4D Connectを経由した4D WebSTARの接続を許可する

このオプションは、4D WebSTARプラグインの4D Webサーバへの接続を許可（チェック）、または禁止（チェックなし）する目的のために設定されています。

4D Connectは、4D WebSTARのWebサーバのためのプラグインで、4DのWebサーバとの通信を行います。

セキュリティ上の理由から、デフォルトでは「4D Connectを経由した4D WebSTARの接続を許可する」オプションが選択されていません。お使いのWebの環境設定に応じ、4D社では以下のような設定をお勧めします。

■ お使いの4D Webサーバが、4D WebSTARプラグインを使用して4D WebSTARサーバに接続していない場合、このオプションはチェックされていない状態のままにしてください。

■ お使いの4D Webサーバが4D WebSTARプラグインを使用して4D WebSTARサーバに接続している場合、接続が正常に行われるようこのオプションをチェックしてください。

この設定の場合、4D Webサーバをファイアーウォールの下で稼働させ、そのファイアーウォールを利用して4Dへのリクエストをフィルタリングすることをお勧めします。

Keep-Alive接続

4th DimensionのWebサーバでは、Keep-Alive接続（持続的接続）を利用することができます。「keep alive接続」オプションを使用すると、1度開かれたTCP接続を維持したままWebブラウザとサーバの間で一連のやり取りを行えるため、システム資源の節約や転送の最適化を図ることができます。

「keep alive接続を使用する」オプションにより、Webサーバに対して持続的なTCP接続を有効または無効に設定することができます。デフォルトとして、このオプションは有効に設定されています。持続的接続を用いるとデータのやり取りが迅速になるため、たいていの場合は、このオプションを選択しておくことをお勧めします。Webブラウザが持続的接続をサポートしていない場合、4D Webサーバは自動的にHTTP/1.0へ切り替わります。

4th Dimension Webサーバのkeep-alive機能は、コンテキストモードおよび非コンテキストモードにおいて、あらゆるTCP/IP接続（HTTP、HTTPS）に影響します。ただし、持続的接続はすべての4D Webプロセスに対して使用されるとは限らない点に注意してください。場合によっては、他の最適化された内部機能が呼び出されることがあります。Keep-Alive接続は、主にスタティックページで有効です。

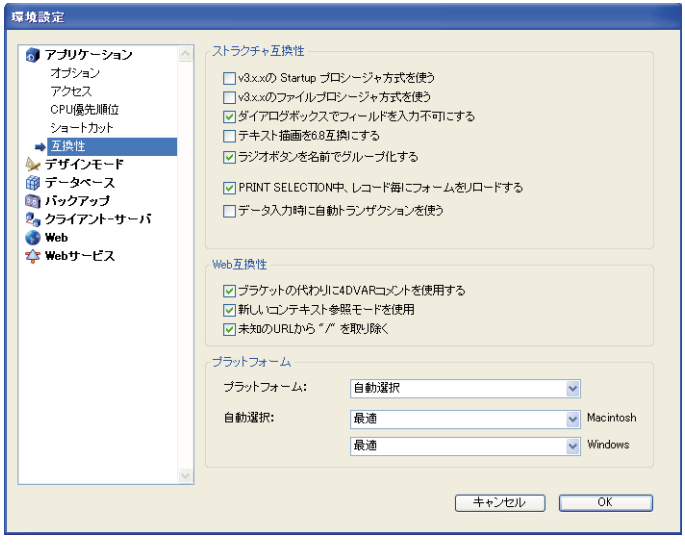
2つのオプションを使用して、Keep-Alive接続の動作を設定することができます。

■ **接続毎のリクエスト数：**持続的接続においてやり取りを行えるリクエストと応答の最大数を設定することができます。接続ごとのリクエスト数を制限することにより、多数のリクエスト受信が原因となるサーバの混雑（フラッディング：ハッカーが利用するテクニック）を避けることができます。
4D Webサーバをホスティングするマシン資源（リソース）に応じて、デフォルト値（100）を増やしたり、または減らすことができます。

■ **タイムアウト：**この値により、最長待機時間（秒単位）を設定します。この間、WebサーバはWebブラウザからのリクエストを受信しない場合でも、開かれたTCP接続を持続します。この時間に達すると、サーバはその接続を閉鎖します。
接続閉鎖後にWebブラウザがリクエストを送信した場合、新しいTCP接続が自動的に作成されます。この処理はユーザの目に触れることはありません。

Web互換性

「環境設定」の「アプリケーション」テーマにある「互換性」ページには、Webサーバの動作を調整できるオプションが含まれています。



ブラケットの代りに4DVARコメントを使用する

このオプションを使用して、スタティックページ上で4D変数を挿入する際に使用するコメントを定義することができます。

■ このオプションを選択した場合（デフォルト値）、使用されるシンタックスは標準のHTMLコメントです。
<！--4DVAR 変数名-->

（4DVARと変数名の間にスペースキャラクタを挿入しなくてはなりません）。

■ オプションを選択しない場合、使用されるシンタックスは角カッコによるコメントです（[MYVAR]）。これは以前のバージョンの4D Webサーバで使用された独自のソリューションです。

新しいコンテキスト参照モードを使用

このオプションを選択すると（デフォルト値）、コンテキストモードにおいてWebサーバは、ブラウザに送信したドキュメントの基本のURLに現在のコンテキスト番号を設定します。

以前のシステムでは（オプションは未選択）、4D Webサーバはページの各要素をブラウザへ送信するたびにコンテキスト番号を送信します。この方法では処理速度が低下してしまいます。

互換性上の理由から、このオプションの選択を解除することができます。ただし、このオプションを変更した後は、新しい設定を有効にするためにデータベースを再起動しなくてはならない点に留意してください。

未知のURLから“/”を取り除く

以前のバージョンの4th Dimensionでは、「On Web Authentication」および「On Web Connection」データベースメソッドにおいて、“/”文字で始まらない未知のURL（既存のページや4D特定のURLのいずれにも該当しないURL）が引数$1に返されていました。4th Dimension 2004より、この特殊なケースが削除されています。

ただし、この特例に基づいたメカニズムを設定しており、以前の動作を維持したい場合は、「未知のURLから“/”を取り除く」オプションをチェックすることができます。デフォルトとして、変換後のデータベースに対してこのオプションが選択され、新規データベースに対しては選択されていません。

参照

接続セキュリティ、SET DATABASE PARAMETER、 SET HOME PAGE、コンテキストモードの使用

Webサイトに関する情報

4DではWebサイトに関する情報を得ることができます。

■ 特定のURL（/4DSTATS、/4DHTMLSTATS、/4DCACHECLEAR、/4DWEBTEST）を使ってサイトの管理ができます。

■ すべてのリクエストのログを生成することができます。

■ ランタイムエクスプローラーウインドウのウォッチページ中でWebサーバに関する情報を取得することができます。

Webサーバ管理URL

4D Webサーバは、4つの特定のURL（/4DSTATS、4DHTMLSTATS、/4DCACHECLEAR、/4DWEBTEST）を受け入れます。

/4DSTATS、4DHTMLSTATS、/4DCACHECLEARのこれらのURLは、4Dパスワードシステムが設定されている場合はデザイナーおよび管理者のみ使用できます。もし、4Dパスワードシステムが設定されていない場合は、すべてのユーザが使用可能になります。

/4DWEBTESTは常に使用可能です。

/4DSTATS

/4DSTATSのURLは、純粋なテキストフォームで下記の情報を返します。

■ ヒット数（低レベル接続）

■ 作成されたコンテキストの数

■ 作成されなかったコンテキストの数

■ パスワードエラーとなった数

■ キャッシュ内に保存されているページの数

■ キャッシュの使用率（%）

■ スタティックホームページのキャッシュ内に、保存されてるページおよびJPEGまたはGIFファイルのリスト（*）

(*) スタティックホームページおよびピクチャに関するより詳しい情報は、「Webサーバセッティング」を参照してください。

この情報は、サーバ機能をチェックし、段階的に各パラメータを最適化することを可能にします。

注：WEB CACHE STATISTICSコマンドは、キャッシュがスタティックホームページにどのように使用されているのかに関する情報を得られるようにします。

/4DHTMLSTATS

/4DHTMLSTATSのURLも、/4DSTATSのURLと同じ情報を純粋なテキストフォームで返します。違いは、最後のフィールドでキャッシュ内に存在するHTMLページのリストだけが返されることです（キャッシュされているJPEGとGIFファイルのリストは含まない）。

/4DCACHECLEAR

/4DCACHECLEARのURLは、スタティックホームページとイメージのキャッシュを即座にクリアします。したがって、変更されたページを「強制的」に更新させることができます。

/4DWEBTEST

/4DWEBTESTのURLはWebサーバのステータスをチェックするように設計されています。このURLが呼び出されると、4Dは次のHTTPフィールドとともにテキストファイルを返します。

Date: current date at the RFC 822 format

例："Date: Wed, 26 Jan 2000 13:12:50 GMT"

Server: 4D WebStar_D/internal version number

例："4D WebStar_D/7.0"

User-Agent: name and version @ IP client address

例："Mozilla/4.08(Macintosh; I; PPC, Nav) @ 192.193.00.00"

接続ログファイル

4Dでは、リクエストのログを取ることができます。ログは、ストラクチャファイルと同じ階層に、"weblog.txt"ファイルとして自動的に作成されます。このファイルは、ほとんどのWebサイト分析ツールで認識できる、CLF（Common Log File）フォーマットまたはNCSAフォーマットになります。

“weblog.txt”ファイルは次の場所へ自動的に配置されます。

- 4th Dimensionおよび4D Serverでは、データベースストラクチャファイルと同じ階層。
- 4D Clientでは、アプリケーションの「.exe」ファイル（Windows）またはソフトウェアパッケージ（Mac OS）と同じ階層。

ファイルの各行は、以下のようなリクエストを表わします。

host rfc931 user[DD/MMM/YYYY:HH:MM:SS] "request" state length

各フィールドは、スペースで分離され、各行はCR/LF（文字コード13/文字コード10）で終わります。

■ host：クライアントのIPアドレス（例192.100.100.10）

■ rfc931：4Dでは生成しない情報で、常に「-（マイナス記号）」です。

■ user：認証されているユーザ名または「-（マイナス記号）」。ユーザ名にスペースがあると「_（下線）」に置き換えられます。

■ DD：日付、MMM：月の名前の3文字の略号（Jan、Feb、...）、YYYY：年HH :時間、MM：分、SS：秒

日付と時間はサーバマシン上の値です。

■ request：クライアントから来たリクエスト（例えば、GET/index.htmHTTP/1.0）

■ state：サーバからの返答

■ length：返答データのサイズ（HTTPヘッダを除く）または0

注：性能上の理由から、ディスクに書き込まれる前にサイズ1KBのパケットとしてメモリ上に保存され、5秒間リクエストが発生しなければディスクに書き込まれます。

stateとして取り得る値は下記の通りです。

200:OK
204:No contents
302:Redirection
304:not modfied
400:Incorrect request
401:Authentication required
404:Not found
500:Internal error

リクエストログで生成される行の例

■ 192.100.100.10 - - [25/Jan/1998:12:54:06] "GET /index.htm" 200 6524
アドレスが192.100.100.10のWebクライアントが認証されなかった。ページ"index.htm"が要求され、送信した（6,524バイト）。

■ 192.100.101.25 - - [25/Jan/1998:12:54:09] "GET /123456.htm" 404 125
アドレスが192.100.101.25のWebクライアントが認証されなかった。ページ"123456.htm"を要求されたが見つけられなかった（4Dは125バイトのメッセージを送りました）。

■ 192.100.101.31 - - [25/Jan/1998:12:54:10] "GET /secret.htm" 401 0
アドレスが192.100.101.31のWebクライアントが認証されなかった。ページ"secret.htm"を要求され、サーバは認証要求をした。

■ 192.100.101.31 - ZZZZ [25/Jan/1998:12:54:11] "GET /secret.htm" 401 0
アドレスが192.100.101.31のWebクライアントが"ZZZZ"として認証された。ページ"secret.htm"を要求され、ユーザ名が不明である。

■ 192.100.101.31 - 4D [25/Jan/1998:12:54:12] "GET /secret.htm" 200 2543
アドレスが192.100.101.31のWebクライアントが"4D"として認証された。ページ"secret.htm"を要求され、送信した（2,543バイト）。

警告：ログファイルはスプレッドシートまたは直接4Dへ読み込み可能です。しかし、データを読み込む前に必ずWebサーバを停止させなければなりません。

デフォルトではリクエストのログファイルは生成されません。すべてのWebリクエストのログファイルの作成を要求するには、データベースの「環境設定」で「Web」テーマの「設定」ページにある「ファイルにリクエストを保存する（logweb.txt）」オプションを選択しなくてはなりません。

ランタイムエクスプローラの情報

ランタイムエクスプローラの「ウォッチ」ページ（“情報”見出し）には、Webサーバに関連する3種類の情報が表示されます。

■ **Webキャッシュ使用率：**Webキャッシュ内にあるページ数とともにその使用率を示します。この情報は、Webサーバがアクティブで、かつキャッシュサイズが0より大きい場合にのみ表示されます。

■ **Webサーバ経過時間：**Webサーバの継続使用時間（時:分:秒形式）を示します。この情報は、Webサーバがアクティブである場合にのみ表示されます。

■ **Webヒット数：**Webサーバの開始以降に受信したHTTPリクエストの総数、および1秒毎の瞬間リクエスト数（2回のランタイムエクスプローラ更新の間で計測）を示します。この情報は、Webサーバがアクティブである場合にのみ表示されます。

注：ランタイムエクスプローラに関する詳細は、『4Dデザインリファレンスマニュアル』を参照してください。

参照

WEB CACHE STATISTICS、Webサーバセッティング

コンテキストモードの使用

4D Webサーバは、非コンテキストモード（標準モード）とコンテキストモードという2種類のモードで動作することができます。この節では、これら2つのモードを説明し、更にコンテキストモードの特性について詳しく述べています。

警告：コンテキストモードは4th Dimensionならびに4D Serverでのみ使用することができます。4D ClientのWebサーバではこのモードがサポートされていません。

注：「Webサーバ入門」の節には、コンテキストモードでデータベースを公開するための例題が提供されています。

非コンテキストモードとコンテキストモード

バージョン2003の4th Dimensionより、デフォルトとして4D Webサーバでは非コンテキストモード（非接続モード）が使用されます。このモードの4D Webサーバの動作は、標準のWebサーバの動作と変わりません。つまり、ブラウザからのHTTPリクエストを受信すると（URL、ポストされたフォーム等）、サーバがリクエストを処理し、必要に応じて応答を返します（例えば、Webページを送信する）。この後、サーバとブラウザの間には特定の接続が維持されません。

非コンテキストモードにおいて、Webサーバはスタティックページまたはセミダイナミックページを送信することができます。セミダイナミックページを使用すると、特別な4Dタグを用いてデータベースのデータへのアクセスやあらゆるタイプの処理の実行が可能になります。この4Dタグはページを送信した時に評価されます。セミダイナミックページを使用することにより、コンテンツのすべて、またはその一部が4Dによる処理を元にしたWebページの作成、管理、送信を行うことができます。通常は、非コンテキストモードによりWebサイト開発における大半の要求に応えることができます。

コンテキストモードでWebブラウザへ接続するとコンテキストが作成され、このコンテキスト内にカレントセレクションや変数等が配置されます。ある意味では、各ブラウザは「カスタム」モードでデータベースに接続している4D Clientとして扱われます。コンテキストは特定のWeb接続プロセスによって処理されます。

このモードを使用すると、Webページを作成しなくても、簡単に4DデータベースをWeb上に公開することができます。つまり、4Dはデータベースのメニューバーやフォームを自動的にHTMLへ変換し、これを元に作成されたダイナミックページを管理してブラウザ側へ送信します。コンテキストモードでも、セミダイナミックページやスタティックページを送信することができます。また、Web上で表示されるページに機能を追加するために、HTMLコードやJavascriptを4Dフォームに挿入することも可能です。

さらに、このモードで4Dはデータへの同時アクセスを自動的に処理します。ブラウザまたは4D Clientマシンがレコードをロードすると、それがブラウザや他の4D Clientマシンであっても、4Dは他のユーザに対してこのレコードを透過的にロックします。また、4Dでは、4th Dimensionや4D Clientと同じ方法で、トランザクション中にWebブラウザを使用したデータ入力を行うことができます。このシステムにより、4D Webサーバ側でブラウザの動作を制御したり、データの整合性を保証できるようになります。

公開が容易である代わりに、コンテキストモードにはいくつかの制約があります。

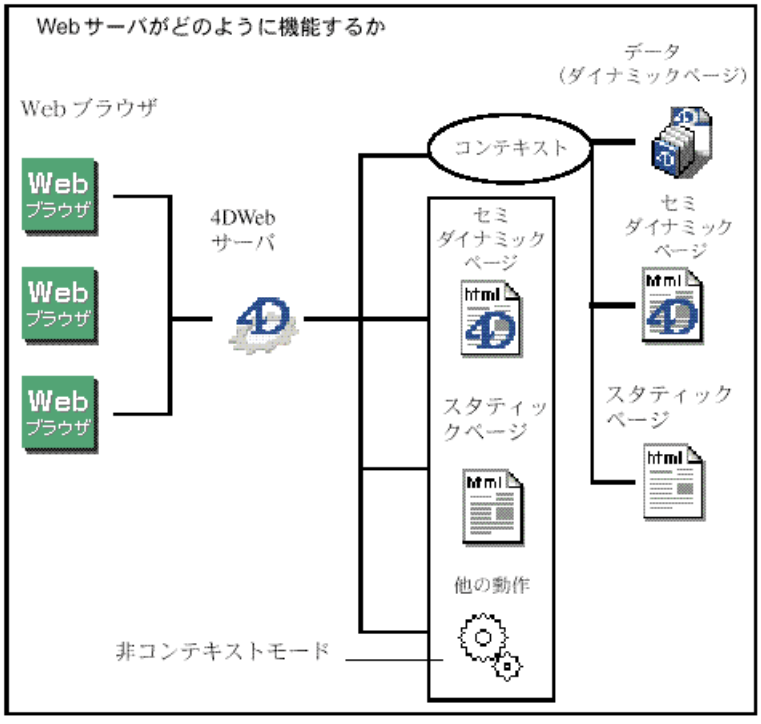
■ Webブラウザでは、あるWebページから別のページへ、あるサイトから別のサイトへと“サーフィン”することができます。クライアント／サーバタイプのデータベースを使用する場合には、データベーストランザクションのロジックを守るため、このナビゲーションを制御しなくてはなりません。ユーザが実行したレコードへの各入力は、不確定な状態のままにならないよう、必ず確定するかキャンセルしなくてはなりません。

4D Webサーバエンジンには、データベースセッションとコンテキストを管理する自動メカニズムが導入されています。これらのメカニズムにより、一部の標準的なブラウザ機能（再読み込み、戻る等。次の節を参照）が使用できなくなります。

■ コンテキスト管理を行う接続プロセスは、データベースの「環境設定」で指定したブラウザのタイムアウトに達するまで有効になります。例えば、ブラウザ側で一時的にそのサイトから抜けると、そのコンテキストは“無駄”になります。

これらの制約が意味するのは、コンテキストモードはどちらかと言えば、イントラネットや特定のインターネットアプリケーションのフレームワーク内での使用を目的としているということです。

4D Webサーバの機能方法を要約すると次の図のようになります。



モードの選択

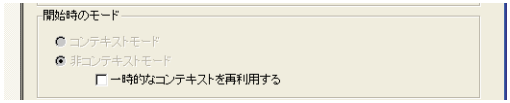
4D Webサーバの動作モードの選択は、次のように行われます。

■ サーバの開始時に、データベースの「環境設定」の「開始時のモード」オプションを使用する。

■ Webサーバの使用中に、送信されたURLや実行されたコマンドに従う。
実際には、いくつかの特別なURLや特定の4Dコマンドを使用して、モードを変更することができます。原則として、URLや4Dコマンドによりモードが変更されるまでの間は、現在のモードが使用されます。

■ 開始時にコンテキストモードを指定する。
デフォルトでは、Webサーバは非コンテキストモードで開始します。しかし、Webサーバを直接コンテキストモードで開始することができます。つまり、ユーザがデータベースに接続する際に、コンテキストが自動的に生成されます。

開始時に非コンテキストモードを指定するには、データベースの「環境設定」ダイアログで「Web」テーマの「設定」ページにある「コンテキストモード（永続的コンテキスト）」オプションを選択します。



■ モード変更を行うコマンドとURL
データベースの操作中に、次の要素を呼び出してモードを変更することができます。

■ 非コンテキストモードへの変更
SEND HTML BLOB：オプションの引数<非コンテキスト>に“True”を渡す。

SEND HTML TEXT：オプションの引数<非コンテキスト>に“True”を渡す。

SEND HTTP REDIRECT

/4DACTIONで始まるURL

■ コンテキストモードへの変更
/4DMETHOD/MyMethodで始まるURL

URL“/4DMETHOD/MyMethod”が送信されると、4D Webサーバは新しいコンテキストを作成し、次の処理を実行します。

- ■「Web Authentication」データベースメソッドを実行します（存在する場合）
- ■「On Web Connection」データベースメソッドを実行します（存在する場合）。この場合、 $1には“/（スラッシュ）”ではなく“/4DMETHOD/MyMethod”が代入されます。
- ■ 最後に、新しく作成したコンテキスト内でリクエストされたメソッドを実行します。

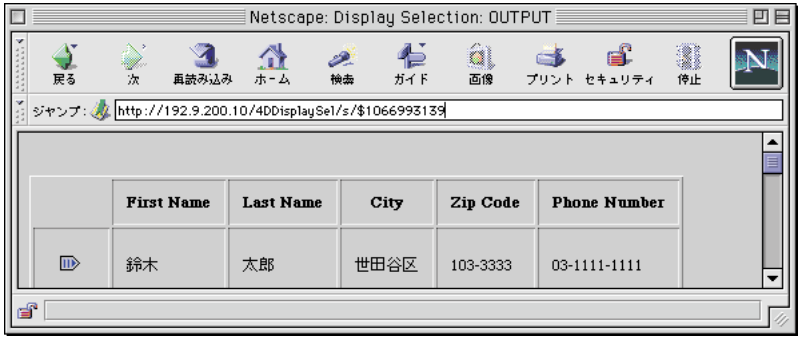
生成されたコンテキストの数を調べる

実行する動作によって、Webコンテキストを使用するWebプロセスもあれば、使用しないものもあります。

PROCESS PROPERTIESコマンドを使用すると、生成されたコンテキストの数を調べることができます。このコマンドは任意のWebプロセスに関して、コンテキストが使用されているか（-11：Web Process with Contex）または使用されていないか（-3：Web Process with no Context）を示す値を引数<派生元>に代入します。

Web接続のコンテキスト管理：コンテキストモード

「Web接続」プロセスの名前に示されている番号を“コンテキストID”といいます。この番号はランダムに生成され、各Web接続を一意に特定します。コンテキストIDは、Web接続中、4D側とブラウザ側の両方で保守されます。この例では、コンテキストIDは“1066993139”です。以下のWebブラウザウインドウでは、ブラウザのジャンプエリアに表示されたURLでこの番号を確認することができます。



URLはコンテキストモードのWebセッション中には4Dによって自動的に保守されます。HTTPリクエストが4DのTCP／IP経由で受信されるたびに、4DはURLからコンテキストIDを抽出し、これによりそのリクエストを正しい「Web接続」プロセスへ転送することができます。

コンテキストIDでは、以下のことを実行することができます。

- 4Dによって各Web接続経由でWebとデータベースの両方のセッションを保守できる。
- 複数の同時実行Web接続を透過的に処理できる。
- 異なるコンテキストIDが各接続で生成されるため、ブックマークを使用するような好ましくない接続を防ぐことができる。

Webとデータベースセッションの同期化：Web接続サブコンテキストID

上記のウインドウでは、コンテキストIDの後にドットと数値の2（.2）のサブコンテキストIDがあります。4Dは、コンテキストモードのブラウザに新しい4DベースのHTMLページが送信されるたびにこの番号を自動的に管理し、1つずつ増分します。サブコンテキストIDは、データベースセッションの保守において重要なものです。

通常、Webブラウザには、戻るボタン、次ボタン、履歴ウインドウ等のナビゲーション制御が含まれています。これらの制御は、ドキュメント、ニュース、掲示板等をブラウズする時に便利です。これらはデータベーストランザクションを実行する時には、あまり魅力的な制御ではありません。

例えば、Webユーザがテーブルにレコードを追加している時に、そのデータ入力が確定されたのかどうか、つまり、Webユーザが4Dフォームの「登録」ボタンまたは「キャンセル」ボタンをクリックしたかどうかを開発者は知る必要があります。この時点で、Webユーザが他のページをナビゲートすると、データ入力は不確定な状態のままになってしまいます。これを防ぐために、4Dはブラウザ側のWebセッションと4D側のデータベースセッションをサブコンテキストIDを使って同期させています。

フォームがサブミットされたり、HTTPリクエストがブラウザから4Dに送信されるたびに、Webとデータベースセッションの同期が失われていることが検出された場合には、4Dは「同期が外れました。ブラウザのナビゲーションコントロールなど（例えば“戻る”ボタン）を使用したために、データ検証を必要とするフォームから離脱してしまいました。検証もしくはキャンセルをするために、今からそのフォームへ戻ります。」というメッセージを送信します。そして、4Dはサブコンテキストを使ってWebデータ入力ページに戻ります。

この同期化は、「Web接続」プロセスにとっても重要なものです。4Dコードの実行を続けるため、例えば、「**ADD RECORD**（[...]）」というコマンドから正確に抜ける必要があります。

同期化は選択的なものです。ブラウザ側に表示されているカレントWebページが4Dフォームである場合（**ADD RECORD**、**DISPLAY SELECTION**、**DIALOG**等）、必然的に同期化が行われます。

カレントWebページが別のWebページからのリンクでアクセスされたHTMLページである（**SEND HTML FILE**コマンドで送信された）場合、ページを自由にナビゲートすることができます。

以下の4Dコードがあるとします。

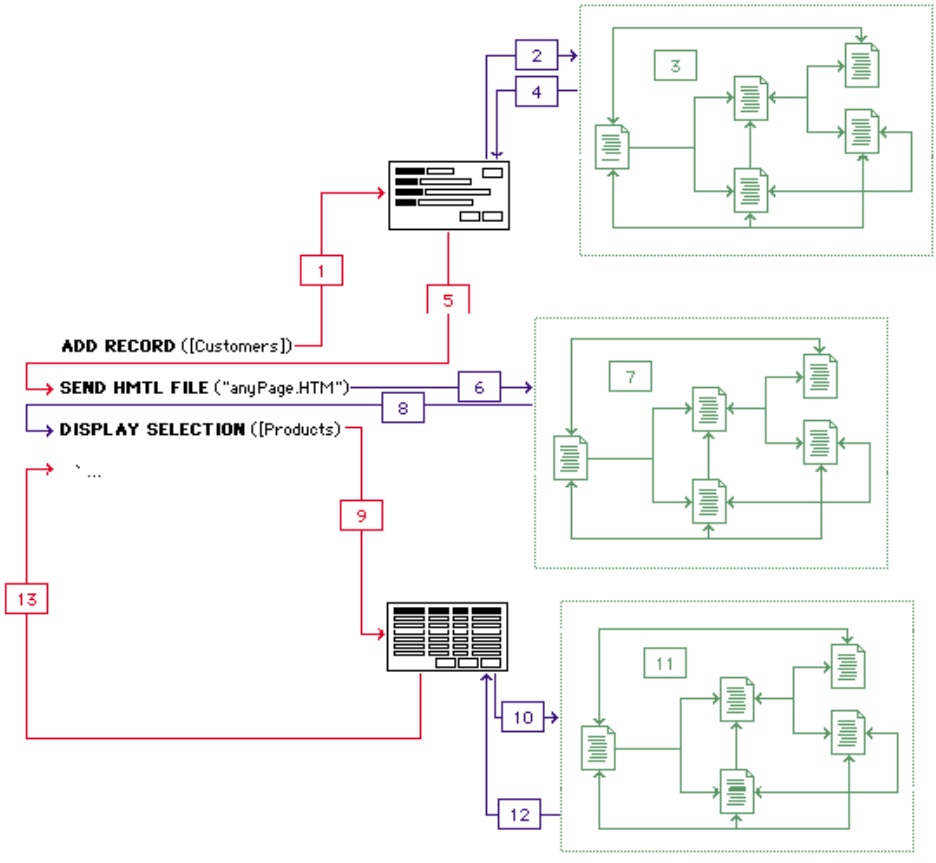
```
ADD RECORD([顧客])
SEND HTML FILE("anyPage.HTM")
DISPLAY SELECTION([商品])
```

以下の図は、実行中に4DとWebブラウザの両方で発生することを詳細に示しています。

■ (1)、(5)、(9)、(13)、は、4Dフォーム変換とサブミッションを示しています。

■ (2)、(4)、(6)、(8)、(10)、(12)、は、4DベースのHTMLページと4DベースではないHTMLページとの切り換えを示しています。

■ (3)、(7)、(11)、は、4DベースではないHTMLページを示しています。



手順の説明

(1) **ADD RECORD**コマンドが発行されます。4Dはテーブルのカレント入力フォームをHTMLページに変換して、それをWebブラウザに送信します。入力フォームがマルチページフォームの場合には、標準4Dページナビゲーションボタンによって、フォームのページ間をナビゲートできます。この4Dベースのナビゲーションが実装され、4Dによって透過的に実行されます（Webフォームサブミッション経由）。

(2) データ入力中（つまり、**ADD RECORD**コール内）に、ボタンがクリックされ、そのオブジェクトメソッドから**SEND HTML FILE**コールを発行します。

(3) **SEND HTML FILE**コール内部で、HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE**（""）」が発行されたときにHTMLモードは終了します。

(4) クリックされたボタンのオブジェクトメソッドと、**ADD RECORD**コマンドで開始されたデータ入力が実行されます。手順(2)と(3)はデータ入力内で数回繰り返すことができます。

(5) 最終的に、データ入力が受け付けられるか、キャンセルされ、「Web接続」プロセスが実行されます。

(6) 以下のコールは、**SEND HTML FILE**コマンドです。

(7) この手順は手順3と似ています。HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE**（""）」が発行された時にHTMLモードが終了します。

(8) 「Web接続」プロセスが実行されます。

(9) **DISPLAY SELECTION**コマンドが発行されます。4Dはテーブルの現在の出力フォームをHTMLページに変換して、それをWebブラウザに送信します。**DISPLAY SELECTION**コマンド中に、4Dは選択ページと個別のレコード表示の間を透過的にナビゲートします。また、4Dは**MODIFY SELECTION**コマンドを使用してデータ入力の管理とレコードロックをWebフォームサブミッションから実行することができます。

(10) セレクションのナビゲーション中に、フォームのフッタエリアにあるボタンがクリックされ、そのオブジェクトメソッドから**SEND HTML FILE**コールを発行します。

(11) この手順は手順3および手順7と似ています。HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE**（""）」が発行された時に、HTMLモードが終了します。

(12) クリックされたボタンのオブジェクトメソッドと、**DISPLAY SELECTION**コマンドで開始されたセレクション表示が実行されます。手順(10)と(11)はセレクションのナビゲーション中に数回繰り返すことができます。

(13) 最終的に、選択表示が終了し、「Web接続」プロセスが実行されます。

以下同様になります。

自由なWebナビゲーション（例えば、戻るボタンや次ボタンをクリックすること）は、任意の**SEND HTML FILE**（コマンド前述の図の緑のエリア）内で可能です。その一方で、任意の4DベースのHTMLページ（データ入力、セレクション表示等、**CONFIRM**コマンドや**Request**関数によって表示されるような標準ダイアログボックスを含む）はブラウザナビゲーション制御の1つを使用して終了され、4Dは最後には「Web接続」プロセス側で現在実行されている発行済の4Dコマンドに対応するサブコンテキストIDを持つWebページに戻ることによって、Webセッションとデータベースセッションを同期化します。

「Web接続」プロセスとWebセッション

ユーザの立場からすると、Webブラウザ側でのユーザのアクションがWebセッションを誘導します。

プログラム作成の立場からすると、「Web接続」プロセスがWebセッションを誘導するのであって、Webセッションが「Web接続」プロセスを誘導するわけではありません。Webブラウザは「Web接続」プロセスが送信したページを表示します。それは、以下のいずれかを行います。

■ 4Dコードを実行する、または

■ カレントWebページのブラウザからのサブミッションを待機する。
設計者の立場からすると、「Web接続」プロセスは実行のドメインが4th Dimensionまたは4D Serverである4Dプロセスとして見えるべきですが、そのユーザインタフェースは接続されたWebブラウザ上でエコーされます。

このようなことから、コンテキストモードのWebデータベースアプリケーションの設計を実行する場合には、この「Web接続」プロセスの二重性を常に考慮してください。以下の例を参照してください。

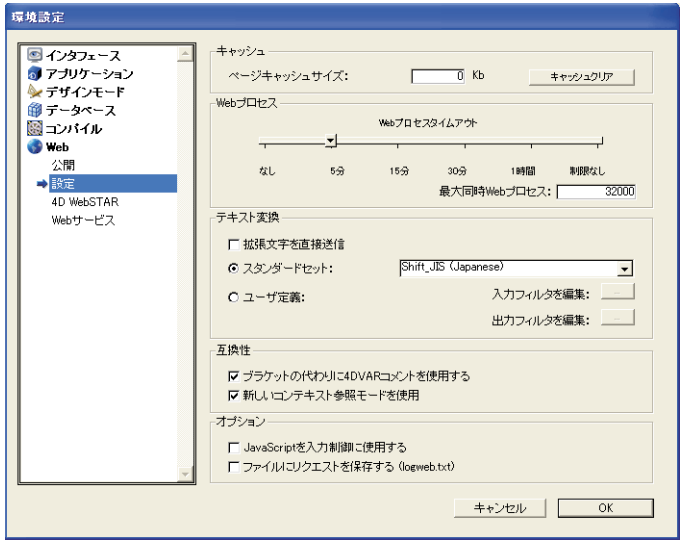
■ どんな種類のデータ入力でも、その間、メインメニューバーはブラウザのものであり、4Dのメニューバーではありません。フォーム内では、4Dメニューバーを頼りにしないでください。4DメニューバーはWebサーバマシン上にあり、Webブラウザマシン上にはありません。

■ Webブラウザで使用されるフォームを設計する場合には、4Dフォームの機能はHTMLのフォーム上で制限されていることを考慮してください（ただし、時には4Dの追加機能もあります）。4Dフォームの機能をすべて使用できるとは限りません（例：オブジェクトタイプやフォームイベント）。詳細については、「Webサービス：HTMLとJavascriptのカプセル化」の節を参照してください。

■ プロセス間通信において、**CALL PROCESS**コマンドを「Web接続」プロセスに適用した場合には、現在アクティブなフォームはWebブラウザに表示されているため、何の動作も行いません。一方、「Web接続」プロセスは別の4Dプロセスに対して**CALL PROCESS**コマンドを発行できます。さらに、プロセス間通信は**GET PROCESS VARIABLE**コマンドおよび**SET PROCESS VARIABLE**コマンドを使用することで、双方向に実行することができます。これらのコマンドは、ユーザインタフェースを持つ必要がないからです。

Web接続タイムアウト

前述したように、「Web接続」プロセスは4Dコードを実行しているか、またはブラウザ側で現在表示されているWebページのサブミッションを待機しているかのいずれかです。後者の場合、「Web接続」プロセスはWebプロセスタイムアウトと同じ時間だけ待機します。このタイムアウトは、「環境設定」ウインドウの「詳細」ページ（Webテーマ）において、あるいは**SET WEB TIMEOUT**コマンドを用いてプログラムから設定されます。



「Webサーバ接続タイムアウト」設定の有効範囲は、データベースセッションです。

すべての「Web接続」プロセスは、タイムアウト時間の値に従います。値の設定が変更されると、プロセスは即座に影響を受けます。デフォルト値は、5分間です。

注：SET WEB TIMEOUT関数はWebプロセスごとにタイムアウト値を指定できます。

必要に応じてこのタイムアウトを長くする、あるいは短くすることができます。例えば、Webユーザがデータベースから提供されるページ内のHTMLリンクを経由して、他のWebサイトに移動することをアプリケーションが許可している場合には、タイムアウトを長くすることができます。タイムアウトを長くすることで、ユーザはデータベースへの接続がクローズされることなく、他のWebサイトをさらに長い時間ナビゲートできます。

警告：「Web接続」プロセスをプログラムで停止する方法はありません。長いタイムアウトを指定した場合には、Webユーザがかなり前にWeb接続での作業を終っていても、プロセスはその遅延を待つことになります。「タイムアウトなし」オプションを指定すると、「Web接続」プロセスはデータベースが終了した時にだけ停止します。しかし、Web接続プロセスはWebサーバが非コンテキストモードに切り替えられると同時に自動的にアボートされます。

Tips：「Webサーバ」プロセスとは違って、「Web接続」プロセスは「アボート」コマンドを使用してアボートすることができます（「プロセスページ」が表示されているときに「ランタイムエクスプローラ」で利用できます）。

HTMLの自動変換

この節では、コンテキストモードにおいて4th DimensionがデータベースをHTMLに変換する際、自動的に処理される要素やオブジェクト、ならびにメカニズムについて説明します。

メニューバー

■ 各メニュータイトルはテキストとしてのみ表示され、4Dメソッドに関連付けられたメニューコマンドは4Dメソッドへのリンクとして表示されます。自動アクションだけに関連付けられたメニューコマンドは、テキストとしてのみ表示されます。

■ Webブラウザ側でメニュー項目をクリックすると、「Web接続」プロセス側で関連する4Dメソッドの実行が開始されます。

注：メニューコマンドに対して「新規プロセス開始」プロパティが指定されている場合、関連するメソッドは、「4DMETHOD」URLを使用して、4D Webサーバにより新しいWeb接続プロセス内で実行されます。この場合、そのメニューのメソッドには「4DACTIONで利用可能」オプション（デフォルトでは新規データベースに対して未選択）が指定されていなくてはなりません。詳細については「接続セキュリティ」の節を参照してください。

■ ピクチャはブラウザのメニューの下に位置するメニューバーと結び付けられています。

フォーム

■ オブジェクトは上から下へ、左から右への順で変換され、それらはブラウザと4Dフォームにあるものと同じ位置あります。

■ マルチページフォームは、透過的にサポートされます。ただし、0ページ目も含まれます。その結果、ページナビゲーションボタンは各ページ上にある必要があります。

■ 適切な場合には、自動アクションが透過的にサポートされます。

■ フォームイベント（**On Load**、**On Unload**、**On Clicked**、**On Timer**）がサポートされます。それ以外のイベントはサポートされません。

■ ヘッダ、ディテイル（詳細）、ブレーク、フッタのタグは、**DISPLAY SELECTION**コマンドと**MODIFY SELECTION**コマンドへの呼び出し中に考慮されます。フォームのヘッダは、HTMLページの先頭に一度表示され、ディテイル（詳細）エリアは必要な数だけ繰り返されます。また、フッタエリアにある変数（ボタン等）はHTMLページの最後、つまり自動選択ページナビゲーションリンクのすぐ下に配置されます。

■ tipsはボタンがピクチャとしてフォームエディタに表示されるのとブラウザで見るのとを結び付けて考えられます。ブラウザは表示するのにこれらのtipsを許可しています。

■ 4Dのフォームエディタ内の座標（0、0、x、x、）に挿入されたリプリケートピクチャ（表示タイプが"繰り返し"）は、ブラウザの背景ピクチャとして送られます。

注意：4D Web ServerはCSS1(Cacading Style Sheet 1)を使って、4Dフォームに極めて近い外観を持つHTMLページを生成します。CSS1の仕様はW3C(World Wide Web Consourtium)により定義されます。これらのスタイルシートはドキュメントの外観に関わる特質、フォント、サイズ、タイトル、ボディ、行間などを設定します。4Dバージョン6.5から.CSSドキュメントはMIMEタイプのテキストまたはCSSとともに送られることになりました。コンテクスト/非コンテクストモードにかかわらずこれらのドキュメントは4Dにより処理されません。互換性により、SET DATABASE PARAMETERコマンド（セレクタ8、ウェブ変換モード）を用いるフォームに利用するために、ウェブ変換モードに変更することができます。

フィールドオブジェクト

4DフォームがHTMLページに変換される時に、フィールドオブジェクトは以下のように変換されます。

4Dフィールドタイプ	HTMLオブジェクト	HTML Markup
文字	テキストフィールド(*)	<INPUT Type="text" ...>
テキスト	テキストフィールド(*)	<TEXTAREA ...>(**) <INPUT Type="text" ...>(***)
実数	テキストフィールド(*)	<INPUT Type="text" ...>
整数	テキストフィールド(*)	<INPUT Type="text" ...>
倍長整数	テキストフィールド(*)	<INPUT Type="text" ...>
日付	テキストフィールド(*)	<INPUT Type="text" ...>
時間	テキストフィールド(*)	<INPUT Type="text" ...>
ブール	ラジオ、または	<INPUT Type="radio" ...>
	チェックボックス(*)	<INPUT Type="checkbox" ...>
ピクチャ	イメージ(常に入力不可)	<IMG SRC="..." ...>
サブテーブル	HTMLではサポートしてない	なし
BLOB	HTMLではサポートしてない	なし(****)

(*) または入力不可の場合にはテキストのみ
(**) テキスト値が複数の行で構成されている場合
(***) テキスト値が1行だけ、または空行の場合
(****) SEND HTML BLOB関数はBLOBタイプのフィールドや変数を送ることができます。

注：入力可能な変数は、変数と同じタイプのフィールドと似た動作になります。

フォームオブジェクト

4DフォームがHTMLページに変換される時に、フォームオブジェクトは以下のように変換されます。

4Dフォームオブジェクト	同等のHTMLオブジェクト	HTML Markup
線	水平ライン(1)	<HR>
矩形	矩形	CSS1で管理
楕円	HTMLではサポートしてない	なし
角の丸い矩形	HTMLではサポートしてない	なし
静止ピクチャ	イメージ / イメージマップ(2)	<IMG SRC="..."> <INPUT Type="image" ...>
グループボックス	テキスト	fontマークアップ付 テキスト(ある場合)
静的テキスト	テキスト	fontマークアップ付 テキスト(ある場合)
ボタン	サブミットボタン	<INPUT Type="submit" ...>
デフォルトボタン	サブミットボタン	<INPUT Type="submit" ...>
ラジオボタン	ラジオボタン(3)	<INPUT Type="radio" ...>
チェックボックス	チェックボックス	<INPUT Type="checkbox" ...>
ポップアップ/	ドロップダウンリスト	<SELECT ...>...</SELECT>
ドロップダウンリスト	ドロップダウンリスト	<SELECT ...>...</SELECT>
コンボボックス	ドロップダウンリスト	<SELECT ...>...</SELECT>
スクロールエリア	スクロールリスト(4)	<SELECT ...>...</SELECT>
透明ボタン	下記の備考2参照	
ハイライトボタン	下記の備考2参照	
タブコントロール	URLリスト(5)	<AHREF= "/4DTAB/4DVar.Onglet...">
3Dボタン	下記の備考2参照	
ボタングリッド	下記の備考2参照	
グラフ	イメージ(入力不可)	<IMG SRC="..." ...>
プラグイン	イメージ(入力不可)	<IMG SRC="..." ...>

以下のオブジェクトは、HTMLではサポートされていないため、トランザクション中は無視されます。

階層ポップアップメニュー、階層リスト、サブフォーム、タブコントロール、ラジオピクチャ、サーモメータ、ルーラ、ダイアル、ピクチャポップアップメニュー、ピクチャボタン、3Dチェックボックス、3Dラジオボタン

備考

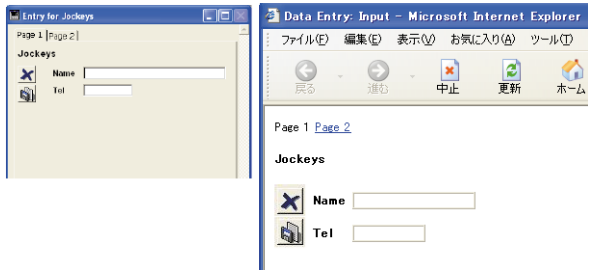
1. 水平方向以外の行は、HTMLではサポートされません。そのため、それらの行は無視されます。

2. 非表示系のボタンは、透明ボタン、ハイライトボタン、3Dボタン、ボタングリッドタイプのオブジェクトです。非表示系のボタンが静止ピクチャの上に重ならない場合には、静止ピクチャは静的イメージとして変換されます。非表示系のボタンが静止ピクチャの上に1つでも重なる場合は、静止ピクチャは「Server-Side Image Map」として変換されます。Webブラウザ側では、このイメージは「Server-Side Image Map」として扱われます。4D側では、サブミッションが受け付けられた時に、ボタンが実際にクリックされたかのように適切なボタン用に「On Clicked」イベントを生成するために、クリックの場所を4Dが再計算します。このようなことから、非表示系のボタンを管理することは、静止ピクチャと重なっている場合には非常に簡単です。非表示系のボタンは、通常の4Dインタフェースで実行するように、「フォーム」メソッドまたはそれらのオブジェクトメソッドによって管理します。また、非表示系のボタンではWeb Image Mappingを処理するのが非常に簡単になります。非表示系のボタンは、静止ピクチャオブジェクトとまったく重ならない場合には、変換中は無視されます。

3. ラジオボタンのグループ化は、変換中は保持されます。

4. グループ化されているスクロールエリアは、HTMLではサポートされません。4Dは、同じ行にある独立したスクロールリストとしてグループ化されたスクロールエリアを変換します。

5. タブコントロール（arrayタイプ又はオブジェクトプロパティで定義された値を使って作成された）はURLリストから変換されています。Array要素が空白の文字だったら、4Dはブラウザに1，2，3を表示します。



6. プラグインエリアはWeb上に公開されています。最初にHTMLにイメージまたはイメージマップとして変換されています。最後にソリューションはプラグインエリア内でマウスのクリックを管理することができます（例えば、4D Chartのプラグインの統合はイメージマップで公開され、4D_Pack AP外部クロックエリアはイメージとして公開されています）。これらのプラグインエリアや4Fフォーム上は、プラグインエディタの詳細によりWeb上に公開されます。

DISPLAY SELECTION／MODIFY SELECTIONコマンド

■ UserSetメカニズムは、サポートされません。

■ 自動セレクションページングメカニズムが4Dから提供されます。詳細については、**SET WEB DISPLAY LIMIT**コマンドの説明を参照してください。

4Dコマンド

4D Webデータベースを開発している時に、コマンドが呼び出されると何が起きるかを知りたい場合があります。コマンドは、Webサーバマシン上で有効になるのか、あるいはWebブラウザマシン上で有効になるのでしょうか？「Web接続」プロセスはWebサーバマシン上で稼働していますが、そのユーザインタフェースは接続されているWebブラウザ上にリモートでエコーされます。したがって、Webデータベースの開発では、4Dコマンドは以下のように規定できます。

「Web接続」プロセス内からの実行には影響を受けないコマンド

CREATE RECORDのようなコマンドは、稼働中のプロセスの中で作動します。この場合、

CREATE RECORDコマンドは「Web接続」プロセス内でレコードを作成します。同じことが**Screen width**等の関数にも当てはまります。この関数は、Webサーバマシン（プロセスが稼働しているマシン）の画面の幅を返します。

透過的なWebサポートのための追加組み込み機能を含むコマンド

コマンド名	コメント
ADD RECORD	フォームの自動変換。マルチページフォームをサポートする
ALERT	ダイアログボックスの自動変換
CONFIRM	ダイアログボックスの自動変換
DIALOG	フォームの自動変換、マルチページフォームをサポートする
DISPLAY SELECTION	フォームの自動変換 内蔵ののWebページングメカニズム。 UserSetメカニズムはサポートしない。
MODIFY RECORD	フォームの自動変換、マルチページフォームをサポートする
MODIFY SELECTION	フォームの自動変換

	内蔵のWebページングメカニズム UserSetメカニズムはサポートしない。
QUERY	標準の「クエリ」ダイアログボックスをサポートする
QUERY BY EXAMPLE	フォームの自動変換、マルチページフォームをサポートする
Request	ダイアログボックスの自動変換
REDRAW	ブラウザ上で表示されたフォームのアップデート

実行したいことがわかっている場合に使用するコマンド

以下のコマンドは、Webサーバマシン上でローカルに実行されます。

例えば、Webブラウザからセレクションの印刷処理を起動できます。ただし、印刷はWebサーバマシン上で行われます。

さらに、ユーザインタフェースコンポーネントが含まれると、そのコンポーネントはWebサーバマシンに表示されます。つまり、「**Open document**（""）」と「**Open documen**("ドキュメント")」の違いです。Webサーバマシン上でダイアログボックスがクローズされるまでWebブラウザが応答を待つため、このような呼び出しは避けるべきです。それに対して、ダイアログボックスが含まれない時にこれらのルーチンを呼び出すことはまったく問題がありません。

コマンド名	コメント
Append document	ファイルダイアログボックスが起動されなければ、問題なし
BEEP	Webサーバマシン上で警告音を出す
Create document	ファイルダイアログボックスが起動されなければ、問題なし
DISPLAY RECORD	何も行わない
EXPORT DIF	ファイルダイアログボックスが起動されなければ、問題なし
EXPORT SYLK	ファイルダイアログボックスが起動されなければ、問題なし
EXPORT TEXT	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT DIF	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT SYLK	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT TEXT	ファイルダイアログボックスが起動されなければ、問題なし
LOAD SET	ファイルダイアログボックスが起動されなければ、問題なし
LOAD VARIABLES	ファイルダイアログボックスが起動されなければ、問題なし
MESSAGE	メッセージは、Webサーバマシン上に表示される
Open document	ファイルダイアログボックスが起動されなければ、問題なし
Open external window	ウインドウは、Webサーバマシン上でオープンする
Open resource file	ファイルダイアログボックスが起動されなければ、問題なし
Open window	ファイルダイアログボックスが起動されなければ、問題なし
PLAY	音は、4Dマシン上で再生される
PRINT FORM	ファイルダイアログボックスが起動されなければ、問題なし
PRINT LABELS	ァイルダイアログボックスが起動されなければ、問題なし
PRINT RECORD	ファイルダイアログボックスが起動されなければ、問題なし

PRINT SELECTION	ファイルダイアログボックスが起動されなければ、問題なし
QUIT 4D	リモートでWebサーバをシャットダウンできる。
SAVE SET	ファイルダイアログボックスが起動されなければ、問題なし
SAVE VARIABLES	ファイルダイアログボックスが起動されなければ、問題なし
SELECT LOG FILE	ファイルダイアログボックスが起動されなければ、問題なし
SET CHANNEL	ファイルダイアログボックスが起動されなければ、問題なし
TRACE	デバッガウインドウは、Webサーバマシン上に表示される

「Web接続」プロセスではサポートされないコマンド

コマンド名	コメント
ADD DATA SEGMENT	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
ADD SUBRECORD	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
CHANGE ACCESS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
EDIT ACCESS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「パスワード」ウインドウは4Dマシン上に表示される。ブラウザはウインドウがクローズされるまで待つ。
GRAPH TABLE	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
MODIFY SUBRECORD	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
ORDER BY	プログラムによるサポートのみ。標準の「並べ替え」ダイアログボックスはWebではサポートされていない。
PRINT SETTINGS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「プリント」ダイアログボックスは4Dマシン上に表示される。 ブラウザはウインドウがクローズされるまで待つ。
QR REPORT	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「クイックレポート」ウインドウは4Dマシン上に表示される。 ブラウザはウインドウがクローズされるまで待つ。

HTMLの埋め込み

HTMLコード（またはJavascript）をフォーム内に埋め込むことにより、HTMLに変換された4Dフォームの内容をカスタマイズすることができます。この結果Webブラウザ側に表示されるフォームには、HTMLと4Dオブジェクトが混在します。

スタティックテキストオブジェクトを使用したHTMLページの挿入

例えば、“{page.HTM}”という文字列を含む4Dフォームのスタティックテキストオブジェクトは、4Dフォーム内にあるそのテキストオブジェクトの場所にHTMLドキュメント“page.HTM”を挿入します。この場合、ドキュメント全体（実際には<BODY>から</BODY>タグの間に含まれるものすべて）が挿入されます。既存のHTMLドキュメントを使用するか、あるいはランゲージを使用してディスクに保存するドキュメントを作成し、この後にそれを参照することもできます。

注：場合によって、バージョン6.0.xで作成された4DフォームにHTMLドキュメント（{mypage.htm}）への参照が含まれていると、バージョン6.7以降の4Dではこのフォームの HTML変換が予期しない結果になることがあります。この場合、SET DATABASE PARAMETERコマンドを使用して、フォームの変換モードを変更することができます。

HTMLコードの挿入

4Dのテキスト変数の1桁目がASCIIコード1である場合、テキスト変数を使用してHTMLコードを4Dフォームに埋め込むことができます（例えば、vtHTML:=Character(1)+"...HTMLコード..."）。

このようにして、コードを挿入することができます。また、この場合はHTMLコードをメモリ上に作成することができます。

ファイル参照とURL

コンテキストモードでデータベースのコンテキストIDとサブコンテキストIDの保守を確実にするために、4Dはファイル参照とURLを自動的に再マップします。例えば、4Dはすべての IMG参照とHREF参照をローカルファイルに再マップします。

テキスト変数を使用して独自のHTMLコードを4Dフォームに挿入する場合には、4D再マッピング構文に従う必要があります。

ローカルのGIFファイルは、"/4DPict/_/GIF_file_pathname/$-2"として再マップされ、ここで「GIF_file_pathname」はファイルが存在するボリュームのルートに相対的なGIFファイルの完全なHTMLパス名になります。

例題

以下の4Dメソッドは、引数として受け取ったパス名に対する再マップされた参照を返します。

```
` 「WWW Local GIF URL」プロジェクトメソッド
` WWW Local GIF URL Project(テキスト)
` WWW Local GIF URL(本来のパス名) → ローカルGIFファイルのURL
C_TEXT($0 ; $1)
$0:="/4Bin/_/"+HTML Pathname($1)
```

注：「HTML Pathname」メソッドの詳細については、Mac to ISO関数の例を参照してください。

次に、テキスト変数を使ってHTMLコードを4Dフォームに挿入する場合は、以下のように指定できます。

```
vtHTML:=Char(1)+"<P><IMG SRC="+Char(34)+ WWW Local GIF URL
                ("F:\ThisImage.HTM"+Char(34)+" ALIGN=MIDDLE></P>"+Char(13)
```

これによって、4D変数vtHTMLの位置にGIFドキュメントが4Dフォームに挿入されます。

重要：カスタムHTMLコードを4Dフォームに挿入するには、単にこの種のコードを作成します。SEND HTML FILEコマンドを使用してHTMLページを送信するだけ、またはADD RECORDコマンド等のコマンドを使用する場合には、4Dが透過的にHTMLを変換し、再マップすることに注意してください。

再マッピングは、以下のプロトコルを持つリンクを変更しません。

- http:
- ftp:
- mailto:
- news:
- gopher:
- javascript:
- nntp:
- wais:
- prospero:
- telnet:

SSLプロトコルの使用

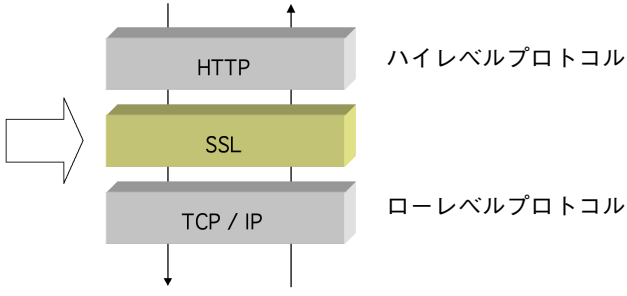
4D WebサーバではSSL（Secured Socket Layer）プロトコルを使用した暗号化モードでの通信が行えるようになりました。

SSLプロトコルの定義

SSLプロトコルは、2つのアプリケーション間、主にWebサーバとブラウザとのデータのやり取りを保護する目的で設計されました。このプロトコルは広く利用されており、また大部分のWebブラウザと互換性があります。

ネットワークレベルでは、SSLプロトコルはTCP/IPレイヤ（ローレベル）とHTTPハイレベルプロトコルの間に挿入されます。SSLは、主にHTTPと共に作業を行うように設計されています。

SSLを使用したネットワーク構成



注：SSLプロトコルは、標準の4D Serverのクライアント／サーバ接続を保護するためにも使用できます。詳細は、『4D Serverリファレンス』マニュアルの「クライアント／サーバ接続の暗号化」の節を参照してください。

SSLプロトコルは、送信者と受信者の認証を行い、やり取りする情報の秘匿性および整合性を保証する目的で作られています。

■ **認証：**送信者と受信者の身元を確認します。

■ **秘匿性：**送信データは暗号化されるため、第三者はメッセージを理解することができません。

■ **整合性：**受信したデータは、偶発的であれ作為的であれ、変更されることはありません。

SSLは、暗号化および復号化のために、公開鍵と秘密鍵という一対の非対称型鍵をベースとした公開鍵暗号化技術を使用します。

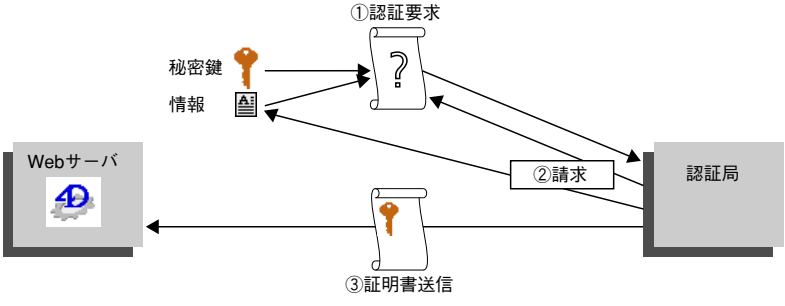
秘密鍵はデータの暗号化に使用されます。送信者（Webサイト）はこの鍵を誰にも与えません。公開鍵は情報の解読に使用され、証明書を介して送信者（Webブラウザ）に送られます。SSLをインターネットで使用する際、この証明書はVerisign®（ベリサイン社）のような認証局を通して届けられます。Webサイトは証明書配達のために認証局へ料金を支払いますが、この証明書によってサーバの認証は保証され、また暗号化モードでのデータのやり取りを行える公開鍵がこれに納められています。

注：暗号化メソッド、および公開鍵と秘密鍵の使用に関する詳細は、ENCRYPT BLOBコマンドの説明を参照してください。

証明書の取得方法

暗号化モードで動作する4D Webサーバには、認証局発行の電子証明書が必要となります。証明書には、サイトIDおよびそのサイトとの通信に使用する公開鍵などの各種情報が納められます。この証明書は、そのサイトに接続するWebサーバへ転送されます。証明書が確認され、受け付けられると、暗号化モードで通信が行われます。

注：ブラウザでは、そのプロパティで参照される認証局発行の証明書しか承認されません。



認証局の選定は、各種条件に応じて行われます。知名度が高い認証局であれば、その証明書は大部分のブラウザで承認されますが、料金は高くなります。

SSL証明書を取得するには、

1 **GENERATE ENCRYPTION KEYPAIR**コマンドを使用して、秘密鍵を生成する。

警告：セキュリティ上の理由から、常に秘密鍵は人に知られないようにしてください。実際上、この鍵はいつもWebサーバマシンのもとに置いてください。また、Key.pemファイルはデータベースストラクチャフォルダ内に配置しなければなりません。

---

2 **GENERATE CERTIFICATE REQUEST**コマンドを使用して、証明書リクエストを発行する。

3 選択した認証局へ証明書リクエストを送信する。

証明書リクエストの必要事項を満たすため、認証局との連絡が必要な場合もあります。認証局では、転送された情報が正しいかを確認します。また、証明書リクエストはPEM（Privacy Enhanced Mail）フォーマットを使用してBLOB内に生成されます。このフォーマットを使用することにより、鍵をテキストとしてコピー＆ペーストし、その内容を変更せずに鍵を電子メールで送信することができます。例えば、証明書リクエストを納めたBLOBをテキストドキュメントに保存（**BLOB TO DOCUMENT**コマンドを使用）した後、これを開いてその内容をメールやWebフォームにコピー＆ペーストし、認証局へ送信することができます。

4 証明書を取得したら、“cert.pem”という名前のテキストファイルを作成し、このファイルに証明書の内容をペーストする。

証明書は、さまざまな方法で受信できます（通常は電子メールまたはHTMLフォーム）。4D Webサーバは、証明書用にすべてのプラットフォーム関連のテキストフォーマットを受け入れます（Mac OS、PC、Linux等）。ただし、証明書は必ずPEMフォーマットでなくてはなりません。

5 “cert.pem”ファイルをデータベースストラクチャフォルダに配置する。

これで、Webサーバは暗号化モードで動作します。証明書は6ヶ月から1年の間有効です。

4DにおけるSSLのインストールとアクティブ化

4D WebサーバでSSLプロトコルを使用したい場合には、次のコンポーネントをサーバ上のそれぞれの場所にインストールしてください。

■ 4DSLI.DLL：SSL管理専用のSecured Layer Interface

このファイルは、データベースを発行する4Dアプリケーションの[4D Extensions]フォルダ内に配置してください。

■ key.pem：暗号化秘密鍵を含むドキュメント

■ 4th Dimensionまたは4D Serverでは、このファイルは必ずデータベースフォルダ内に配置してください。

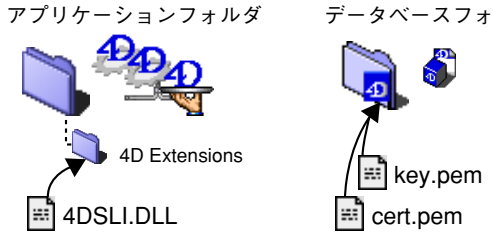
■ 4D Clientでは、このファイルは必ず4D Clientのアプリケーションフォルダ内に配置してください。

■ cert.pem：“証明書”を納めたドキュメント

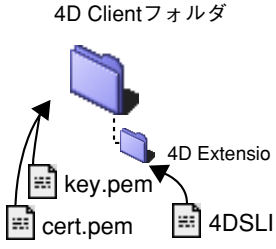
■ 4th Dimensionまたは4D Serverでは、このファイルは必ずデータベースフォルダ内に配置してください。

■ 4D Clientでは、このファイルは必ず4D Clientのアプリケーションフォルダ内に配置してください。

4D WebサーバでSSLを実装するために必要なファイル（4th Dimensionおよび 4D Server）

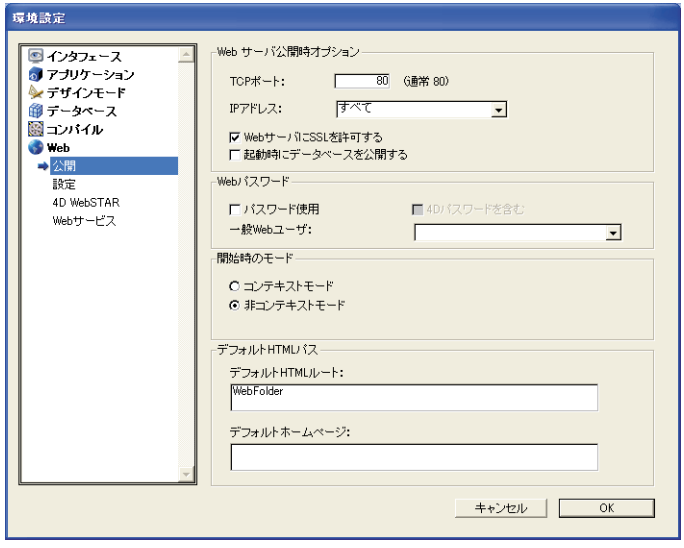


4D Webサーバ（4D Client）でSSLを実装するために必要なファイル



注：暗号化コマンドであるENCRYPT BLOBおよびDECRYPT BLOBを使用する際にも、4DSLI.DLLが必要となります。

これらのファイルがインストールされると、SSLを使用して4D Webサーバへ接続することができます。ただし、4D WebサーバがSSL接続を受け入れるには、SSLを“アクティブ”にしなくてはなりません。このパラメータは、データベースの「環境設定」において、「Web」テーマの「設定」ページで利用することができます。



デフォルトでは、SSL接続が許可されています。WebサーバでSSL機能を使用したくない場合や、同一マシン上で暗号化接続を許可する他のWebサーバが動作している場合には、このオプションの選択を解除することができます。

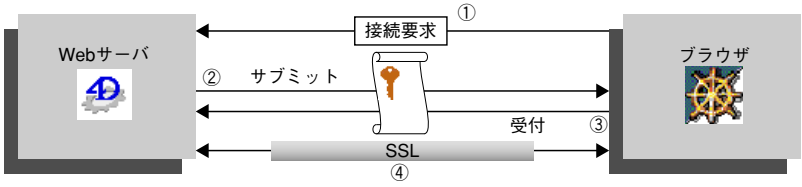
デフォルトでは、SSLデータのやり取り専用のTCPポートは443です。例えば、Webサーバのセキュリテイを強化するために、「HTTPSポート番号」エリアでこのポート番号を変更することができます（この件に関する詳細は、「Webサーバ設定」の節を参照してください）。

注：4D Webサーバの管理のために設定した「環境設定」のその他パラメータ（パスワード、タイムアウト、キャッシュサイズ等）は、サーバがSSLモードで動作しているかどうかに関わらず適用されます。

SSLを使用したブラウザ接続

Web接続を暗号化モードで実行するには、ブラウザから送信するURLを（“http”ではなく）“https”で開始します。

この場合、ブラウザには警告ダイアログが表示されます。ユーザが「OK」をクリックすると、Webサーバは証明書をブラウザへ送信します。



次に、接続に使用される暗号化アルゴリズムがブラウザとWebサーバ側で決定されます。デフォルトとして、サーバはRC4（128ビット）暗号化アルゴリズムを提供し、このアルゴリズムは40ビットで暗号化した鍵を使って行われる接続に対しても使用されます。その国の法律や使用するブラウザのバージョンによっては、128ビットのアルゴリズムが使用できない場合があります。最も強力で一般的なアルゴリズムが使用されます。

警告：許可される暗号化のレベルは、それを利用する国の現行法によって異ります。また、4D Webサーバから提供される暗号化のレベルは、使用する暗号化システムライブラリに依存します。デフォルトとして、4Dはこのライブラリの“Export”バージョンを提供しているため、対称アルゴリズムは40ビットに制限されます。

接続モードの管理

4D WebサーバでSSLを使用する場合に、特別なシステム構成は必要ありません。しかし、SSL Webサーバは非暗号化モードでも動作できるという点に注意してください。また、接続モードは、ブラウザ側の要求があれば（例えば、ブラウザのURLエリアでユーザが“HTTPS”を“HTTP”で置き換えた場合）、もう一方のモードへ切り替えることができます。開発者は、非暗号化モードで行われたリクエストを禁止したり、リダイレクトすることが可能です。**Secured Web connection**関数を使用すると、現在の接続モードを取得できます。

参照

DECRYPT BLOB、ENCRYPT BLOB、GENERATE CERTIFICATE REQUEST、GENERATE ENCRYPTION KEYPAIR、Secured Web connection、Webサーバセッティング

XMLとWMLのサポート

WML

4D WebサーバではWML（Wireless Markup Language）テクノロジーがサポートされます。この機能により、携帯電話やPDAの所有者が4Dデータベースのデータの読み込みや入力を行えるようになります。

注：WAP（Wireless Application Protocol）に関連するWML言語は、複数の企業により開発されました。WAPテクノロジーにより、一連のネットワーク通信ツールが提供されますが、この結果、携帯電話やPDAユーザがWebページ上に発行されたテキストを表示できるようになります。WMLテクノロジーはオープンかつ無償で提供されています。WMLに関する詳細は、Phone.com社のWebサイト、http://www.phone.com/を参照してください。

4DVARや4DSCRIPTタグを使用したWMLページ経由で、データの入力や読み込みを行うことができます。

4D Webサーバでサポートされる WMLドキュメントのリストを以下に示します。

拡張子	MIMEタイプ	説明
.wml	text/vnd.wap.wml	WMLページ（4Dにより常にサポートされる*）
.wmls	text/vnd.wap.wmlscript	WMLスクリプト（クライアント側）
.wmlc	application/vnd.wap.wmlc	WMLバイナリページ
.wmlsc	application/vnd.wap.wmlscript	WMLバイナリスクリプト
.wbmp	picture/vnd.wap.wbmp	携帯電話用ビットマップイメージ（サポートされない場合もある）

* 4DVARや4DSCRIPTタグにより、ダイナミックなデータ挿入を行うことができます。

XML

4D Webサーバは、.xml、.xls 、.dtdドキュメントをサポートします。これらのドキュメントは、MIMEタイプ“text/xml”および“text/xsl”を使用して送信されます。

送信ドキュメントに適用されるモードに関わらず（コンテキストモードまたは非コンテキストモード）、4Dはダイナミックな XMLを生成するために、そのコンテンツを解析して4DVARや4DSCRIPTタグ（存在する場合）を処理します。

注：スタティックテキストに含まれる{mypage.xml}のようなタグを使って、4Dフォームから XMLフォーマットを送信することはできません。

参照

HTMLオブジェクトと4Dオブジェクトのバインド

CGIの使用

4D Webサーバでは、CGI（Common Gateway Interface）がサポートされます。WebサーバとCGIとの関係は、4Dメソッドとプラグインの関係に似ています。CGIはWebサーバによって呼び出され、タスクを実行して応答、つまり完全なWebページまたはサーバより送信されたページに挿入されたHTMLコードを返します。CGIは、ビジタカウンタの表示、ゲストブックの作成、フォームーメールの受信等を行うため、頻繁に使用されます。現在では数多くのCGIが利用可能であり、その大半はフリーウェアです。

注：本来、CGIは外部アプリケーションとHTTPサーバがやり取りを行うための規約でした。今ではCGIという用語は、外部アプリケーションそのものに対して使用されています。

4D Webサーバは、2種類の方法でCGIをサポートしています。

■ 4D Webサーバでは自動モードまたは手動モードでCGIを使用することができます。

■ 4D Webサーバでは、CGIエクステンションを使用して他のHTTPサーバと対話することができます（Windowsのみ）。

4D WebサーバからCGIを実行する

Mac OS XおよびWindowsにおいて、4Dはあらゆるタイプの CGIをサポートします。CGIは、アプリケーションやPERLスクリプトであったり、またはWebサーバとの対話を行うDLLの場合もあります。

■ 実行形式（.EXE）：“コンソール（標準入出力）”と環境変数を使用します。通常、ソースコードはクロスプラットフォームです（WindowsおよびUnix）。CGI名は一般的に、nnn.exeまたはnph-nnn.exeとなります。この種類のCGIに関する詳細は、インターネットサイトhttp://hoohoo.ncsa.uiuc.edu/cgi/ を参照してください。

■ DLL ISAPI：IIS（Internet Information Server）用のエクステンションです。CGI名はnnn.dll またはnph-nnn.dllとなります。Webサーバが処理実行のため中断されると、DLLがダウンロードされます。
この種類のCGIに関する詳細は、インターネットサイトhttp://www.microsoft.com/iis/を参照してください。

■ PERLスクリプト：“コンソール”と環境変数を使用します。CGIの実行にはインタプリタが必要です。しかし、これらはクロスプラットフォームです（Windows、UnixおよびMac OS）。CGI名はnnnn.pl、nph-nnnn.pl、nnnn.cgiまたはnph-nnnn.cgiとなります。
この種類のCGIに関する詳細は、インターネットサイトhttp://www.perl.com/ を参照してください。

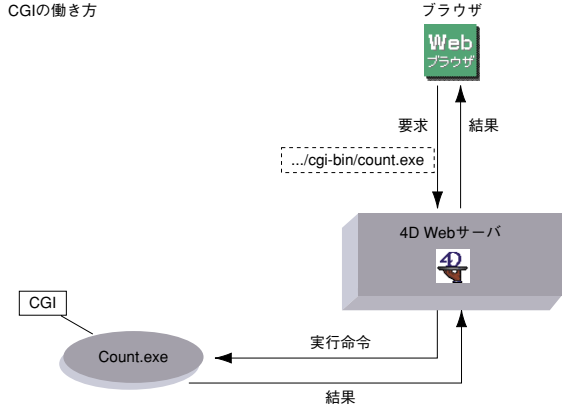
自動モードでCGIを実行する

自動モードでのCGIの呼び出しは、CGIによって実行されるタスクに応じて、ページに挿入されたURLやアクション、またはHTMLタグを介して行われます。いかなる場合でも、HTML文字列には“/cgi-bin/”に続けてCGI名、あるいはHTML構文を使用したパスと検索文字列を含む必要があります。

例えば、“http://195.1.2.3/cgi-bin/search.exe”というURLにより、search.exeというCGIが起動されます。同様に、<IMG SRC="/special/cgi-bin/counter.exe">というマーカーがHTMLページに置かれている場合、このページが送信されるとcounter.exeというCGIが起動されます。

CGIは、「cgi-bin」という名前のフォルダのルートに配置する必要があります。このフォルダは、Webサーバのルート、またはサブフォルダに置きます。サーバには複数のcgi-binフォルダを保持しておくことができます。また、このフォルダには実行形式のアプリケーション以外のファイルを入れておくことができますが、最新のものだけがWebクライアントから呼び出されます。

CGI“Count.exe”を使用したインストールの例



いくつかの例、または場所と対応するURLを以下に示します。

項目の場所	対応するURL
（Webサーバルート）	
[mybase]フォルダ	
mybase.4db（ストラクチャ）	(http://195.1.2.3/)
[cgi-bin]フォルダ	
counter.exe	(http://195.1.2.3/cgi-bin/counter.exe)
[Misc]フォルダ	
[cgi-bin]フォルダ	
script.pl	(http://195.1.2.3/Misc/cgi-bin/script.pl)

手動モードでCGIを実行する

手動モードでのCGIの呼び出しには、**SET CGI EXECUTABLE**コマンドを使用する必要があります。特に、このコマンドを使用すると、Webユーザに対してURL上に表示されないようにCGIを実行することができます。

この件に関する詳細は、このコマンドの説明を参照してください。

4D WebサーバとCGIとの対話

CGIの呼び出しによって、4D環境が変更されることはありません（セレクション、変数等）。

4Dは応答サイズの制限を行いませんが、CGIに割り当てられる最大処理時間には 30 秒という制限がある点に注意してください。この時間を超えると、Webサーバはエラーを返します。

呼び出し元のモードに関わらず、CGIは常にコンテキストを使用せずに実行されます。しかし、コンテキストモードでは、CGIがコンテキストを非同期化してしまう可能性があるため、HTMLコードを送り返すCGIは使用しないようお勧めします。

4Dから返されるCGI呼び出しに関するエラー（WindowsおよびMac OS）

CGIの呼び出しによりエラーが発生すると、4Dは以下の応答のいずれか1つを標準のHTMLページに納めて返します。

■ Not found（見つからない）：4DはCGIを見つけられません、またはPERLインタプリタが見つかりません。

■ Forbidden（禁止）：Webクライアントは、[cgi-bin]フォルダ内の実行形式ファイル以外のものを要求しています。

■ Timeout（タイムアウト）：リクエストは、CGIにより 30 秒以内に処理されませんでした。

■ Bad Answer（不当な応答）：4DはCGIの応答を処理できません、またはISAPI DLLが例外処理を引き起こしました。

■ Internal Error（内部エラー）：メモリーフルなど

注：CGIが動作しない場合は、CGIに関する実行権が適切であるか、またCGIスクリプトの改行が正しいかどうかを調べてください。

ISAPI DLLに関する4Dの保護（Windowsのみ）

ダイレクトに呼び出されていても、ISAPI DLLが例外処理（ユーザ定義によらない処理）を引き起こした場合に、4Dが中断されることはありません。4Dは“Bad Answer”エラー（DLLは応答できません）を返すだけです。

CGIディベロッパへの情報

この節は、4Dデータベース用に特定のCGIを開発しようとするプログラマを主に対象としています。

環境変数

CGI/1.1仕様、および以下の情報に従って環境変数の定義を行います。

GATEWAY_INTERFACE：常に“CGI/1.1”

SERVER_SOFTWARE：常に“4D WebStar_D/version”

SERVER_PROTOCOL：常に“HTTP/1.0 ”

SERVER_PORT_SECURE：HTTP接続が暗号化モードである場合には“1”が、それ以外は“0”が納められます。

PATH_TRANSLATED：HTMLサーバルートへのフルパス、およびCGI名に続けてパス部分が納められます。セキュリティ上の理由から、この部分には文字シーケンス“//”や“..”を入れることはできません。

例：サーバのルートC:/web

/ccgi-bin/cgi.exe/パスのようなCGI呼び出しに対し、PATH_TRANSLATEDの値は“C:/web/パス”になります。/cgi-bin/cgi.exe/../パスのようなCGI呼び出しに対して、4Dは“Forbidden”エラーを返します。

REMOTE_IDENT：ユーザ名（適切であれば）、それ以外は未定義

HTTP_AUTHORIZATION、HTTP_CONTENT_LENGTH、HTTP_CONTENT_TYPE：未定義

ALL_HTTPおよびURLは、ISAPI DLL呼び出しの場合に定義されます。

CERT_xxxおよびHTTPS_xxxは、接続が保護されている場合に定義されます（DLLのみ）。

注：SET ENVIRONMENT VARIABLEコマンドを使用して、これらの変数を設定することができます。

標準の環境変数に加え、4Dではテキスト変数のFORMVAR_変数名が提供されます。

■ リクエストが「POST」メソッドで送信された場合、バイナリフィールド（INPUT TYPE="FILE"）を除き、これらの変数にはフォームの入力エリア（例えば、FORMVAR_NAME、FORMVAR_FIRSTNAME等）が代入されます。このシステムは、“www/url-encoded”および“multipart/form-data”暗号化フォームとともに使用できます。

■ リクエストが「GET」メソッドで送信された場合、これらの変数にはリクエストの文字列に渡された値が代入されます（例えば、.../cgi.exe?name=smith&code=75というURLの場合、FORMVAR_NAMEには“smith”という値が、FORMVAR_CODEには“75”という値が入ります）。

この機能により、フォームの処理はさらに楽になりますが（“a=1&b=2&...”のような文字列を解析する必要がない）、このCGIは4D特定のものになります。

CGIより送信された応答の処理

CGI（Windows実行形式またはPERLスクリプト）の名前が“nph-（No Parsing Header）”で始まる場合、4Dはその応答を“現状のまま”Webクライアントに送信します。この場合、HTTPの規約に準拠するかどうかはCGI次第です。ISAPI DLLに関しては、4Dは名前の先頭が何であろうと、その応答を解析しません。

上記以外の場合、4DはHTTPヘッダを送信します。

■ “Content-Type”がCGIにより指定されていない場合、4Dは常に“Content-Type: text/html”を送信します。

■ “Location”が指定されている場合、4Dはこの応答の他の要素を考慮せずに、HTTPのリダイレクションを実行します。

■ “Status”が指定されていない場合、4Dは“HTTP/1.0 200 OK”を送信します。

4Dは、HTTP応答のヘッダ内にある、あらゆる種類の改行コンビネーション（Windows-CRLF、Mac OS-CR、Unix-LF）を受け付け、再フォーマットします。

ISAPI DLLに関して4Dは非同期処理（HttpExtensionProcはHSE_STATUS_PENDINGを返します）を受け付けます。**ServerSupportFunction**（HSE_REQ_DONE_WITH_SESSION）の呼び出しは、必ず次の30秒の間に行われなければなりません。関数**Terminate Extension**が定義されると、常にHSE_TERM_MUST_UNLOADの値を使って呼び出されます。

CGIを使用した4D Webサーバ呼び出し（Windowsのみ）

4th Dimensionでは、2つのエクステンション、4DISAPI.DLLとNPH-CGI4D.EXEが提供されます。これらのエクステンションは、HTTPサーバが4DのHTTPサーバへリクエストを送信できるように設計されています。例えば、非暗号化モードの4D Web サーバは、暗号化モードで動作する他のHTTPサーバを介して問い合わせを行うことができます。

警告：これら2つのエクステンションはWindowsでのみ使用できます。

■ 4DISAPIエクステンションは、ISAPI（Internet Services Application Programming Interface）によって定義された仕様に準拠しています。ISAPIテクノロジーは本来Microsoft ィ社によってIISサーバ用に開発されましたが、Netscape®、Apache®、Sambar®等の各種HTTPサーバと互換性があります。

■ NPH-CGI4D.EXEエクステンションは、CGI（Common Gateway Interface）仕様に準拠しており、すべてのCGI互換サーバで使用できます。

これら2つのエクステンションは同じような働きをします。CGI互換はHTTPサーバで広く使用されていますが、CGIエクステンションのパフォーマンスは通常、ISAPIのものよりは劣ります。

動作方法

これらのエクステンションは次のように働きます。

HTTPサーバ“A”はインターネット上にページを発行し、別のHTTPサーバ“B”はイントラネットで使用されている4D Serverであるものとします。この2つのサーバが通信できるようにするには、サーバ“A”の[Scripts]ディレクトリに、4DISAPIまたはNPH-4DCGIエクステンションを追加するだけです。

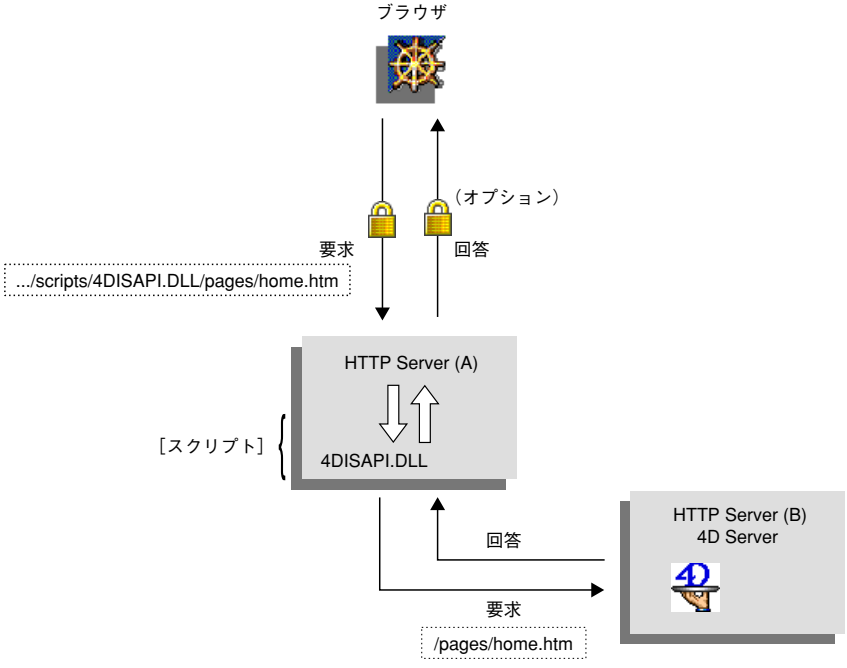
Webブラウザがサーバ“A”にリクエストを送信すると、サーバ“A”はURLを介し、4DISAPIまたはNPH-4DCGIエクステンションを使用してこのリクエストをサーバ“B”に転送します（通常、このエクステンションはサーバ“B”に対し、リクエストアイテムを含め、呼び出しの後に置かれたURL部分を転送します）。その後、応答がブラウザに返されます。CGI名がNPH（No Parsing Header）で始まる場合、サーバは応答のHTTPヘッダを解析する必要はなく、CGIがフォーマットを行います。

このエクステンションがHTTPリクエストや応答のボディを変更することはありません。

サーバ“A”への最初のリクエストは、ノーマルモードまたは暗号化モード（SSL）で送信することができます。2つのHTTPサーバと4DISAPI.DLLエクステンションとの通信は、非暗号化モードで行われます。

注：4DISAPIおよびNPH-4DCGIエクステンションは、4D Webサーバのコンテキストモードと互換性がありません。

次の図は、この原理を示しています。



■ エクステンションは、「GET」、「HEAD」および「POST」メソッドを識別し、これらのメソッドは4Dより返される各種ステータス（200 OK、302 Moved Temporarily、404 Not Found等）を管理します。

■ 4DISAPIやNPH-CGI4Dエクステンションを介し、HTTPレベルで認証を行うことはできません。これを行うには、HTMLフォームを使用する必要があります（暗号化接続で実行可能）。

注：これらのエクステンションは、ダイナミックデータの送受信、特にデータのpostを行えるように設計されています。ISAPIやCGIエクステンションを使用する際、ベーシックなWebページサービスでは良好なパフォーマンスが得られません。

インストールと環境設定

4DISAPIおよびNPH-CGI4Dエクステンションをインストールするには、4DISAPI.DLLまたはNPH-CGI4D.EXEファイルをHTTPサーバの[Scripts]フォルダにコピーします。

インストールしたエクステンションファイルはそれぞれ、環境設定ファイル（.INIファイル）とともに提供されます。.INIファイルには、エクステンションと同じ名前が付きます（例えば、4DISAPI.INI）。エクステンションファイルと環境設定ファイルは同じフォルダに入れます。

HTTPサーバは、別の複数のHTTサーバを対象とするように設定できます。この場合、HTTPサーバの[Scripts]フォルダには、対象となるサーバの数だけのエクステンションをコピーしてください。後はこれらのファイル名を変更するだけです（例えば、4DISAPI2.DLL、4DISAPI3.DLL等）。その際、環境設定ファイルが各エクステンションに関連付けられており、その名前が正しいことを確認してください（4DISAPI2.INI、4DISAPI3.INI等）。

.INIファイルには、1つのセクション[Forward]が含まれます。[Forward]は以下のコマンドを許可します。

■ TargetServer =

対象とするWebサーバのIP名またはIPアドレスです（例えば、myserver.netや192.193.194.195）。名前によって決定できない場合、この行をブランクのままにして、アドレスで対象サーバを決めることができます。

“ローカルホスト”名はアドレス127.0.0.1で識別されます。

デフォルトとしてアドレス127.0.0.1が使用されます。

■ TargetPort =

対象サーバが使用するポートです（例、81）。デフォルトではポート番号8080が使用されます。

■ Timeout =

サーバ応答の最大タイムアウトです（秒単位）。デフォルト値は30秒です。

■ Allowed =

許可されるURLのリストで、コンマで区切られます。例えば、/pages, /imgの場合、/pagesおよび/imgで始まるURLにのみアクセスを許可します。サイト全体へのアクセスを許可するには、スラッシュ記号（/）を入力します（デフォルト設定）。

■ Forbidden =

禁止されるURLのリストで、コンマで区切られます。例えば、/4DMETHOD, /pages2の場合、/4DMETHODおよび/pages2で始まるURLへのアクセスを禁止します。無制限のアクセスを許可するには、このリストに何も入力しないでください（デフォルト設定）。

上記のルールに従い、以下のURLへのアクセスの認否を示します。

/pages/document.html	アクセス可
/pages1/onepage.html	アクセス可
/www/picture.gif	アクセス不可
/pages2/mypage.html	アクセス不可
/4dmethod/myproc	アクセス不可

禁止されたURLが呼び出されると、エクステンションはエラー“HTTP/1.0 403 Forbidden”をダイレクトに返します。

エクステンションの使用

4DISAPIおよびNPH-CGI4Dエクステンションは、以下のURLをサポートします。

■ 4D呼び出し（4Dはエクステンション名の後にあるURL部だけを受信）
http://server-address/cgi-bin/4disapi.dll/[Path]
http://server-address/cgi-bin/nph-cgi4d.exe/[Path]

■ エクステンションのチェック（リクエストのエコー）

http://server-address/cgi-bin/4disapi.dll/~~echo
http://server-address/cgi-bin/nph-cgi4d.exe/~~echo

■ エクステンションに関する情報（テクニカルサポート）

http://server-address/cgi-bin/4disapi.dll/~~info
http://server-address/cgi-bin/nph-cgi4d.exe/~~info

以下の情報が返されます。

エクステンションの名前およびバージョン。例えば、"Script name:4disapi.dll(6.7.0b1.2)"

エクステンションを呼び出しているサーバの名前およびバージョン。例えば、"Server software:4D_WebStar_D/6.7"。

HTTPプロトコルのバージョン。例えば、"Server protocol:HTTP/1.0"。

CGIプロトコルのバージョン。例えば、"Gateway interface:CGI/1.1"。

■ 対象サーバのチェック（サーバは使用可能か？）

http://server-address/cgi-bin/4disapi.dll/~~target
http://server-address/cgi-bin/nph-cgi4d.exe/~~target

この答えは、次のいずれかになります。

- "Good:target server reached."：対象サーバは応答した（応答内容に関わらず）。
- "Bad:target server not reached."：対象サーバが見当たらない、または応答がない。

この場合、以下の原因が考えられます。

- 環境設定ファイルが見つからない
- ターゲットアドレスまたはポートが正しくない
- ターゲットサーバが接続されていない
- ターゲットサーバはリクエストを受信したが、応答できない

4D WebSTARに関する注意：4D WebSTAR®はMac OS上で人気が高いWebサーバのひとつです。4Dと4D WebSTARとの間でさまざまなやり取りの可能性が考えだされてきました。具体的に言うと、4D Connectおよび4D Linkという4D WebSTARのプラグインがあります。これらのプラグインに関する詳細は、4D WebSTARのドキュメントを参照してください。

参照

SET CGI EXECUTABLE、SET ENVIRONMENT VARIABLE

START WEB SERVER

START WEB SERVER

説明

START WEB SERVERコマンドは、アプリケーション（4th Dimension、4D Server、4D Client）が実行されるマシン上で、4th DimensionアプリケーションのWebサーバを開始します。この結果、イントラネットネットワークまたはインターネット上にデータベースが公開されます。

Webサーバが正常に起動された場合には、システム変数OKに1が設定され、そうでなければシステム変数OKは0（ゼロ）が設定されます。例えば、TCP/IPネットワークプロトコルが正しく設定されている場合には、システム変数OKに0が代入されます。

参照

STOP WEB SERVER

システム変数とシステムセット

Webサーバが正常に開始された場合はシステム変数OKに1、そうでない場合はシステム変数OKに0が設定されます。

STOP WEB SERVER

STOP WEB SERVER

説明

STOP WEB SERVERコマンドは、アプリケーション（4th Dimension、4D Server、4D Client）が実行されるマシン上で、4th DimensionアプリケーションのWebサーバを終了します。Webサーバが開始している場合は、すべてのWeb接続が停止され、すべてのWebプロセスは終了します。

Webサーバが開始していない場合、このコマンドは何も行いません。

参照

START WEB SERVER

SET WEB TIMEOUT

SET WEB TIMEOUT(タイムアウト)

引数	タイプ		説明
timeout	数値	→	Web接続タイムアウトに設定する秒数

説明

SET WEB TIMEOUTコマンドは、コンテキストモードにおけるWeb接続プロセスへのタイムアウトを設定します。タイムアウトのデフォルトは5分です。

引数<パラメータ>に時間を渡すことにより、値を増減することができます。新しいタイムアウトのデフォルトは秒で表されます。

■ **SET WEB TIMEOUT**がWebプロセスから呼ばれると、<タイムアウト>引数はそのプロセスだけ適用されます。

■ **SET WEB TIMEOUT**がWebプロセス以外から呼ばれると、すべての「Web接続」プロセスが影響を受けます。

参照

コンテキストモードの使用、Webサーバセッティング

SET HTML ROOT

SET HTML ROOT(パス名HTML)

引数	タイプ		説明
パス名HTML	数値	→	HTMLファイル用のデフォルトディレクトリに対するHTMLパス名

説明

SET HTML ROOTコマンドを使用すると、**SEND HTML FILE**コマンドの引数としてHTMLファイルを渡す場合に4Dがこのファイルを検索するデフォルトのディレクトリやフォルダを変更することができます。

警告：SET HTML ROOTコマンドはコンテクストモードでのみ機能します。非コンテクストモードでデフォルトのHTMLのルートフォルダを設定するためには、「環境設定」ダイアログボックスの「デフォルトHTMLルート」エリアを使用します。パフォーマンス上の理由から、Webサーバの実行モードに関わらず、通常は「環境設定」でデフォルトのHTMLルートフォルダを設定することをお勧めします。

指定するパス名はHTMLパス名である必要があり、プラットフォームに関係なく、そのパス名はディレクトリまたはフォルダ名をスラッシュ（"/"）文字で区切ります。HTMLパス名の詳細については、書店にあるHTMLに関する書籍のランゲージリファレンスを参照してください。

無効なパス名を指定した場合には、OSのファイルマネージャエラーが生成されます。**ON ERR CALL**メソッドでこのエラーを処理することができます。エラーメソッド内部から警告やメッセージを表示する場合には、ブラウザ側に表示されます。

注意：SET HTML ROOTコマンドは、データベースの「環境設定」で定義されたデフォルトのHTMLルートフォルを考慮します。

例題

▼ **SEND HTML FILE**コマンドの例を参照してください。

参照

ON ERR CALL

エラー処理

無効なパス名を指定すると、OSのファイルマネージャエラーが生成されます。**ON ERR CALL**メソッドでこのエラーを処理することができます。

SET WEB DISPLAY LIMIT

SET WEB DISPLAY LIMIT(レコード数 { ; ページ数 { ; ピクチャ参照番号}})

引数	タイプ		説明
レコード数	数値	→	各HTMLページに表示する最大レコード数
ページ数	数値	→	各HTMLページの下部にあるページ参照の最大数
ピクチャ参照番号	数値	→	フルページレコードボタン用のピクチャ参照番号

説明

SET WEB DISPLAY LIMITコマンドは、ユーザが**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドを呼び出した際に、Webブラウザ側におけるレコードセレクションの4th Dimensionによる表示方法を変更します。このコマンドはコンテキストモードでのみ動作します。

4th Dimensionを使用してレコードセレクションを表示するとき、プログラムはセレクション中のすべてのレコードをロードするわけではなく、一度にウインドウに表示できる数のレコードを（ディスクから）ロードするだけです。そうすることによって、何千件ものレコードからなるセレクションを作成した場合でも、レコードの表示は非常に高速に行われます。その後、ウインドウのスクロールやサイズ変更をすると、4Dはそれに対応してレコードをロードします。

Webでは、4Dはページに表示されるレコードセレクションを分割します。ページング体系がなければ、数千レコードからなるセレクションの場合、結果としてインターネット上またはイントラネット上で1つのWebページだけに数千のレコードを表示することになります。また、これらのレコードのダウンロードにはかなりの時間を要し、Webブラウザのメモリ不足につながります。

デフォルトでは、4th Dimensionはセレクションの最初の20レコードを表示し、また、各HTMLページの最後に、最初の20ページのセレクションへのリンクを20個含みます。つまり、デフォルトでは、各セレクションページの最後にあるページリンク上をクリックすることにより、セレクションの最初の400レコードをブラウズできるということです。このページングシステムはコーディングに対して透過的であることに注意してください。すべての作業は、**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドへの呼び出しの内部で行われます。

SET WEB DISPLAY LIMITコマンドでこれらの設定を変更できます。引数<レコード数>には、各セレクションページに表示したい最大レコード数を指定します。引数<ページ数>には、各セレクションページの最後に配置したい、セレクションページの最大リンク数を指定します。

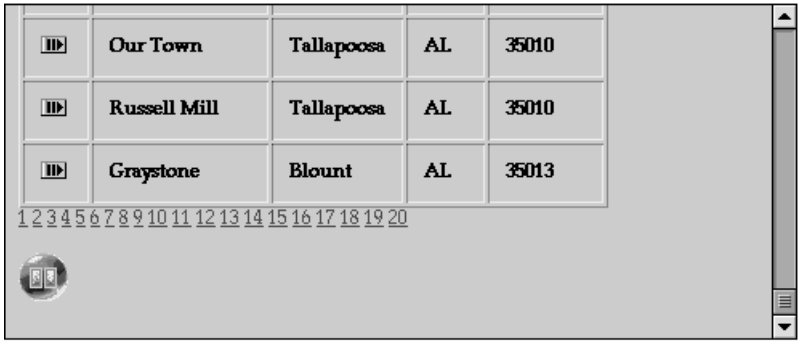
例えば、10,000件のレコードセレクションがあり、1回のセレクション表示ですべてをブラウズしたい場合には、＜レコード数＞=100、＜ページ数＞=100を渡すことができます。ただし、このデータがネットワークやインターネットを経由するということに注意してください。インターネットの場合は、セレクションの表示設定を変更する際に、スピード要因を考慮する必要があります。

さらに、**SET WEB DISPLAY LIMIT**コマンドはオプションとして、フルページレコードボタンのデフォルトアイコンを変更できます。引数＜ピクチャ参照番号＞に、データベースのピクチャライブラリに格納されている、新しいアイコンとして使用したいピクチャのピクチャ参照番号を指定します。

SET WEB DISPLAY LIMITコマンドは、その後の**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドへの呼び出しだけに影響を与え、その有効範囲はカレントプロセス内です。

例題

以下の例では、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドが[郵便番号テーブル用に発行されます。デフォルトで、4Dは以下に示すようにWebブラウザ側にレコードを表示します。



最初の400レコードがブラウズできることに注意してください。

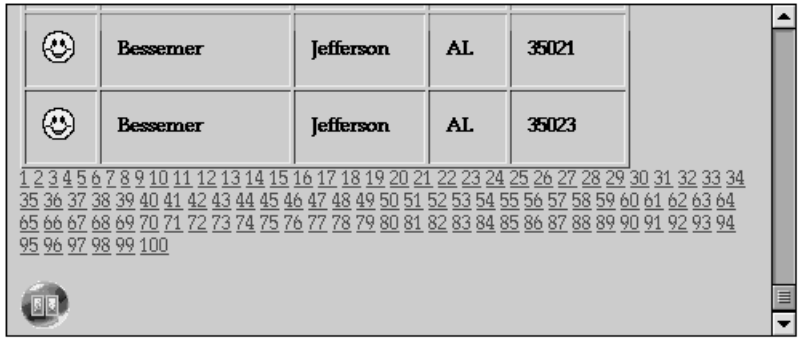
以下のピクチャがデータベースのピクチャライブラリに追加されたものとします。



セレクションを表示するプロジェクトメソッドが次に示す**SET WEB DISPLAY LIMIT**コマンドの呼び出しを実行する場合、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドへの呼び出しの前に以下を行います。

```
SET WEB DISPLAY LIMIT(50 ; 100 ; 17877)
```

次に、Webブラウザ側の選択が、以下のように表示されて終ります。



これで選択の最初の5,000レコードをブラウズできるようになります。

参照

DISPLAY SELECTION、MODIFY SELECTION、コンテキストモードの使用

SET HOME PAGE

SET HOME PAGE(ホームページ)

引数	タイプ		説明
ホームページ	文字列	→	ページ名またはHTMLアクセスパス、またはカスタムホームページを送らないようにする空白文字列

説明

SET HOME PAGEコマンドを使用すると、現在のWebプロセス用のカスタムホームページを変更することができます。

定義されたページはWebプロセスに関連付けられているため、接続されたユーザごとに違うホームページを定義することもできます。このページはスタティックにもセミダイナミックにもどちらにでもなることができます。

HTMLホームページの名前またはページのHTMLアクセスパスをホームページ引数に渡します。デフォルトホームページを有効にするには、空白をホームページに渡します。

注：4Dでは、「環境設定」ダイアログボックスでデフォルトホームページを定義することができます。この場合、Webサーバのスタートアップモード（コンテキストモードまたは非コンテキストモード）に関わらず、デフォルトホームページはすべてのWeb接続に適用されます。

参照

Webサーバセッティング

SEND HTML FILE

SEND HTML FILE(HTMLファイル)

引数	タイプ		説明
HTMLファイル		→	HTMLファイルへのHTMLアクセスパス、または**SEND HTML FILE**を終了させる空白文字列

説明

SEND HTML FILEコマンドは、HTMLドキュメントに保存されたWebページをWebブラウザへ送信します。このドキュメントのパス名は＜HTMLファイル＞に渡します。

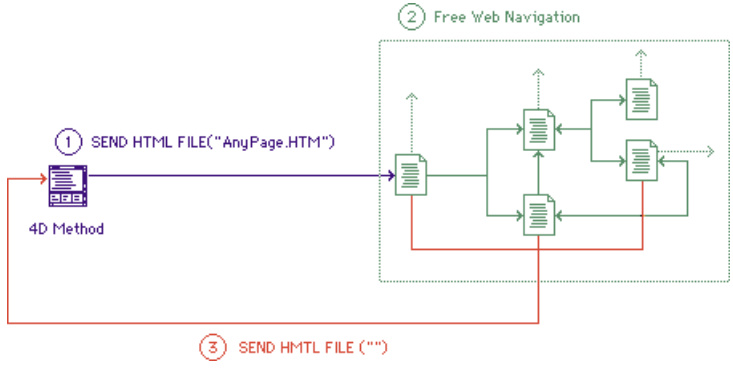
デフォルトとして4th Dimensionは、アプリケーションの「環境設定」で指定したHTMLルートフォルダ内でこのHTMLドキュメントを探します。

このコマンドは、HTML構文で表わされたパス名だけを引数として受け入れます。つまり、プラットフォームに関わらず、ディレクトリ名やフォルダ名はスラッシュ（“/”）で区切らなくてはなりません。

無効なHTMLパス名を指定した場合、4Dは「リクエストされたHTMLページが見つかりません」というメッセージをWebブラウザに送信します。

＜HTMLファイル＞に空の文字列を渡すもう一つのシンタックス、**SEND HTML FILE**("")は、コンテキストモードでHTMLモードを開始した**SEND HTML FILE**コマンドの呼び出しを終了します。

これを次の図で説明します。



1. コンテキストモードにおいて、4Dメソッド（プロジェクト、オブジェクト、またはデータベース）は**SEND HTML FILE**をコールして、HTMLドキュメントをブラウザへ送信します。

2. ブラウザへ送られた最初のWebページは、別のWebページへのHTMLリンクを含むか、またはそのページ内には別のWebページを送信する**SEND HTML FILE**をコールする4Dメソッドへの参照を含みます。さらに、これらの別のページには、別のページをアクセスする4Dメソッドのリンクや参照を含む、というように続きます。Webページを参照し続けている間でも、戻るボタン等でブラウザのナビゲーションコントロールを使用することができます。

3. 任意のWebページに**SEND HTML FILE**（""）を呼び出す4Dメソッドへの参照を組み込むことができます。この呼び出しによって、すべての始まりである**SEND HTML FILE**のコールを終了して戻り、自由なWebナビゲーションの開始元である4Dメソッドの実行を続けます。

SEND HTML FILEコマンドは、システム変数OKをセットするようになりました。送るファイルが存在していて、タイムアウトになっていなければ、システム変数OKは1になります。そうでなければ0になります。

注：WebプロセスではないプロセスからSEND HTML FILEを実行すると、コマンドは何もせずエラーが戻ります。呼び出しは無視されます。

いずれのモードでも、ページ中の4D変数および4DSCRIPTタイプのタグへの参照は、常に解析されます。

1. データベースのHTMLルートフォルダは「WebDocs」フォルダです。このフォルダには次の要素があります。

```
..\WebDocs\HTM\MyPage.HTM
```

Webページ“MyPage.HTM”の送信は、次の方法で実行しなくてはなりません。

```
SEND HTML FILE("HTM/MyPage.HTM")
```

2. コンテキストモードの例題：4D Webセッション中に、4Dフォームを使用してレコードを追加します。このフォームには「bHelp」ボタンがあり、そのオブジェクトメソッドは次の通りです。

```
'bHelpボタンのオブジェクトメソッド
 SEND HTML FILE("Help.HTM")
```

Help.HTMドキュメントから始めた場合、非常に多くのWebサイトからのデータベースのヘルプシステムのあるHTMLページの間でも自由にナビゲートできます。

Submitボタンで、データ入力に戻られます。

このようなHTMLドキュメントはsubmitボタンの定義を含まなければなりません。

```
<!--# bDone submit button ->
<P><INPUT TYPE="submit" NAME="bDone" VALUE="Done"></P>
```

post actionフォームの定義と同様

```
<!--# Execute the 4D htm_Help_Done when a submit button is hit ->
<FORM action="/4DMETHOD/htm_Help_Done" method="POST"
```

4D側では、htm_Help_Doneプロジェクトメソッドが**SEND HTML FILE**の開始をbHelpによってを終了させます。

```
` htm_Help_Doneプロジェクトメソッド
SEND HTML FILE("")
```

BHelpボタンのオブジェクトメソッドでの**SEND HTML FILE**の呼び出しはメソッドの最終行で行います。メソッドが完成した時、データ入力に戻ります。

システム変数とセット

送られたファイルが存在し、タイムアウトが実行されていない場合は、システム変数OKは1がセットされ､それ以外は0がセットされます。

参照

HTMLオブジェクトと4Dオブジェクトのバインド、SEND HTML BLOB、Webサービス：入門編

SEND HTML BLOB

SEND HTML BLOB(blob;タイプ {; 非コンテキスト })

引数	タイプ		説明
blob	BLOB	→	ブラウザへ送る内容を持つBLOB
タイプ	文字列	→	BLOB内のデータのデータタイプ
非コンテキスト	ブール	→	True=非コンテキストモードへの切り替え False=現在のモード維持

説明

このコマンドは、blobの内容をブラウザへ送ります。

blobに含まれるデータのタイプは＜タイプ＞によって示されます。この引数には下記のタイプの内の1つを指定します。

■ type = 空白（""）：この場合、それ以上の情報は何も供給する必要はありません。ブラウザはBLOB内の内容を解釈しようとします。

■ type = ファイル拡張子（例：".HTM"、".GIF"、".JPEG"等）：この場合、BLOB内に含まれるデータのMIMEタイプを拡張子を使って指定します。BLOBの内容は拡張子に応じて解釈されますが、ブラウザが正確に解釈できるように拡張子は標準のものでなければなりません。

■ type = Mime/Type（例："text/html"、"image/tiff"等）：この場合、BLOB内に含まれるデータのMIMEタイプを直接指定します。標準タイプの他に、固有のドキュメントをイントラネット経由で送るためにカスタムMIMEタイプを渡すこともできます。これを実行するには、送られるタイプを認識し、適切なアプリケーションが開けるようにブラウザを設定するだけです。この場合、typeに渡す形式は "application/x-[TypeName]" です。クライアントワークステーションのブラウザでは、このタイプを "アプリケーション起動" 動作に関連付けます。この方法で、**SEND HTML BLOB**コマンドはすべてのタイプのドキュメントを送ることが可能となり、イントラネットクライアントは関連するアプリケーションを自動的に開くことができるようになります。

注：BLOBがタイプ"text/html"（.htm、.html、.shtm、.shtml）である場合、HTMLファイルとして4Dによって内容を解析され、必要な処置が施されます。この場合、コンテクストモードが使われているときは、SEND HTML BLOBはSEND HTML FILEとして正確に機能します。それは、SEND HTML BLOB(“”)コールを発行する4Dのメソッドが、オリジナルのSEND HTML BLOBを終了させるため、HTMLページのひとつから呼び出されます。詳細はSEND HTML FILEコマンドを参照してください。

最も一般的なMIMEタイプのリストです。

拡張子	Mime/Type
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

非コンテキスト引数は、4D Webサーバに「コンテキストモード」から「非コンテキストモード」への切り替えができるようにするものです。非コンテキストモードを使用するには非コンテキストへTrueを渡します。コンテキストモードを使用するにはFalseを渡します。引数が省略されるかFalseが含まれる場合、現在のモードが使用されます。4D変数およびページ内の4DACTIONタグへの参照は、モードがなんであれ、常に解釈されます。

注：「非コンテキスト」モードのサポートはバージョン6.5で新しく追加された機能です。

例題

▼ **PICTURE TO GIF**ルーチンの例を参照してください。

参照

SEND HTML FILE

SEND HTML TEXT

SEND HTML TEXT(htmlテキスト{;非コンテキスト})

引数	タイプ		説明
htmlテキスト	テキスト	→	WebブラウザへHTML送信されるテキストフィールドまたは変数
非コンテキスト	ブール	→	True=非コンテキストモードへ移動 Falseまたは省略=現行のモードを維持

説明

SEND HTML TEXTコマンドは、HTML フォーマットのテキストデータをダイレクトに送信します。

引数＜htmlテキスト＞には、送信するデータを納めます。4Dは引数の内容をチェックしないため、HTMLコードが正しいことを確認してください。

注：このコマンドは、SEND HTML BLOBコマンドで“html/txt”タイプのBLOBを使用する場合とよく似ています。

引数＜非コンテキスト＞により、このコマンドの実行時にコンテキストモードから非コンテキストモードへ切り替えることを4D Webサーバに対して指示します。

非コンテキストモードを使用したい場合、引数＜非コンテキスト＞にはTrueを渡し、現行のモードを使用したい場合にはFalseを渡すか、または何も指定しません。

いずれのモードでも、テキスト中の4D変数および4DSCRIPTタグ（存在する場合）への参照は、常に解析されます。

例題

以下にメソッドを示します。

```
TEXT TO BLOB("<html><head></head><body>"+String(Current time)
                                    +"</body></html>";$blob;Text without length)
SEND HTML BLOB($blob;"text/html")
... 次の 1 行で置き換えることができます。
SEND HTML TEXT("<html><head></head><body>"+String(Current time)
                                                +"</body></html>")
```

参照

Mac to ISO、SEND HTML BLOB

PROCESS HTML TAGS

PROCESS HTML TAGS(入力データ; 出力データ)

引数	タイプ		説明
入力データ	テキストまたはBLOB	→	処理するHTMLタグを含むデータ
出力データ	テキストまたはBLOB	←	処理されたデータ

説明

PROCESS HTML TAGSコマンドにより、4th Dimensionは引数＜入力データ＞（テキストまたはBLOBタイプのフィールドや変数）に格納された4D HTMLの処理を開始し、結果のデータを＜出力データ＞に返します。

このコマンドを使用すると、**SEND HTML BLOB**タイプのコマンドを利用したり、またはURLを用いて接尾辞“.shtml”が付加されたページをリクエストすることにより、Webサーバがページを送信しなくても、タグ付けされたHTMLコードの処理を実行することができます。4th Dimension Webサーバを開始する必要さえありません。

引数＜入力データ＞には、処理するタグを含むデータを渡します。この引数は、テキストやBLOBタイプのフィールドまたは変数です。BLOBには文字数の制限がないため（テキストタイプの場合は32,000桁まで）、一般的にBLOBタイプを使用する方が望ましいということを覚えておいてください。

Webサーバの処理モード（コンテキストまたは非コンテキスト）とは関係なく、またWebサーバが開始していなくても、すべての4D HTMLタグが考慮されます（4DVAR、4DSCRIPT、4DLOOPなど）。

注：Webサーバ（Webプロセス）の枠外で4DINCLUDEタグを使用する場合、

・4th Dimensionまたは4D Serverを使用している場合、デフォルトフォルダはデータベースストラクチャファイルが格納されているフォルダです。

・4D Clientを使用している場合、デフォルトフォルダは4D Clientアプリケーションが格納されているフォルダです。

コマンドの実行後、引数＜出力データ＞には引数＜入力データ＞のデータが返され、さらに、データに含まれているあらゆる4D HTMLタグの処理結果も一緒に代入されます（該当する場合）。引数＜入力データ＞に4D HTMLタグがひとつも含まれていない場合、＜出力データ＞と＜入力データ＞の内容は同じになります。

引数＜出力データ＞はフィールドまたは変数のいずれでも構いませんが、引数＜入力データ＞と同じタイプでなくてはなりません。

このコマンドにより、データベースにあるHTMLタグの処理結果である値が送信される前に、それを保存しておくことができます。

また、このコマンドはWebサーバの使用だけではなく、4D HTMLタグの解析も行えます。特に、コマンドを用いて、データベース内のデータに関する処理手続きや、その参照を含む電子メールメッセージを4D Internet Commandsを介してHTMLフォーマットで送信することができます。

例題

▼次の例題は、このコマンドの機能を示しています。

```
C_BLOB($in)
C_BLOB($out)
C_TEXT($in_text)
C_TEXT(Var)
C_TEXT(HTMLvar)

Var:="<B>"
$in_text:="<p><!--#4DVAR Var--></p>"
TEXT TO BLOB($in_text;$in;Text without length)
PROCESS HTML TAGS($in;$out)
HTMLvar:=BLOB to text($out;Text without length)
      `HTMLvarには<p>&#60;B&#62;</p>が格納される
```

参照

4D HTML Tags

GET WEB FORM VARIABLES

GET WEB FORM VARIABLES(名前配列;値配列)

引数	タイプ		説明
名前配列	テキスト配列	←	Web フォーム変数の名前
値配列	テキスト配列	←	Web フォーム変数の値

説明

GET WEB FORM VARIABLESコマンドは、“サブミットされた”（つまり、Webサーバに送信された）Webフォームの変数の名前および値を、テキスト配列である＜名前配列＞と＜値配列＞に代入します。

このコマンドは、HTMLページに含まれるすべての変数の値を取得します。つまり、テキストエリア、ボタン、チェックボックス、ラジオボタン、ポップアップメニュー、選択リスト等の値です。

注：チェックボックスに関しては、チェックボックスが実際にチェックされている場合にのみ、変数の名前と値（“On”）が返されます。

Webサーバへ送信したURLやフォーム（POSTまたはGETメソッド）の種類とは関係なく、このコマンドは非コンテキストモードまたはコンテキストモードのいずれかで有効です。

■ フォームが「POST」メソッドでサブミットされる場合（/4DACTIONまたは/4DMETHODまたは/4DCGIで開始するアクション）

■ フォームが「GET」メソッドでサブミットされる場合（/4DACTIONまたは/4DMETHODまたは/4DCGIで開始するアクション）

■ リクエストの文字列を含むURLがWebサーバに送信される場合

必要があれば、このコマンドを**On Web Connection**データベースメソッドや、フォームのサブミットの結果として発生するその他の4Dメソッド内で呼び出すことができます。

Webフォームと関連するアクションについての詳細

各フォームには、名前の付いたデータ入力エリア（テキストエリア、ボタン、チェックボックス）が含まれます。

フォームがサブミットされると（リクエストがWebサーバに送信されると）、そのリクエストには(その他の項目の中に）データ入力エリアのリストとこれに関連する値が納められます。

フォームは、次の2種類のメソッドを使用してサブミットできます（両方とも4Dと共に使用できます）。

■ POST：通常はWebサーバへデータを追加するために使用される－データベースへの追加。

■ GET：通常はWebサーバのリクエストに使用される

例題

▼ フォームにはvNameとvCityという 2 つのフィールドがあり、それぞれ“ROBERT”と“DALLAS”という値が納められています。このフォームに関連付けられたアクションは、“/4DACTION/WEBFORM”です。

■ フォームメソッドが「POST」（最も頻繁に使用される）の場合、入力データはURL上には表示されません（http://127.0.0.1/4DACTION/WEBFORM）。

■ フォームメソッドが「GET」の場合、入力データはURL 上に表示されます（http://127.0.0.1/4DACTION/WEBFORM?vNAME=ROBERT&vCITY=DALLAS ）。

WEBFORMメソッドは次のようになります。

```
ARRAY TEXT($anames;0)
ARRAY TEXT($avalues;0)
GET WEB FORM VARIABLES($anames;$avalues)
```

結果は次の通りです。

```
$anames{1} = "vNAME"
$anames{2} = "vCITY"
$avalues{1} = "ROBERT"
$avalues{2} = "DALLAS"
```

変数vNAME には“ROBERT”が、変数vCITY には“DALLAS”が納められます。

参照

HTMLオブジェクトと4Dオブジェクトのバインド、URLとフォームアクション

Web Context

Web Context → ブール

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	ブール	←	True=コンテキストモード False=非コンテキストモード

説明

この関数はWebプロセスから呼び出されなければなりません。これは、Webの接続がコンテキストモード（True）で実行されているのか、それとも非コンテキストモード（False）で実行されているのかを返します。

注：次の場合、Web Context関数は常に“False”を返します。

・Webプロセスではないプロセスから呼び出された場合

・4D Clientマシン上で実行された場合

この関数の使用は、**On Web Connection**データベースメソッド内で行なうことをお勧めします。

例題

▼ **On Web Connection**データベースメソッドの例を示します。

```
If(Web Context)
    WithContext($1;$2;$3;$4;$5;$6)
Else
    WithoutContext($1;$2;$3;$4;$5;$6)
End if
```

参照

On Web Connection Databaseメソッド、PROCESS PROPERTIES、コンテキストモードの使用

SET HTTP HEADER

SET HTTP HEADER(ヘッダー | フィールド{; 数値})

引数	タイプ		説明
ヘッダー	文字列	→	HTTPヘッダリクエストまたはHTTPヘッダ
フィールド			フィールドを含むフィールドや変数
数値	文字列	→	HTTPヘッダ内のフィールド

説明

このコマンドは、4DによってWebブラウザへ送り返されるHTTPヘッダ内のフィールドを設定できるようにするものです。これは、非コンテキストモードでのWebプロセスにのみ影響します。

このコマンドは、"cookies"の管理も可能にします。

このコマンドは2通りの書き方ができます。

構文 1 ：SET HTTP HEADER(header)

HTTPヘッダ内のフィールドを設定したTextタイプ（変数またはフィールド）のデータを引数フィールドへ渡します。

この構文により”HTTP/1.0 200 OK”+Char(13)+”Set-Cookie:C=HELLO”のようなヘッダータイプを記述することができます。

WindowsやMac OSでヘッダーフィールドはCRかCR+LF(Carriage return + Line feed)の順で分割されます。

▼ カスタム“cookie”の例を以下に示します。

```
C_TEXT($vTcookie)
$vTcookie:="SET-COOKIE: USER="+String( Abs( Random))+"; PATH=/"
SET HTTP HEADER($vTcookie)
```

注：このコマンドは、引数＜ヘッダ＞としてリテラルテキストタイプの定数を受け付けません。必ず4D変数またはフィールドを指定しなければなりません。

構文についてのより詳しい情報は、インターネットアドレスhttp://www.w3c.org で得られるR.F.Cs（Request For Comments）を参照してください。

注：フィールドは常にcr/lfシーケンス（キャリッジリターン／ラインフィード）で分離されていなければなりません。構文についてのより詳しい情報は、インターネットアドレス www.w3c.orgで得られる R.F.Cs（Request For Comments）を参照してください。

構文 2 ：SET HTTP HEADER(fieldArray; valueArray)

HTTPヘッダーはふたつのテキスト配列、フィールド配列と変数配列によって定義されます。ヘッダーは次のように記述されます。

fieldArray{1} = “X-VERSION” valueArray{1} = “HTTP/1.0”*
fieldArray{2} = “X-STATUS” valueArray{2} = “200 OK”*
fieldArray{3} = “Set-Cookie” valueArray{3} = “C=HELLO”

*はじめのふたつの項目は返答の一行目です。それらが入力されたとき、配列の１番目、２番目の項目となります。しかし、これらは記述するだけ、または省略することができます。

4Dはヘッダーフォーマットに注意します。

fieldArray{1} = “Set-Cookie” valueArray{1} = “C=HELLO”

通信プロトコルを指定しないと自動的に“HTTP/1.0 200 OK”が設定されます。

いくつかの**SET HTTP HEADER**は同じウェブプロセス中でoccurを呼ぶとき、最後の呼び出しがアカウントに加えられます。

サーバ、日付やConten-lengthフィールドは常に4Dが設定します。

参照

GET HTTP HEADER

GET HTTP HEADER

GET HTTP HEADER(ヘッダ|フィールド配列{;値配列})

引数	タイプ		説明
ヘッダ¦	テキスト	←	リクエストのHTTPヘッダ
フィールド配列	テキスト配列	←	HTTPヘッダフィールド
値配列	テキスト配列	←	HTTPヘッダフィールド内容

説明

GET HTTP HEADERコマンドは、1つの文字列、あるいは2種類の配列を返し、現在処理中のリクエストで使用されるHTTPヘッダを提供します。

このコマンドは非コンテキストモードでのみ有効です。非コンテキストモードのWeb プロセス内で実行されるあらゆるメソッド（**On Web Connection**、**On Web Authentication**データベースメソッド、または「/4DACTION」でコールされたメソッド）から呼び出すことができます。**GET HTTP HEADER**コマンドがコンテキストモードで呼び出されると、空の文字列を返します。

■ 第一のシンタックス：**GET HTTP HEADER**(ヘッダ)

このシンタックスを使用すると、＜ヘッダ＞の変数に返される戻り値は次のようになります。

```
"GET /page.html HTTP/1.0"+Char(13)+Char(10)+"User-Agent: browser"+
                                Char(13)+Char(10)+"Cookie: C=HELLO"
```

WindowsおよびMac OSにおいて、各ヘッダフィールドは、「CR+LF」（キャリッジリターン+ラインフィード）シーケンスで区切られます。

■ 第二のシンタックス：**GET HTTP HEADER**(フィールド配列;値配列)

このシンタックスを使用すると、＜フィールド配列＞と＜値配列＞に返される戻り値は次のようになります。

フィールド配列{1} = "X-METHOD"	フィールド配列{1} = "GET" *
フィールド配列{2} = "X-URL"	フィールド配列{2} = "/page.html" *
フィールド配列{3} = "X-VERSION"	フィールド配列{3} = "HTTP/1.0" *
フィールド配列{4} = "User-Agent"	フィールド配列{4} = "browser"
フィールド配列{5} = "Cookie"	フィールド配列{5} = "C=HELLO"

*最初の3つの項目は、HTTPフィールドではありません。リクエストの1行目の一部分です。

HTTPの標準に合わせるため、フィールド名は常に英語で記載されます。

以下に、リクエストで使用されるHTTPフィールドをいくつか示します。

■ Accept：ブラウザで許可されるコンテンツ

■ Accept-Language：ブラウザで使用できる言語（情報として）。ブラウザで定義した言語を使用してWebページの選択が可能

■ Cookie：cookieリスト

■ From：ブラウザユーザの電子メールアドレス

■ Host：サーバの名前またはアドレス（例えば、「http://www.mycomp.co.jp/mypage.html」というURLを使用すると、Hostは値として<<www.mycomp.co.jp>>を取得）。同じIPアドレスを指す複数の名前を管理可能（仮想ホスト処理）

■ Referer：リクエスト元（例えば、http://www.mycomp.co.jp/mypage1.html）、つまり「戻る」ボタンをクリックした際に表示されるページ

■ User-Agent：ブラウザまたはプロキシの名前とバージョン

例題

次のメソッドを使用して、任意のHTTPリクエストヘッダフィールドのコンテンツを取得することができます。

```
` GetHTTPFieldプロジェクトメソッド
` GetHTTPField(テキスト) -> テキスト
` GetHTTPField(HTTPヘッダ名) -> HTTPヘッダコンテンツ
C_TEXT($0;$1)
C_LONGINT($vlItem)
ARRAY TEXT($names;0)
ARRAY TEXT($values;0)
$0:=""
GET HTTP HEADER($names;$values)
$vlItem:=Find in array($names;$1)
If($vlItem>0)
    $0:=$values{$vlItem}
End if
```

このプロジェクトメソッドを作成したら、次のように呼び出すことができます。

```
` Cookieヘッダコンテンツ
$cookie:=GetHTTPField("Cookie")
```

▼ ブラウザで設定された言語に応じて、異なるページを送信できます（例えば、**On Web Connection**データベースメソッドにおいて）。

```
$language:=GetHTTPField("Accept-Language")
Case of
	¥($language="@fr@")		`フランス語（ISO 639 リスト参照）
		SEND HTML FILE("index_fr.html")
	¥($language="@sp@")		`スペイン語（ISO 639 リスト参照）
		SEND HTML FILE("index_es.html")
	Else
		SEND HTML FILE("index.html")
End case
```

注意：Webブラウザはデフォルトでいくつかの言語の定義することができます。それらは";"で区切られた「受け入れられた言語」フィールドにリストされます。優先順位は文字列上の位置により決められます。したがって言語の文字列上の位置を分析してもよいでしょう

▼ 以下は、仮想ホスト処理の例です（例えば、**On Web Connection**データベースメソッドにおいて）。次の名前、“home_site.com”、“home_site1.com”、“home_site2.com”は、192.1.2.3 のような同じIPアドレスを示しています。

```
$host:=GetHTTPField("Host")
Case of
	¥($host="www.site1.com")
		SEND HTML FILE("home_site1.com")
	¥($host="www.site2.com")
		SEND HTML FILE("home_site2.com")
	Else
		SEND HTML FILE("home_site.com")
End case
```

参照

GET HTTP BODY、SET HTTP HEADER

GET HTTP BODY

GET HTTP BODY(ボディ)

引数	タイプ		説明
ボディ	BLOBまたはText	←	HTTPリクエストのボディ部

説明

GET HTTP BODYコマンドは、処理されるHTTPリクエストのボディ部（body）を返します。HTTPリクエストのボディは、処理や解析が行われずにそのままの状態で返されます。

このコマンドは非コンテキストモードでのみ動作し、Webデータベースメソッド（**On Web Authentication**、**On Web Connection**）または非コンテキストモードで実行される任意のWebメソッドから呼び出すことができます。

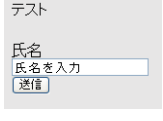
＜ボディ＞には、BLOBまたはテキストタイプのフィールドや変数を渡すことができます。BLOBには文字数の制限がないため、一般的にBLOBタイプを使用する方が望ましいということを覚えておいてください。これに反してテキストタイプの場合は、32,000桁までに制限され、この桁数を超えると、受信されたデータのうち超過した部分が切り捨てられます。

このコマンドを使用して、リクエストのボディ部でクエリを実行することができます。また、上級ユーザの場合は、4th Dimensionデータベース内でWebdevサーバを設置することも可能です。

例題

▼ この例題では、4th Dimension Webサーバへシンプルなリクエストを送信し、デバッガ上にHTTPリクエストのボディの内容を表示します。4th Dimension Webサーバへ送信されたフォームならびに対応するHTMLコードを次に示します。

フォーム



ボディ

```
<HTML>
<HEAD>
 <TITLE>テストページ</TITLE>
 <META http-equiv="content-type" content="text/html;charset=shift_JIS">
</HEAD>
<BODY>
テスト

<FORM ACTION="/4DACTION/Web_test" METHOD="post">
<br>氏名</br>
<input type="text" value="氏名を入力" name="vName">
<br>
<input type="submit">
</FORM>
</BODY>

</HTML>
```

次に示すのはTest4D2004メソッドです。

```
C_BLOB($request)
C_TEXT($requestText)
```

```
GET HTTP BODY($request)
$requestText:=BLOB to text($request;Text without length)
SEND HTML FILE("page.html")
```

注：このメソッドに対して「4DACTION、4D METHODおよび4D SCRIPTで利用可能」プロパティが定義されています。

このフォームをWebサーバへサブミットすると、変数$requestTextにはHTTPリクエストのボディのテキストが返されます。

参照

GET HTTP HEADER

SEND HTTP RAW DATA

SEND HTTP RAW DATA(データ{;*})

引数	タイプ		説明
データ	BLOB→		送信するHTTPデータ
*	*	→	“chunked（塊）”送信

説明

このコマンドを使用して、4D Webサーバは“raw（ロウ）”HTTPデータを送信することができます。また、このデータは“chunked（塊）”形式にすることができます。このコマンドは非コンテキストモードでのみ動作します。

引数<データ>には、HTTPレスポンスの2つの標準部分、つまりヘッダ部とボディ部を格納します。データは、サーバによる事前のフォーマット処理が行われないまま送信されます。しかし、4th Dimensionはレスポンスのヘッダとボディに関する簡単なチェックを実行して、これらのデータが有効であることを確認します。

■ ヘッダが不完全であったり、HTTPプロトコル仕様に準拠していない場合、4th Dimensionは適宜に変更を行います。

■ HTTPリクエストが不完全である場合、4th Dimensionは不足している情報を追加します。例えば、リクエストのリダイレクトを行いたい場合は、次のように記述しなくてはなりません。

```
HTTP/1.1 302
Location:http://...
```

しかし、次の文だけを渡した場合、

```
Location:http://...
```

4th Dimensionは“HTTP/1.1 302”を付加してリクエストを完成させます。

任意の引数<*>を使用して、レスポンスを“chunked（塊）形式”で送信するように指定することができます。サーバがその合計長を認識せずにレスポンスを送信する場合は、レスポンスをチャンクに分けると便利です（例えば、レスポンスがまだ生成されていない場合）。HTTP/1.1に対応するブラウザはすべてチャンク形式のレスポンスを受け入れます。

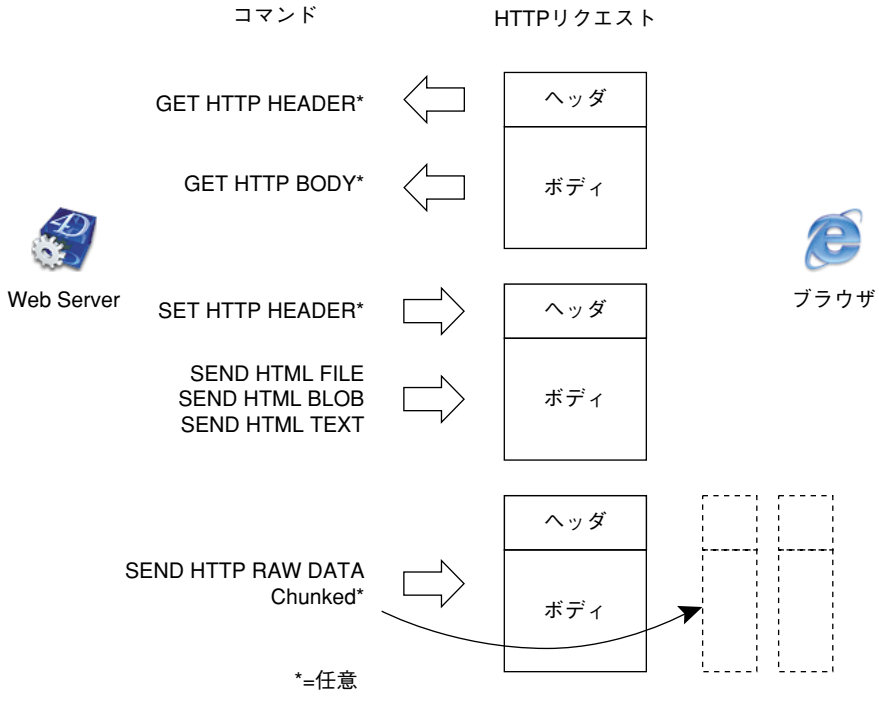
引数<*>を渡すと、必要に応じてWebサーバは自動的に「transfer-encoding: chunked（チャンク形式エンコーディング）」フィールドをレスポンスのヘッダに組み込みます（希望する場合は、レスポンスヘッダを手動で処理することが可能）。レスポンスの残りの部分もまた、“chunked”オプションのシンタックスに合うようにフォーマットされます。“chunked”形式のレスポンスには、1つのヘッダと不定数のボディのチャンクが納められます。

同一メソッド内において**SEND HTTP RAW DATA**(data;*) の実行に続けて行われるすべての**SEND HTTP RAW DATA**命令は、レスポンスの一部とみなされます（引数<*>が指定されているかどうかに関わらず）。このメソッドの実行が完了すると、サーバは“chunked”形式の送信を終了します。

注：WebクライアントがHTTP/1.1をサポートしていない場合、4th Dimensionはそのレスポンスを自動的にHTTP/1.0互換フォーマットへ変換します（送信されるデータは“chunked”形式にはならない）。ただし、この場合は望ましい結果にならない可能性があります。したがって、WebブラウザがHTTP/1.1をサポートするかどうかを調べて、適切なレスポンスを送信することをお勧めします。これを実現するため、次のようなコードを使用することができます。

```
C_BOOLEAN($0)
ARRAY TEXT(arFields;0)
ARRAY TEXT(arValues;0)
GET HTTP HEADER(arFields;arValues)
$0:=False
If(Size of array(arValues)>=3)
	If(Position("HTTP/1.1";arValues{3})>0)
		$0:=True	` ブラウザは HTTP/1.1をサポートする
	End if
End if
```

このコマンドを**GET HTTP BODY**コマンドや「Webサーバ」テーマの他のコマンドと組み合わせて使用すると、4D開発者が利用できるツールの種類がすべて揃い、受信側と送信側のHTTP接続の処理を完全にカスタマイズすることができます。これらの各種ツールを次の図に示します。



例題

▼ この例題は、**SEND HTTP RAW DATA**コマンドとともに“chunked”オプションを使用する例を示しています。ループにおいて処理中に生成された100個のチャンクとしてデータが送信されます。レスポンスのヘッダは明示的に設定されないということを覚えておいてください。つまり、引数<*>が使用されているため、**SEND HTTP RAW DATA**コマンドはヘッダを自動的に送信し、「transfer-encoding:chunked」フィールドをその中に挿入します。

```
C_LONGINT($cpt)
C_BLOB($my_blob)
C_TEXT($output)
For($cpt;1;100)
    $output:="["+String($cpt)+"]"
    TEXT TO BLOB($output;$my_blob;Text without length)
    SEND HTTP RAW DATA($my_blob;*)
End for
```

参照

GET HTTP BODY、GET HTTP HEADER

SEND HTTP REDIRECT

SEND HTTP REDIRECT(url {; *})

引数	タイプ		説明
url	文字列	→	新しいURL
*	*	→	指定されている場合=4Dは URLをエンコードしない 省略されている場合=4Dは URLをエンコードする

説明

このコマンドは、URLのリダイレクトを可能にします。

引数＜url＞には、リクエストを転送する新しいURLを渡します。この引数がファイルへのURLである場合、このファイルへの参照を含まなくてはなりません。

例：**SEND HTTP REDIRECT**（"/MyPage.HTM"）

このコマンドがコンテキストモードで呼び出されると、実行された直後にWebプロセスはアボートされ、Webライセンスは開放されます。このコマンドは、同じメソッド内にあるデータを送るコマンド（**SEND HTML FILE**、**SEND HTML BLOB**等）より優先されます。

さらに、このコマンドを使用すると、他のWebサーバへリクエストを転送することができます。

4DはURLを自動的にエンコードしますが、*を渡すとエンコードしません。

例題

コマンドを使用して、スタティックページを用いて4D内でカスタムリクエストを実行することができます。スタティックHTMLページ内に下記のエレメントを置いたと仮定します。



注：POSTアクション"/4dcgi/rech"はテキストエリアとOKボタン、CANCELボタンに関連づけられています。

非コンテキストモードを管理する**On Web Connection**データベースメソッド部分（またはサブパート）では、下記のコードを挿入します。

```
Case of
	￥($1="/4dcgi/rech")`4Dはこのurlを受け取ると
		`OKボタンが使用され、'name'フィールドにavalueがある場合
	If((bOK="OK") &(name # ""))
		`同じメソッド内のはるか下に置かれた
		`リクエストコードを実行するURLを変更する
		SEND HTTP REDIRECT("/4dcgi/rech?"+name)
	Else
		`そうでなければ、始めのページに戻る
		SEND HTTP REDIRECT("/page1.htm")
	End if
:
￥($1="/4dcgi/rech?@") `URLがリダイレクトされている場合
	... `リクエストコードはここに挿入する
END case
```

参照

なし

WEB CACHE STATISTICS

WEB CACHE STATISTICS(ページ ; ヒット数 ; 使用率)

引数	タイプ		説明
ページ	テキスト配列	←	キャッシュされている参照されたページの名前
ヒット数	倍長整数配列	←	各ページのヒット数
使用率	数値	←	キャッシュの使用率

説明

このコマンドは、Webサーバのキャッシュ内にロードされている、ページについての情報を得ることができるようにするものです。そのため、これらの統計はスタティックホームページ、GIFピクチャ、JPEGピクチャ（100KB未満の）およびスタイルシート（.css）のみに適応されます。

コマンドは、キャッシュされている参照されたページの名前でテキスト配列を作成します。倍長整数配列には各ページのヒット数がセットされ、＜使用率＞引数にはWebキャッシュの使用率がセットされます。

例題

▼ Webキャッシュの統計を表示するセミダイナミックページを作成したいと仮定します。このためには、"stats.shtm"1と名付けられたスタティックHTMLページ内に、タグ<!--#4DACTION/STATS-->を置き、2つの4D変数vPageおよびvUsageを挿入します。

プロジェクトメソッドSTATSの内容です。

```
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))
        `キャッシュ内に存在する各ページ
    vPages:=vPages+pages{$i}+"   "+String(hits{$i})+"<br>"
        `ページの名前とHTMLコードを挿入
End for
SEND HTML FILE("stats.shtm")
```

参照

Webサーバセッティング

Secured Web connection

Secured Web connection →ブール

引数	タイプ		説明
このコマンドには引数はありません。			
戻り値	ブール	←	True=Web接続は暗号化モード False=Web接続は非暗号化モード

説明

Secured Web connection関数はブール値を返し、4D Webサーバ接続がSSLを使用した暗号化モード（リクエストが“http:”ではなく“https:”で始まる）で行われたかどうかを示します。

■ 接続がSSL経由で行われている場合、この関数はTrueを返します。

■ 接続が非暗号化モードで行われている場合、この関数はFalseを返します。

注：SSLプロトコルに関する詳細は、「SSLプロトコルの使用」の節を参照してください。

この関数を使用すると、非暗号化モードで行われた接続（存在する場合）を拒否することもできます。

参照

GENERATE CERTIFICATE REQUEST、SSLプロトコルの使用

SET CGI EXECUTABLE

SET CGI EXECUTABLE(url1{; url2})

引数	タイプ		説明
url1	文字列	→	アクセスURL
url2	文字列	→	アクセスURL

説明

SET CGI EXECUTABLEコマンドは、Webユーザに対してURL上にCGIを表示しないで、それを実行するために使用します。

特に、このコマンドは「On Web Authentication」データベースメソッドにおいて、例えば、実行するCGIを決定するために使用されます。また、Mac OS XおよびWindows双方で動作します。

注：CGIについての詳細は「CGIの使用」の節を参照してください。

＜url1＞には、実行するCGIのアクセスURLを渡します。例えば、**SET CGI EXECUTABLE**("/myfile.pl")という命令を記述した場合、4D Webサーバは CGI「myfile.pl」を実行します。その際、このアプリケーションをWebサーバのデフォルトフォルダ内に配置しなくてはなりません。

＜url1＞に空の文字列（""）を渡すと、4th Dimensionはブラウザから直接送信されたURL内に指定されたCGIを実行します（適用可能な場合）。

任意の引数＜url2＞には、そのCGIで処理したいファイルのアクセスURLを渡します。例えば、**SET CGI EXECUTABLE**("cgi-bin/Perl2.cgi";"Perl2.pl")という命令を記述した場合、Webサーバは「Perl2.pl」ファイルを渡した上で、CGIである「Perl2.cgi」を実行します。

＜url2＞に空の文字列（""）を渡すと、4th Dimensionはブラウザから送信されたURLに指定されたファイルをCGIへ渡して処理します。特にPHPでこのメカニズムが頻繁に用いられます（例：**SET CGI EXECUTABLE**("/cgi-bin/php";"")。

コマンドにより指定されたアクセスURLが正しくない場合、ブラウザは“File not found”エラーページを表示します。

ただし、**SET CGI EXECUTABLE**コマンドはエラーを直接返さないということを覚えておいてください。このコマンドはCGIがコールされた時に次に使用される“現在値”を設定するだけです。このコマンドを何度も呼び出した場合は、一番最後の呼び出しで指定された値が使用されます。

例題

▼ この例題では、「cgi-bin」フォルダに置かれていない「example.php」ファイルが、「cgi-bin」フォルダ内のCGI「Perl2.cgi」によって処理されます。

```
SET CGI EXECUTABLE("/cgi-bin/Perl2.cgi";"example.php")
```

参照

CGIの使用

OPEN WEB URL

OPEN WEB URL(url {; *})

引数	タイプ		説明
url	文字列	→	スタートアップURL
*	*	→	指定されている場合=4Dは URLを翻訳しない 省略されている場合=4Dは URLを翻訳する

説明

OPEN WEB URLコマンドは、デフォルトのWebブラウザを起動し、引数＜url＞に渡されたURLでブラウザを開きます。

Webブラウザが既に開かれている場合、このコマンドを実行すると、

■ Windowsでは、もう1つブラウザが起動し、＜url＞引数で渡されたURLを開きます。

■ Macintoshでは、現在のページに＜url＞引数で渡されたURLを開きます。

このコマンドを実行したコンピュータ上から起動するブラウザがない場合は、何も行ないません。

注意：Mac OSでは、このコマンドはコントロールパネルのインターネットで定義されたインターネットの設定（デフォルトのブラウザ）を読み込みます。

4DはURLを自動的にエンコードしますが、*を渡すと翻訳しません。このオプションは、下記のタイプのURLにアクセスする、またはこれを送ることができます。

"http://www.server.net/page.htm?q=something "

注：このコマンドは、Webプロセスから呼び出された場合、何も行ないません。

1　下記のコードの行が実行されると、

```
OPEN WEB URL("file:///D:/web file.htm")
```

Webブラウザが起動して、URLは下記のように翻訳されます。

"file:///D%3A/web%20file.htm"

2　下記のコードの行が実行されると、

```
OPEN WEB URL("file:///D:/web file.htm";*)vPages:=Char(1)
```

Webブラウザが起動して、URLは"file:///D:/web file.htm"のままです。

3 ローカル上でファイルを開く場合、システムセパレータを引数＜url＞で使用することができます。

```
$ref:=Open document("";"";Get Pathname)
CLOSE DOCUMENT($ref)
OPEN WEB URL("file:///"+document;*)
```

4 下記のコードの行はブラウザを起動して、4Dのホームページに接続します。

```
OPEN WEB URL("http://www.4d-japan.co.jp/")
```

Webサービス（クライアント）

この節では、4th Dimensionから外部のWebサービスへサブスクライブする（クライアント側）際に使用するコマンドについて説明します。Webサービス（サーバ側）の公開に使用するコマンドの説明は、「Webサービス（サーバ)」のコマンドを参照してください。

4th Dimensionを使用したWebサービスへのサブスクライブは、「Webサービスウィザード」を用いて簡単に行うことができます（4th DimensionにおけるWebサービスのセットアップに関する詳細は、前述の「Webサービス」の章を参照してください)。大抵の場合、このウィザードだけでWebサービスを利用することができます。しかし、特定の仕組みをカスタマイズしたい場合には、4th Dimensionのクライアント**SOAP**コマンドを使用しなければなりません。

注：規約により、サーバ側とクライアント側のコマンド（および定数）名を区別するため、“SOAP”と“Webサービス”という用語がそれぞれ使用されています。それ以外の点では、これら2つの概念は同じ技術について触れています。

SET WEB SERVICE PARAMETER

SET WEB SERVICE PARAMETER(名前; 値{; soapタイプ})

引数	タイプ		説明
名前	文字列	→	SOAPリクエストに含めるパラメータ名
値	変数	→	パラメータの値を納めた4D変数
soapタイプ	文字列	→	パラメータのSOAPタイプ

説明

このコマンドにより、クライアントSOAPリクエストで使用するパラメータを定義することができます。リクエスト内の各パラメータに対して、このコマンドを呼び出してください（このコマンドを呼び出す回数は、パラメータの数によって決まります）。

＜名前＞には、SOAPリクエスト中に出現するものと同じパラメータ名を渡します。

＜値＞には、パラメータの値を含む4D変数を渡します。プロキシメソッドの場合、通常この変数は、$1、$2、$3等であり、コールされた時にプロキシメソッドへ渡された4D変数に対応します。ただし、中間変数を使用することができます。

注：使用する各4D変数や配列は、「コンパイラ」および「配列」テーマのコマンドを使用して、最初に定義しておかなくてはなりません。

デフォルトとして、4th Dimensionは＜値＞の内容に応じ、引数＜名前＞に対して最も適したSOAPタイプを自動的に決定します。このタイプ指定はリクエスト中に組み込まれます。しかし、パラメータのSOAPタイプを“強制的に”定義したい場合があります。この場合には、以下の文字列のいずれかを使用して、任意の引数である＜soapタイプ＞を渡すことができます（プライマリーデータタイプ）。

SOAPタイプ	説明
string	文字列
int	倍長整数
boolean	ブール
float	32ビットの実数
decimal	小数点付き実数
double	64ビットの実数
duration	年、月、日、時、分、秒単位での期間。 例：P1Y2M3DT10H30M
datetime	ISO8601フォーマットの日付と時間。 例：2003-05-31T13:20:00
time	時間。例：13:20:00
date	日付。例：2003-05-31
gyearmonth	年と月（太陽暦）。例：2003-05
gyear	年（太陽暦）。例：2003
gmonthday	月と日（太陽暦）。例：--05-31
gday	日（太陽暦）。例：---31
gmonth	月（太陽暦）。例：--10--
hexbinary	16進数で表わした値
base64binary	BLOB
anyuri	Uniform Resource Identifier(URI)。 例：http://www.company.com/page
qname	正規のXML名（名前空間とローカルパート）
notation	Notation属性

注：XMLデータタイプに関する詳細は、次のURLを参照してください。

http://www.w3.org/TR/xmlschema-2/

例題

▼ この例題は、2つのパラメータを定義します。

```
C_TEXT($1)
C_TEXT($2)
SET WEB SERVICE PARAMETER("city";$1)
SET WEB SERVICE PARAMETER("country";$2)
```

参照

GET WEB SERVICE RESULT

SET WEB SERVICE OPTION

SET WEB SERVICE OPTION(オプション; 値)

引数	タイプ		説明
オプション	倍長整数	→	設定するオプションのコード
値	倍長整数	→	オプションの値

注：このコマンドは上級Webサービスユーザを対象としています。このコマンドの使用は任意です。

説明

SET WEB SERVICE OPTIONコマンドを使用すると、**CALL WEB SERVICE**コマンドで次回のSOAPリクエストが開始された時に使用する各種オプションを設定することができます。

CALL WEB SERVICEコマンドの前に、設定したいオプションの数だけ、このコマンドを呼び出すことができます。

引数＜オプション＞には、設定するオプションの値を引数＜値＞に代入して渡します。引数＜オプション＞に対し、事前に定義された「Web Services(Client)」テーマの定数のいずれかを使用することができます。

定数（＜オプション＞引数）	タイプ	値
Web Service HTTP Timeout	倍長整数	1
Web Service SOAP Header	倍長整数	2
Web Service SOAP Version	倍長整数	3
定数（＜値＞引数）	**タイプ**	**値**
Web Service SOAP_1_1	倍長整数	0
Web Service SOAP_1_2	倍長整数	1

以下では、これらの利用可能なオプションと値についてそれぞれ説明します。

■ Web Service HTTP Timeout

＜値＞=クライアント側のタイムアウト（秒単位）

クライアント側のタイムアウトとは、サーバからの応答がない場合のWebサービスクライアントの待機時間です。この待機時間が過ぎると、クライアントはセッションをクローズし、リクエストは失われます。

デフォルトのタイムアウトは10秒間です。特別な理由から（ネットワークステータス、Webサービスの特殊事情など)、この値を変更することができます。

■ Web Service SOAP Header

＜値＞=SOAPリクエストのヘッダとして挿入するXMLルート要素の参照

このオプションにより、**CALL WEB SERVICE**コマンドを用いて生成されたSOAPリクエストにヘッダを挿入することができます。デフォルトでは、SOAPリクエストに特定のヘッダは含まれていません。しかし、identificationパラメータの管理のような一定のWebサービスでは、ヘッダが必要になります。

■ Web Service SOAP Version

＜値＞=Web Service SOAP_1_1またはWeb Service SOAP_1_2（「Web Services(Client)」テーマの定数）

このオプションにより、リクエストで使用するSOAPプロトコルのバージョンを指定することができます。バージョン1.1を指定するには、＜値＞にWeb Service SOAP_1_1を渡し、バージョン1.2を指定するには＜値＞に定数Web Service SOAP_1_2を渡します。

このオプションの呼び出し順序は重要ではありません。同じ＜オプション＞が何度も設定された場合、最後の呼び出しの値が考慮されます。

例題

(1)次の例題は、独自のヘッダをSOAPリクエストに挿入します。

```
`XML参照の作成
C_STRING(16;vRootRef;vElemRef)
vRootRef:=DOM Create XML element("RootElement")
vxPath:="/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
`参照を用いてSOAPヘッダを変更する
SET WEB SERVICE OPTION(Web Service SOAP Header;vElemRef)
```

(2)次の例題は、SOAPプロトコルバージョン1.2を使用します。

```
SET WEB SERVICE OPTION(Web Service SOAP Version; Web Service SOAP_1_2)
```

参照

CALL WEB SERVICE

CALL WEB SERVICE

CALL WEB SERVICE(アクセスURL; soapAction; メソッド名; 名前空間{; 複合型})

引数	タイプ		説明
アクセスURL	文字列	→	WebサービスへのアクセスURL
soapAction	文字列	→	SOAPActionフィールドの内容
メソッド名	文字列	→	メソッドの名前
名前空間	文字列	→	名前空間
複合型	倍長整数	→	複合型の設定（省略すると、単純型）

説明

CALL WEB SERVICEコマンドは、HTTPリクエストを送信してWebサービスを呼び出すために使用します。このリクエストには、**SET WEB SERVICE PARAMETER**コマンドを使用して事前に作成されたSOAPメッセージが納められます。

次に**SET WEB SERVICE PARAMETER**コマンドを呼び出すと、新しくリクエストが作成されます。**CALL WEB SERVICE**コマンドを実行した場合も、以前に呼び出したWebサービスの結果が消去され、新しい結果と置き換えられます。

＜アクセスURL＞には、Webサービスへのアクセスが可能である、完全なURLを渡します（このURLとWSDLファイルのURLを混同しないでください。WSDLファイルはWebサービスを説明するものです）。

暗号化モードでのアクセス（SSL）：SSLを使用した暗号化モードでWebサービスを利用したい場合は、URLの初めに“http://”ではなく“https://”を渡します。この設定により、自動的に暗号化モードでの接続が可能になります。

＜soapAction＞には、リクエストのSOAPActionフィールドの内容を渡します。通常、このフィールドには、“サービス名#メソッド名”という値を納めます。

＜メソッド名＞には、実行しようとするリモートメソッド（Webサービスに属する）の名前を渡します。

＜名前空間＞には、SOAPリクエストで使用するXML名前空間を渡します。XML名前空間に関する詳細は、前述の「名前空間のカスタマイズ」の節を参照してください。

任意の引数である＜複合型＞は、送受信するWebサービスのパラメータの設定（**SET WEB SERVICE PARAMETER**コマンドと**GET WEB SERVICE RESULT**コマンドを使用して定義する）を指定します。引数＜複合型＞の値は、Webサービスの公開モード（DOCまたはRPC、前述の「複合型の処理」の節を参照）と独自のパラメータによって決定します。

＜複合型＞には、「Web Services(Client)」テーマにある以下の定数のうちいずれかを渡します。

定数	タイプ	値
Web Service Dynamic	倍長整数	0（デフォルト）
Web Service Manual In	倍長整数	1
Web Service Manual Out	倍長整数	2
Web Service Manual	倍長整数	3

定数はそれぞれ、Webサービスの“設定”に対応しています。この設定は、公開モード（RPC／DOC）と実装するパラメータタイプ（入力／出力、単純／複合）の組み合わせを表わします。

注：パラメータの“入力”や“出力”特性はプロキシメソッドやWebサービスの観点から判断される点に留意してください。

・“入力”パラメータは、プロキシメソッドへ渡される値であり、したがってWebサービスへ渡される値です。

・“出力”パラメータは、Webサービスから返される値であり、したがってプロキシメソッドから返される値です（通常は $0を使用）。

次の表は、考えられる設定とそれに対応する定数を示しています。

	入力パラメータ	
出力パラメータ	単純	複合
単純	Web Service Dynamic（RPCモード）	Web Service Manual In（RPCモード）
複合	Web Service Manual Out（RPCモード）	Web Service Manual（RPCまたは DOCモード）

したがって、以下に説明する5つの設定を実装することができます。すべてのケースにおいて、4th DimensionはWebサービスへ送信されるSOAPリクエストのフォーマットとそのエンベロープを処理します。使用した設定に従ってリクエストの内容をフォーマットするかどうかは、ユーザ次第です。

注：データ配列に関しては、複合型のXMLであるにも関わらず、4Dは単純型として処理します。

RPCモード、単純入力と単純出力

これは最も利用しやすい設定です。この場合、引数＜複合型＞にはWeb Service Dynamic定数を指定するか、または省略します。送信されたパラメータと受信した応答は、事前処理なしに直接処理されます。

後述する**GET WEB SERVICE RESULT**コマンドの例題を参照してください。

RPCモード、複合入力と単純出力

この場合、引数＜複合型＞にはWeb Service Manual In定数を指定します。この設定を使用すると、**SET WEB SERVICE PARAMETER**コマンドを使用し、“手動で”各XMLソース要素をBLOB形式でWebサービスに渡さなければなりません。

最初のBLOBを妥当なXML要素としてフォーマットするかどうかはユーザ次第です。その一番目の要素として、このBLOBには最終的なリクエストの<Body>要素の最初の明示的な“子”要素を必ず納めなければなりません。

例題

```
C_BLOB($1)
C_BOOLEAN($0)
SET WEB SERVICE PARAMETER("MyXMLBlob";$1)
CALL WEB SERVICE("http://my.domain.com/my_service";
                 "MySoapAction";"http://my.namespace.com/"; Web Service Manual In)
GET WEB SERVICE RESULT($0;"MyOutputVar";*)
```

RPCモード、単純入力と複合出力

この場合、引数＜複合型＞にはWeb Service Manual Out定数を指定します。各出力パラメータは、WebサービスからXML要素の形式でBLOB内に納めて返されます。**GET WEB SERVICE RESULT**コマンドを使用して、このパラメータを取得します。この後、4DのXMLコマンドを使用して、取得したBLOBの内容を解析することができます。

例題

```
C_BLOB($0)
C_BOOLEAN($1)
SET WEB SERVICE PARAMETER("MyInputVar";$1)
CALL WEB SERVICE("http://my.domain.com/my_service";
                 "MySoapAction";"http://my.namespace.com/"; Web Service Manual Out)
GET WEB SERVICE RESULT($0;"MyXMLOutput";*)
```

RPCモード、複合入力と複合出力

この場合、引数＜複合型＞にはWeb Service Manual定数を指定します。前述した2種類の設定と同様に、それぞれの入力および出力パラメータはBLOB内にXML要素の形式で納めなければなりません。

例題

```
C_BLOB($0)
C_BLOB($1)
SET WEB SERVICE PARAMETER("MyXMLInputBlob";$1)
CALL WEB SERVICE("http://my.domain.com/my_service";
                 "MySoapAction";"http://my.namespace.com/"; Web Service Manual)
GET WEB SERVICE RESULT($0;"MyXMLOutput";*)
```

DOCモード

DOC Webサービスを呼び出すためのプロキシメソッドは、複合型の入力および出力パラメータを用いてRPC Webサービスを呼び出すプロキシメソッドとよく似ています。

これら2つの設定における唯一の相違点は、送受信を行うBLOBパラメータのXML内容のレベルにあります。4th Dimensionの観点から見ると、SOAPリクエストの作成と送信は全く同じです。

例題

```
C_BLOB($0)
C_BLOB($1)
SET WEB SERVICE PARAMETER("MyXMLInput";$1)
CALL WEB SERVICE("http://my.domain.com/my_service";
                 "MySoapAction";"http://my.namespace.com/"; Web Service Manual)
GET WEB SERVICE RESULT($0;"MyXMLOutput";*)
```

注：DOC Webサービスの場合、パラメータとして渡した文字列の値（上の例では“MyXMLInput”および“MyXMLOutput”）は重要ではなく、空の文字列を渡すことさえ可能です。実際のところ、これらの値はXML文書を含むSOAPリクエストでは使用されません。それでも、これらのパラメータは必ず渡さなくてはなりません。

DOCモード（または複合型を用いたRPCモード）で公開されたWebサービスを使用する際には、次のような処理を行うことをお勧めします。

■ クライアントWebサービスウィザードを用いてプロキシメソッドを生成します。プロキシメソッドを次のように呼び出します。

```
$XMLresultBlob:=$DOCproxy_Method($XMLparamBlob)
```

■ オンラインテスト（例えば、http://soapclient.com/soaptest.html）を使用して、Webサービスへ送信するSOAPリクエストの内容を把握しておいてください。この種類のツールは、WebサービスのWSDLを基にしたHTMLテストフォームを生成する際に使用されます。

■ <body>の最初の子要素をもとに生成されたXMLコンテンツをコピーします。

■ 実際のパラメータ値をXMLコードに配置するメソッドを作成します。この後、このコードはBLOBである$XMLparamBlobに配置しなければなりません。

■ 応答を解析するため、オンラインテストを使用したり、または帰される要素を定義しているWSDLを利用することもできます。

リクエストが正しく送り出され、Webサービスがそれを受け付けた場合、システム変数OKには1が代入されます。それ以外の場合には、0が代入され、エラーが返されます。

参照：

SET WEB SERVICE PARAMETER、GET WEB SERVICE RESULT

GET WEB SERVICE RESULT

GET WEB SERVICE RESULT(戻り値{; パラメータ名}{; *})

引数	タイプ		説明
戻り値	変数	←	Webサービスから返される値
パラメータ名	文字列	→	取り出されるパラメータの名前
*		→	メモリの解放

説明

GET WEB SERVICE RESULTコマンドは、実行された処理の結果としてWebサービスから返される値を取得するために使用します。

注：このコマンドは、CALL WEB SERVICEコマンドを実行した後に使用しなければなりません。

引数<戻り値>には、Webサービスから返される値を受け取ります。この引数には4D変数を渡してください。通常、この変数は$0であり、プロキシメソッドから返される値に相当します。ただし、中間変数を使用することができます（プロセス変数のみ使用可能）。

注：使用される4D変数や配列はそれぞれ、「コンパイラ」および「配列」テーマのコマンドを使用して事前に定義しなければなりません。

任意の引数<パラメータ名>を使用して、取り出されるパラメータの名前を指定することができます。しかし、大半のWebサービスは単一の値だけを返すため、通常このパラメータは必要ありません。

任意の引数である<*>は、リクエストの処理に占有されていたメモリを解放するようプログラムに指示します。Webサービスから送信された最後の値を取得した後は、この引数を渡さなくてはなりません。

例題

▼Webサービスにより、世界中の任意の都市における現在時刻が返される場合を想定してみましょう。Webサービスへ送信するパラメータは、都市の名前とその国のコードです。引き換えに、Webサービスからは対応する時刻が送信されます。Webサービスを呼び出すプロキシメソッドは、次の通りです。

```
C_TEXT($1)
C_TEXT($2)
C_TIME($0)
SET WEB SERVICE PARAMETER("city";$1)
SET WEB SERVICE PARAMETER("country_code";$2)
```

```
CALL WEB SERVICE("http://www.citiesoftheworld.com/WS"; "WSTime#City_time";
                 "City_time"; "http://www.citiesoftheworld.com/namespace/default")
If(OK=1)
   GET WEB SERVICE RESULT($0;"return";*)
End if
```

参照：

SET WEB SERVICE PARAMETER

AUTHENTICATE WEB SERVICE

AUTHENTICATE WEB SERVICE(名前; パスワード)

引数	タイプ		説明
名前	文字列	→	ユーザ名
パスワード	文字列	→	ユーザパスワード

説明

AUTHENTICATE WEB SERVICEコマンドにより、クライアントアプリケーションの認証を必要とするWebサービスを使用できるようになります（シンプル認証）。

引数＜名前＞と＜パスワード＞には、要求される識別情報（ユーザ名とパスワード）を渡します。この情報は暗号化され、**CALL WEB SERVICE**コマンドを使用してWebサービスへ送信されるHTTPリクエストに追加されます。したがって、**AUTHENTICATE WEB SERVICE**コマンドは、**CALL WEB SERVICE**コマンドを呼び出す前にコールする必要があります。

認証情報は、各リクエストの後で0へリセットされるため、各**CALL WEB SERVICE**コマンドの前に、**AUTHENTICATE WEB SERVICE**コマンドを使用しなくてはなりません。

認証に失敗すると、SOAPサーバからエラーが返され、**Get Web Service error info**コマンドを用いてこのエラーを識別することができます。

参照：

CALL WEB SERVICE

Get Web Service error info

Get Web Service error info(情報タイプ) → 文字列

引数	タイプ		説明
情報タイプ	倍長整数	→	取得する情報
戻り値	文字列	←	前回のSOAPエラーに関する情報

説明

このコマンドは、リモートWebサービスへ送信したSOAPリクエストの実行中に発生した最後のエラーに関する情報を返します。

引数＜情報タイプ＞を使用して、取得しようとする情報のタイプを指定することができます。「Web Services(Client)」テーマ内にある次の定数のいずれかを渡さなくてはなりません。

定数	タイプ	値
Web Service Error Code	倍長整数	0
Web Service Detailed Message	倍長整数	1
Web Service HTTP Error code	倍長整数	2
Web Service Fault Actor	倍長整数	3

これらの定数は、次の値を取得するために使用します。

■ Web Service Error Code：メインエラーコード（4Dにより定義される）。このコードは、システム変数Errorへも返されます。
返されるコードの一覧は次の通りです。

- ■ 9910：Soap fault（Web Service Fault Actorも参照）
- ■ 9911：パーサーfault
- ■ 9912：HTTP fault（Web Service HTTPエラーコードも参照）
- ■ 9913：ネットワークfault
- ■ 9914：インターネットfault

■ Web Service Detailed Message：エラーを解説する詳細なメッセージ。メッセージタイプは、メインエラータイプによって異なります。

- ■ メインエラーコード = 9910（Soap fault）の場合：

  SOAP faultの原因が返されます（例：“リモートメソッドが存在しない”）。

- ■ メインエラーコード = 9911（パーサー fault）の場合：

XML文書内のエラー箇所が返されます。

■ メインエラーコード = 9912（HTTP fault）の場合：

- HTTPエラーが300～400の間である場合（リクエストされた文書の場所に関連した問題）、リクエストされたURLの新しい場所が返されます。

- その他任意のHTTPエラーコードに対しては、 <body> が返されます。

■ メインエラーコード = 9913（ネットワークfault）の場合：

ネットワークfaultの原因が返されます（例：“ServerAddress: DNS検索失敗”）。

■ メインエラーコード = 9914（インターネットfault）の場合：

インターネットfaultの原因が返されます。

■ Web Service HTTP Error code：HTTPエラーコード（メインエラーコード9912の場合に使用される）。

■ Web Service Fault Actor：エラーの原因（SOAPプロトコルから返され、メインエラーコード9910の場合に使用される）。
次のような原因が返されます。

- ■ Version Mismatch
- ■ Must Understand（サーバは必須属性として定義されたパラメータを理解できない）
- ■ Client Fault
- ■ Server Fault
- ■ Encoding Unknown

情報が取得できない場合には、空の文字列が返されます。

Webサービス（サーバ）

バージョン2003以降の4th DimensionはWebサービスをサポートします。つまり、プログラムを使用して、作成したデータベースから直接Webサービスの公開（サーバ側）や利用（クライアント側）を行うことができます。

この節では、4th DimensionにおいてWebサービス（サーバ側）の公開を行うために使用するコマンドについて説明します。Webサービスへのサブスクライブを行うために使用するコマンドの説明は、後述の「Webサービス（クライアント）」の節を参照してください。

4th Dimensionを使用したWebサービスの公開は、メソッドプロパティのオプションを利用して簡単に実行することができます（4th DimensionにおけるWebサービスのセットアップに関する詳細は、前述の「Webサービス」の章を参照してください）。大抵の場合、この操作だけでWebサービスの公開を行うことができます。ただし、データ配列を使用する場合など、特定の仕組みをカスタマイズしたい場合には、4th DimensionのサーバSOAPコマンドを使用しなければなりません。

注：規約により、サーバ側とクライアント側のコマンド（および定数）名を区別するため、“SOAP”と“Webサービス”という用語がそれぞれ使用されています。それ以外の点では、これら2つの概念は同じ技術について触れています。

SOAP DECLARATION

SOAP DECLARATION(変数; タイプ; 入出力{; エイリアス})

引数	タイプ		説明
変数	4D変数	→	SOAP入力、または出力引数を参照する変数
タイプ	倍長整数	→	引数が示す4Dタイプ
入出力	倍長整数	→	1= SOAP入力 2= SOAP出力
エイリアス	文字列	→	SOAP交換の際にこの引数用に公開された名前

説明

このコマンドを使用して、Webサービスとして公開された4Dメソッドで使用する引数のSOAPタイプを明示的に宣言します。

メソッドをWebサービスとして公開する場合、標準的な引数である$0、$1～$nを使用して、外部に対し（特にWSDLファイル内で）Webサービスのパラメータを表現します。SOAPプロトコルでは引数に明示的に名前が付けられている必要があります。デフォルトとして、4th Dimensionは“FourD_arg0、FourD_arg1～ FourD_argn”という名前を使用します。

しかし、次の理由からデフォルトとしてのこの運用は問題点が多いことが判明しています。

- $0や$1は配列として宣言することができません。したがって、ポインタを使う必要がありますが、その場合にはWSDLファイルの生成時に値のタイプが分かっていなければなりません。
- 次に、パラメータ名（入力と出力）のカスタマイズが有効であったり、必要となる場合があります。
- 最後に、この運用では、RPCコール毎に複数の値を返すことが不可能になります（$0内に）。

SOAP DECLARATIONコマンドを使用すると、これらの制約から解放されます。このコマンドをそれぞれの入力および出力引数に対して実行し、引数の名前やタイプを割り当てることができます。

注：SOAP DECLARATIONコマンドを使用した場合でも、常に「コンパイラ」テーマのコマンドを使用し、Compiler_Webメソッドで4D変数や配列を宣言する必要があります。

＜変数＞には、RPCコールで参照される4D変数を渡します。

警告：参照できるのは、プロセス変数または4Dメソッドの引数（$0～ $n）だけです。ローカル変数とインタープロセス変数は使用することができません。

＜タイプ＞には、対応する4Dタイプを渡します。大半の4D変数や配列を使用することができます。「Field and Variable Types」テーマにある次の定義済定数を使用できます。

定数	タイプ	値
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Real	倍長整数	1
Boolean array	倍長整数	22
String array	倍長整数	21
Date array	倍長整数	17
Integer array	倍長整数	15
LongInt array	倍長整数	16
Real array	倍長整数	14
Text array	倍長整数	18
Is Text	倍長整数	2
Is Date	倍長整数	4
Is Time	倍長整数	11
Is String Var	倍長整数	24

注：SOAPメソッドで使用されない定数は次の通りです：Is Alpha Field、Is Pointer、Array 2D、Picture array、Pointer array、Is Picture、Is Subtable、Is Undefined

＜入出力＞には、処理される引数が“入力”タイプ（つまり、メソッドが受け取る値に相当する）か、“出力”タイプ（つまり、メソッドから返される値に相当する）かを示す値を渡します。「Web Services(Server)」テーマにある次の定義済定数を使用することができます。

定数	タイプ	値
SOAP Input	倍長整数	1
SOAP Output	倍長整数	2

注：4D変数（4Dメソッドの引数ではなく）を使用して参照されるSOAP入力の引数は、COMPILER_WEBプロジェクトメソッド内で最初に宣言しなければなりません。詳細は、後述の「COMPILER_WEBメソッドの使用」の節を参照してください。

＜エイリアス＞には、WSDLやSOAP交換で現われるものと同じ引数名を渡します。

警告：RPCコールにおいて、この名前は一意でなくてはなりません（入力引数と出力引数は一緒とみなされる）。そうでない場合、4Dは最後に行われた宣言だけを考慮します。

注：マーカー名に関するXML規格に準拠し、引数名には空白や拡張文字を決して使用しないでください。次のLatin文字だけを使用することができます：([A-Za-z0-9._] ¦ '-')*

引数＜エイリアス＞を省略すると、デフォルトとして4th Dimensionは4Dメソッドの引数（$0～$n）に対して、変数名やFourD_argNという名前を使用します。

SOAP DECLARATIONコマンドは、Webサービスとして公開されたメソッド内に含まれていなくてはなりません。他のメソッドからこのコマンドを呼び出すことはできません。

(1) この例題はエイリアス名を指定します。

```
` WSDLファイルの生成およびSOAPコールの際に、
` fourD_arg1の代わりにzipcodeという文言を使用します。
C_LONGINT($1)
SOAP DECLARATION($1;Is LongInt;SOAP Input;"zipcode")
```

(2) この例題は、倍長整数配列の郵便番号を取得します。

```
`COMPILER_WEBメソッドにおいて
ARRAY LONGINT(codes;0)

`Webサービスメソッドにおいて
SOAP DECLARATION(codes;LongInt array;SOAP Input;"in_codes")
```

(3) この例題は、エイリアス名を指定せずに2つの戻り値返します。

```
SOAP DECLARATION(ret1;Is LongInt;SOAP Output)
SOAP DECLARATION(ret2;Is LongInt;SOAP Output)
```

(4) この例題は、32kb以上の引数値を4D SOAP Serverが返すことができます。

```
C_BLOB($0)
SOAP DECLARATION($0; Is Text; SOAP Output)
```

タイプには、Is Text（Is Blobではない）を指定します。これの記述は、実引数を正しく処理します。

定数

フィールドと変数のタイプ、Webサービス（サーバ）テーマ

参照

Get SOAP info、Is data file locked、SEND SOAP FAULT

SEND SOAP FAULT

SEND SOAP FAULT(faultタイプ; 説明)

引数	タイプ		説明
faultタイプ	倍長整数	→	1= クライアント側のfault 2= サーバ側のfault
説明	文字列	→	SOAPクライアントに送信されるエラーの説明

説明

このコマンドは、SOAPクライアントに対してエラーを返し、障害（fault）の原因（クライアントまたはサーバ）を伝えるために使用します。

このコマンドを使用すると、結果を返さなくてもクライアントにエラーを知らせることができます。

例えば、Webサービス“Square_root”を公開し、クライアントが負数を用いてリクエストを送信した場合、クライアント側の誤りが検出されます。その場合、このコマンドを使用して、正数が必要であることをクライアントに知らせることができます。

例えば、サーバ側の障害としては、メソッド実行時に起きるメモリ不足があります。

＜faultタイプ＞には、エラーの原因を渡します。「Webサービス（サーバ）」テーマ内の次の定義済定数を使用することができます。

定数	タイプ	値
SOAP Client Fault	倍長整数	1
SOAP Server Fault	倍長整数	2

＜説明＞には、エラーの説明を渡します。クライアントの実行がこれに適合する場合、エラーが処理されます。

例題

▼ コマンドの説明で取り上げた、Webサービス“Square_root”の例題に戻ると、負数を用いたリクエストの処理には、次のコマンドを使用することができます。

SEND SOAP FAULT(Client Fault; "正数が必要です。")

Is SOAP request

Is SOAP request→ ブール

引数	タイプ		説明
このコマンドに引数はありません。			
戻り値	ブール	←	SOAPリクエストの場合、True（真） それ以外はFalse（偽）

説明

このコマンドは、実行されるコードがSOAPリクエストの一部である場合にTrueを返します。

セキュリティ上の理由から、受信したリクエストの種類を判断するために「On Web Authentication」データベースメソッドでこのコマンドを使用することができます。

Get SOAP info

Get SOAP info(情報番号) → 文字列

引数	タイプ		説明
情報番号	倍長整数	→	取得するSOAP情報のタイプ番号
戻り値	文字列	←	SOAP情報

説明

このコマンドを使用し、SOAPリクエストに関する各種情報を文字列の形で取得することができます。

SOAPリクエストを処理する際、RPCパラメータの値以外にリクエストに関する追加情報を取得すると便利な場合があります。例えば、セキュリティ上の理由から、このコマンドを「On Web Authentication」データベースメソッドで使用し、要求されたWebサービスメソッドの名前を調べることができます。

＜情報番号＞には、調べようとするSOAP情報タイプの番号を渡します。「Web Services(Server)」テーマにある次の定義済定数を使用することができます。

定数	タイプ	値
SOAP Method Name	倍長整数	1
SOAP Service Name	倍長整数	2

SOAP Method Name：実行しようとしているWebサービスメソッドの名前。

SOAP Service Name：メソッドが属しているWebサービスの名前。

注：4Dへ送信するWebサービスリクエストに対して、最大サイズを設定することができます。この設定は、SET DATABASE PARAMETERコマンド（「ストラクチャアクセス」テーマ）を使用して行います。このコマンドについては後述しています。

ウインドウについて

ウインドウは、ユーザに対して情報を表示するのに使用します。ウインドウには、データの入力、データの表示、メッセージやダイアログでのユーザに対する告知の3つの機能があります。

最低でも1つのウインドウが開かれています。これは、タイトルバーとサイズボックスを持った標準ウインドウです。ユーザにウインドウよりも大きいフォームでスクロールさせる必要のある場合は、スクロールバーがそれに追加されます。また、ウインドウにコントロールメニュー（Macintosh版では、クローズ）ボックスを持たせることもできます。

「ユーザ」モードにおいて、ウインドウはレコードリスト（出力フォーム）、またはデータ入力画面（入力フォーム）のいずれかを表示します。「カスタム」モードにおいてウインドウは、スプラッシュ画面（カスタムグラフィック）を表示します。スプラッシュ画面はフォームを表示するコマンドでデータ表示画面に置き換えられます。コマンドの処理が終了すると、デフォルトとしてスプラッシュ画面が再び表示されます。

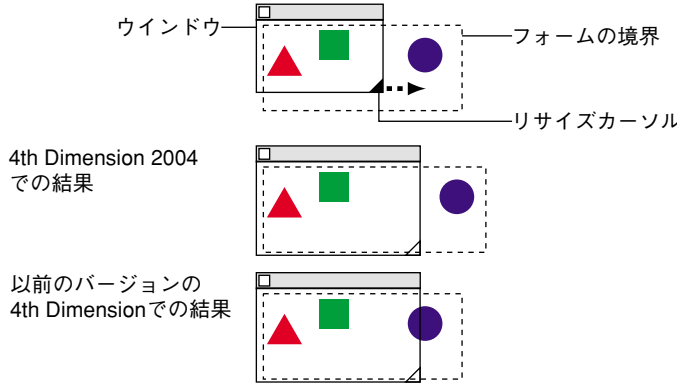
カスタムウインドウは、**Open window**関数で開くことができます。カスタムウインドウには、Windows（Macintosh）のさまざまな形式のウインドウを利用することができます。不要になったウインドウは、**CLOSE WINDOW**コマンドか、コントロールメニューボックス（Windows）、クローズボックス（Macintosh）をクリックして閉じます。

コマンドの中には、自らウインドウを開くものがあります。**GRAPH TABLE**コマンド、**REPORT**コマンド、**PRINT LABEL**コマンド等は、自ら最前面にウインドウを開きます。

新規プロセスを開始して、プロセスメソッドの冒頭で任意のウインドウを開かないと、4th Dimensionは任意フォームが表示されるとすぐにデフォルトのウインドウを自動的に開きます。

サイドプッシャー

バージョン2004の4th Dimensionより、デフォルトとしてウインドウの右側および下側は“プッシャー”スプリッタの機能を持ちます。つまり、ウインドウを大きくすると、画面上のウインドウの右側や下側の境界にあるオブジェクトが右方向や下方向へ自動的に押し出されます。



このメカニズムにより、「エクスプローラ」ウインドウのような格納式のウインドウを管理することができます（**SET FORM SIZE**コマンドの例題を参照）。

注：この機能は、スクロールバー付きのウインドウでは動作しません。

参照

Open window、ウインドウのタイプ

ウインドウのタイプ

下記のあらかじめ定義された定数の1つを使って、**Open window**関数で開くウインドウのタイプを指定することができます。

定数	タイプ	値	フローティングウインドウ
Plain window	倍長整数	8	不可
Plain no zoom box window	倍長整数	0	不可
Plain fixed size window	倍長整数	4	不可
Modal dialog box	倍長整数	1	不可
Alternate dialog box	倍長整数	3	可
Movable dialog box	倍長整数	5	可
Plain dialog box	倍長整数	2	可
Palette window	倍長整数	1984	可

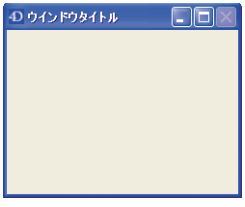
Round corner window	倍長整数	16	不可
Pop up window	倍長整数	32	不可
Sheet window	倍長整数	33	不可
Resizeable sheet window	倍長整数	34	不可

フローティングウインドウ

Open window関数に上記の定数の1つを渡すと、一般的なウインドウを開きます。フローティングウインドウを開くには、**Open window**関数に負数を渡します。

以下の図は、各ウインドウタイプを示しています。左側がWindows、右側がMacintoshのウインドウです。

Plain window(8)





■ タイトル付加：可

■ 「閉じる」ボタンまたはクローズボックス付加：可

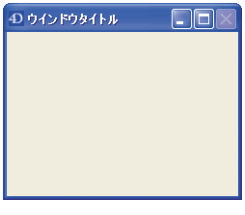
■ サイズ変更：可

■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：可

■ スクロールバーの使用：可

■ 用途：**DISPLAY SELECTION**コマンド、**MODIFY SELECTION**コマンド等

Plain no zoom box window(0)





■ タイトル付加：可

■ 「閉じる」ボタンまたはクローズボックス付加：可

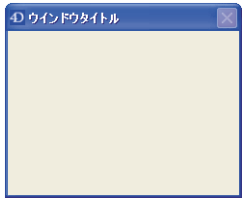
■ サイズ変更：可

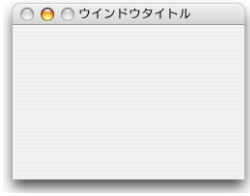
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：Macintosh上では、不可

■ スクロールバーの使用：可

■ 用途：スクロールバー付きのデータ入力、**DISPLAY SELECTION**コマンド、**MODIFY SELECTION**コマンド等

Plain fixed size window(4)





■ タイトル付加：可

■ 「閉じる」ボタンまたはクローズボックス付加：可

■ サイズ変更：Macintosh上では、不可

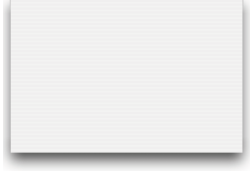
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

■ スクロールバーの使用：可／不可

■ 用途：**ADD RECORD**（...；...*）または同等の機能付きのデータ入力

Modal dialog box(1)





■ タイトル付加：不可

■ 「閉じる」ボタンまたはクローズボックス付加：不可

■ サイズ変更：不可

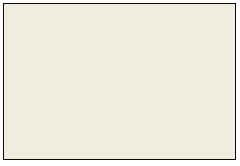
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

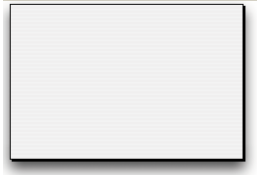
■ スクロールバーの使用：不可

■ 用途：**DIALOG**コマンド、**ADD RECORD**（...；...*）または同等の機能

■ このタイプのウインドウは、モーダル（形式）ウインドウです

Alternate dialog box(3)





■ タイトル付加：不可

■ 「閉じる」ボタンまたはクローズボックス付加：不可

■ サイズ変更：不可

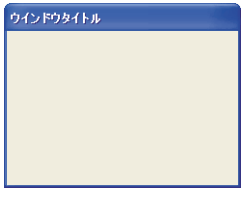
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

■ スクロールバーの使用：不可

■ 用途：**DIALOG**コマンド、**ADD RECORD**（...；...*）または同等の機能

■ フローティングウインドウとして使用されない場合、このタイプのウインドウはモーダル（形式）ウインドウです

Movable dialog box(5)





■ タイトル付加：可

■ 「閉じる」ボタンまたはクローズボックス付加：不可

■ サイズ変更：不可

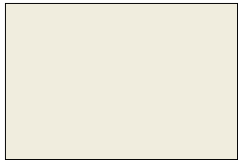
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

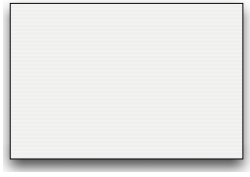
■ スクロールバーの使用：不可

■ 用途：**DIALOG**コマンド、**ADD RECORD**（...；...*）または同等の機能

■ このタイプのウインドウはモーダル（形式）ウインドウですが、フローティングウインドウとして移動、または使用することが可能

Plain dialog box(2)





■ タイトル付加：不可

■ 「閉じる」ボタンまたはクローズボックス付加：不可

■ サイズ変更：不可

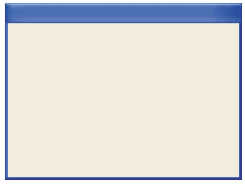
■ 「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

■ スクロールバーの使用：不可

■ 用途：**DIALOG**コマンド、**ADD RECORD**（...；...*）または同等の機能、スプラッシュスクリーン

■ フローティングウインドウとして使用されない場合、このタイプのウインドウはモーダル（形式）ウインドウです

Palette window(1984 {+ 1} {+ 2} {+ 4} {+ 8})





Open window関数を呼び出すと、パレットウインドウに下記の定数を1つまたは複数を追加して、そのウインドウのさまざまな機能を取得することができます。

定数	タイプ	値
Has zoom box	倍長整数	8
Has grow box	倍長整数	4
Has window title	倍長整数	2
Has highlight	倍長整数	1

■ タイトル付加：可、「Has window title」定数が指定された場合

■ 「閉じる」ボタンまたはクローズボックス付加：可

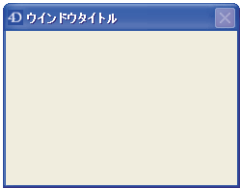
■ サイズ変更：可、「Has grow box」定数が指定された場合

■「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：可、「Has zoom box」定数が指定された場合

■ スクロールバーの使用：可、「Has grow box」定数が指定された場合

■ 用途：**DIALOG**コマンド、**DISPLAY SELECTION**コマンド（データ入力はなし）を用いたフローティングウインドウ

Round corner window(16)





■ タイトル付加：可

■「閉じる」ボタンまたはクローズボックス付加：可

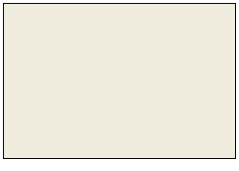
■ サイズ変更：Macintosh上では、不可

■「最大化」／「最小化」ボタンまたは「ズーム」ボックス付加：不可

■ スクロールバーの使用：Macintosh上では、不可

■ 用途：現在は使用しません（互換性のために存在）。

Pop up window(32)



このタイプのウインドウは、標準のダイアログボックスタイプ（2）のウインドウの基本的な特性と、次に示す高度な仕様を備えています。

■ 次の場合に、ウインドウは自動的に閉じられ、“キャンセル”イベントがウインドウに渡されます。

- ■ ウインドウの外側でクリックが行われた場合。
- ■ 背面のウインドウまたはMDI（Multiple Document Interface）ウインドウが移動された場合。
- ■ ユーザがEscキーを押した場合。

■ このウインドウは、“親”ウインドウの前面に表示されます（そのプロセスのメインウインドウとして使用してはいけません）。背面のウインドウは使用不可ではありませんが、イベントを受け取りません。

■ マウスを使用して、ウインドウのリサイズや移動を行うことはできません。しかし、これらの動作をプログラムから実行すると、背面にある項目のリドロウが最適化されます。

■ **用途：**主としてこのタイプのウインドウは、3Dの“ベベル”や“ツールバー”タイプのボタンに関連付けたポップアップメニューの処理に使用されます。

Sheet window(33)とResizable sheet window(34)



シートウインドウはMac OS X特定のものです。これらのウインドウは、アニメーションを用いてメインウインドウのタイトルバーの上に“ドロップダウン”され、メインウインドウの前面に表示されます。また、ウインドウはメインウインドウ上でセンタリングされます。そのプロパティはモーダルダイアログボックスと同じです。通常、これらのウインドウは、メインウインドウで行われた動作に直接関連するアクションを実行するために使われます。

■ 最後に開いたウインドウが表示されており、しかもドキュメントタイプ（フォーム）である場合にのみ、Mac OS X上でシートウインドウを作成することができます。

■ 次の場合、コマンドはタイプ33のウインドウの代わりにタイプ1のウインドウ（モーダルダイアログボックス）を開き、タイプ34のウインドウの代わりにタイプ8のウインドウ（標準）を開きます。

  ■ 最後に開いたウインドウが表示されていないか、またはドキュメントタイプではない場合。

  ■ Windows上で実行された場合。

■ シートウインドウはフォームの前面に描画されます。そのため、シートウインドウが表示されるタイミングは、最初のフォームを表示（例えば**DIALOG**コマンド）時のOn Loadイベント発生のときまで持ち越しされます（**Open window**関数の例題4を参照）。

■ **用途：**Mac OSにおいて（Windowsでは標準ではない）、**DIALOG**、**ADD RECORD** (...;...*)またはこれと同等のコマンド。

参照

Open exteenal window、Open window

Open window

Open window(左 ; 上 ; 右 ; 下 {;タイプ} {; タイトル} {; コントロールメニュー (Macintosh版では、クローズ) ボックス}}}) → {ウインドウ参照番号}

引数	タイプ		説明
左	数値	→	ウインドウ左端の画面上の位置（ピクセル）
上	数値	→	ウインドウ上端の画面上の位置（ピクセル）
右	数値	→	ウインドウ右端の画面上の位置（ピクセル）または、フォームのデフォルトサイズを使用する場合は、-1
下	数値	→	ウインドウ下端の画面上の位置（ピクセル）または、フォームのデフォルトサイズを使用する場合は、-1
タイプ	数値	→	ウインドウのタイプ
タイトル	文字列	→	ウインドウのタイトル、またはデフォルトのフォームタイトルを使用する場合は、""
コントロールメニューボックス	文字列	→	ボックスをダブルクリックしたときに呼び出されるメソッド
戻り値	参照番号	←	ウインドウ参照番号

説明

Open window関数は、最初の4つの引数で指定した大きさの新しいウインドウを開きます。

■ <左>は、画面の左端からウインドウの内側の左端までの距離をピクセル数で示します。

■ <上>は、画面の上端からウインドウの内側の上端までの距離をピクセル数で示します。

■ <右>は、画面の左端からウインドウの内側の右端までの距離をピクセル数で示します。

■ <下>は、画面の上端からウインドウの内側の下端までの距離をピクセル数で示します。

<右>と<下>の両方に-1を渡した場合、以下の方法でウインドウサイズを自動的に設定させることができます。

■ 任意のフォームをデザインして、「デザイン」モードの「フォームプロパティ」ウインドウでそのフォームの「サイズオプション」を設定する。

■ **Open window**関数を呼び出す前に、サイズ設定したフォームとオプション引数<*>を渡した**INPUT FORM**コマンドを使用する。

重要：このウインドウの自動サイズオプションは、このコマンドの実行前に、表示されるフォーム用のINPUT FORMコマンドを呼び出し、かつそのINPUT FORMコマンドにオプション引数<*>を渡した場合にのみ機能します。

■引数<タイプ>はオプションで、表示するウインドウのタイプを指定します。ウインドウのタイプは、前節に示したウインドウに対応します。ウインドウのタイプに負の値が指定された場合は、作成されるウインドウはフローティングウインドウになります。ウインドウのタイプが指定されない場合には、デフォルトとしてタイプ1が使用されます。

■引数<タイトル>は、オプションです。これで、表示するウインドウのタイトルを設定します。

<タイトル>に空の文字列（""）を渡すと、「デザイン」モードの「フォームプロパティ」ウインドウ内にある「ウインドウタイトル」を表示されるフォームに設定するように4th Dimensionに指示します。

重要：デフォルトのフォームタイトルは、このコマンドの実行前に、表示されるフォーム用のINPUT FORMコマンドを呼び出し、そしてそのINPUT FORMコマンドにオプション引数<*>が渡された場合にのみウインドウにセットされます。

注：「ユーザ」モードでは、4th Dimensionは自動的にウインドウタイトルを変更します。例えば、データ入力を実行する場合、ウインドウタイトルは“更新：テーブル名”となります。ウインドウタイトルを変更すると、4th Dimensionはそれを上書きします。これに対して、「カスタム」モードでは、4th Dimensionはウインドウタイトルを変更しません。

■引数<コントロールメニュー（クローズ）ボックス>はオプションで、ウインドウにコントロールメニュー（クローズ）ボックス用のメソッドを設定します。<コントロールメニュー（クローズ）ボックス>を指定すると、コントロールメニュー（クローズ）ボックスがウインドウに追加されます。ユーザがウインドウ上のクローズボックスをクリックすると、<コントロールメニュー（クローズ）ボックス>に受け渡されるメソッドを呼び出します。

バージョン6の注意：On Close Boxイベントが生じる際に、ウインドウに表示されるフォームのフォームオブジェクトの中からウインドウのクローズ処理を管理することもできます。これに関する詳細は、Form event関数の節を参照してください。

1つのプロセス内で複数のウインドウが開いている場合は、最後に開かれたウインドウがそのプロセスにおけるアクティブウインドウ（最前面のウインドウ）になります。アクティブウインドウの情報のみを修正することができます。その他のウインドウは参照のみ行えます。ユーザのキー入力によって、アクティブウインドウは常に最前面に配置されます。

フォームは、開かれたウインドウ内に表示されます。**MESSAGE**コマンドのテキストもウインドウ内に表示することができます。

ウインドウの大きさを画面の大きさに依存しないようにするには、**Screen height**関数と**Screen width**関数を使用してウインドウの左上端と右下端の位置を計算することができます。この技法の関する詳細は、次ページの例を参照してください。

例題

1. 以下のプロジェクトメソッドは、画面の中央にウインドウを開きます。2個、3個または4個の引数を受け取り、その数によって処理を振り分ける点に注目してください。

```
      ` プロジェクトメソッド：ウインドウ中央表示
      ` $1ーウインドウの幅
      ` $2ーウインドウの高さ
      ` $3ーウインドウのタイプ(オプション)
      ` $4ーウインドウのタイトル(オプション)
$S幅:=Screen width / 2     ` 画面の中心を求める(幅)
$S高さ:=(Screen height / 2) -10   ` 画面の中心を求める(高さ)
$W高さ:=$1 / 2       ` ウインドウの高さの半分を求める
$W幅:=$2 / 2 ` ウインドウの幅の半分を求める
Case of
   ¥(Count parameters=2)
      $WinRef:=OPEN WINDOW($S幅-$W幅 ; $S高さ-$W高さ ; $S幅+$W幅 ;
                                                $S高さ+$W高さ)
   ¥(Count parameters=3)
      $WinRef:=OPEN WINDOW($S幅-$W幅 ; $S高さ-$W高さ ; $S幅
                                          +$W幅 ; $S高さ+$W高さ ; $3)
   ¥(Count parameters=4)
      $WinRef:=OPEN WINDOW($S幅-$W幅 ; $S高さ-$W高さ ; $S幅
                                       +$W幅 ; $S高さ+$W高さ ; $3 ; $4)
End case
```

プロジェクトメソッドを作成したら、以下のようにこのメソッドを使用することができます。

```
ウインドウ中央表示(400 ; 250 ; Movable dialog box ; "更新")
DIALOG([テーブル1];"更新オプション")
CLOSE WINDOW
If(OK=1)
   …
End if
```

2. 以下の例は、コントロールメニューボックス（クローズボックス）のメソッドを持ったフローティングウインドウを開きます。このウインドウは、アプリケーションウインドウの右上端に開きます。このメソッドが画面の大きさを気にしなくてよいように**Screen width**関数を使用している点に注目してください。

```
$WinRef:=Open window(Screen width-149;33;Screen width-4;178;
                                        -Palette window;""; "CloseBox")
DIALOG([ダイアログ] ; "カラーパレット")
```

“CloseBox”メソッドは、**CANCEL**コマンドを呼び出します。

```
CANCEL
```

3. 以下の例は、ウインドウに表示されるフォームのプロパティから得たサイズとタイトルを持つウインドウを開きます。

```
INPUT FORM([顧客] ; "レコード追加" ; *)
$WinRef:=Open window(10 ; 80 ; -1 ; -1 ; Plain window ; "")
Repeat
    ADD RECORD([顧客])
Until(OK=1)
```

注：Open window関数でフォームプロパティを自動的に使用できるようにするには、オプション引数の＜*＞を指定してINPUT FORMコマンドを呼び出し、さらに「デザイン」モードでそのフォームのプロパティがそれに応じて設定されている必要があります。

4. この例題は、Mac OS X上でシートウインドウを表示する際の“遅延”メカニズムを示しています。

```
$myWindow:=Open window(10;10;400;400;Sheet window)
    `現時点では、ウインドウは作成されたが非表示のままです
DIALOG([Table];"dialForm")
    `「On Load」イベントが生成された後、シートウインドウが表示されます。
    `ウインドウはタイトルバーの下側から“ドロップダウン”されます。
```

参照

CLOSE WINDOW、Open external window

Open external window

Open external window(左 ; 上 ; 右 ; 下 ; タイプ ; タイトル ; プラグインエリア)

→ 整数

引数	タイプ		説明
左	数値	→	ウインドウ左端の画面上の位置（ピクセル）
上	数値	→	ウインドウ上端の画面上の位置（ピクセル）
右	数値	→	ウインドウ右端の画面上の位置（ピクセル）
下	数値	→	ウインドウ下端の画面上の位置（ピクセル）
タイプ	数値	→	ウインドウのタイプ
タイトル	文字列	→	ウインドウのタイトル
プラグインエリア	文字列	→	プラグインエリアコマンド
戻り値	数値	←	ウインドウ参照番号

説明

Open external window関数は、新しいウインドウを開きます。4Dプラグインで提供される＜プラグインエリア＞のコマンドでサポートされるプラグインエリアを表示します。通常、＜プラグインエリア＞に渡されるコードは、“_プラグイン名”という形式になります（例：_4D Write、_4D View、_4D Draw）。

Open external window関数は、倍長整数の値を返します。この値は、（他のウインドウコマンドで使用される）ウインドウ参照番号および（4Dプラグインで提供される他のルーチンで使用される）ウインドウに表示されるプラグインエリアへの参照番号として使用されます。

最初の6個の引数は**Open window**関数と同じです。しかし、いずれの引数もオプションではありません。すべて引数は指定します。

Open external window関数は、モードレスウインドウを作成します。このコマンドはユーザの入力を待機しません。したがって、複数のアクティブウインドウを一度に開くことができます。各ウインドウは、クリックすることによって最前面に移動して操作することができます。タイトルバーを持っているウインドウタイプの場合、コントロール-メニューボックス（Windows）またはクローズボックス（Macintosh）が追加され、ユーザはウインドウを閉じることができます。

例題

▼ 以下の例は、プラグインウインドウを開き、プラグインエリアに“4D Draw”を表示します。

```
X:=Open external window(50 ; 50 ; 350 ; 450 ; 8 ; "4D Draw" ; "_4D Draw")
```

▼ 以下の例は、上記の例で開いたプラグインウインドウを閉じます。

```
CLOSE WINDOW(X)
```

参照

CLOSE WINDOW、Open window

CLOSE WINDOW

CLOSE WINDOW {(ウインドウ)}

引数	タイプ		説明
ウインドウ	数値	→	プラグインウインドウ参照番号、または省略した場合、カレントプロセスの最前面ウインドウ

説明

CLOSE WINDOWコマンドは、**Open window**関数で開いたカレントプロセスのウインドウを閉じます。**CLOSE WINDOW**コマンドは、カスタムウインドウが開かれていない場合には、何も行いません。また、標準ウインドウに対しても何も行いません。フォームがアクティブな状態の時に**CLOSE WINDOW**コマンドを使用した場合には、何も行いません。**Open window**関数で開いたウインドウは、必ず**CLOSE WINDOW**コマンドで閉じてください。

オプション引数＜ウインドウ＞を指定すると、プラグインウインドウ参照番号で指定されたプラグインウインドウを閉じます。プラグインウインドウに関する詳細は、**Open external window**関数を参照してください。

Open window関数で開いたウインドウを閉じる際に、ある番号を**CLOSE WINDOW**コマンドへ渡しても意味がありません。一連の**Open window**関数を呼び出して、複数ウインドウを開いた場合は、ウインドウの作成された順序と逆の順番で各ウインドウが閉じられます。

例題

▼以下の例は、ウインドウを開き**ADD RECORD**コマンドを使用して新しいレコードを追加します。レコードを追加した時点で、**CLOSE WINDOW**コマンドによってウインドウを閉じます。

```
$WinRef:=Open window(5 ; 40 ; 250 ; 300 ; 0 ; "新規採用")
Repeat          ` 取り消すまでループする
   ADD RECORD([従業員])      ` 新しいレコードを追加する
Until(OK=0)
CLOSE WINDOW     ` カスタムウインドウを閉じる
```

参照

Open external window、Open window

ERASE WINDOW

ERASE WINDOW {(ウインドウ)}

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

説明

ERASE WINDOWコマンドは、引数＜ウインドウ参照番号＞に渡した参照番号を持つウインドウを内容を消去します。

＜ウインドウ参照番号＞を省略すると、**ERASE WINDOW**コマンドはカレントプロセスの最前面ウインドウの内容を消去します。

通常、**ERASE WINDOW**コマンドは、**MESSAGE**コマンドと**GOTO XY**コマンドと組み合わせて使用します。この場合、**ERASE WINDOW**コマンドはウインドウ内容を消去し、カーソルをウインドウの左上端、「**GOTO XY**（0；0）」の位置に移動します。

ウインドウ内容を消去する**ERASE WINDOW**コマンドと画面からウインドウ自体を消去する**CLOSE WINDOW**コマンドを混同しないように注意してください。

参照

GOTO XY、MESSAGE

REDRAW WINDOW

REDRAW WINDOW {(ウインドウ参照番号)}

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

説明

REDRAW WINDOWコマンドは、引数＜ウインドウ参照番号＞に渡した参照番号のウインドウをグラフィカルに更新することができます。

＜ウインドウ参照番号＞を省略すると、**REDRAW WINDOW**コマンドはカレントプロセスの最前面ウインドウに適用します。

注：4th Dimensionでは、ウインドウの移動、サイズ変更、前面への移動等を実行する度にグラフィカル更新を行います。また、ウインドウ内のフォームやウインドウ内に表示されている値を変更する場合もグラフィカル更新を行います。

参照

ERASE WINDOW

DRAG WINDOW

DRAG WINDOW

引数	タイプ	説明

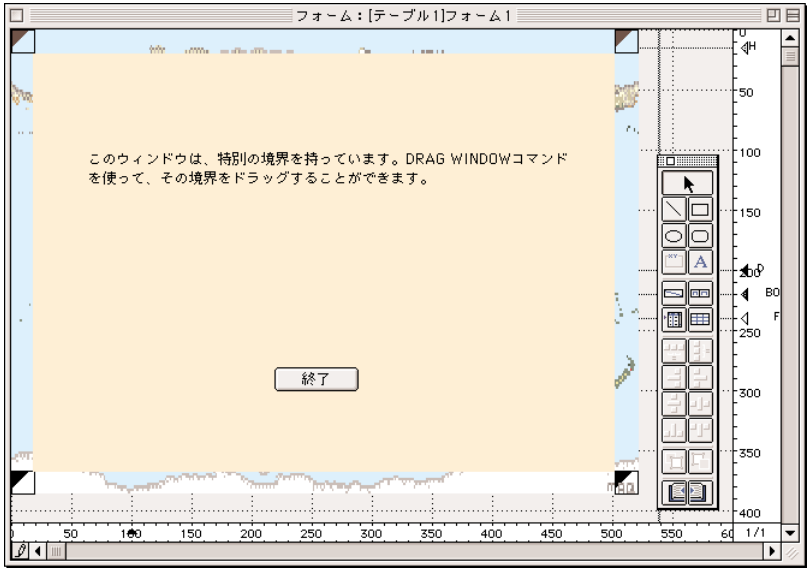
このコマンドには、引数はありません。

説明

DRAG WINDOWコマンドは、マウス移動にしたがって、現在の最前面ウインドウをドラッグします。通常、このコマンドは、マウスクリックに瞬時に反応できるオブジェクト（例えば、透明ボタン）のオブジェクトメソッドから呼び出します。

例題

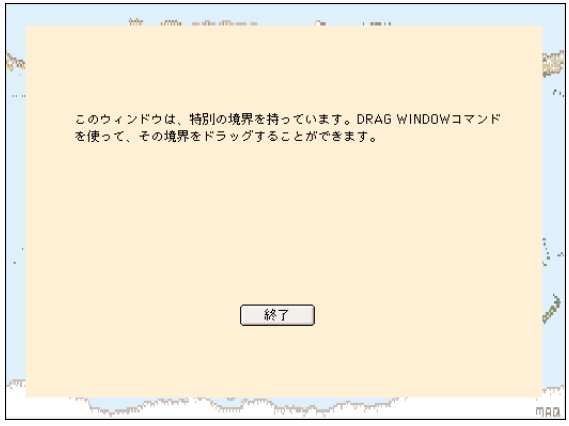
以下のフォームは、「デザイン」モードで表示されていますが、固定のピクチャで作成された枠を持っています。このピクチャの枠の各コーナーには4つの透明ボタンが設定されています。



各ボタンには、以下のメソッドが設定されています。

DRAG WINDOW `クリックすると、ウインドウのドラッグを開始する

「ユーザ」モードまたは「カスタム」モードでは、以下のプロジェクトメソッドを実行すると、次ページのような図が表示されます。




```
$WinRef:=Open window(50 ; 50 ; 50+400 ; 50+300 ; 2)
DIALOG([Table1] ; "Custom Drag")
CLOSE WINDOW
```

各コーナーの任意の場所をクリックすると、ウインドウをドラッグすることができます。

参照

GET WINDOW RECT、SET WINDOW RECT

HIDE WINDOW

HIDE WINDOW {(ウインドウ)}

引数	タイプ		説明
ウインドウ	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ

説明

HIDE WINDOWコマンドを使用すると、引数＜ウインドウ＞で渡す番号を持つウインドウ、あるいはこの引数を省略した場合には、カレントプロセスの最前面ウインドウを隠すことができます。例えば、このコマンドを使用すると、複数のプロセスから構成されるプロセスで、アクティブウインドウだけを表示することができます。

ウインドウは画面上には表示されなくなりますが、開かれたままになっています。したがって、プログラムからこのウインドウに対し、4Dウインドウがサポートするあらゆる変更を実行することができます。

HIDE WINDOWコマンドで隠されたウインドウを表示するには、以下の方法があります。

■ **SHOW WINDOW**コマンドを使用してウインドウ参照番号を渡す。

■ ランタイムエクスプローラのプロセスページを使用する。ウインドウを処理しているプロセスを選択し、「表示」ボタンをクリックする。

プロセスのすべてのウインドウを隠すには、**HIDE PROCESS**コマンドを使用してください。

例題

以下の例は、入力フォームに配置されたボタンのメソッドを示したものです。このボタンは、同じプロセスに属する新しいウインドウにダイアログボックスを開きます。この例では、ダイアログボックスが表示されている間、そのプロセスの他のウインドウ（入力用フォームとツールパレット）を隠そうとしています。ダイアログボックスが確定されると、他のウインドウが再び表示されます。

```
`「情報」ボタンのオブジェクトメソッド
HIDE WINDOW(Entry)`入力ウインドウを隠す
HIDE WINDOW(Palette)`パレットを隠す
$Infos:=Open window(20;100;500;400;8)`情報ウインドウを作成する
...`ダイアログの管理に関する指示をここに入れる
    CLOSE WINDOW($Infos)  `ダイアログを閉じる
    SHOW WINDOW(Entry)
    SHOW WINDOW(Palette)  `他のウインドウを表示する
```

参照

SHOW WINDOW

SHOW WINDOW

SHOW WINDOW {(ウインドウ)}

引数	タイプ		説明
ウインドウ	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ

説明

SHOW WINDOWコマンドを使用すると、引数<ウインドウ>で渡す番号を持つウインドウを表示することができます。この引数を省略した場合には、カレントプロセスの最前面ウインドウが表示されます。

SHOW WINDOWコマンドを使用するには、**HIDE WINDOW**コマンドを使ってウインドウが隠されている必要があります。ウインドウが既に表示されている場合には、このコマンドは何も行いません。

例題

HIDE WINDOWコマンドの例を参照してください。

参照

HIDE WINDOW

MAXIMIZE WINDOW

MAXIMIZE WINDOW {(ウインドウ)}

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスのすべての最前面ウインドウ（Windows）、またはカレントプロセスの最前面ウインドウ（Macintosh）

説明

MAXIMIZE WINDOWコマンドは、引数＜ウインドウ参照番号＞に渡す参照番号を持つウインドウを拡大します。引数＜ウインドウ参照番号＞を省略してもコマンドの効果は変わりませんが、その場合にはカレントプロセスのすべての最前面ウインドウ（Windows）、またはカレントプロセスの最前面ウインドウ（Macintosh）に適用されます。

このコマンドは、4Dアプリケーションウインドウのズームボックスをクリックしたのと同じ効果を持ちます。

Windowsの場合

ウインドウのサイズは、アプリケーションウインドウの現在のサイズに一致するように拡大されます。最大表示されたウインドウは、最前面ウインドウとして設定されます。＜ウインドウ参照番号＞引数を渡さなかった場合には、このコマンドはすべてのアプリケーションウインドウに適用されます。


Windows上のズームボックス

Macintoshの場合

ウインドウのサイズは、メインスクリーンのサイズに一致するように拡大されます。＜ウインドウ参照番号＞引数を渡さなかった場合には、このコマンドはカレントプロセスの最前面ウインドウに適用されます。

Macintosh上のズームボックス

このコマンドは、ズームボックスを持つウインドウにだけ適用されます。ウインドウがズームボックスを持たないタイプのものである場合には、このコマンドは何も行いません。詳細については、「ウインドウのタイプ」の節を参照してください。

MAXIMIZE WINDOWコマンドは、ウインドウをその「最大」サイズに設定します。ウインドウが実際にはフォームプロパティに定義したサイズのフォームである場合は、ウインドウのサイズはその値に設定されます。ウインドウが既に最大表示されている場合には、このコマンドは何も行いません。

ウインドウを拡大した後にズームボックスをクリックするか、**MINIMIZE WINDOW**コマンドを呼び出すと、ウインドウはその初期サイズに縮小されます。Windowsでは、引数を指定せずに**MINIMIZE WINDOW**コマンドを呼び出すと、すべてのアプリケーションウインドウのサイズがその初期サイズに設定されます。

例題

以下の例は、データベースを開くときに、データベースアプリケーションのウインドウサイズをフルスクリーンに設定します。これを実行するには、以下のコードを「On Startup」データベースメソッドに記述します。

```
`「On Startup」データベースメソッド
MAXIMIZE WINDOW
```

参照

MINIMIZE WINDOW

MINIMIZE WINDOW

MINIMIZE WINDOW {(ウインドウ参照番号)}

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスのすべての最前面ウィンドウ（Windows）、またはカレントプロセスの最前面ウインドウ（Macintosh）

説明

MINIMIZE WINDOWコマンドは、引数＜ウインドウ参照番号＞に渡す番号を持つウインドウのサイズを、最大表示される前のサイズに設定します。引数＜ウインドウ参照番号＞を省略すると、このコマンドはアプリケーションの各ウインドウ（Windowsの場合）、またはプロセスの最前面ウインドウ（Macintoshの場合）に適用されます。

このコマンドは、4Dアプリケーションの縮小ボックスを1回クリックするのと同じ効果を持ちます。

Windowsの場合

ウインドウのサイズはその初期サイズ、つまり最大表示される前のサイズに設定されます。引数＜ウインドウ＞を省略すると、すべてのアプリケーションウインドウがその初期サイズに設定されます。


Windows上の「元のサイズに戻す」ボックス

Macintoshの場合

ウインドウのサイズはその初期サイズ、つまり最大表示される前のサイズに設定されます。引数＜ウインドウ参照番号＞を省略すると、カレントプロセスの最前面ウインドウがその初期サイズに設定されます。

Macintosh上の縮小／ズームボックス

コマンドが適用されるウインドウが（手動で、または**MAXIMIZE WINDOW**を使って）最大表示されていない場合や、ウインドウがズームボックスを持たないタイプのものである場合には、このコマンドは何も行いません。ウインドウのタイプについての詳細は、「ウインドウのタイプ」の節を参照してください。

注：Windowsでは、この機能を、下図のボタンをクリックすることで実行される、ウインドウをボタンに最小化する操作と混同しないようにしてください。

 windows

 Mac OS

参照

MAXIMIZE WINDOW

Get window title

Get window title {(ウインドウ)} → 文字列

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
戻り値	文字列	←	ウインドウのタイトル

説明

Get window title関数は、引数＜ウインドウ＞に渡される参照番号を持つウインドウのタイトルを返します。ウインドウが存在しない場合は、空の文字列が返されます。

引数＜ウインドウ＞を省略した場合は、**Get window title**関数はカレントプロセスの最前面ウインドウのタイトルを返します。

例題

▼ **SET WINDOW TITLE**コマンドの例を参照してください。

参照

SET WINDOW TITLE

SET WINDOW TITLE

SET WINDOW TITLE(タイトル {; ウインドウ})

引数	タイプ		説明
タイトル	文字列	→	ウインドウタイトル
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ

説明

SET WINDOW TITLEコマンドは、引数<ウインドウ>に渡した参照番号を持つウインドウのタイトルを引数<タイトル>（最大80バイト）で指定したテキストに変更します。ウインドウが存在しない場合、**SET WINDOW TITLE**コマンドは何も行いません。<ウインドウ>を省略すると、**SET WINDOW TITLE**コマンドはカレントプロセスの最前面ウインドウに適用します。

注：「ユーザ」モードでは、4th Dimensionは自動的にウインドウタイトルを変更します。例えば、データ入力を実行する場合、ウインドウタイトルは“更新：テーブル名”となります。ウインドウタイトルを変更すると、4th Dimensionはそれを上書きします。これに対して、「カスタム」モードでは、4th Dimensionはウインドウタイトルを変更しません。

例題

以下の例は、任意のフォームでデータ入力を行っている最中に、（プログラムでサブフォームに表示されるリレートレコードをブラウズするような）長い処理を実行するボタンをクリックします。現在のウインドウタイトルを使って、その処理の進行状況を確認し続けることができます。

```
`「bAnalysis」ボタンのオブジェクトメソッド
Case of
	¥(Form event=On Clicked)
		`ローカル変数に現在のウインドタイトルを保存する
		$vsCurTitle:=Get window title
		`長い処理を開始する
		FIRST RECORD([送り状明細])
		For($vlRecord ; 1 ; Records in selection([送り状明細]))
			DO SOMETHING
			`処理経過を表示する
			SET WINDOW TITLE( "処理数 #"+String($vlRecord))
		End for
			`元のウインドウタイトルに戻す
		SET WINDOW TITLE($vsCurTitle)
End case
```

参照

Get window title

WINDOW LIST

WINDOW LIST(ウインドウ参照番号 {; *})

引数	タイプ		説明
ウインドウ	配列 参照番号	←	ウインドウ参照番号の配列
*	*	→	指定した場合は、フローティングウインドウを考慮 省略した場合は、フローティングウインドウを無視

説明

WINDOW LISTコマンドは、動作しているすべてのプロセス（カーネルプロセスまたはユーザプロセス）の中で現在開いているウインドウのウインドウ参照番号を引数＜ウインドウ参照番号＞の配列に格納します。

オプション引数＜*＞を指定しなかった場合は、フローティングウインドウは無視されます。

例題

以下のプロジェクトメソッドは、フローティングウインドウとダイアログボックスを除いて、現在開いているウインドウ全部を並べます。

```
`「TILE WINDOWS」プロジェクトメソッド
WINDOW LIST($alWnd)
$vlLeft:=10
$vlTop:=80`ツールバー用の空間を確保する
For($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
```

注：このメソッドは、メインウインドウのサイズ（Windows上）、または画面サイズと位置（Macintosh上）を調べるテストを追加して、さらに改良することができます。

参照

Window kind、Window process

Window kind

Window kind {(ウインドウ参照番号)} → 数値

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
戻り値	数値	←	ウインドウのタイプ

説明

Window kind関数は、引数<ウインドウ参照番号>に渡された参照番号を持つウインドウの4th Dimensionウインドウタイプを返します。ウインドウが存在しない場合は、**Window kind**関数は0を返します。

それ以外は、**Window kind**関数は以下の値の1つを返します。

定数	タイプ	値
Regular window	倍長整数	8
Modal dialog	倍長整数	9
External window	倍長整数	5
Floating window	倍長整数	14

引数<ウインドウ>を省略した場合は、**Window kind**関数はカレントプロセスの最前面ウインドウのタイプを返します。

例題

▼ **WINDOW LIST**コマンドの例を参照してください。

参照

GET WINDOW RECT、Get window title、Window process

Window process

Window process {(ウインドウ参照番号)} → 数値

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
戻り値	数値	←	プロセス参照番号

説明

Window process関数は、引数<ウインドウ参照番号>に渡された参照番号を持つウインドウを動作するプロセス番号を返します。ウインドウが存在しない場合は、**Window process**関数は0を返します。

引数<ウインドウ参照番号>を省略した場合は、**Window process**関数はカレント最前面ウインドウのプロセスを返します。

参照

Current process

GET WINDOW RECT

GET WINDOW RECT(左 ; 上 ; 右 ; 下 {; ウインドウ})

引数	タイプ		説明
左	数値	→	ウインドウの内容エリアの左座標
上	数値	→	ウインドウの内容エリアの上座標
右	数値	→	ウインドウの内容エリアの右座標
下	数値	→	ウインドウの内容エリアの下座標
ウインドウ	数値	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

説明

GET WINDOW RECTコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウのグローバル座標を返します。ウインドウが存在しない場合は、引数として指定した変数は更新されません。

＜ウインドウ＞を省略すると、**GET WINDOW RECT**コマンドはカレントプロセスの最前面ウインドウに適用します。

座標は、Windows上ではアプリケーションウインドウ、Macintosh上ではメイン画面の内容エリアの左隅との相対位置で表されます。座標は、（タイトルバーと境界線を除いた）ウインドウの内容エリアと一致する矩形を返します。

参照

SET WINDOW RECT

SET WINDOW RECT

SET WINDOW RECT(左 ; 上 ; 右 ; 下 {; ウインドウ})

引数	タイプ		説明
左	数値	→	ウインドウの内容エリアの左座標
上	数値	→	ウインドウの内容エリアの上座標
右	数値	→	ウインドウの内容エリアの右座標
下	数値	→	ウインドウの内容エリアの下座標
ウインドウ	数値	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

説明

SET WINDOW RECTコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウのグローバル座標を返します。ウインドウが存在しない場合は、**SET WINDOW RECT**コマンドは何も行いません。

＜ウインドウ数値番号＞を省略すると、**SET WINDOW RECT**コマンドはカレントプロセスの最前面ウインドウに適用します。

このコマンドは、引数で受け渡された新しい座標でウインドウサイズを変更、またはウインドウを移動することができます。

座標は、Windows上ではアプリケーションウインドウ、Macintosh上ではメイン画面の内容エリアの左隅との相対位置で表す必要があります。座標は、（タイトルバーと境界線を除いた）ウインドウの内容エリアと一致する矩形を返します。

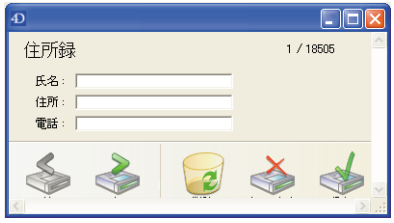
警告：このコマンドを使用することにより、Windows上ではメインウインドウ、Macintosh上ではメイン画面の表示範囲を越えてウインドウを移動してしまうことがあります。これを避けるには、Screen width関数やScreen height関数を使って、ウインドウの新しい座標を2重チェックします。

このコマンドはフォームオブジェクトに影響を与えません。ウインドウにフォームが含まれている場合、このコマンドによりフォームオブジェクトの移動やリサイズは行われず（そのプロパティとは関係なく）、ウインドウだけが変更されます。リサイズプロパティおよびフォーム上に存在する各オブジェクトを考慮に入れながら、フォームウインドウを変更するには、必ず**RESIZE FORM WINDOW**コマンドを使用してください。

例題

1. **WINDOW LIST**コマンドの例題を参照してください。

2. 次に示すウインドウの場合



次の行を実行した後、

SET WINDOW RECT(100;100;300;300)

ウインドウは次のようになります。



参照

DRAG WINDOW、GET WINDOW RECT

Frontmost window

Frontmost window {(*)} → ウインドウ参照番号

引数	タイプ		説明
*	*	→	指定した場合は、フローティングウインドウを考慮 省略した場合は、フローティングウインドウを無視
戻り値	数値	←	ウインドウ参照番号

説明

Frontmost window関数は、最前面ウインドウのウインドウ参照番号を返します。

参照

Frontmost process、Next window

Next window

Next window {(ウインドウ)} → 数値

引数	タイプ		説明
ウインドウ参照番号	数値	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
戻り値	数値	←	ウインドウ参照番号

説明

Next window関数は、（ウインドウの前後の順番をもとにした）引数＜ウインドウ参照番号＞に渡された参照番号を持つウインドウの“背面”にあるウインドウのウインドウ参照番号を返します。もし、ウインドウが存在しない場合は、**Window process**関数は0を返します。

参照

Frontmost window

Find window

Find window(左 ; 上 { ; ウインドウ部分}) → ウインドウ参照番号

引数	タイプ		説明
左	数値	→	ウインドウ左端のグローバル座標
上	数値	→	ウインドウ上端のグローバル座標
ウインドウ部分	数値	←	ウインドウ部分のID番号
戻り値	数値	←	ウインドウ参照番号

説明

Find window関数は、引数＜左＞と＜上＞で渡された画面上の位置に触れている最初のウインドウの参照番号を返します（存在する場合）。

座標は、アプリケーションウインドウ（Windows）の内容エリアの左上隅またはメインスクリーン（Macintosh）の相対位置で表されなければなりません。

オプション引数＜ウインドウ部分＞を指定した場合、ウインドウの有無に関係なく、この引数は以下の値の1つを返します。

定数	タイプ	値	プラットフォーム
In menu bar	倍長整数	1	Macintoshのみ
In system window	倍長整数	2	Macintoshのみ
In contents	倍長整数	3	WindowsまたはMacintosh
In drag	倍長整数	4	Macintoshのみ
In grow	倍長整数	5	Macintosh のみ
In go away	倍長整数	6	Macintosh のみ
In zoom box	倍長整数	7	Macintoshのみ

参照

Frontmost window、Next window

Open form window

Open form window({テーブル ;} フォーム名 {; タイプ {; 水平位置 {; 垂直位置 {;*}}}})→ ウインドウ参照番号

引数	タイプ		説明
テーブル	テーブル	→	フォームのテーブル 省略されている場合デフォルトテーブル
フォーム名	文字列	→	フォームの名前
タイプ	倍長整数	→	ウインドウのタイプ
水平位置	倍長整数	→	ウインドウの水平位置
垂直位置	倍長整数	→	ウインドウの垂直位置
*	*	→	ウインドウの現在位置とサイズをセーブ
ウインドウ	倍長整数	←	ウインドウの参照番号

説明

Open form windowコマンドは、フォーム名引数で指定したフォームのサイズおよびサイズ変更プロパティを使用して新しいウインドウを開きます。

フォームの内容はウインドウに表示されないことに注意してください。フォームを表示したい場合には、フォームをロードするコマンドを呼び出さなければなりません（例えば、**ADD RECORD**）。

デフォルトでは（タイプ引数が渡されていない場合）クローズボックス付きの標準ウインドウが開かれます。**Open window**コマンドとは異なり、ウインドウのクローズボックスには何のメソッドも定義されません。このクローズボックスをクリックすると、**On Close Box**フォームイベントがフォーム用に起動されている場合を除き、ウインドウをキャンセルして閉じます。この場合、**On Close Box**イベントに定義されたコードが実行されます。

フォームのサイズ変更が可能であれば、開かれたウインドウはズームボックスならびにグローボックスを持ちます。

注：フォームの主なプロパティを知るには、GET FORM PROPERTIESコマンドを使用します。

オプションのタイプ引数はウインドウのタイプが指定でき、下記の“Open window”テーマ内にある定数の内から1つを渡します。

定数	タイプ	値
Standard form window	倍長整数	8
Modal form dialog box	倍長整数	1
Movable form dialog box	倍長整数	5
Palette window	倍長整数	1984
Pop up form window	倍長整数	32
Sheet form window	倍長整数	33

注：ウインドウタイプに関する詳細は、「ウインドウのタイプ」の節を参照してください。Open form window関数では、「Open form window」テーマに記載されているタイプだけが使用できるという点に注意してください。

オプションの引数水平位置は、ウインドウの水平位置を定義します。ポイント単位で位置を指定します（**Open window**コマンドを参照してください）。または、下記の“Open form window”テーマ内にある定数の内から1つを渡します。

定数	タイプ	値
Horizontally Centered	倍長整数	65536
On the Left	倍長整数	131072
On the Right	倍長整数	196608

オプションの引数垂直位置は、ウインドウの垂直位置を定義します。ポイント単位で位置を指定します（**Open window**コマンドを参照してください）。または、下記の“Open form window”テーマ内にある定数の内から1つを渡します。

定数	タイプ	値
Vertically Centered	倍長整数	262144
At the Top	倍長整数	327680
At the Bottom	倍長整数	393216

これらの引数は、ツールバーおよびメニューバーの存在ならびにアプリケーションウインドウの現在のサイズ（Windowsの場合）を考慮に入れます。

オプションの引数*を渡すと、ウインドウをクローズした時の位置およびサイズが記憶されます。ウインドウが再度開かれる時に、その前の位置とサイズが優先されます。この場合、垂直位置と水平位置の引数は最初にウインドウが開かれる時のみに使用されます。

例題

▼下記のステートメントは標準のウインドウをクローズボックス付きで開き、自動的にそれを「入力」フォームと同じサイズになるように調整します。フォームは、サイズ変更可能なものとして定義されているので、ウインドウもグローボックスおよびズームボックスを持ちます。

```
$winRef := Open form window([Table1];"Enter")
```

▼下記のステートメントは、画面の左上部にあるフローティングパレットを開くものです。このパレットは、開かれるたびに前回ユーザが閉じた時の位置に表示されます。

```
$winRef:= Open form window([Table1]; "Tools"; Palette form window; On the Left;
                                                                  At the Top;*)
```

参照

Open window、GET FORMS PROPERTIES

Current form window

Current form window → ウインドウ参照番号

引数	タイプ		説明
このコマンドには引数は必要ありません。			
戻り値	数値	←	カレントフォームウインドウの参照番号

説明

このコマンドは、カレントフォームウインドウの参照を返します。カレントフォームに対してウインドウが定義されていない場合、コマンドは0を返します。

カレントフォームウインドウは、ユーザアクションの後、または**Open window**や**Open form window**関数を使用することにより、**ADD RECORD**などのコマンドを用いて自動的に生成されます。

参照

Open form window、Open window、RESIZE FORM WINDOW

RESIZE FORM WINDOW

RESIZE FORM WINDOW (幅; 高さ)

引数	タイプ		説明
幅	倍長整数	→	カレントフォームウインドウの幅に追加、または減らすピクセル数
高さ	倍長整数	→	カレントフォームウインドウの高さに追加、または減らすピクセル数

説明

このコマンドを使用し、カレントフォームウインドウのサイズを変更することができます。

引数＜幅＞と＜高さ＞には、カレントウインドウのサイズに加算したいピクセル数を渡します。カレントサイズを変更したくない場合は、どちらの引数にも0を渡します。サイズを小さくするには、引数＜幅＞と＜高さ＞に負の値を渡します。

このコマンドは、リサイズボックスを使用して手動でウインドウサイズを変更した場合（それが可能なウインドウタイプの場合）とまったく同じ結果をもたらします。結果として、このコマンドは、フォームプロパティで指定されたオブジェクトのサイズ調整プロパティやサイズ制限を考慮します。例えば、コマンドを用いて、フォームに許可されたサイズよりも大きなサイズにウインドウを変更すると、コマンドは何も行いません。

注目すべきなのは、この動作が**SET WINDOW RECT**コマンドの動作とは異なるところです。**SET WINDOW RECT**コマンドは、ウインドウのリサイズを行う際に、フォームプロパティや内容を考慮しません。

また、このコマンドは必ずしもフォームサイズを変更しないということに注意してください。プログラムからフォームサイズを変更するには、後述の**SET FORM SIZE**コマンドを参照してください。

参照

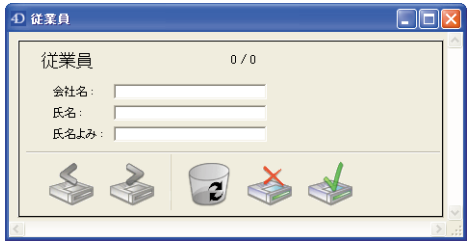
以下のウインドウで考えてみましょう（ウインドウにはフィールドおよびフレームを横方向にサイズ変更するための"Grow"プロパティがあります）。



次のコマンドを実行します。

RESIZE FORM WINDOW（25;0）

コマンドの実行後、ウインドウは次のように変わります。



参照

GET FORM PROPERTIES, SET FORM SIZE, SET WINDOW RECT

XMLについて

XMLのサポートに関して、4th DimensionはApache Foundation company開発による「Xerces.dll」というライブラリを使用します。4th DimensionはXMLバージョン1.0をサポートします。

注：4Dはインポート／エクスポートエディタを使用してXMLフォーマットのデータを直接インポート／エクスポートできます。

DOMコマンドとSAXコマンド

4th Dimensionでは、2種類のXMLコマンドが提供され、それぞれコマンド名の先頭に「DOM」または「SAX」が付けられています。

DOMおよびSAXとは？

DOM（Document Object Model）およびSAX（Simple API XML）は、XML文書用の2つの解析モードです。

- DOMモードは、XMLソースを解析し、その構造（“ツリー（木）”）をメモリ上に作成します。このため、ソースの各要素へのアクセスが非常に高速に行われます。しかし、ツリー構造全体がメモリ上に保持されるため、大規模なXML文書の処理を行う場合はメモリ容量を超えてしまい、エラーが発生する可能性があります。

- SAXモードは、ツリー構造をメモリ上に作成しません。このモードでは、ソースの解析時に“イベント”（要素の初めと終わり等の）が生成されます。このモードにより、利用可能なメモリ量とは関係なく、あらゆるサイズのXML文書を解析することができます。

XML規格に関する詳細は、以下のサイトを参考してください。

http://www.saxproject.org/?selected=event
http://www.w3schools.com/xml/

DOMを介したXML文書の作成、オープン、クローズ

4D DOMコマンドにより作成、変更、解析されるオブジェクトとして、テキスト、URL、ドキュメント、またはBLOBがあります。4th DimensionでXMLオブジェクトのオープンに使用されるDOMコマンドは、**DOM Parse XML source**関数ならびに**DOM Parse XML variable**関数です。

次に、数々のコマンドを使用し、要素や属性の読み込みや解析、書き込みを行うことができます。各種エラーは、**GET XML ERROR**コマンドを用いて回復することができます（両XML規格で共通）。

最後に**DOM CLOSE XML**コマンドを使用してソースをクローズします。

SAXを介したXML文書の作成、オープン、クローズ

SAXコマンドは、4th Dimension標準のドキュメント参照（DocRef、時間タイプの参照）を用いて動作します。したがって、**SEND PACKET**や**Append document**等の文書管理用4Dコマンドと一緒に、これらのコマンドを使用することができます。

プログラムからのXML文書の作成やオープンは、**Create document**関数および**Open document**関数を使用して行います。その後、これらの文書でXMLコマンドを使用すると、符号化などのXMLのメカニズムが自動的に実装されます。例えば、<?xml version= "1.0" encoding="...encoding..." standalone= "no"?>というヘッダが自動的に文書へ記述されます。

XML文書のクローズは、**CLOSE DOCUMENT**コマンドを使用して行わなければなりません。オープンされているXML要素が存在すれば、それも自動的にクローズされます。

XPathノーテーションの使用（DOM）

2つのDOMコマンド（**DOM Create XML element**および**DOM Find XML element**）は、XML要素へアクセスするためのXPathノーテーションに対応します。

XPathノーテーションはXPath言語に由来し、XML構造内をナビゲーションすることを目的としています。XML構造内のパス名により、フルアクセスパスを指定しなくても、XML構造内でダイレクトに要素を設定することができます。例えば、次のような構造の場合、

```
<RootElement>
    <Elem1>
        <Elem2>
            <Elem3 Font=Verdana Size=10> </Elem3>
        </Elem2>
```

```
        </Elem1>
    </RootElement>
```

XPathノーテーションを使用すると、シンタックス“RootElement/Elem1/Elem2/Elem3”を用いて要素3にアクセスすることができます。

4th Dimensionでは、シンタックス“要素[要素番号]”を用いて、指標付けされたXPath要素にも対応します。

```
<RootElement>
    <Elem1>
        <Elem2>aaa</Elem2>
        <Elem2>bbb</Elem2>
        <Elem2>ccc</Elem2>
    </Elem1>
</RootElement>
```

XPathノーテーションにより、シンタックス“RootElement/Elem1/Elem2[3]”を用いて値“ccc”にアクセスすることができます。

XPathノーテーションの説明は、**DOM Create XML element**関数および**DOM Find XML element**関数の例題を参照してください。

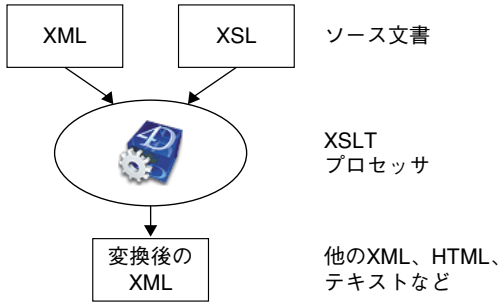
XSL Transformationsの管理

4th Dimensionは、XSLスタイルシート（eXtended Stylesheet Language）のアプリケーションをサポートします。XSL言語を使用し、XML文書のタグを変更することができます。

XSL言語には2種類の機能があります。

- Formatting（フォーマッティング）：この機能により、HTMLのCSS（Cascading Style Sheets）と同じように、XML要素に対してスタイルや表示規則を適用することができます。

- Transformations（トランスフォーメーション）：この機能により、XMLタグを別のタグシステムに変換することができます（例えば、HTML）。この機能は、XSLTとして知られています。XSLスタイルシートは、文書内のXML要素を選択することにより、要素全体を再編成し、別の要素に変換することができます。例えば、この機能は一連の異なる文書を同期化する場合に役立ちます。



注：4th Dimensionは、XSL変換を実行するために「Xalan-C_1_6_0.dll」ライブラリを利用します。Xalanはフリーウェアの XSLプロセッサです。詳細については、http://xml.apache.org/xalan-c/index.htmlを参考にしてください。

XSLスタイルシートはテキストタイプの文書で（拡張子“.xsl”が付属）、手動または専門のアプリケーションを用いて生成されます。その全体的な構造はCSSとほぼ同じです。XSL言語にはさまざまな要素や機能が用意されており、あらゆるタイプのダイナミック変換を実行することができます。この言語に関する詳細は、http://xmlfr.org（一例として）を参考にしてください。

4th Dimensionでは、既存のXSLスタイルシートを用いてXML文書を変換することができます（**APPLY XSLT TRANSFORMATION**コマンドを参照）。また、4th Dimensionでは、**SET XSLT PARAMETER**コマンドを使用して、XSLスタイルシートのパラメータをオンザフライで変更することも可能です。

注：書き出し用のダイアログボックスのオプションを使用すると、XMLの書き出し時にXSLスタイルシートを利用することが可能になり、その結果、変換済みのXML文書が生成されます。

用語

XML言語には数々の特別な用語や頭文字が使われています。この節では、4th Dimension 2003のコマンドや関数で使用される主なXMLの概念について説明します。

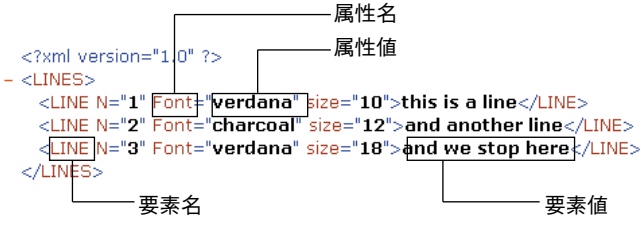
■ **属性**：XMLのサブタグで、要素と関連付けられています。属性は常に名前と値を持ちます（図を参照）。

■ **子**：XML構造で他のすぐ下のレベルに位置する要素。

■ **DTD**：Document Type Declaration（文書型定義）の略。DTDには、XMLが従わなくてはならない特定の規則とプロパティ一式が記録されています。これらの規約として、特に各タグの名前と内容、ならびにそのコンテキストが定義されています。各要素を形式的に定義することにより、XML文書の整合性をチェックすることができます（その場合、文書は“妥当”であると宣言されます）。
DTDは、XML文書内（内部DTD）あるいは別の文書内（外部DTD）に組み込まれます。DTDは必須ではないという点に注意してください。

■ **要素**：XMLタグ。ある要素は常に名前と値を持ちます。オプションとして、要素には属性を含めることができます（図を参照）

要素と属性



■ **elementRef**：4DのXMLコマンドでXML構造を特定するために使用するXMLリファレンス。
この要素リファレンスは、16進表記の8つのコード化文字、すなわち16桁で構成されます。

■ **親**：XML構造で他のすぐ上のレベルに位置する要素。

■ **解析、パーサ**：有用な情報を取り出すために、構造化オブジェクトの内容を解析する処理。「XML」テーマのコマンドは、任意のXMLオブジェクトの内容を解析するために使用されます。

■ **ルート**：XML構造の一番始めのレベルに位置する要素。

■ **XML構造**：構造化されたXMLオブジェクト。このオブジェクトは、文書、変数、要素のいずれでも構いません。

■ **妥当性検証**：XML文書は、それが“整形式”であり、DTD仕様に準拠している場合、パーサにより“妥当”であるものと判断されます。

■ **整形式**：XML文書が一般的なXML規格に準拠している場合、パーサにより“整形式”であるものと宣言されます。

■ **XML**：eXtensible Markup Language（拡張可能なマーク付け言語）の略。電子化されたデータの変換を行う際の標準言語で、データならびにその構造を転送することができます。XML言語は、HTML言語を踏まえて、タグと特定のシンタックスの使用を基本としています。ただし、HTMLとは異なり、XML言語ではカスタマイズしたタグを定義することができます。

■ **XSL**：eXtensible Stylesheet Languageの略。XSL文書内容の処理や表示に使用するスタイルシートを定義する言語。

DOM Create XML Ref

DOM Create XML ref(ルート{; 名前空間{; 名前空間名; 名前空間値}
{; 名前空間名X; 名前空間値X}}) → 文字列

引数	タイプ		説明
ルート	文字列	→	ルート要素の名前
名前空間	文字列	→	名前空間の値
名前空間名	文字列	→	名前空間の名前
名前空間値	文字列	→	名前空間の値
戻り値	文字列	←	ルートXML要素リファレンス

説明

このコマンドは、メモリ上に空のXMLツリーを作成し、その要素リファレンスを返します。

引数＜ルート＞には、XMLツリーのルート要素の名前を渡します。

任意の引数＜名前空間＞には、ツリーの名前空間の値の宣言を渡します（例：“http:www.4d.com”）。

その際、引数＜ルート＞の前には、名前空間の値に続けてコロン（:）を付加しなくてはなりません（例：“MyNamespace:MyRoot”）。

注：＜名前空間＞は文字列であり、この値によりXML変数名が確実に一意となるように設定することができます。通常、http://www.mysite.com/myurlのようなURLが用いられます。このURLは必ずしも有効でなくても構いませんが、一意（ユニーク）でなければなりません。

＜名前空間名＞と＜名前空間値＞のペアを使用して、生成されるXMLツリー内に1つ以上の名前空間をさらに宣言することができます。名前空間の名前／値のペアを必要な数だけ渡すことができます。

例題

(1)1つのXMLツリーを作成します。

```
C_STRING(16;vElemRef)
vElemRef:=DOM Create XML ref("MyRoot")
```

このメソッドの結果は次の通りです。

```
<?xml version="1.0"encoding="UTF-8"standalone="no"?>
<MyRoot/>
```

(2)1つの名前空間があるXMLツリーを作成します。

```
C_STRING(16;vElemRef)
$Root:="MyNameSpace:MyRoot"
$Namespace:="http://www.4D.com/tech/namespace"
vElemRef:=DOM Create XML ref($Root;$Namespace)
```

このメソッドの結果は次の通りです。

```
<MyNameSpace MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/namespace"/>
```

(3)複数の名前空間があるXMLツリーを作成します。

```
C_STRING(16;vElemRef)
C_STRING(80;$aNSName1;$aNSName2;$aNSValue1;$aNSValue2)
$Root:="MyNameSpace:MyRoot"
$Namespace:="http://www.4D.com/tech/namespace"
$aNSName1:="NSName1"
$aNSName2:="NSName2"
$aNSValue1:="http://www.4D.com/Prod/namespace"
$aNSValue2:="http://www.4D.com/Mkt/namespace"
vElemRef:=DOM Create XML Ref
        ($Root;$Namespace;$aNSName1;$aNSValue1;$aNSname2;$aNSValue2)
```

このメソッドの結果は次の通りです。

```
<MyNameSpace MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/nameSpace"
NSName1="http://www.4D.com/Prod/namespace"
NSName2="http://www.4D.com/Mkt/namespace"/>
```

参照

DOM SET XML OPTIONS

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

DOM SET XML OPTIONS

DOM SET XML OPTIONS(elementRef;エンコード{; スタンドアロン

{; インデント}})

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
エンコード	文字列	→	XML文書の文字セット
スタンドアロン	ブール	→	True=文書はスタンドアロン False（デフォルト）=文書はスタンドアロンではない
インデント	ブール	→	True（デフォルト）=文書はインデントされる False=文書はインデントされない

説明

DOM SET XML OPTIONSコマンドにより、＜要素リファレンス＞を用いて指定されたXMLツリーの作成に役立つ各種オプションを定義することができます。これらのオプションは、符号化方式、スタンドアロン属性、ならびにツリーのインデントに関連します。

■ **符号化方式：**文書で使用する文字セットを示します。デフォルトでは（このコマンドが呼び出されない場合）、UTF-8文字セット（圧縮Unicode）が使用されます。

■ **スタンドアロン：**ツリーが独立しているか（True）、または動作するために他のファイルや外部リソースが必要となるか（False）を示します。デフォルトでは（このコマンドが呼び出されない場合や引数が省略された場合）、ツリーは独立していません。

■ **インデント：**XMLキーの階層構造に従ってツリーのインデントを表示するか（True）、表示しないか（False）を示します。デフォルトでは（このコマンドが呼び出されない場合や引数が省略された場合）、ツリーはインデントされます。

例題

▼ 次の例題は、elementRefで使用する符号化方式とスタンドアロンオプションを設定します。

```
DOM SET XML OPTIONS(elementRef;;"UTF16";True)
```

参照

DOM Create XML Ref

DOM Parse XML source

DOM Parse XML source(document{; validation{; dtd}}) → 文字列

引数	タイプ		説明
document	文字列	→	ドキュメントのアクセスパス
validation	ブール	→	True= DTDによる妥当正検証 False= 妥当性検証なし
dtd	変数	→	DTDの場所
戻り値	文字列	←	XML要素リファレンス（16桁）

説明

Parse XML source関数は、XML構造を含むドキュメントを解析し、XML要素リファレンスを返します。また、このコマンドはドキュメントの妥当性を検証することができます（または、検証を行いません）。

文書の場所は、ディスク上であっても、インターネット／イントラネット上でも構いません。

引数＜document＞には、次のいずれかを渡すことができます。

■ 標準的なフルアクセスパス（Windowsでは、C:¥Folder¥File¥...、Mac OS上では、MacintoshHD:Folder:File）

■ Unixパス（http://www.site.com/Fileまたはfile://Myfile）
＜document＞にファイル名だけを渡した場合、コマンドはデータベースのストラクチャファイルと同じ階層内を検索します。Mac OSのソフトウェアパッケージの場合には、コマンドはソフトウェアパッケージと同じ階層内のファイルを検索します。

ブールタイプの引数＜validation＞を使用し、DTDを用いて構造の妥当性検証を行うかどうかを指定します。

■ ＜validation＞がTrue（真）の場合、XML構造の妥当性が検証されます。この場合、パーサーは、文書内で定義または参照されるDTD、あるいは引数＜dtd＞で指定されたDTDに基づいて、文書のXML構造の妥当性検証を試みます。

■ ＜validation＞がFalse（偽）の場合、XML構造の妥当性検証は行われません。
3番目の引数＜dtd＞を使用し、文書解析に使用する特定のDTDを指定します。この引数を使用する場合、コマンドはXML文書内で参照されるDTDを無視します。

DTDを指定する方法には2通りあります。

■ **外部参照する：**これを行うには、引数＜dtd＞に新しいDTDのフルアクセスパスを渡します。指定された文書に有効なDTDが含まれていない場合、引数＜dtd＞は無視され、エラーが生成されます。

■ **テキスト内に直接指定：**この場合、引数の内容が“<xml!”で開始していれば、それをDTDであるものとみなします。それ以外の場合、それをアクセスパスであるものとみなします。

妥当性検証が行われない場合（DTDなし、DTDへの不正なURL等）、エラーが生成されます。システム変数Errorにはエラー番号が代入されます。**ON ERR CALL**コマンドでインストールされたメソッドを使用して、このエラーを阻止することができます。

このコマンドは、メモリ上でのドキュメントの仮想構造へアクセスするための要素リファレンス（16バイトの文字列）を返します。要素リファレンスは、他のXML解析用コマンドで使用します。

例題

▼ ディスク上にあるXML文書を開き、妥当性検証は行いません。

```
$xml_Struct_Ref:=DOM Parse XML source("C:¥¥import.xml")
```

▼ データベースのストラクチャファイルと同じ階層上にあるXML文書を開き、妥当性検証は行いません。

```
$xml_Struct_Ref:=DOM Parse XML source("import.xml")
```

▼ ディスク上にあるXML文書を開き、ディスク上のDTDを使用して妥当性を検証します。

```
$xml_Struct_Ref:=DOM Parse XML source("C:¥¥import.xml";True;
                                      "C:¥¥import_dtd.xml")
```

▼ 指定されたURLにあるXML文書を開き、妥当性検証は行いません。

```
$xml_Struct_Ref:=DOM Parse XML source("http://www.4D.com/xml/import.xml")
```

参照

Parse XML variable

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。それ以外の場合には0が代入されます。

DOM Parse XML variable

DOM Parse XML variable(変数{; validation{; dtd}}) → 文字列

引数	タイプ		説明
変数	BLOB／テキスト	→	変数の名前
validation	ブール	→	True= DTDによる妥当正検証 False= 妥当性検証なし
dtd	変数	→	DTDの場所
戻り値	文字列	-	XML要素リファレンス（16桁）

説明

Parse XML variable関数は、XML構造を含むBLOBまたはテキストタイプの変数を解析し、要素リファレンスを返します。このコマンドはXMLドキュメントの妥当性を検証することができます（または、検証を行いません）。

引数＜変数＞には、XMLオブジェクトを含むBLOBやテキストタイプの変数の名前を渡します。

ブールタイプの引数＜validation＞を使用し、DTDを用いて構造の妥当性検証を行うかどうかを指定します。

■ ＜validation＞がTrue（真）の場合、XML構造の妥当性が検証されます。この場合、パーサーは、文書内で定義または参照されるDTD、あるいは引数＜dtd＞で指定されたDTDに基づいて、文書のXML構造の妥当性検証を試みます。

■ ＜validation＞がFalse（偽）の場合、XML構造の妥当性検証は行われません。

3番目の引数＜dtd＞を使用し、文書解析に使用する特定のDTDを指定します。この引数を使用する場合、コマンドはXML変数内で参照されるDTDを無視します。

DTDを指定する方法には2通りあります。

■ **外部参照する：**これを行うには、引数＜dtd＞に新しいDTDのフルアクセスパスを渡します。指定された文書に有効なDTDが含まれていない場合、引数＜dtd＞は無視され、エラーが生成されます。

■ **テキスト内に直接指定：**この場合、引数の内容が“<xml!”で開始していれば、それをDTDであるものとみなします。それ以外の場合、それをアクセスパスであるものとみなします。

妥当性検証が行われない場合（DTDなし、DTDへの不正なURL等）、エラーが生成されます。システム変数Errorにはエラー番号が代入されます。**ON ERR CALL**コマンドでインストールされたメソッドを使用して、このエラーを阻止することができます。

このコマンドは、メモリ上でのドキュメントの仮想構造へアクセスするための要素リファレンス（16バイトの文字列）を返します。要素リファレンスは、他のXML解析用コマンドで使用します。

例題

▼4Dのテキスト変数内にあるXMLオブジェクトを開き、妥当性検証は行いません。

```
C_TEXT(myTextVar)
C_TIME(vDoc)
C_STRING(16;$xml_Struct_Ref)

vDoc:=Open document("Document.xml")
If(OK=1)
    RECEIVE PACKET(vDoc;myTextVar;32000)
    CLOSE DOCUMENT(vDoc)
    $xml_Struct_Ref:=DOM Parse XML variable(myTextVar)
End if
```

▼4DのBLOB内にあるXML文書を開き、妥当性検証は行いません。

```
C_BLOB(myBlobVar)
C_STRING(16;$xml_Struct_Ref)

DOCUMENT TO BLOB(“c:¥¥import.xml” ;myBlobVar)
$xml_Struct_Ref:=DOM Parse XML variable(myBlobVar)
```

参照

Parse XML source

システム変数またはセット

コマンドが正しく実行されると、システム変数OKには1が代入されます。それ以外の場合には0が代入されます。

DOM Create XML element

DOM Create XML element(elementRef; xPath{; 属性名; 属性値}...
{; 属性名X; 属性値X}) → 文字列

引数	タイプ		説明
elementRef	文字列	→	ルートXML要素リファレンス
xPath	テキスト	→	作成するXML要素のXPathパス
属性名	文字列	→	設定する属性
属性値	文字列	→	属性の新しい値
戻り値	文字列	←	作成したXML要素の参照

説明

この関数を使用し、引数＜xPath＞に指定したパスのXML要素リファレンス＜elementRef＞内に新しい要素を作成することができます。

＜elementRef＞には、ルート要素リファレンスを渡してください（例えば、**DOM Create XML element**コマンドで作成したルート要素）。

＜xPath＞には、作成する要素のアクセスパスをXMLフォーマットで渡します（前述の「xPathノーテーションの管理」の節を参照）。そのパスに要素が存在しない場合は、要素が作成されます。

注：要素リファレンスで指定したツリーに対し、1つ以上の名前空間を定義している場合（前述のDOM Create XML Ref関数を参照）、使用する名前空間名を引数＜xPath＞の前におかなければなりません（例：“MyNameSpace:MyElement”）。

任意の引数＜属性名＞と＜属性値＞には、属性と値のペアを渡すことができます（変数、フィールド、リテラル値の形で）。必要な数のペアを渡すことができます。

コマンドは結果として作成された要素のXML参照を返します。

例題

▼次のような要素を作成してみましょう。

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
        <Elem1>
                <Elem2>
                        <Elem3/>
                </Elem2>
        </Elem1>
```

```
</RootElement>
```

これを作成するには、次のように記述します。

```
C_STRING(16;vRootRef;vRefElement)
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Create XML element(vRootRef;vxPath)
```

▼次のような要素を作成してみましょう（属性を含む）。

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
        <Elem1>
                <Elem2>
                        <Elem3 Font="Verdana" Size="10"/>
                </Elem2>
        </Elem1>
</RootElement>
```

これを作成するには、次のように記述します。

```
C_STRING(16;vRootRef;vElemRef)
C_STRING(80;$aAttrName1;$aAttrName2;$aAttrVal1;$aAttrVal2)
$aAttrName1:="Font"
$aAttrName2:="Size"
$aAttrValue1:="Verdana"
$aAttrValue2:="10"
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element
        (vRootRef;vxPath;$aAttrName1;$aAttrValue1;$aAttrName2;$aAttrValue2)
```

参照

DOM Get XML element、DOM REMOVE XML ELEMENT

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

エラー処理

エラーが返される条件は次の通りです。

■ ルート要素リファレンスが無効。

■ 作成する要素の名前が無効（例えば、数値で始まる名前）。

DOM Find XML element

DOM Find XML element(elementRef; xPath) → 文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素の参照
xPath	テキスト/BLOB	→	検索する要素のXPathパス
戻り値	文字列	←	発見された要素リファレンス（該当する場合）

説明

このコマンドを使用し、XML構造内において特定のXML要素を検索することができます。検索は、引数＜elementRef＞で指定した要素から開始されます。

検索するXMLノードは、引数＜xPath＞を用いて指定します。xPathフォーマットを使用することができます（前述の「XPathノーテーションの管理」の節を参照）。指標付きの要素も使用可能です。

注：XML標準に準拠して、検索では大文字小文字が区別されます。

コマンドは、発見された要素のXML要素リファレンスを返します。

例題

▼ この例題により、XML要素を迅速に検索し、その値を表示することができます。

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]/Title")
DOM Get XML ELEMENT VALUE(vFound;value)
ALERT("要素の値：\""+value+"\"")
```

▼ 同様の検索は、次の式を用いて実行することもできます。

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]")
vFound:=DOM Find XML element(vFound;"Book/Title")
DOM GET XML ELEMENT VALUE(vFound;value)
ALERT("要素の値： \""+value+"\"")
```

注：上記の例題の通り、xPathの記述は現在のエレメント名から始めなければなりません。このことは相対的なxPathのパスを扱う上で重要です。

参照

DOM Count XML elements、DOM Create XML element

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。エラーが返される条件は次の通りです。

エラー処理

■ ルート要素リファレンスが無効

■ 指定したxPathパスが無効

DOM Count XML elements

DOM Count XML elements(elementRef; elementName) → 倍長整数

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementName	文字列	→	カウントするXML要素の名前
戻り値	倍長整数	←	要素の数

説明

このコマンドは、引数＜elementRef＞に指定した親要素と＜elementName＞の名前に対応する“子”要素の数を返します。

参照

DOM Get XML element

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM Get XML element

DOM Get XML element(elementRef; elementName; index; elementValue)

→ 文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementName	文字列	→	取得する要素名
index	倍長整数	→	取得する要素のインデックス番号
elementValue	変数	←	要素の値
戻り値	文字列	←	XML参照（16桁）

説明

このコマンドは、引数＜elementName＞と＜index＞に対応する“子”要素への要素リファレンスを返します。

また、要素の値が引数＜elementValue＞に返されます。

参照

DOM GET XML ELEMENT VALUE

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM Get parent XML element

DOM Get parent XML element（elementRef{; parentElemName
{; parentElemValue}}）→文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
parentElemName	文字列	←	親XML要素の名称
parentElemValue	文字列	←	親XML要素の値
戻り値	文字列	←	親XML要素の参照（16文字）

説明

DOM Get parent XML element関数は、<elementRef>に渡されたXML要素リファレンスに対応する要素の、親要素のXML要素リファレンスを返します。このXML要素リファレンスは他のXML解析関数で使用します。

オプションの引数<parentElemName>および<parentElemValue>を渡すと、それぞれ、該当する親要素名および値を受け取ります。

参照

DOM Get first child XML element、DOM Get last child XML element

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入されます。そうでない場合は、0が代入されます。

DOM Get First Child XML element

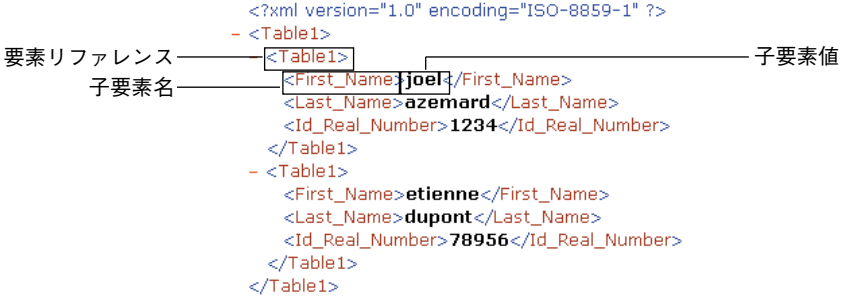
DOM Get First Child XML element(elementRef{; childElemName
{; childElemValue}}) → 文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
childElemName	文字列	←	選択されたフィールドの名前
childElemValue	文字列	←	選択されたフィールドの値
戻り値	文字列	←	XML要素リファレンス（16桁）

説明

このコマンドは、＜elementRef＞で渡したXML要素の最初の“子”への要素リファレンスを返します。この要素リファレンスは他のXML解析用コマンドで使用することができます。

引数＜childElemName＞と＜childElemValue＞を指定すると、子要素の名前と値をそれぞれ受け取ります。



例題

(1)親ルートの最初のXML要素リファレンスを検索します。まず初めに、XML構造（C:¥import.xml）がBLOBにロードされます。

```
C_BLOB(myBlobVar)
C_STRING(16;$xml_Parent_Ref;$xml_Child_Ref)

DOCUMENT TO BLOB("c:¥¥import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get First Child XML element($xml_Parent_Ref)
```

(2)親ルートの最初のXML要素リファレンスと名前、値を検索します。XML構造（C:¥import.xml）がBLOBにロードされます。

```
C_BLOB(myBlobVar)
C_STRING(16;$xml_Parent_Ref;$xml_Child_Ref)
C_TEXT($childName;$childValue)
DOCUMENT TO BLOB("c:¥¥import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get First Child XML element($xml_Parent_Ref;
                                                $childName;$childValue)
```

参照

DOM Get next sibling XML element

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。それ以外の場合には0が代入されます。

DOM Get next sibling XML element

DOM Get next sibling XML element(elementRef{; siblingElemName {; siblingElemValue}}) → 文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
siblingElemName	文字列	←	兄弟要素のフィールドの名前
siblingElemValue	文字列	←	兄弟要素のフィールドの値
戻り値	文字列	←	XML要素リファレンス（16桁）

説明

このコマンドは、渡したXML要素リファレンスの次の“兄弟”への要素リファレンスを返します。この要素リファレンスは他のXML解析用コマンドで使用することができます。

引数＜siblingElemName＞と＜siblingElemValue＞を指定すると、兄弟要素の名前と値をそれぞれ受け取ります。

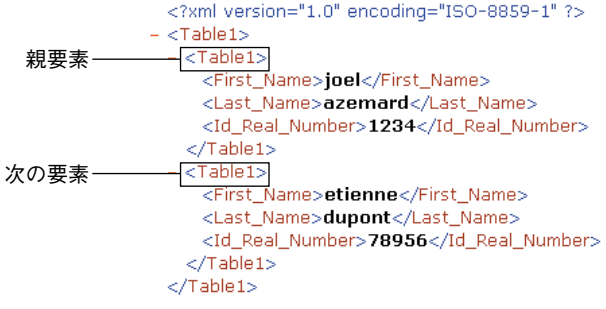
このコマンドは、引数として渡されたXML要素のすべての“兄弟”を連続して解析するために使用します。

最後の“兄弟”の解析後、システム変数OKには0が代入されます。

例題

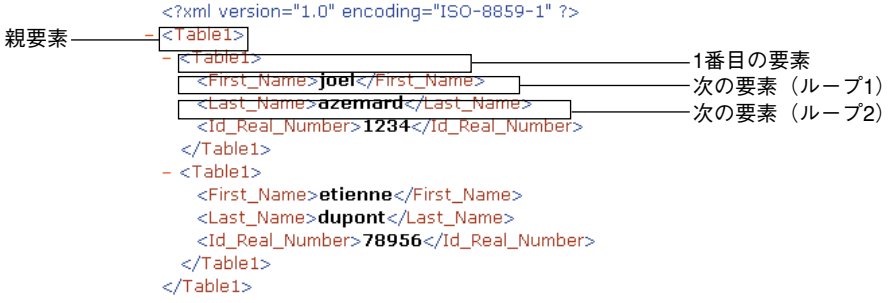
(1)引数として渡した要素に続く、次のXML要素への参照を検索します。

```
C_STRING(16;$xml_Parent_Ref;$next_XML_Ref)
$next_XML_Ref:=DOM Get next sibling XML element($xml_Parent_Ref)
```



(2)引数として渡した親要素に続く、XML要素の参照ループを検索します。

```
C_STRING(16;$xml_Parent_Ref;$first_XML_Ref;$next_XML_Ref)
$first_XML_Ref:=DOM Get First Child XML element($xml_Parent_Ref)
$next_XML_Ref:=$first_XML_Ref
While(OK=1)
    $next_XML_Ref:=DOM Get next sibling XML element($next_XML_Ref)
End while
```



参照

Get First XML element

システム変数またはセット

コマンドが正常に終了し、解析された要素が参照要素の最後の“兄弟”でなければ、システム変数OKに1が代入されます。エラーが発生するか、解析された要素が最後の“兄弟”である場合、システム変数OKには0が代入されます。

DOM Get previous sibling XML element

DOM Get previous sibling XML element (elementRef

{; siblingElemName {; siblingValue}})→文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
siblingElemName	文字列	←	兄弟要素の名前
siblingValue	文字列	←	兄弟要素の値
戻り値	文字列	←	兄弟要素のリファレンス（16文字）

説明

このコマンドは、指定した要素リファレンスの前の兄弟要素（兄）のリファレンスを返します。

このリファレンスは他のXML解析用コマンドで使用することができます。オプションの＜siblingElemName＞と＜siblingValue＞を指定すると、兄弟要素（兄）の名前と値をそれぞれ受け取ります。

このコマンドは、XML要素の“子（children）”の間でのナビゲーションに使用することができます。

最初の兄弟要素（兄）の前が指定された場合、システム変数OKには 0 が代入されます。

システム変数またはセット

コマンドが正常に実行され、要素リファレンスが最初の“兄弟（sibling）”ではない場合、システム変数OKには1が代入されます。エラーが発生した場合や解析された要素が最初の“兄弟（sibling）”であった場合には0が代入されます。

DOM Get last child XML element

DOM Get last child XML element (elementRef

{;childElemName {;childElemValue}})→文字列

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
childElemName	文字列	←	子要素の名前
childElemValue	文字列	←	子要素の値
戻り値	文字列	←	XML要素リファレンス（16文字）

説明

このコマンドは、＜elementRef＞で指定したXML要素リファレンスの最後の“子（child）”へのXMLリファレンスを返します。

このリファレンスは他のXML解析用コマンドで使用することができます。オプションの＜childElemName＞と＜childElemValue＞を指定すると、前の“子（child）”要素の名前と値をそれぞれ受け取ります。

例題

▼親ルートの最後のXML要素のリファレンスを復元します。まず初めに、XML構造（C:¥¥import.xml）がBLOBにロードされます。

```
C_BLOB(myBlobVar)
C_STRING(16;$ref_XML_Parent;$ref_XML_Child)
C_TEXT($childName;$childValue)
DOCUMENT TO BLOB("c: ¥¥import.xml";myBlobVar)
$ref_XML_Parent:=DOM Parse XML variable(myBlobVar)
⇒ $ref_XML_Child:=DOM Get last child XML element
                    ($ref_XML_Parent;$childName;$childValue)
```

参照

DOM Get first child XML element

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入されます。それ以外では0が代入されます。

DOM SET XML ELEMENT NAME

DOM SET XML ELEMENT NAME(elementRef; elementName)

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementName	文字列	→	要素の新しい名前

説明

このコマンドを使用し、＜elementRef＞で指定した要素の名前を変更することができます。

＜elementRef＞には名前を変更する要素の参照を渡し、＜elementName＞には要素の新しい名前を渡します。さらに、このコマンドは要素の開始タグと終了タグの更新も行います。

例題

▼次のようなXMLソースの場合

```
<Book>
	<Title>The Best Seller</Title>
</Book>
```

要素“Book”への参照をvElemRefに代入し、次のコードを実行します。

```
DOM SET XML ELEMENT NAME(vElemRef;"BestSeller")
```

結果は次の通りです。

```
<BestSeller>
	<Title>The Best Seller</Title>
</BestSeller>
```

参照

DOM GET XML ELEMENT NAME

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

エラー処理

エラーが返される条件は次の通りです。

■ 要素リファレンスが無効

■ 作成する要素の新しい名前が無効（例えば、数値で始まる名前）

DOM GET XML ELEMENT NAME

DOM GET XML ELEMENT NAME(elementRef; elementName)

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementName	変数	←	要素の名前

説明

このコマンドは、＜elementRef＞で指定されたXML要素の名前を引数＜elementName＞に代入して返します。XML要素名についての詳細は「XMLについて」の節を参照してください。

例題

▼ このメソッドは、要素$xml_Element_Refの名前を返します。

```
C_STRING(16;$xml_Element_Ref)
C_TEXT($name)
DOM GET XML ELEMENT NAME($xml_Element_Ref;$name)
```

参照

DOM Get XML element、DOM GET XML ELEMENT VALUE、DOM SET XML ELEMENT NAME

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM SET XML ELEMENT VALUE

DOM SET XML ELEMENT VALUE(elementRef; elementValue)

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementValue	文字列 ¦ 変数	→	要素の新しい値

説明

このコマンドを使用し、＜elementRef＞で指定した要素の値を変更することができます。

＜elementRef＞には、指定した要素の新しい値を格納した文字列、または変数（またはフィールド）を渡します。

■ 文字列を渡すと、その値は“現状のまま”XML構造上で使用されます。

■ 変数またはフィールドを渡すと、4th Dimensionは＜elementValue＞のタイプに応じてその値を処理します。配列、ピクチャとポインタ以外のすべてのデータタイプを使用することができます。

例題

▼次のようなXMLソースの場合

```
<Book>
	<Title>The Best Seller</Title>
</Book>
```

要素“Title”への参照をvElemRefに代入し、次のコードを実行します。

```
DOM SET XML ELEMENT VALUE(vElemRef;"The Loser")
```

結果は次の通りです。

```
<Book>
	<Title>The Loser</Title>
</Book>
```

参照

DOM GET XML ELEMENT NAME.

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます（例えば、要素リファレンスが無効）。

DOM GET XML ELEMENT VALUE

DOM GET XML ELEMENT VALUE(elementRef; elementValue{; cDATA})

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
elementValue	変数	←	要素の値
cDATA	変数	←	cDATAセクションの内容

説明

このコマンドは、＜elementRef＞で指定されたXML要素リファレンスの値を＜element Value＞に代入して返します。4th Dimensionは、取得した値を引数として指定された変数と同じタイプに変換します。

任意の引数＜cDATA＞を使用して、XML要素＜要素リファレンス＞のcDATAセクションの内容を受け取ります。引数＜elementValue＞の場合と同様に、4th Dimensionは取得した値を変換し、引数として渡した変数と同じタイプに変えようとします。

例題

▼ このメソッドは、要素$xml_Element_Refの値を返します。

```
C_STRING(16;$xml_Element_Ref)
C_REAL($value)
DOM GET XML ELEMENT VALUE($xml_Element_Ref;$value)
```

参照

DOM Get XML element、DOM GET XML ELEMENT NAME、DOM SET XML ELEMENT VALUE

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM REMOVE XML ELEMENT

DOM REMOVE XML ELEMENT(elementRef)

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス

説明

このコマンドをは、＜elementRef＞で指定した要素を削除します。

参照

DOM Create XML element

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。エラーが返される条件は次の通りです。

■ 要素リファレンスが正しくない

■ 要素が空

DOM Count XML attributes

DOM Count XML attributes(elementRef) → 倍長整数

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
戻り値	倍長整数	←	属性の数

説明

このコマンドは、＜elementRef＞で指定されたXML要素内にあるXML属性の数を返します。XML属性についての詳細は「XMLについて」の節を参照してください。

例題

▼配列内の要素の値を取得する前に、次のXML要素内の属性の数を知りたいものとします。

```
<?xml version="1.0" ?>
- <LINES>
    <LINE N="1" Font="verdana" size="10">this is a line</LINE>
    <LINE N="2" Font="charcoal" size="12">and another line</LINE>
    <LINE N="3" Font="verdana" size="18">and we stop here</LINE>
  </LINES>
```

```
C_BLOB(myBlobVar)
C_STRING(16;$xml_Parent_Ref;$xml_Child_Ref)
C_TEXT(myResult)
C_LONGINT($numAttributes)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get First Child XML element($xml_Parent_Ref)
$numAttributes:=DOM Count XML attributes($xml_Child_Ref)
ARRAY TEXT(tAttrib;$numAttributes)
Loop($i;1;$numAttributes)
    DOM GET XML ATTRIBUTE BY INDEX($xml_Child_Ref;$i;tAttrib{$i})
End of loop
```

上記の例題では、$numAttributesは3であり、tAttrib{1}には“Font”、tAttrib{2}には“N”、tAttrib{3}には“size”が代入されます。

注：テキストフォーム上に表示されたXMLファイル内の属性の位置とインデックス番号は対応していません。XMLにおいて、属性のインデックスは、属性が（属性名に基づいて）アルファベット順に並べられた際の位置を表わします。

参照

DOM Count XML elements

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM SET XML ATTRIBUTE

DOM SET XML ATTRIBUTE(elementRef; attribName; attribValue...

{; attribName X; attribValue X})

引数	タイプ		説明
要素参照	文字列	→	XML要素参照
属性名	文字列	→	セットする属性
属性値	文字列	→	属性の新しい値

説明

このコマンドを使用すると、引数＜要素参照＞に渡した参照番号を持つXML要素に対し、1つ以上の属性を追加することができます。また、定義される各属性の値も設定することができます。

設定する属性ならびにその値は、それぞれ引数＜属性名＞と＜属性値＞に渡します（変数、フィールド、リテラル値の形で）。必要な数の属性と値のペアを渡すことができます。

例題

▼ 次のXMLソースの場合

```
<Book>
<Title>The Best Seller</Title>
</Book>
```

次のコードを実行します。

```
vAttrName:="Font"
vAttrVal:="Verdana"
DOM SET XML ATTRIBUTE(vElemRef;vAttrName;vAttrVal)
```

この結果は次の通りです。

```
<Book>
<Title Font=Verdana>The Best Seller</Title>
</Book>
```

参照

DOM GET XML ATTRIBUTE BY INDEX、DOM GET XML ATTRIBUTE BY NAME

システム変数とセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

DOM GET XML ATTRIBUTE BY INDEX

DOM GET XML ATTRIBUTE BY INDEX(elementRef; attribIndex; attribName {; attribValue})

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
attribIndex	文字列	→	属性のインデックス番号
attribName	変数	←	属性の名前
attribValue	変数	←	属性の値

説明

このコマンドを使用し、インデックス番号で指定した属性の名前、ならびにオプションとしてその値を調べることができます。

＜elementRef＞にはXML要素の参照を渡し、＜attribIndex＞には名前を知りたい属性のインデックス番号を渡します。引数＜attribName＞には、属性名が返され、引数＜attribValue＞を渡した場合には、この引数に属性の値が返されます。この場合、4th Dimensionは、取得した値を引数として渡された変数と同じタイプに変換します。

注：インデックス番号は、テキストフォーム上に表示されたXMLファイル内の属性の位置と対応していません。XMLにおいて、属性のインデックスは、属性が（属性名に基づいて）アルファベット順に並べられた際の位置を表わします。実例に関しては、後述するDOM Compter attributs XML関数の例題を参照してください。

＜attribIndex＞に渡された値が、XML要素内にある属性の数よりも大きい場合、エラーが返されます。

例題

▼ 後述する**Count XML attributes**関数の例題を参照してください。

参照

DOM GET XML ATTRIBUTE BY NAME

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

DOM GET XML ATTRIBUTE BY NAME

DOM GET XML ATTRIBUTE BY NAME(elementRef; attribName; attribValue)

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
attribName	文字列	→	属性の名前
attribValue	変数	←	属性の値

説明

このコマンドを使用し、名前により指定された属性の値を取得することができます。

＜elementRef＞にはXM要素リファレンスを渡し、＜attribName＞には値を知りたい属性の名前を渡します。引数＜attribValue＞には、属性の値が返されます。4th Dimensionは、取得した値を引数として渡された変数と同じタイプに変換します。

XML要素内に＜attribName＞で指定した属性が存在しない場合、エラーが返されます。指定されたものと同じ名前を持つXML要素の属性がいくつか存在する場合、最初に発見された属性の値だけが返されます。

例題

▼ このメソッドを使用し、名前を用いてXML属性の値を取得します。

```
C_BLOB(myBlobVar)
C_STRING(16;$xml_Parent_Ref;$xml_Child_Ref)
C_LONGINT($LineNum)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get First Child XML element($xml_Parent_Ref)
DOM GET XML ATTRIBUTE BY NAME($xml_Child_Ref;"N";$LineNum)
```

このメソッドを下記の例に対して実行すると、$LineNumには値1が納められます。

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove Isate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

参照

GET XML ATTRIBUTE BY INDEX

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。それ以外の場合には0が代入されます。

DOM Get XML information XML

DOM Get XML information(elementRef; xmlInfo) → 文字列

引数	タイプ		説明
elementRef	文字列	→	XMLルート要素のリファレンス
xmlInfo	倍長整数	→	取得する情報のタイプ
戻り値	文字列	←	XML情報の値

説明

この関数を使用して、引数<elementRef>で指定したXML要素についての様々な情報を取得します。

<xmlInfo>には、取得する情報のタイプを示すコードを渡します。「XML」テーマ内にある以下の定義済み定数を使用することができます。

定数	タイプ	値
PUBLIC ID	倍長整数	1
SYSTEM ID	倍長整数	2
DOCTYPE Name	倍長整数	3
Encoding	倍長整数	4
Version	倍長整数	5
Document URI	倍長整数	6

これらの定数は、次の情報を表します。

■ PUBLIC ID：ドキュメントファイル（document）に一致するDTDの公開識別子（FPI）（DOCTYPE xxx PUBLICタグがある場合）

■ SYSTEM ID：システム識別子

■ DOCTYPE Name：DOCTYPEマーカーで定義されたルート要素の名前

■ Encoding：使用される符号化方式（UTF-8、ISOなど）

■ Version：使用可能なXMLのバージョン

■ Document URI：DTDのURL

参照：

GET XML ERROR

定数

XML theme

DOM EXPORT TO FILE

DOM EXPORT TO FILE(elementRef; filePath)

引数	タイプ		説明
elementRef	文字列	→	ルートXML要素リファレンス
filePath	テキスト	→	ファイルのフルアクセスパス

説明

このコマンドを使用し、XMLツリーをディスク上のファイルに保存することができます。

引数＜elementRef＞には、書き出すルート要素の参照を渡します。

＜filePath＞には、使用または作成する書き出しファイルのフルアクセスパスを渡します。ファイルが存在しない場合は作成されます。

ファイル名だけを渡した場合（アクセスパスなし）、そのファイルの検索が行われるか、またはストラクチャファイルと同じ階層にファイルが作成されます。

空の文字列（""）を渡すと、ファイルや新規ファイルを開く標準のダイアログボックスが表示されます。

この例題は、ツリー「vElemRef」を「MyDoc.xml」ファイルに保存します。

```
DOM EXPORT TO FILE(vElemRef;"C:¥¥folder¥MyDoc.xml")
```

参照

DOM EXPORT TO VAR

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

エラー処理

エラーが返される条件は次の通りです。

■ 要素リファレンスが無効

■ 指定したアクセスパスが無効

■ 保存先のボリュームからエラーが返される（ディスクの空き容量が不十分など）

DOM EXPORT TO VAR

DOM EXPORT TO VAR(elementRef; vXmlVar)

引数	タイプ		説明
elementRef	文字列	→	ルートXML要素のリファレンス
vXmlVar	テキスト¦BLOB	→	XMLツリーを受け取る変数

説明

このコマンドを使用し、XMLツリーをテキストまたはBLOBタイプの変数に保存することができます。

引数＜elementRef＞には、書き出すルート要素のリファレンスを渡します。

＜vXmlVar＞には、XMLツリーを格納する変数の名前を渡します。この変数は、テキストまたはBLOBのいずれかのタイプでなくてはなりません。次に実行しようとする動作や、ツリーの予想サイズ（テキストタイプの変数のサイズは、32,000バイトまでに制限される点に注意）に応じて、そのタイプを選択することができます。

要素がテキスト変数に保存される場合、日本語環境ではシフトJISコードが使用されます。。つまり、返されたテキストのオリジナルのエンコード（encoding=”xxx”）は失われます。この場合は、＜vVarXml＞変数によりコードの閲覧または保存ができますが、有効なXMLドキュメント（例えば**SEND PACKET**コマンドを使用した場合のような）は生成されません。

例題

▼ この例題は、ツリー「vElemRef」をテキスト変数に保存します。

```
C_TEXT(vtMyText)
DOM EXPORT TO VAR(vElemRef;vtMyText)
```

参照

DOM EXPORT TO FILE

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます（例えば、要素リファレンスが無効）。

DOM CLOSE XML

DOM CLOSE XML(elementRef)

引数	タイプ		説明
elementRef	文字列	→	XMLルート要素リファレンス

説明

このコマンドは、＜elementRef＞で指定したXMLオブジェクトが占有していたメモリを解放します。

＜elementRef＞がXMLのルートオブジェクトではない場合、エラーが生成されます。

参照

DOM Parse XML source, DOM Parse XML variable

システム変数またはセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

SAX SET XML OPTIONS

SAX SET XML OPTIONS(document; encoding{; standalone{; indentation}})

引数	タイプ		説明
document	時間	→	開かれているドキュメントの参照番号
encoding	文字列	→	XML文書の文字セット
standalone	ブール	→	True=文書はスタンドアロン False（デフォルト）=文書はスタンドアロンではない
indentation	ブール	→	True（デフォルト）=文書はインデントされる False=文書はインデントされない

説明

SAX SET XML OPTIONSコマンドは、引数に渡された値を用いて、＜document＞で参照されるXML文書を初期化します。これらの引数を使用し、エンコード、スタンドアロン属性、ならびに文書のインデントを決定することができます。

■ encoding（符号化方式）：文書で使用する文字セットを示します。デフォルトとして（このコマンドが呼び出されない場合）、UTF-8文字セット（圧縮Unicode）が使用されます。

■ standalone：文書が独立しているか（True）、または動作するために他のファイルや外部リソースを必要とするか（False）を示します。デフォルトでは（このコマンドが呼び出されない場合や引数が省略された場合）、ドキュメントは独立していません。

■ indentation：XMLキーの階層構造に従って文書のインデントを表示するか（True）、表示しないか（False）を示します。デフォルトでは（このコマンドが呼び出されない場合や引数が省略された場合）、文書はインデントされます。

このコマンドは、文書内の最初のセットコマンドの前、XML文書毎に一度呼び出さなくてはなりません。これを行わないと、エラーメッセージが生成されます。

例題

▼ 次の命令の場合

```
SAX SET XML OPTIONS($RefDoc;"UTF-16";True)
```

文書内には、次の行が作成されます。

```
<?xml version="1.0"encoding="UTF-16"standalone="yes"?>
```

参照

SAX GET XML DOCUMENT VALUES

SAX GET XML DOCUMENT VALUES

SAX GET XML DOCUMENT VALUES(document; encoding; version, standalone)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
encoding	文字列	←	XML文書の文字セット
version	文字列	←	XMLバージョン
standalone	ブール	←	True=文書は独立している、それ以外の場合はFalse

参照

SAX GET XML DOCUMENT VALUESコマンドは、＜document＞で参照されるXML文書のXMLヘッダから基本情報を取得します。

コマンドは、符号化方式のタイプ、バージョン、文書の“independent”属性を引数＜encoding＞、＜version＞、＜standalone＞にそれぞれ返します。このコマンドはSAXイベントである「**XML Start Document**」と一緒に使用しなくてはなりません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

参照

SAX Get XML node、SAX SET XML OPTIONS

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX ADD XML DOCTYPE

SAX ADD XML DOCTYPE(document; docType)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
docType	文字列	→	追加されるDocType

説明

SAX ADD XML DOCTYPEコマンドは、＜document＞で参照されるXML文書にDocType命令を追加します。

DocType命令により、文書が書き込まれたXMLのタイプを示し、使用されるDocument Type Declaration（DTD）を指定することができます。一般的に、DocType命令は次の形式になります。

<!DOCTYPE XML_type "DTD_address”>

例題

▼次の命令の場合

```
vDocType:= "SYSTEM Books¥"Book.DTD¥""
SAX ADD XML DOCTYPE($RefDoc;vDocType)
```

文書内には、次の行が作成されます。

<!DOCTYPE SYSTEM Books "Book.DTD”>

参照

SAX ADD XML COMMENT

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

エラー処理

エラーの場合、コマンドはエラーを返し、このエラーはエラー処理メソッドを用いて遮ることができます。

SAX Get XML node

SAX Get XML node(document) → 倍長整数

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
戻り値	倍長整数	←	関数から返されるイベント

説明

このコマンドは倍長整数を返し、＜document＞で参照されるXML文書の解析時に返されたSAXイベントのタイプを示します。

返されるイベントの値は、「XML」テーマの定数として入手することができます。

定数	タイプ	値
XML Start Document	倍長整数	1
XML Comment	倍長整数	2
XML Processing Instruction	倍長整数	3
XML Start Element	倍長整数	4
XML End Element	倍長整数	5
XML DATA	倍長整数	6
XML CDATA	倍長整数	7
XML Entity	倍長整数	8
XML End Document	倍長整数	9

例題

▼次の例題は任意のイベントを処理します。

```
$RefDoc:=Open document("";"";Read Mode)
If (OK=1)
    Repeat
        $MyEvent:=SAX Get XML node($RefDoc)
        Case of
            ¥ ($MyEvent=XML Start Document)
                `処理1を実行
            ¥ ($MyEvent=XML Comment)
                `処理2を実行
        End case
    Until($MyEvent=XML End Document)
End if
CLOSE DOCUMENT($RefDoc)
```

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX ADD XML COMMENT

SAX ADD XML COMMENT(document; comment)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
comment	文字列	→	追加されるコメント

説明

SAX ADD XML COMMENTコマンドは、＜document＞で参照されるXML文書にコメントを追加します。

XMLコメントはテキストであり、その内容はXMLインタプリタによって解析されません。XMLコメントは“<!--”と“-->”文字で囲まなくてはなりません。

例題

▼次の命令の場合

```
vComment:= "Created by 4th Dimension"
SAX ADD XML COMMENT($RefDoc;vComment)
```

文書内には、次の行が作成されます。

<!--Created by 4th Dimension-->

参照

SAX ADD XML DOCTYPE

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

エラー処理

エラーの場合、コマンドはエラーを返し、このエラーはエラー処理メソッドを用いて遮ることができます。

SAX GET XML COMMENT

SAX GET XML COMMENT(document; comment)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
comment	文字列	←	XMLコメント

説明

このコマンドは、＜document＞で参照されるXML文書においてXML Commentタイプの SAXイベントが生成されると、コメントを返します。SAXイベントについての詳細は **SAX Get XML node**コマンドを参照してください。

参照

SAX ADD XML COMMENT、SAX Get XML node

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0 が代入されてエラーが返されます。

SAX OPEN XML ELEMENT

SAX OPEN XML ELEMENT(document; tag{; attribName; attribValue}
...{;attribNameX;attribValueX})

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
tag	文字列	→	オープンする要素の名前
attribName	文字列	→	属性の名前
attribValue	文字列	→	属性の値

説明

SAX OPEN XML ELEMENTコマンドを使用し、＜document＞により参照されるXML文書に新しい要素を追加し、さらにオプションとして属性とその値を追加することができます。

追加された要素は文書内において“オープン”な状態（終了タグが追加されていない）です。このコマンドを使用して作成された要素をクローズするには、次のいずれかを行います。

■ **SAX CLOSE XML ELEMENT**コマンドを使用します。

■ または、XML文書をクローズします。この場合、必要に応じて4th Dimensionは自動的にクローズXMLタグを付加します。

＜tag＞には、作成される要素の名前を渡します。この名前には、文字、数値、“.”、“-”、“_”、“:”などの記号だけを含めることができます。

オプションとして、コマンドは引数＜attribName＞と＜attribValue＞を用いて1つ以上の属性と値のペアを渡すことができます（変数やフィールド、リテラル値の形で）。必要な数の属性と値のペアを渡すことができます。

例題

▼次の命令の場合、

```
vElement:= "Book"
SAX OPEN XML ELEMENT($RefDoc;vElement)
```

文書内には、次の行が作成されます。

```
<Book>
```

参照

SAX CLOSE XML ELEMENT、SAX OPEN XML ELEMENT ARRAYS

エラー処理

無効な文字を＜tag＞に渡すと、エラーが生成されます。

SAX OPEN XML ELEMENT ARRAYS

SAX OPEN XML ELEMENT ARRAYS(document; tag{; attribNamesArray;
attribValuesArray}{;attribNamesArray2; attribValuesArray2; ...;
attribNamesArrayN; attribValuesArrayN})

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
tag	文字列	→	オープンする要素の名前
attribNameArray	文字列配列	→	属性名の配列
attribValueArray	文字列配列	→	属性値の配列

説明

SAX OPEN XML ELEMENT ARRAYSコマンドを使用すると、＜document＞に渡された参照番号を持つXML文書に新しい要素を追加し、さらにオプションとして配列形式で提供される属性とその値を追加することができます。

配列がサポートされる点を除けば、このコマンドは**SAX OPEN XML ELEMENT**コマンドと同じです。このコマンドの動作に関する詳細については、**SAX OPEN XML ELEMENT**コマンドの説明を参照してください。

オプションとして、**SAX OPEN XML ELEMENT ARRAYS**コマンドは、引数＜attribNamesArray＞と＜attribValuesArray＞の配列を用いて属性と値のペアを渡すことができます。

この配列は事前に作成し、属性と属性値を対で使用しなくてはなりません。配列のペアおよび各ペア内の項目は、必要な数だけ渡すことができます。

例題

▼次の4Dコードの場合

```
ARRAY STRING(80;tAttrNames;2)
ARRAY STRING(80;tAttrValues;2)
vElement:="Book"
tAttrNames{1}:="Font"
tAttrValues{1}:="Arial"
tAttrNames{2}:="Style"
```

```
tAttrValues{2}:="Bold"
SAX OPEN XML ELEMENT ARRAYS($RefDoc;vElement;tAttrNames;tAttrValues)
```

文書内には、次の行が作成されます。

```
<Book Font="Arial" Style="Bold">
```

参照

SAX CLOSE XML ELEMENT、SAX OPEN XML ELEMENT

SAX CLOSE XML ELEMENT

SAX CLOSE XML ELEMENT(document)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号

説明

＜document＞で参照されるXML文書において、**SAX CLOSE XML ELEMENT**コマンドは、**SAX OPEN XML ELEMENT**コマンドで前回オープンされた要素のクローズに必要な命令を記述します。

このコマンドの使用は任意です。実際、4th DimensionはXML文書のクローズ時に、閉じられていない要素に対して必要な終了タグを自動的に追加します。

例題

▼ 前回オープンされた要素が <Book> である場合、次の命令を実行します。

```
SAX CLOSE XML ELEMENT($RefDoc)
```

文書内には、次の行が作成されます。

```
</Book>
```

参照

SAX OPEN XML ELEMENT、SAX OPEN XML ELEMENT ARRAYS

SAX GET XML ELEMENT

SAX GET XML ELEMENT (document; name; prefix; attrNames; attrValues)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
name	文字列	←	要素の名前
prefix	文字列	←	名前空間
attribName	文字列配列	←	属性の名前
attribValue	文字列配列	←	属性の値

説明

このコマンドは、引数＜document＞で参照されるXML文書内に存在する要素＜要素名＞に関する各種情報を返します。このコマンドは、「**XML Start Element**」イベントまたは「**XML End Element**」イベントとともに呼び出さなくてはなりません。「**XML End Element**」イベントの特例として、属性パラメータが処理されません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

引数＜name＞には要素の名前が代入されます。

引数＜prefix＞には要素の名前空間（ネームスペース）が代入されます。要素に名前空間が関連付けられていない場合、この引数は空の文字列を返します。

コマンドは、配列＜attribName＞に対象となる要素の属性名を代入します。必要に応じて、コマンドは配列を自動的に作成し、そのサイズを調整します。

また、コマンドは、配列＜attribValue＞に対象となる要素の属性の値を代入します。必要に応じて、コマンドは配列を自動的に作成し、そのサイズを調整します。

例題

▼ 次のXMLコードについてみてみましょう。

```
<RootElement>
	<Child Att1="111" Att2="222" Att3="333">MyText</Child>
</RootElement>
```

次の命令を実行すると

	SAX GET XML ELEMENT(DocRef;vName;vPrefix;attrNames;attrValues)

vNameには“Child”

vPrefixには“”（空）

attrNames{1}には“Att1”、attrNames{2}は“Att2”、attrNames{3}は“Att3”

attrValues{1}には“111”、attrValues{2}には“222”、attrValues{3}には“333”

がそれぞれ代入されます。

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX ADD XML ELEMENT VALUE

SAX ADD XML ELEMENT VALUE(document;data)

引数	タイプ		説明
document	時間	→	開かれているドキュメント参照番号
data	テキスト¦変数	→	文書に挿入するテキストまたは変数

説明

SAX ADD XML ELEMENT VALUEコマンドは、＜document＞で参照されるXML文書内に、テキストや変数の内容＜data＞を変換せずに直接追加します。このコマンドは、例えば、電子メールの本文に添付書類を挿入する動作と同じです。

＜data＞には、4D変数もしくは文字列を直接渡すことができます。変数の内容はXMLドキュメントに格納される前にテキストに変換されます（数値タイプを渡した場合は、自動的に数値の文字列に変換され、ブールを渡した場合は“True”または“False”、時間を渡した場合は“0000-00-00T20:33:58”、日付を渡した場合は“2005-11-16T00:00:00”に変換されます）。

＜data＞の内容を符号化したい場合は、**ENCODE**コマンドを使用しなくてはなりません。その際、もちろん＜data＞にはBLOBを渡さなくてはいけません。

このコマンドが正しく動作するために、要素は必ずオープンな状態にしておいてください。オープンな状態でなければ、エラーが生成されます。コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

例題

▼ この例題は、開かれているXML要素内にwhitepaper.pdfファイルを挿入します。

```
C_BLOB(vBMyBLOB)
DOCUMENT TO BLOB("c:¥¥whitepaper.pdf";vBMyBLOB)
SAX ADD XML ELEMENT VALUE($RefDoc;vBMyBLOB)
```

参照

SAX GET XML ELEMENT VALUE

SAX GET XML ELEMENT VALUE

SAX GET XML ELEMENT VALUE(document;value)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
value	テキスト¦BLOB	←	要素の値

説明

SAX GET XML ELEMENT VALUEコマンドを使用し、引数＜document＞で参照されるXML文書内に存在するXML要素の＜value＞を取得することができます。このコマンドは、「**XML DATA**」イベントとともに呼び出さなくてはなりません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

引数＜value＞には、テキストまたはBLOBタイプの変数を渡します。BLOBを渡した場合、テキストはそのままの状態で返されます（修正されない）。

例題

▼次のXMLコードについてみてみましょう。

```
<RootElement>
<Child Att1="111" Att2="222" Att3="333">MyText</Child>
</RootElement>
```

次の命令を実行すると、vValueには“MyText”が返されます。

```
SAX GET XML ELEMENT VALUE(DocRef;vValue)
```

参照

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX ADD XML CDATA

SAX ADD XML CDATA(document; data)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
data	テキスト¦BLOB	→	文書内のCDataタグの間に挿入するテキストまたはBLOB

説明

SAX ADD XML CDATAコマンドは、＜document＞で参照されるXML文書内へテキストやBLOBタイプの＜data＞を追加します。この＜data＞は、自動的に<CData>と </CData>タグの間に入れられます。CDataセクションに組み込まれたテキストは、XMLインタプリタより無視されます。

＜data＞の内容を暗号化したい場合は、**ENCODE**コマンドを使用しなくてはなりません。その際はもちろん、＜data＞にBLOBを渡さなくてはなりません。

このコマンドが正しく動作するために、要素は必ずオープンな状態にしておいてください。オープンな状態でなければ、エラーが生成されます。

例題

▼ XML文書に次の行を挿入したいものとします。

```
function matchwo(a,b)
{
	if(a< b&& a< 0) then
	{
		return 1
	}
else
	{
		return 0
	}
}
```

これを行うには、次のコードを実行するだけです。

```
C_TEXT(vtMytext)
	` 変数vtMytextのテキストをここに配置する
SAX ADD XML CDATA($RefDoc;vtMytext)
```

結果は次のようになります。

```
<![CDATA[
function matchwo(a,b)
{
if(a<b&&a<0)then
        {
        return 1
        }
else
        {
        return 0
        }
}
]]>
```

参照

SAX GET XML CDATA

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

SAX GET XML CDATA

SAX GET XML CDATA(document; value)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
value	BLOB	←	要素の値

説明

SAX GET XML CDATAコマンドを使用し、引数＜document＞で参照されるXML文書内に存在するXML要素のcDATAの＜value＞を取得することができます。このコマンドは、「**XML CDATA**」イベントとともに呼び出さなくてはなりません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

データはそのままの状態で返されます（データは修正されない）。

例題

▼ 次のXMLコードについてみてみましょう。

```
<RootElement>
<Child>MyText<![CDATA[MyCData</Child>
</RootElement>
```

次の4Dコードを実行した場合

```
C_BLOB(vData)
C_TEXT(vTextData)
SAX GET XML CDATA(DocRef;vData)
vTextData:=BLOB to text(vData;C string)
```

vTextDataには“MyCData”が返されます。

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX ADD PROCESSING INSTRUCTION

SAX ADD PROCESSING INSTRUCTION(document; statement)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
statement	テキスト	→	文書に挿入するテキストまたはBLOB

説明

SAX ADD PROCESSING INSTRUCTIONコマンドは、＜document＞で参照されるXML文書内へXML処理手続きである＜statement＞を追加します。

処理命令により、アプリケーションタイプを指定したり、必要な場合は未処理の外部エンティティを処理する追加パラメータを指定することができます。

このコマンドは、XMLに準拠して命令文のデータの書式を整えます。しかし、命令文そのものは解析されないため、開発者の責任において構文が有効であることを確認してください。

例題

▼次のコードの場合

```
vtStatement:="xml-stylesheet type="+Char(Quote)+"text/xsl"+
Char(Quote)+ "href="+Char(Quote)+
"style.xsl"+Char(Quote)
SAX ADD PROCESSING INSTRUCTION($RefDoc;vtStatement)
```

文書内には、次の行が作成されます。

```
<?xml-stylesheet type="text/xsl"href="style.xsl"?>
```

参照

SAX GET XML PROCESSING INSTRUCTION

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

SAX GET XML PROCESSING INSTRUCTION

SAX GET XML PROCESSING INSTRUCTION(document; name; value)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
name	文字列	←	命令の名前
value	文字列	←	命令の値

説明

このコマンドは、＜document＞で参照されるXML文書において処理されたXML命令の＜name＞と＜value＞を返します。このコマンドは、「**XML Processing Instruction**」イベントと一緒に呼び出さなくてはなりません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

例題

▼次のXMLコードについてみてみましょう。

```
<?xml version="1.0"encoding="UTF-8"?>
<!--Edited with XML Spy v3.0.7 NT(http://www.xmlspy.com)by Myself(4D SA)-->
<?PI TextProcess?>
<!DOCTYPE RootElement SYSTEM"ParseTest.dtd">
```

次の命令を実行すると

SAX GET XML PROCESSING INSTRUCTION($RefDoc;vName;vValue)

vNameには“PI”が、vValueには“TextProcess”が返されます。

参照

SAX ADD PROCESSING INSTRUCTION、SAX Get XML node

SAX GET XML ENTITY

SAX GET XML ENTITY(document; name; value)

引数	タイプ		説明
document	時間	→	開かれている文書の参照番号
name	文字列	←	エンティティ（実体）の名前
value	文字列	←	エンティティの値

説明

SAX GET XML ENTITYコマンドを使用し、引数＜document＞で参照されるXML文書内に存在するXMLエンティティの＜value＞を取得することができます。このコマンドは、「**XML Entity**」イベントとともに呼び出さなくてはなりません。SAXイベントについての詳細は**SAX Get XML node**コマンドを参照してください。

例題

▼次のXMLコードについてみてみましょう。

```
<?xml version="1.0"encoding="UTF-8"?>
<!DOCTYPE body[
        <!ELEMENT body(element*)>
        <!ELEMENT element(#PCDATA)>
        <!ENTITY name"Replacement">
]>
<body>
        <element>Entity updated by&name;</element>
</body>
```

次の命令を実行すると

SAX GET XML ENTITY(DocRef;vName;vValue)

vNameには“name”が、vValueには“Replacement”が返されます。

参照

SAX Get XML node

システム変数

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されてエラーが返されます。

GET XML ERROR

GET XML ERROR(elementRef; errorText{; row{; column}})

引数	タイプ		説明
elementRef	文字列	→	XML要素リファレンス
errorText	変数	←	エラーテキスト
row	変数	←	行番号
column	変数	←	列番号

説明

このコマンドは、引数＜elementRef＞で指定されたXML要素を処理する際に発生したエラーの説明を引数＜errorText＞に代入して返します。返される情報は、Xerces.DLLライブラリから提供されます。

任意の引数＜row＞と＜column＞は、エラー箇所を示します。指定した変数には行番号、およびエラーの原因となる表現式の最初の桁位置がそれぞれ返されます。

参照

DOM Parse XML information

システム変数とセット

コマンドが正常に終了すると、システム変数OKには1が代入されます。エラーが発生した場合には0が代入されます。

APPLY XSLT TRANSFORMATION

APPLY XSLT TRANSFORMATION(xmlソース; xslシート; 結果)

引数	タイプ		説明
xmlソース	文字列¦BLOB	→	XMLソース文書の名前またはアクセスパス、またはXMLソースを納めたBLOB
xslシート	文字列¦BLOB	→	XSLスタイルシートを納めた文書の名前またはアクセスパス、またはXSLスタイルシートを納めたBLOB
結果	文字列¦BLOB	→	XSLT変換の結果を受け取る文書の名前またはアクセスパス、またはXSLT変換の結果を受け取るBLOB

説明

APPLY XSLT TRANSFORMATIONコマンドは、XMLを納めた文書やBLOBに対してXSLT変換を適用し、文書またはBLOBを生成します。このコマンドのスコープはカレントプロセスです。

注：XSL変換（XSLT）についての詳細は、「XMLについて」の節を参照してください。

コマンドは、3つのBLOBまたは文字列タイプの引数を必要とします。

警告：このコマンドは、変数またはフィールドのみを引数として受け入れます。

文字列を渡した場合は、文書を指定します。その際、文書の名前（文書はデータベースストラクチャと同じ階層に置かなくてはならない）またはフルアクセスパスだけを渡すことができます。

一度の呼び出しにおいて、異なるタイプの引数を混ぜて使用することができます。しかし、空の引数を渡すことはできません（BLOBまたはテキスト）。

■ 引数＜xmlソース＞には、変換するXMLソースを納めなくてはなりません。コマンドはXMLコードの妥当性を検証しないため、XMLコードは必ず妥当でなくてはなりません。

■ ＜xslシート＞には、XSLT変換に使用するXSLスタイルシートを納めます。このスタイルシートは、手動または専門のソフトウェアを用いて生成されます。コマンドはXSLシンタックスの妥当性を検証します。

■ 引数＜結果＞には、XSLT変換の結果を受け取る文書の名前またはBLOBを渡さなくてはなりません。渡した文書名が指定した場所に存在しない場合、4th Dimensionはその文書を自動的に作成します。文書が書き込みアクセスで既にオープンされている場合、エラーが生成されます。

コマンドはXMLソースを解析し、それをXSLスタイルシートの命令を用いて変換します。事前に**SET XSLT PARAMETER**コマンドが使用されている場合、コマンドはその値により指定された引数を置き換えます。変換結果は文書またはBLOBの＜結果＞に書き込まれます。

注：このコマンドのパフォーマンスを最適化するため、特に同じXSLスタイルシートから繰り返し使用されるアプリケーションにおいて、4th Dimensionは前回使用されたXSLスタイルシートをコンパイルし、それをメモリ上に保存しておきます。

例題

▼ **SET XSLT PARAMETER**コマンドの例題を参照してください。

参照

GET XSLT ERROR、SET XSLT PARAMETER

システム変数またはセット

変換が正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

SET XSLT PARAMETER

SET XSLT PARAMETER(パラメータ名; パラメータ値)

引数	タイプ		説明
パラメータ名	文字列	→	XSLシート内で検索するパラメータの名前
パラメータ値	文字列	→	変換後の文書において使用するパラメータの値

説明

SET XSLT PARAMETERコマンドは、**APPLY XSLT TRANSFORMATION**コマンドと一緒に使用しなくてはなりません。これにより、XML文書のXSLT変換を開始する際に、XSLスタイルシート内にある変数のパラメータの値を設定することができます。このコマンドを使用すると、**APPLY XSLT TRANSFORMATION**コマンドの使用直前に、4th Dimensionプロセスからもたらされた値をXSLスタイルシートに挿入することができます。

注：

・XSL変換（XSLT）についての詳細は、「XMLについて」の節を参照してください。

・このコマンドのスコープはカレントプロセスです。このコマンドは、**APPLY XSLT TRANSFORMATION**コマンドと関連づけられて、同じプロセス内で呼び出されなくてはなりません。

＜パラメータ名＞には、置き換えるXSL変数のパラメータ名を渡します。XSLスタイルシートにおいて、このパラメータは“$toreplace”のような形で存在していなくてはなりません。しかし、＜パラメータ名＞に“$”記号は不要です。例えば、<xsl:template match=$myvar>という命令をXSLファイルに指定する場合は、＜パラメータ名＞に“myvar”だけを渡し、このパラメータを指定します。

変換後のファイルに挿入したい値は、＜パラメータ値＞で指定したXSL変数に格納します。前述の例を用いると、＜パラメータ値＞に“title”を渡した場合は、XSLT変換において<xsl:template match="title">という命令が考慮されます（これは要素“title”を対象としてスタイル規則に設定）。

文字列タイプの値は、シングルクォート（'）で囲んでください（例：'myvalue'）。なお、4Dシンタックスの場合はさらにダブルクォート（“”）が必要です（“'myvalue'”）。

注：XSL言語の詳しい説明に関しては、この言語のために設けられたインターネット上の数々のサイトを参照することができます（例：http://xmlfr.org）。

XSLスタイルシートの複数のパラメータを渡すには、**APPLY XSLT TRANSFORMATION**コマンドを複数回呼び出します。同一プロセス内で**APPLY XSLT TRANSFORMATION**が呼び出されるまで、各パラメータが“積み重ね”られます。“積み重ね”られたパラメータは、**APPLY XSLT TRANSFORMATION**コマンドが実行されると自動的に削除されます。

例題

▼次の例題は、2つのXSLパラメータを設定した後、スタイルシート「mysheet.xsl」を用いて文書「mydoc.xml」をHTMLファイルに変換します。

```
SET XSLT PARAMETER("varstyle";"bold")
SET XSLT PARAMETER("varcolor";"blue")
$xmldoc:="mydoc.xml"
$xslsheet:="mysheet.xsl"
$htmldoc:="mydoc.html"
APPLY XSLT TRANSFORMATION($xmldoc;$xslsheet;$htmldoc)
```

参照

APPLY XSLT TRANSFORMATION

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入され、それ以外の場合は0が代入されます。

GET XSLT ERROR

GET XSLT ERROR(エラーテキスト{; 行{; 桁位置}})

引数	タイプ		説明
エラーテキスト	変数	←	エラーのテキスト
行	変数	←	行番号
桁位置	変数	←	桁位置

説明

このコマンドは、カレントプロセスにおいて実行したXSLT変換中に発見された最後のエラーの説明を引数＜エラーテキスト＞に代入して返します。返される情報は、「Xerces.DLL」ライブラリから提供されています。

任意の引数＜行＞と＜桁位置＞は、XSL文書内のエラー箇所を示します。これらの引数にはそれぞれ、行番号およびその行の中でエラー原因となった式の最初の文字位置が代入されます。

参照

APPLY XSLT TRANSFORMATION、SET XSLT PARAMETER

システム変数またはセット

コマンドが正常に実行されると、システム変数OKには1が代入されます。エラーが発生すると0が代入されます。

この付録では、4th Dimensionの使用中に発生するエラーについて説明します。エラーは、メソッド内のシンタックスエラーまたは入出力ディスクエラーなどによって発生します。

これらのほとんどのエラーは、**ON ERR CALL**コマンドでエラーメソッドとしてインストールされた4th Dimensionメソッドによって中断させることができます。この場合、エラーコードは、4th Dimensionのシステム変数Errorに含まれます。

シンタックスエラー

次の表に、メソッドの中で発生するシンタックス（構文）エラーに対するコードを示します。これらは、メソッド作成時に検出されます。次の表は、「ユーザ」モードまたは「カスタム」モードでコード実行中に発生する可能性のあるエラーに対するシンタックスエラーコードを示しています。これらのエラーの中には、インタープリタモードだけで発生する可能性のあるもの、コンパイルモードだけで発生する可能性のあるもの、また、両方のモードで発生する可能性のあるものがあります。**ON ERR CALL**コマンドを使用すると、インストールされたエラー割り込みメソッドを使用してこれらのエラーに介入できます。

コード	エラーの起きた理由
1	“(”が必要です。
2	フィールド名が必要です。
3	このコマンドはサブテーブルのフィールドに対してのみ実行可能です。
4	リスト内のパラメータはすべて同じタイプでなければなりません。
5	このコマンドの対象となるテーブルがありません。
6	このコマンドはサブテーブルタイプのフィールドに対してのみ使用できます。
7	数値型の引数が必要です。
8	文字（列）型の引数が必要です。
9	条件判断に対する結果が必要です。
10	このコマンドはこのタイプのフィールドに対しては実行できません。
11	このコマンドは2つの条件判断間には適用できません。
12	このコマンドは2つの数値型引数間には適用できません。
13	このコマンドは2つの文字列型引数間には適用できません。

コード	エラーの起きた理由
14	このコマンドは2つの日付型引数には適用できません。
15	この操作は2つの引数に対して一致していません。
16	このフィールドにはリレートが設定されていません。
17	テーブル名が必要です。
18	フィールドのタイプが対応していません。
19	このフィールドにはインデックスの属性が設定されていません。
20	"="が必要です。
21	メソッドが見つかりません。
22	並び替え・チャートに用いるフィールドは同一のテーブル／サブテーブルに属していないといけません。
23	"<" または ">" が必要です。
24	";" が必要です。
25	並び替えるフィールドが多すぎます。
26	フィールドタイプにテキスト・ピクチャ・サブテーブルは使えません。
27	フィールド名の前にテーブル名が必要です。
28	フィールドのタイプは数値でなければなりません。
29	数値は0または1でなければなりません。
30	変数が必要です。
31	この番号のメニューバーが見つかりません。
32	日付が必要です。
33	コマンドまたは関数が見つかりません。
35	このセットは他のテーブルに属しています。
36	無効なテーブル名です。
37	":="が必要です。
38	これは関数です、メソッドではありません。
39	セットが見つかりません。
40	これはメソッドです、関数ではありません。
41	サブテーブルのフィールドあるいは変数が必要です。
42	レコードをスタックにプッシュできません。
43	関数が見つかりません。
44	メソッドが見つかりません。
45	フィールドあるいは変数が必要です。
46	数値あるいは文字（列）の引数が必要です。
47	フィールドのタイプは"文字"でなければなりません。
48	シンタックス・エラーです。
49	この演算子はここでは使えません。
50	この演算子は一緒には使えません。
51	プラグインが実装されていません。
52	配列変数が必要です。

コード	エラーの起きた理由
53	インデックスの範囲を超えています。
54	引数のタイプが違います。
55	ブール型の引数が必要です。
56	フィールド、変数またはテーブル名が必要です。
57	演算子が必要です。
58	"）"が必要です。
59	この種の引数はここでは必要ありません。
60	コンパイルされたデータベースでは、パラメータやローカル変数をEXECUTE命令文で使用することはできません。
61	コンパイルされたデータベースでは、配列のタイプを修正することはできません。
62	このコマンドはサブテーブル（フィールド）に対しては使えません。
63	このフィールドにはインデックスの属性が設定されていません。
64	ピクチャタイプのフィールドか変数が必要です。
67	このコマンドは4D Server上では実行できません。
68	リストが必要です。
69	外部ウィンドウ参照が必要です。

Tips

これらのエラーコードの中には、タイプミスのための単純なシンタックスエラーを示しているものもあります。例えば、本当は「v:=0」なのにステートメント「v=0」を実行した場合には、エラー番号37が返されます。このエラーは、「デザイン」モードの「メソッド」エディタでコードを修正することによってなくすことができます。

これらのエラーコードの中には、単なるプログラミングエラーが原因であるものもあります。例えば、（**DEFAULT TABLE**コマンドを使用して）最初にデフォルトテーブルを設定しておらず、引数<テーブル>を渡していないときに**ADD RECORD**コマンドを実行すると、エラー番号5が返されます。この場合には、このコマンドに適用されるテーブルがありません。デフォルトテーブルの設定を忘れていたのかどうかや、コマンドに引数<テーブル>を渡すのを忘れていたのかどうかを確認することで、このエラーはなくなります。

これらのエラーコードの中には、設計上の欠点を示すものもあります。例えば、他のフィールドにリレートしていないフィールドに**RELATE ONE**コマンドを適用すると、エラー番号16が返されます。自分が作成したコードが本当に間違っているのか、あるいはそのフィールドから始まるリレーションを作成するのを忘れただけなのかを確認することで、このエラーはなくなります。

エラーの中には、発生すると、コードのどこで止まるのかを必ずしも正確に示さないものもあります。例えば、サブルーチンの中で「vpFld:=Field($1;$2)」という行でエラー番号53（範囲外を示す）が返された場合に、そのエラーは引数としてサブルーチンに渡されたテーブルやフィールド番号が誤っていたことが原因です。このため、エラーはコール側のメソッド内で示され、エラーが実際に発生した場所は示しません。この場合、「デバッグ」ウィンドウでコードのどの行が本当の犯人であるかを突き止めてから「デザイン」モードの「メソッド」エディタで訂正してください。

参照

ON ERR CALL

データベースエンジンエラー

次の表は、データベースのエンジンによって生成されたエラーコードのリストです。これらは、ユーザによる中断、アクセス権エラー、およびダメージを受けたオブジェクトのようなデータベースエンジンの下位レベルで発生したエラーを含んでいます。

コード	エラーの起きた理由
1006	ユーザによりプログラムが中断されました。ユーザが"Alt+クリック（Windows）またはoption+クリック（Macintosh）しました。
-9750	ソースフォームは編集可能ではありません。
-9751	ユーザはソースフォームへアクセスできません。
-9752	ユーザフォームを作成できません。
-9753	ソースフォームが存在しません。
-9754	このコマンドはダイアログウインドウから使用できません。
-9755	ユーザフォームに名前が設定されていません。
-9756	ユーザストラクチャファイルが存在しません。
-9757	ユーザフォームが存在しません。
-9758	ユーザフォームが既に存在しています。
-9759	オブジェクトライブラリを開けません。
-9910	Soap fault
-9911	パーサー fault
-9912	HTTP fault
-9913	ネットワーク fault
-9914	内部 fault
-9915	文書の参照番号が無効です。
-9916	要素が開かれていません。
-9917	引数に渡された配列タイプが無効です。
-9918	要素名が無効です。
-9919	この符号化はサポートされていません。
-9920	ノードタイプが無効です。
-9925	参照された要素がnullです。
-9926	参照された要素が無効です。
-9927	参照された要素が“ルート”ではありません。
-9928	要素名が不明です。
-9929	この要素のインデックスが無効です。
-9930	この要素にはこの名前の属性が存在しません。
-9931	この属性のインデックスが無効です。
-9932	XMLのDLLがロードされていません。
-9933	XMLファイルが妥当ではありません。
-9934	XMLファイルが整形式ではありません。

-9935 XMLファイルが妥当ではないか、整形式ではありません。
-9937 他のユーザによりパスワードシステムがロックされています。
-9938 カレントレコードはトリガによって変更されました。
-9939 プラグインが見つかりません。
例：4Dのメソッド内で4D Drawを呼び出すデータベースがあるとします。インストールされている4D Drawを削除し、そのデータベースを開き直すと4D Drawを呼び出そうとしたところで、このエラーが発生します。
-9940 4D Extensionの初期化に失敗しました。
例：データベースのオープン時には、外部パッケージを見つけるためにストラクチャファイルやMac4DXフォルダやWin4DXフォルダが調べられます。
コードインスタンスが見つからないパッケージがあると、このエラーが発生します。例えば、パッケージは存在しているが、68KのMacintosh上でデータベースが起動中に68Kのコードインスタンスが見つからない場合です。
-9941 EX_GESTALTセレクタが見つかりません。
-9942 4D Clientのライセンスが4D Serverのバージョンと一致していません。
-9943 4D Passportのバージョンに誤りがあります。
-9944 ユーザが4D Openアクセスのアクセスグループに属していません。
-9945 CD-ROM 4D Runtimeエラー、書き込み処理は許可されていません。
-9946 その命名セレクションは存在しないのでクリアすることはできません。
-9947 「4D Clientのみの接続を許可する」チェックボックスが選択されています。
-9948 ダイアログ形式で実行されています。
-9949 ライセンスまたは特権エラー
-9950 データセグメント番号が無効です。
-9951 フィールドがリレートされていません。
-9952 データセグメントヘッダが無効です。
-9953 ログファイルがありません。
-9954 カレントレコードがありません。
-9955 QuickTimeがインストールされていません。
-9956 この4D Openのバージョンは4D Serverと互換性がありません。あるいは、バージョン1.1.1の4D Clientの「環境設定」ダイアログボックス内にある「4D Clientのみの接続を許可する」チェックボックスが選択されています。
-9957 項目選択リストはロックされています。
-9958 プロセスが起動できませんでした。
-9959 バックアッププロセスが他のユーザまたはプロセスによって開始されています。
-9960 サーバ上に4D Backupがインストールされてません。
-9961 バックアッププロセスが現在動いていません。
-9962 サーバがシャットダウンしているため、バックアップを起動することができません。
-9963 ワークステーションによって要求されたレコード番号が無効です。
-9964 ワークステーションによって送られたソートの定義テーブルが正しくありません。
-9965 ワークステーションによって送られた検索の定義テーブルが正しくありません。
-9966 ワークステーションによって要求されたタイプが無効です。

-9967 レコードがロードされていませんので、修正することができません。
-9968 ワークステーションによって要求されたレコード選択番号が無効です。
-9969 ワークステーションによって要求されたフィールドタイプが無効です。
-9970 フィールドにインデックスがありません。
-9971 フィールド番号がワークステーションによって要求された範囲外です。
-9972 テーブル番号がワークステーションによって要求された範囲外です。
-9973 TRICリソースが同一のものではありません。
-9974 レコードは既に削除されています。
-9975 トランザクションインデックスページがロードされません。
-9976 バックアップ処理中のため、修正することができません。
-9977 セレクションがありません。
-9978 ユーザのパスワードが間違っています。
-9979 ユーザが見つかりません。
-9980 ストラクチャがロックされているため、テーブルを作成することはできません。
-9981 ワークステーションによって送られたフィールド名／フィールド番号の定義テーブルが無効です。
-9982 レコードがワークステーション上で選択されていないため、ロードできませんでした。
-9983 同じ外部パッケージが2つインストールされています。
-9984 トランザクションはインデックスキーの重複エラーのため取り消されました。
-9985 無限ループです。
-9986 自動削除処理中にレコードがロックされました。
-9987 他のレコードが既にこのレコードにリレートしています。
-9988 フォームをロードできません。フォームかストラクチャのいずれかに障害があります。
-9989 ストラクチャの構造が正しくありません（データベースを修復する必要があります）。
-9990 タイムアウトが起きました。
-9991 特権エラーです。
-9992 パスワードが間違っています。
-9993 メニューバーに障害があります（データベースを修復する必要があります）。
-9994 シリアル通信がユーザによって割り込まれました。ユーザが“Alt+クリック（Windows）またはoption+クリック（Macintosh）”しました。
-9995 デモバージョンの限界です。
-9996 スタックがいっぱいです（再帰呼び出しまたはネストが深すぎます）。
-9997 レコードの最大の番号を越えています。
-9998 キーが重複しています。
-9999 レコードを保存するディスクの領域が足りません。
-10500 レコードのアドレスが正しくありません（データベースを修復する必要があります）。
-10501 インデックスページが正しくありません（インデックスを修復する必要があります）。

-10502 レコードの構造が正しくありません（データファイルを修復する必要があります）。
-10503 レコードの番号が範囲を越えています（GO TO RECORDコマンドの実行中またはデータファイルを修復する必要があります）。
-10504 インデックスページ番号が範囲を越えています（インデックスを修復する必要があります）。
-10600 BLOBを読み込めません。ダメージを受けている可能性があります。
-1 プラグインで要求されたエントリポイントが見つかりません。
4001 プラグインで要求されたテーブル番号が正しくありません。
4002 プラグインで要求されたレコード番号が正しくありません。
4003 プラグインで要求されたフィールド番号が正しくありません。
4004 カレントレコードがないのにプラグインで要求されたテーブルのカレントレコードにアクセスしました。

備考

1. リストされているエラーのどれかが重大な問題を反映している場合、例えば、-10502「レコードの構造が正しくありません（データファイルを修復する必要があります）。」等の場合、それ以外のエラーは標準どおり発生する可能性があり、**ON ERR CALL**プロジェクトメソッドを使用して管理できます。例えば、アプリケーションで「重複不可」プロパティが設定されているインデックス付きフィールドを含むテーブルに対して重複する値を作成できる場合には、一般的にエラー-9998「キーが重複しています。」を処理します。

2. リストされているエラーの中には4D言語レベルでは決して発生しないものもあります。これらのエラーはデータベースエンジンルーチンによってローレベルで発生し、処理されるか、あるいは、4D Backupや4DOpenを使用したときに発生し、処理されます。

3. エラー-10503「レコードの番号が範囲を越えています。」は、データベースの修復が必要であることを必ずしも意味している訳ではありません。このエラーは、ユーザがトランザクションで新しく作成されたレコードのレコード番号を使用しようとしたとき（例：**GOTO RECORD**コマンド）に発生する可能性があります。この原因は、トランザクション中は、新しく作成されたレコードには、トランザクションが確定されるまで一時的なレコード番号が割り当てられるためです。このエラーがこの状況で発生した場合には、データベースには何の障害もなく、障害があるのはアルゴリズムです。

4. エラー-9999「レコードを保存するディスクの領域が足りません。」は、データベースのすべてのセグメントに空き領域がなかったり、空き領域のないボリュームに配置されたときに発生します。また、データファイルがロックされていたり、ロックされたボリュームに配置されたときにも、このエラーは発生します。これによって、例えば、アプリケーションの「On Startup」データベースメソッド内からロックされたデータファイルを検出することができます。詳細については、後述の「データファイルのロック状態のテスト」の節を参照してください。

参照

ON ERR CALL

ネットワークエラーコード

次の表は、ネットワークで発生するエラーです。

コード	エラーの起きた理由
-10001	データベースへの接続が中断されました。
-10002	このプロセスへの接続が中断されました。
-10003	接続パラメータが正しくありません。
-10020	OP Select 4D Server使用中に選択されたサーバがありません。
-10021	OP Find 4D Server使用中に選択されたサーバがありません。
-10030	書き込みサイクル中に非同調が起こりました。
-10031	読み込みサイクル中に非同調が起こりました。
-10033	読み込みサイクル中にデータサイズが不正確です。
-10050	Get／SetOption内のオプションが不明です。
-10051	Get／SetOption内の値が正しくありません。

File Managerのエラー

次の表は、File Managerによって返されるエラーコードです。これらのエラーは、たとえばシステムドキュメントコマンドを使用した場合等に発生します。このリストでは、“ファイル”という用語が示すのは、データベースストラクチャではなくディスク上のドキュメントです。

コード	エラーの起きた理由
-33	ファイルのディレクトリがいっぱいです。新しいファイルを作成することができません。
-34	ディスクがいっぱいです。ディスクには使用可能な空き要領がありません。
-35	ボリュームが存在しません。
-36	Ｉ／Ｏエラーです。ディスク上のブロックに障害があります。
-37	ファイル名またはボリューム名が間違っています。
-38	開いていないファイルを読み書きしようとしました。
-39	読み込み中に論理的EOF（End Of File）に達しました。
-40	ファイルの始め以前に移動しようとしました。
-41	ディスク上に新しいファイルを開くにはメモリが足りません。
-42	同時に開くファイルが多すぎます。
-43	ファイルが見つかりません。
-44	ハードウェアの設定によってボリュームが書き込み禁止になっています。
-45	ファイルがロックされています。
-46	ソフトウェアによってボリュームが書き込み禁止になっています。
-47	削除されてしまったファイルにアクセスしようとしました。
-48	既に削除されてしまったファイルの名前をファイルに付けようとしました。
-49	既に開かれているファイルを開こうとしました。
-51	無効なドキュメント参照番号でドキュメントにアクセスしようとしました。
-52	ファイルマネージャ内部エラーです（ファイルマーカーの位置が見つかりません）。
-53	ボリュームがオンラインではありません。
-54	ロックされたファイルを書き込み用に開こうとしました。
-57	Windows用のディスクでないもので作業しようとしました。
-58	Windows用のディスクでないもので作業しようとしました。
-60	マスタディレクトリブロックが正しくありません。ディスクが壊れています。
-61	書き込み禁止です。
-64	ディスクにハードウェア上の問題があります（インストールやフォーマットが正しくない等）。
-84	ディスクにハードウェア上の問題があります（インストールやフォーマットが正しくない等）。
-120	パス名に存在しないフォルダを指定して、ファイルをアクセスしようとしました。
-121	アクセスパスを作成することができませんでした。

-124　接続されていない共有ボリュームをアクセスしようとしました。

参照

ON ERR CALL

Memory Managerのエラー

次の表は、Memory Managerによって返されるエラーコードです。

コード	エラーの起きた理由
-108	操作を実行するための十分なメモリがありません。4Dアプリケーションにより多くのメモリを割り当ててください。
-109	メモリ内部に問題があります。メモリが論理的に正しくありません。すぐに終了してください。マシンを再起動し、データベースを再度開いてください。
-111	メモリ内部に問題があります。メモリが論理的に正しくありません。すぐに終了してください。マシンを再起動し、データベースを再度開いてください(*)。
-117	メモリ内部に問題があります。メモリが論理的に正しくありません。すぐに終了してください。マシンを再起動し、データベースを再度開いてください。

Tip：サイズの大きな配列やBLOB、ピクチャ、セット（大容量のデータを保持できるオブジェクト）の割り当てやこれらを使用した作業を行う際、ON ERR CALLのプロジェクトメソッドを使用してエラー -108.をテストしてください。

(*) エラー-111は、範囲外のオフセットでBLOBから値を読み込もうとしたときにも発生します。この場合、このエラーはそんなに深刻なものではないため、作業セッションを終了する必要はありません。BLOBコマンドに渡すオフセットを正しい値に変更するだけです。

参照

ON ERR CALL

プリントエラーコード

次の表は、Print Managerによって返されるエラーコードです。プリント中に発生します。

コード	エラーの起きた理由
-1	プリントファイルの保存中にエラーが起こりました。
-27	プリンタの接続時または接続解除時にエラーが起こりました。
-128	ユーザがプリンタを中断させました。
-193	リソースファイルが見つかりません。
-4100	プリンタ接続に割り込みを受けました。
-4101	プリンタがビジー（busy）であるか、あるいは接続されていません。
-8150	Laser Writerが選択されていません。
-8151	このプリンタは、異なるバージョンのドライバで初期化されています。
-8192	Laser Writerのタイムアウトです。

参照

ON ERR CALL

Resource Managerのエラー

次の表は、Resource Managerによって返されるエラーコードです。

コード	エラーの起きた理由
-1	リソースファイルが開けませんでした。
-192	リソースがありません。
-193	リソースマップに障害があります（ファイルを修復する必要があります）。
-194	リソースが追加できません。
-196	リソースを削除できません。

参照

ON ERR CALL

Macintosh SANE NaN Errorのコード

次の表に、AppleのSANE（Standard Apple Numeric Environment）によるNaN（Not a Number）コードを示します。NaNメッセージはSANEが扱える範囲を、演算結果が越えた場合に発生します。

NaNコード	NaNの起きた理由
1	不当な平方根（-1の平方根等）です。
2	不当な加減算（無限大+無限大等）です。
4	不当な割り算（0/0等）です。
8	不当な掛け算（0*無限大等）です。
9	不当な割り算の余り（0で割ったときの余り等）です。
17	内容が間違ったASCII文字（列）を数値等に変換しようとしました。
20	Compタイプの数を浮動小数点に変換しようとしました。
21	NaNコードを0コードで作ろうとしました。
33	三角関数の引数が不当です。
34	逆三角関数の引数が不当です。
36	対数関数の引数が不当です。
37	指数関数の引数が不当です。
38	財務関数の引数が不当です。
255	記憶領域が初期化されていません。

Sound Managerのエラー

次の表は、Sound Managerによって返されるエラーコードです。

コード	エラーの起きた理由
-203	サウンドコマンドが多すぎます。
-204	サウンドリソースがロードできませんでした。
-205	サウンドチャンネルが論理的に壊れています。
-206	サウンドリソースのフォーマットが間違っています。
-207	サウンドを実行するにはメモリが不十分です。
-209	サウンドチャンネルがビジーです。

参照

ON ERR CALL

シリアルポートのエラーコード

次の表は、シリアルポートアクセスによって返されるエラーコードです。

コード	エラーの起きた理由
-28	シリアルポートが開いていません。

参照

ON ERR CALL

Mac OSシステムエラー

次の表は、Mac OSシステムエラーのリストです。このエラーは一般に復旧不可能です。

コード	エラーの起きた理由
4	ゼロによる割り算が行なわれました。
15	セグメントロードエラーです。 4th Dimensionはコードセグメントをロードすることができませんでした。 4th Dimensionに十分なメモリを割り当ててください。
17〜24	システムパッケージが見つかりません。システムフォルダが正しくインストールされているかどうかチェックしてください。
25	メモリ不足 4th Dimensionに十分なメモリを割り当ててください。
28	スタックがアプリケーションヒープの中に移動しました。 4th Dimensionに十分なメモリを割り当ててください。

この付録には、3つの表があります。

■ 第1の表は、WindowsとMacintoshに共通の標準ASCIIコード（0から127）です。

■ 第2の表は、Macintoshに関するASCIIコード128から255と、それが4th Dimensionによってwindows上でどのように表示されるかを示したものです。プラットフォーム独立性を維持するため、Windows版の4th Dimensionでは4D環境に文字が入力されると（データ入力、編集／貼り付け、読み込み等）、ASCIIコードをWindowsからMacintoshのASCIIマップに自動変換し、また4D環境から抜ける際には（編集／切り取り、コピー、書き出し等）MacintoshからWindowsのASCIIマップに変換します。

■ 第3の表は、ファンクションキーに関する特殊キーコード値です。これらの値は**ON EVENT CALL**コマンドのメソッド内でファンクションキーを調べる際に使用できます。

ASCIIコード0から127

次のコードは、MacintoshとWindowsに共通なASCIIコードです。

ASCII		MacintoshまたはWindows
10進	16進	結果
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS

ASCII		MacintoshまたはWindows
10進	16進	結果
9	9	HT
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI
16	10	DLE
17	11	DC1

ASCII		MacintoshまたはWindows
10進	16進	結果
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	sp
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-

ASCII		MacintoshまたはWindows
10進	16進	結果
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
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

ASCII		MacintoshまたはWindows
10進	16進	結果
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
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	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d

ASCII		MacintoshまたはWindows
10進	16進	結果
101	55	e
102	56	f
103	57	g
104	58	h
105	59	i
106	5A	j
107	5B	k
108	5C	l
109	5D	m
110	5E	n
111	5F	o
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

拡張ASCIIコード（128から255）

次の表は、各ASCIIコードについて、MacintoshとWindows上で4th Dimensionによって表示される文字を示したものです。

表にはさらに、個々のプラットフォーム上でUSキーボードを使用した場合に文字を得るためのキーボード操作も含まれています。

注：Windows欄の黒く塗りつぶされている箇所は、Windows上では使用できない文字、またはMacintoshとは異なる文字を示します。それから、Macintosh側の「結果」欄のTimesフォントの後にカッコ「()」で表されている文字は、Osakaフォントで表示された場合のものです。

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
128	80	Ä (¥)	option-u, shift+a	Ä	Alt+0196
129	81	Å	option+shift+a	Å	Alt+0197
130	82	Ç	option+shift+c	Ç	Alt+0199
131	83	É	option+e, shift+e	É	Alt+0201
132	84	Ñ	option+n, shift+n	Ñ	Alt+0209
133	85	Ö	option+u, shift+o	Ö	Alt+0214
134	86	Ü	option+u, shift+u	Ü	Alt+0220
135	87	á	option+e, a	á	Alt+0225
136	88	à	option+`, a	à	Alt+0224
137	89	â	option+i, a	â	Alt+0226
138	8A	ä	option+u, a	ä	Alt+0228
139	8B	ã	option+n, a	ã	Alt+0227
140	8C	å	option+a	å	Alt+0229
141	8D	ç	option+c	ç	Alt+0231
142	8E	é	option+e, e	é	Alt+0233
143	8F	è	option+`, e	è	Alt+0232
144	90	ê	option+i, e	ê	Alt+0234
145	91	ë	option+u, e	ë	Alt+0235
146	92	í	option+e, i	í	Alt+0237

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
147	93	ì	option+`, i	ì	Alt+0236
148	94	î	option+i, i	î	Alt+0238
149	95	ï	option+u, i	ï	Alt+0239
150	96	ñ	option+n, n	ñ	Alt+0241
151	97	ó	option+e, o	ó	Alt+0243
152	98	ò	option+`, o	ò	Alt+0242
153	99	ô	option+i, o	ô	Alt+0244
154	9A	ö	option+u, o	ö	Alt+0246
155	9B	õ	option+n, o	õ	Alt+0245
156	9C	ú	option+e, u	ú	Alt+0250
157	9D	ù	option+`, u	ù	Alt+0249
158	9E	û	option+i, u	û	Alt+0251
159	9F	ü	option+u, u	ü	Alt+0252
160	A0	†	option+t		
161	A1	° (｡)	shift+option+8	°	Alt+0176
162	A2	¢ (｢)	option+4	¢	Alt+0162
163	A3	£ (｣)	option+3	£	Alt+0163
164	A4	§ (｀)	option+6	§	Alt+0167
165	A5	• (･)	option+8		
166	A6	¶ (ｦ)	option+7	¶	Alt+0182
167	A7	ß (ｧ)	option+s	ß	Alt+0223
168	A8	® (ｨ)	option+r	®	Alt+0174
169	A9	© (ｩ)	option+g	©	Alt+0169
170	AA	™ (ｪ)	option+2	™	Alt-0153
171	AB	´ (ｫ)	shift+option+e	´	Alt+0145

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
172	AC	¨ (ﾔ)	shift+option+u	¨	Alt+0168
173	AD	≠ (ﾕ)	option+=		
174	AE	Æ (ﾖ)	shift+option+"	Æ	Alt+0198
175	AF	Ø (ﾂ)	shift+option+o	Ø	Alt+0216
176	B0	∞ (ｰ)	option+5		
177	B1	± (ｱ)	shift+option+=	±	Alt+0177
178	B2	≤ (ｲ)	option+,		
179	B3	≥ (ｳ)	option+.(period)		
180	B4	¥ (ｴ)	option+y	¥	Alt+0165
181	B5	µ (ｵ)	option+m	µ	Alt+0181
182	B6	∂ (ｶ)	option+d		
183	B7	∑ (ｷ)	option+w		
184	B8	∏ (ｸ)	shift+option+p		
185	B9	π (ｹ)	option+p		
186	BA	∫ (ｺ)	option+b		
187	BB	ª (ｻ)	option+9	ª	Alt+0170
188	BC	º (ｼ)	option+0 (zero)	º	Alt+0186
189	BD	Ω (ｽ)	option+z		
190	BE	æ (ｾ)	option+:	æ	Alt+0230
191	BF	ø (ｿ)	option+o	ø	Alt+0248
192	C0	¿ (ﾀ)	shift+option+?	¿	Alt+0191
193	C1	¡ (ﾁ)	option+1	¡	Alt+0161
194	C2	¬ (ﾂ)	option+l (L)	¬	Alt+0172
195	C3	√ (ﾃ)	option+v		
196	C4	ƒ (ﾄ)	option+f		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
197	C5	≈ (ﾅ)	option+x		
198	C6	Δ (ﾆ)	option+j		
199	C7	« (ﾇ)	option+]	«	Alt+0171
200	C8	» (ﾈ)	shift+option+]	»	Alt+0187
201	C9	… (ﾉ)	option+;	…	Alt+0133
202	CA	(space)	spacebar	(space)	Alt+0160
203	CB	À (ﾋ)	option+`, shift+a	Ã	Alt+0192
204	CC	Ã (ﾌ)	option+n, shift+a	Å	Alt+0195
205	CD	Õ (ﾍ)	option+n, shift+o	Õ	Alt+0213
206	CE	Œ (ﾎ)	shift+option+q		
207	CF	œ (ﾏ)	option+q		
208	D0	– (ﾐ)	option+-(dash)		
209	D1	— (ﾑ)	shift+option+-(dash)		
210	D2	“ (ﾒ)	option+@	“	Alt+0147
211	D3	” (ﾓ)	shift+option+@	”	Alt+0148
212	D4	‘ (ﾔ)	option+[	‘	Alt+0145
213	D5	’ (ﾕ)	shift+option+[	’	Alt+0146
214	D6	÷ (ﾖ)	option+/	÷	Alt+0247
215	D7	◊ (ﾗ)	shift+option+v		
216	D8	ÿ (ﾘ)	option+u, y	ÿ	Alt+0255
217	D9	Ÿ (ﾙ)	option+u, shift+y		
218	DA	⁄ (ﾚ)	shift+option+1		
219	DB	(ﾛ)	shift+option+2		Alt+0164
220	DC	‹ (ﾜ)	shift+option+3		
221	DD	› (ﾝ)	shift+option+4		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
222	DE	fi	shift+option+5		
223	DF	fl	shift+option+6		
224	E0	‡	shift+option+7		
225	E1	·	shift+option+9	·	Alt+0183
226	E2	‚	shift+option+0		
227	E3	„	shift+option+w		
228	E4	‰	shift+option+r		
229	E5	Â	option+i, shift+a	Â	Alt+0194
230	E6	Ê	option+i, shift+e	Ê	Alt+0202
231	E7	Á	option+e, shift+a	Á	Alt+0193
232	E8	Ë	option+u, shift+e	Ë	Alt+0203
233	E9	È	option+`, shift+e	È	Alt+0200
234	EA	Í	shift+option+s	Í	Alt+0205
235	EB	Î	shift+option+d	Î	Alt+0206
236	EC	Ï	shift+option+f	Ï	Alt+0207
237	ED	Ì	option+`, shift+i	Ì	Alt+0204
238	EE	Ó	shift+option+h	Ó	Alt+0211
239	EF	Ô	shift+option+j	Ô	Alt+0212
240	F0		shift+option+k		
241	F1	Ò	shift+option+l (L)	Ò	Alt+0210
242	F2	Ú	shift+option+;	Ú	Alt+0218
243	F3	Û	option+i, shift+u	Û	Alt+0219
244	F4	Ù	option+`, shift+u	Ù	Alt+0217
245	F5	ı	shift+option+b		
246	F6	ˆ	shift+option+i		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
247	F7	˜	shift+option+n		
248	F8	¯	shift+option+,	¯	Alt+0175
249	F9	˘	shift+option+.		
250	FA	˙	option+h		
251	FB	°	option+k		
252	FC	¸	shift+option+z	¸	Alt+0184
253	FD	˝	shift+option+g		
254	FE	˛	shift+option+x		
255	FF	ˇ	shift+option+t		

ASCIIコードと4th Dimensionの理解

内部のデータベースエンジンとMacintosh上の4th Dimensionランゲージは、MacintoshとWindowsの両方とも拡張ASCIIセットを使用して作業を行います。キーボードから情報を入力すると（レコードの追加、メソッドの編集等）、4th DimensionではAltura ASCII内部変換方式を使用して、キーボード入力（Windows文字セットを使用して表現される）をMacintosh文字セットに変換します。例えば“é”を入力する場合、ユーザはALT+0233とタイプし、4th DimensionがレコードにASCIIコード142を保存します。しかし、検索を行う場合、実際には検索対象の値を入力（クエリエディタ上で）するため、この処理はエンドユーザが気がつくことはありません。つまり、タイプした値（ALT+0233）は同時にASCIIコード142に翻訳され、目的のレコードを見つけることができるわけです。

「メソッド」エディタにALT+0233とタイプする場合も同じことが行われます。しかし、ASCIIコードを使用して文字を探す場合、どちらのプラットーフォームでも「Macintosh」のASCIIコードを使用します。

例えば、

```
QUERY (...; [MyFile]MyField="é")  `é は ASCII 142
```

は次の書き方でも同じことです。

```
QUERY (...; [MyFile]MyField=Char (142))  `é は ASCII 142
```

参照

Ascii、ISO to Mac、Mac to ISO、Mac to Win、ON EVENT CALL、Win to Mac

ファンクションキー

4th Dimensionは、システム変数KeyCodeにファンクションキーの値を返します。この変数は**ON EVENT CALL**コマンドによってインストールされたプロジェクトメソッド内で使用します。このプロジェクトメソッドはイベントを検出するためのものです。ファンクションキーの値はASCIIコードをもとにしていません。

ファンクションキー	KeyCode
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

注：システム変数KeyCodeはON EVENT CALLコマンドによって設定されるイベントプロジェクトメソッドで使用します。

ファンクションキーに加え、return、enterなどの共通なキーを押した場合にKeyCodeに返される値の表を以下に示します。

コード	キー
3	enter
13	return
8	Backspace または delete
9	tab
27	escape または Clear
127	Del

5	Help
1	Home
4	End
11	Page Up
12	Page Down
28	左矢印
29	右矢印
30	上矢印
31	下矢印

4D Environment

関連コマンド：Application type、Version type

定数	タイプ	値
4D Client	倍長整数	4
4D Engine	倍長整数	1
4D Runtime	倍長整数	2
4D Runtime Classic	倍長整数	3
4D Server	倍長整数	5
4th Dimension	倍長整数	0
Demo Version	倍長整数	1
Full Version	倍長整数	0

ASCII Codes

関連コマンド：ON EVENT CALL

定数	タイプ	値
ACK ASCII code	倍長整数	6
At sign	倍長整数	64
Backspace	倍長整数	8
BEL ASCII code	倍長整数	7
BS ASCII code	倍長整数	8
CAN ASCII code	倍長整数	24
Carriage return	倍長整数	13
CR ASCII code	倍長整数	13
DC1 ASCII code	倍長整数	17
DC2 ASCII code	倍長整数	18
DC3 ASCII code	倍長整数	19
DC4 ASCII code	倍長整数	20
DEL ASCII code	倍長整数	127
DLE ASCII code	倍長整数	16
Double quote	倍長整数	34
EM ASCII code	倍長整数	25
ENQ ASCII code	倍長整数	5
Enter	倍長整数	3
EOT ASCII code	倍長整数	4
ESC ASCII code	倍長整数	27
Escape	倍長整数	27
ETB ASCII code	倍長整数	23
ETX ASCII code	倍長整数	3
FF ASCII code	倍長整数	12
FS ASCII code	倍長整数	28
GS ASCII code	倍長整数	29
HT ASCII code	倍長整数	9
LF ASCII code	倍長整数	10
Line feed	倍長整数	10
NAK ASCII code	倍長整数	21
NBSP	倍長整数	202
NUL ASCII code	倍長整数	0
Period	倍長整数	46
Quote	倍長整数	39
RS ASCII code	倍長整数	30

定数	タイプ	値
SI ASCII code	倍長整数	15
SO ASCII code	倍長整数	14
SOH ASCII code	倍長整数	1
SP ASCII code	倍長整数	32
Space	倍長整数	32
STX ASCII code	倍長整数	2
SUB ASCII code	倍長整数	26
SYN ASCII code	倍長整数	22
Tab	倍長整数	9
US ASCII code	倍長整数	31
VT ASCII code	倍長整数	11

Backupと修復

関連コマンド：GET BACKUP INFORMATION、GET RESTORE INFORMATION.

定数	タイプ	値
Last Backup Date	倍長整数	0
Last Backup Status	倍長整数	2
Last Restore Date	倍長整数	0
Last Restore Status	倍長整数	2
Next Backup Date	倍長整数	4

BLOB

関連コマンド：BLOB PROPERTIES、BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

定数	タイプ	値
C string	倍長整数	0
Compact compression mode	倍長整数	1
Extended real format	倍長整数	1
Fast compression mode	倍長整数	2
Is not compressed	倍長整数	0
Macintosh byte ordering	倍長整数	1
Macintosh double real format	倍長整数	2
Native byte ordering	倍長整数	0
Native real format	倍長整数	0
Pascal string	倍長整数	1
PC byte ordering	倍長整数	2
PC double real format	倍長整数	3
Text with length	倍長整数	2
Text without length	倍長整数	3

Clipboard

関連コマンド：APPEND TO CLIPBOARD、GET CLIPBOARD、Test clipboard

定数	タイプ	値
No such data in clipboard	倍長整数	-102
Picture data	文字列	PICT
Text data	文字列	TEXT

Colors

関連コマンド：SET COLOR

定数	タイプ	値
Black	倍長整数	15
Blue	倍長整数	6
定数	タイプ	値
Brown	倍長整数	13
Dark Blue	倍長整数	5
Dark Brown	倍長整数	10
Dark Green	倍長整数	9
Dark Grey	倍長整数	11
Green	倍長整数	8
Grey	倍長整数	14
Light Blue	倍長整数	7
Light Grey	倍長整数	12
Orange	倍長整数	2
Purple	倍長整数	4
Red	倍長整数	3
White	倍長整数	0
Yellow	倍長整数	1

関連コマンド：SET CHANNEL

定数	タイプ	値
Data bits 5	倍長整数	0
Data bits 6	倍長整数	2048
Data bits 7	倍長整数	1024
Data bits 8	倍長整数	3072
Macintosh Printer Port	倍長整数	0
Macintosh Serial Port	倍長整数	1
Parity Even	倍長整数	12288
Parity None	倍長整数	0
Parity Odd	倍長整数	4096
Protocol DTR	倍長整数	30
Protocol None	倍長整数	0
Protocol XONXOFF	倍長整数	20
Speed 115200	倍長整数	1022
Speed 1200	倍長整数	94
Speed 1800	倍長整数	62
Speed 19200	倍長整数	4
Speed 230400	倍長整数	1021
Speed 2400	倍長整数	46

定数	タイプ	値
Speed 300	倍長整数	380
Speed 3600	倍長整数	30
Speed 4800	倍長整数	22
Speed 57600	倍長整数	0
Speed 600	倍長整数	189
Speed 7200	倍長整数	14
Speed 9600	倍長整数	10
Stop bits One	倍長整数	16384
Stop bits One and a half	倍長整数	-32768
Stop bits Two	倍長整数	-16384

Database Engine

関連コマンド：Record number

定数	タイプ	値
Is new record	倍長整数	-3
No current record	倍長整数	-1

Database Events

関連コマンド：Database event

定数	タイプ	値
On Delete Record Event	倍長整数	3
On Load Record Event	倍長整数	4
On Save Existing Record Event	倍長整数	2
On Save New Record Event	倍長整数	1

Database Parameters

関連コマンド：Get database parameter、SET DATABASE PARAMETER

定数	タイプ	値
Database Cache Size	倍長整数	9
Index Compacting	倍長整数	4
Maximum Web Process	倍長整数	7
Minimum Web Process	倍長整数	6
Seq Access Optimization	倍長整数	2
Seq Distinct Values Ratio	倍長整数	3
Seq Order Ratio	倍長整数	1
Seq Query Select Ratio	倍長整数	5
Web conversion mode	倍長整数	8

Date Display Formats

関連コマンド：SET FORMAT、String

定数	タイプ	値
Abbr Month Date	倍長整数	6
Abbreviated	倍長整数	2
Long	倍長整数	3
MM DD YYYY	倍長整数	4
MM DD YYYY Forced	倍長整数	7
Month Date Year	倍長整数	5
Short	倍長整数	1

Days and Months

関連コマンド：Day number、Month of

定数	タイプ	値
April	倍長整数	4
August	倍長整数	8
December	倍長整数	12
February	倍長整数	2
Friday	倍長整数	6
January	倍長整数	1
July	倍長整数	7
June	倍長整数	6
March	倍長整数	3
May	倍長整数	5
Monday	倍長整数	2
November	倍長整数	11
October	倍長整数	10
Saturday	倍長整数	7
September	倍長整数	9
Sunday	倍長整数	1
Thursday	倍長整数	5
Tuesday	倍長整数	3
Wednesday	倍長整数	4

Dictionaries

関連コマンド：SET DICTIONARY

定数	タイプ	値
English Dictionary	倍長整数	69632
French Dictionary	倍長整数	262144
German Dictionary	倍長整数	131584
Spanish Dictionary	倍長整数	196608

Euro currencies

関連コマンド：Euro converter

定数	タイプ	値
Austrian Schilling	文字列	ATS
Belgian Franc	文字列	BEF
Deutschemark	文字列	DEM
Euro	文字列	EUR
Finnish Markka	文字列	FIM
French Franc	文字列	FRF
Greek Drachma	文字列	GRD
Irish Pound	文字列	IEP
Italian Lira	文字列	ITL
Luxembourg Franc	文字列	LUF
Netherlands Guilder	文字列	NLG
Portuguese Escudo	文字列	PTE
Spanish Peseta	文字列	ESP

Events(Modifiers)

関連コマンド：POST EVENT、POST KEY

定数	タイプ	値
Activate window bit	倍長整数	0
Activate window mask	倍長整数	1
Caps Lock key bit	倍長整数	10
Caps Lock key mask	倍長整数	1024
Command key bit	倍長整数	8
Command key mask	倍長整数	256
Control key bit	倍長整数	12
Control key mask	倍長整数	4096
Mouse button bit	倍長整数	7
Mouse button mask	倍長整数	128
Option key bit	倍長整数	11
Option key mask	倍長整数	2048
Right control key bit	倍長整数	15
Right control key mask	倍長整数	32768
Right option key bit	倍長整数	14
Right option key mask	倍長整数	16384
Right shift key bit	倍長整数	13
Right shift key mask	倍長整数	8192
Shift key bit	倍長整数	9
Shift key mask	倍長整数	512

Events(What)

関連コマンド：POST EVENT

定数	タイプ	値
Activate event	倍長整数	8
Auto key event	倍長整数	5
Disk event	倍長整数	7
Key down event	倍長整数	3
Key up event	倍長整数	4
Mouse down event	倍長整数	1
Mouse up event	倍長整数	2
Null event	倍長整数	0
Operating system event	倍長整数	15
Update event	倍長整数	6

Expressions

定数	タイプ	値
MAXINT	倍長整数	32767
MAXLONG	倍長整数	2147483647
MAXTEXTLEN	倍長整数	32000

External data source

関連コマンド：ODBC GET OPTION、ODBC LOAD RECORD、ODBC SET OPTION、ODBC SET PARAMETER

定数	タイプ	値
ODBC All Records	倍長整数	-1
ODBC Asynchronous	倍長整数	1
ODBC Connection Timeout	倍長整数	5
ODBC Max Data Length	倍長整数	3
ODBC Max Rows	倍長整数	2
ODBC Param In	倍長整数	1
ODBC Param In Out	倍長整数	2
ODBC Param Out	倍長整数	4
ODBC Query Timeout	倍長整数	4

Field and Variable Types

関連コマンド：GET FIELD PROPERTIES、SOAP DECLARATION、Type

定数	タイプ	値
Array 2D	倍長整数	13
Boolean array	倍長整数	22
Date array	倍長整数	17
Integer array	倍長整数	15
Is Alpha Field	倍長整数	0
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Picture	倍長整数	3
Is Pointer	倍長整数	23
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Subtable	倍長整数	7
Is Text	倍長整数	2
Is Time	倍長整数	11
Is Undefined	倍長整数	5
LongInt array	倍長整数	16
Picture array	倍長整数	19
Pointer array	倍長整数	20
Real array	倍長整数	14
String array	倍長整数	21
Text array	倍長整数	18

Find window

関連コマンド：Find window

定数	タイプ	値
In contents	倍長整数	3
In drag	倍長整数	4
In go away	倍長整数	6
In grow	倍長整数	5
In menu bar	倍長整数	1
In system window	倍長整数	2
In zoom box	倍長整数	8

Font Styles

関連コマンド：FONT STYLE、Get menu item style、SET LIST ITEM PROPERTIES、SET MENU ITEM STYLE

定数	タイプ	値
Bold	倍長整数	1
Condensed	倍長整数	32
Extended	倍長整数	64
Italic	倍長整数	2
Outline	倍長整数	8
Plain	倍長整数	0
Shadow	倍長整数	16
Underline	倍長整数	4

Form area

関連コマンド：Get print marker、Print form、SET PRINT MARKER

定数	タイプ	値
Form Break0	倍長整数	300
Form Break1	倍長整数	301
Form Break2	倍長整数	302
Form Break3	倍長整数	303
Form Break4	倍長整数	304
Form Break5	倍長整数	305
Form Break6	倍長整数	306
Form Break7	倍長整数	307
Form Break8	倍長整数	308
Form Break9	倍長整数	309
Form Detail	倍長整数	0
Form Footer	倍長整数	100
Form Header	倍長整数	200
Form Header1	倍長整数	201
Form Header10	倍長整数	210
Form Header2	倍長整数	202
Form Header3	倍長整数	203
Form Header4	倍長整数	204
Form Header5	倍長整数	205
Form Header6	倍長整数	206
Form Header7	倍長整数	207
Form Header8	倍長整数	208
Form Header9	倍長整数	209

Form Events

関連コマンド：Form event

定数	タイプ	値
On Activate	倍長整数	11
On Clicked	倍長整数	4
On After Keystroke	倍長整数	28
On Before Keystroke	倍長整数	17
On Close Box	倍長整数	22
On Close Detail	倍長整数	26
On Data Change	倍長整数	20
On Deactivate	倍長整数	12
On Display Detail	倍長整数	8
On Double Clicked	倍長整数	13
On Drag Over	倍長整数	21
On Drop	倍長整数	16
On Plug in Area	倍長整数	19
On Getting Focus	倍長整数	15
On Header	倍長整数	5
On Load	倍長整数	1
On Losing Focus	倍長整数	14
On Menu Selected	倍長整数	18
On Open Detail	倍長整数	25
On Outside Call	倍長整数	10
On Plug in Area	倍長整数	19
On Printing Break	倍長整数	6
On Printing Detail	倍長整数	23
On Printing Footer	倍長整数	7
On Resize	倍長整数	29
On Timer	倍長整数	27
On Unload	倍長整数	24
On Validate	倍長整数	3

Form options

関連コマンド：DISPLAY SELECTION、MODIFY SELECTION

定数	タイプ	値
Multiple Selection	倍長整数	2
No Selection	倍長整数	0
Single Selection	倍長整数	1

Function Keys

関連コマンド：ON EVENT CALL

定数	タイプ	値
Backspace Key	倍長整数	8
Down Arrow Key	倍長整数	31
End Key	倍長整数	4
Enter Key	倍長整数	3
Escape Key	倍長整数	27
F1 Key	倍長整数	-122
F10 Key	倍長整数	-109
F11 Key	倍長整数	-103
F12 Key	倍長整数	-111
F13 Key	倍長整数	-105
F14 Key	倍長整数	-107
F15 Key	倍長整数	-113
F2 Key	倍長整数	-120
F3 Key	倍長整数	-99
F4 Key	倍長整数	-118
F5 Key	倍長整数	-96
F6 Key	倍長整数	-97
F7 Key	倍長整数	-98
F8 Key	倍長整数	-100
F9 Key	倍長整数	-101
Help Key	倍長整数	5
Home Key	倍長整数	1
Left Arrow Key	倍長整数	28
Page Down Key	倍長整数	12
Page Up Key	倍長整数	11
Return Key	倍長整数	13
Right Arrow Key	倍長整数	29
Tab Key	倍長整数	9
Up Arrow Key	倍長整数	30

Hierarchical Lists

関連コマンド：GET LIST PROPERTIES、SET LIST PROPERTIES

定数	タイプ	値
Ala Macintosh	倍長整数	1
Ala Windows	倍長整数	2
Macintosh node	倍長整数	860
Use PicRef	倍長整数	131072
Use PICT resource	倍長整数	65536
Windows node	倍長整数	138

Is license available

関連コマンド：Is license available

定数	タイプ	値
4D Draw License	倍長整数	808464694
4D for OCI License	倍長整数	808465208
4D View License	倍長整数	808465207
4D Web License	倍長整数	808464945
4D Write License	倍長整数	808464697
Client Web License	倍長整数	808465209

ISO Latin Character Entities

定数	タイプ	値
ISO L1 a acute	文字列	á
ISO L1 a circumflex	文字列	â
ISO L1 a grave	文字列	à
ISO L1 a ring	文字列	å
ISO L1 a tilde	文字列	ã
ISO L1 a umlaut	文字列	ä
ISO L1 ae ligature	文字列	æ
ISO L1 Ampersand	文字列	&
ISO L1 c cedilla	文字列	ç
ISO L1 Cap A acute	文字列	Á
ISO L1 Cap A circumflex	文字列	Â
ISO L1 Cap A grave	文字列	À
ISO L1 Cap A ring	文字列	Å
ISO L1 Cap A tilde	文字列	Ã
ISO L1 Cap A umlaut	文字列	Ä
ISO L1 Cap AE ligature	文字列	&AELig;
ISO L1 Cap C cedilla	文字列	Ç
ISO L1 Cap E acute	文字列	É
ISO L1 Cap E circumflex	文字列	Ê
ISO L1 Cap E grave	文字列	È
ISO L1 Cap E umlaut	文字列	Ë
ISO L1 Cap Eth Icelandic	文字列	Ð
ISO L1 Cap I acute	文字列	Í
ISO L1 Cap I circumflex	文字列	Î
ISO L1 Cap I grave	文字列	Ì
ISO L1 Cap I umlaut	文字列	Ï
ISO L1 Cap N tilde	文字列	Ñ
ISO L1 Cap O acute	文字列	Ó
ISO L1 Cap O circumflex	文字列	Ô
ISO L1 Cap O grave	文字列	Ò
ISO L1 Cap O slash	文字列	Ø
ISO L1 Cap O tilde	文字列	Õ
ISO L1 Cap O umlaut	文字列	Ö
ISO L1 Cap THORN Icelandic	文字列	Þ
ISO L1 Cap U acute	文字列	Ú
ISO L1 Cap U circumflex	文字列	Û
ISO L1 Cap U grave	文字列	Ù

定数	タイプ	値
ISO L1 Cap U umlaut	文字列	Ü
ISO L1 Cap Y acute	文字列	Ý
ISO L1 Copyright	文字列	©
ISO L1 e acute	文字列	é
ISO L1 e circumflex	文字列	ê
ISO L1 e grave	文字列	è
ISO L1 e umlaut	文字列	ë
ISO L1 eth Icelandic	文字列	ð
ISO L1 Greater than	文字列	>
ISO L1 i acute	文字列	í
ISO L1 i circumflex	文字列	î
ISO L1 i grave	文字列	ì
ISO L1 i umlaut	文字列	ï
ISO L1 Less than	文字列	<
ISO L1 n tilde	文字列	ñ
ISO L1 o acute	文字列	ó
ISO L1 o circumflex	文字列	ô
ISO L1 o grave	文字列	ò
ISO L1 o slash	文字列	ø
ISO L1 o tilde	文字列	õ
ISO L1 o umlaut	文字列	ö
ISO L1 Quotation mark	文字列	"
ISO L1 Registered	文字列	®
ISO L1 sharp s German	文字列	ß
ISO L1 thorn Icelandic	文字列	þ
ISO L1 u acute	文字列	ú
ISO L1 u circumflex	文字列	û
ISO L1 u grave	文字列	ù
ISO L1 u umlaut	文字列	ü
ISO L1 y acute	文字列	ý
ISO L1 y umlaut	文字列	ÿ

List box

関連コマンド：Get listbox information、SELECT LISTBOX ROW

定数	タイプ	値
Add to listbox selection	倍長整数	1
Display listbox header	倍長整数	0
Display listbox hor scrollbar	倍長整数	2
Display listbox ver scrollbar	倍長整数	4
Listbox header height	倍長整数	1
Listbox hor scrollbar height	倍長整数	3
Listbox ver scrollbar width	倍長整数	5
Position listbox hor scrollbar	倍長整数	6
Position listbox ver scrollbar	倍長整数	7
Remove from listbox selection	倍長整数	2
Replace listbox selection	倍長整数	0

Math

関連コマンド：Arctan、Cos、Sin、Tan

定数	タイプ	値
Degree	実数	0.0174532925199432958
e number	実数	2.71828182845904524
Pi	実数	3.141592653589793239
Radian	実数	57.29577951308232088

Object alignment

関連コマンド：Get alignment、SET ALIGNMENT

定数	タイプ	値
Align default	倍長整数	1
Align left	倍長整数	2
Align right	倍長整数	4
Center	倍長整数	3

Open form window

関連コマンド：Open form window

定数	タイプ	値
At the Bottom	倍長整数	393216
At the Top	倍長整数	327680
Horizontally Centered	倍長整数	65536
Modal form dialog box	倍長整数	1
Movable form dialog box	倍長整数	5
On the Left	倍長整数	131072
On the Right	倍長整数	196608
Palette form window	倍長整数	1984
Standard form window	倍長整数	8
Vertically Centered	倍長整数	262144

Open window

関連コマンド：Open external window、Open window

定数	タイプ	値
Alternate dialog box	倍長整数	3
Has grow box	倍長整数	4
Has highlight	倍長整数	1
Has window title	倍長整数	2
Has zoom box	倍長整数	8
Modal dialog box	倍長整数	1
Movable dialog box	倍長整数	5
Palette window	倍長整数	720
Plain dialog box	倍長整数	2
Plain fixed size window	倍長整数	4
Plain no zoom box window	倍長整数	0
Plain window	倍長整数	8
Round corner window	倍長整数	16

Picture Compression

関連コマンド：COMPRESS PICTURE、COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE

定数	タイプ	値
QT Animation compressor	文字列	rle
QT Compact video compressor	文字列	cdvc
QT Graphics compressor	文字列	smc
QT Photo compressor	文字列	jpeg
QT Raw compressor	文字列	raw
QT Video compressor	文字列	rpza

Picture Display Formats

関連コマンド：CREATE THUMBNAIL、SET FORMAT

定数	タイプ	値
On Background	倍長整数	3
Scaled to Fit	倍長整数	2
Scaled to fit prop centered	倍長整数	6
Scaled to fit proportional	倍長整数	5
Truncated Centered	倍長整数	1
Truncated non Centered	倍長整数	4

Platform Interfaces

関連コマンド：Get platform interface、SET PLATFORM INTERFACE

定数	タイプ	値
Automatic interface	倍長整数	-1
Copland interface	倍長整数	3
Macintosh interface	倍長整数	0
Windows NT 3.51 Interface	倍長整数	1
Windows 95 interface	倍長整数	2

Platform Properties

関連コマンド：PLATFORM PROPERTIES

定数	タイプ	値
INTEL 386	倍長整数	386
INTEL 486	倍長整数	486
Macintosh 68K	倍長整数	1
Pentium	倍長整数	586
Power Macintosh	倍長整数	2
PowerPC 601	倍長整数	601
PowerPC 603	倍長整数	603
PowerPC 604	倍長整数	604
Windows	倍長整数	3
PowerPC G3	倍長整数	510

Print options

関連コマンド：GET PRINT OPTION、PRINT OPTION VALUES、SET PRINT OPTION

定数	タイプ	値
Color option	倍長整数	8
Destination option	倍長整数	9
Double sided option	倍長整数	11
Number of copies option	倍長整数	4
Orientation option	倍長整数	2
Paper option	倍長整数	1
Paper source option	倍長整数	5
Scale option	倍長整数	3
Spooler document name option	倍長整数	12

Process state

関連コマンド：PROCESS PROPERTIES、Process state

定数	タイプ	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

Process Type

関連コマンド：PROCESS PROPERTIES

定数	タイプ	値
Apple Event Manager	倍長整数	-7
Cache Manager	倍長整数	-4
Created from Menu Command	倍長整数	2
Created from Programming	倍長整数	1
Created from User Mode	倍長整数	3
Design Process	倍長整数	-2
Event Manager	倍長整数	-8
External Task	倍長整数	-9
Indexing Process	倍長整数	-5
None	倍長整数	0
Other 4D Process	倍長整数	-10
Other User Process	倍長整数	4
Serial Port Manager	倍長整数	-6
User or Custom Menus Process	倍長整数	-1
Web Process with Context	倍長整数	-11
Web Process with no Context	倍長整数	-3

QR Area Properties

関連コマンド：QR Get area property、QR SET AREA PROPERTY

定数	タイプ	値
qr view color toolbar	倍長整数	6
qr view column toolbar	倍長整数	5
qr view contextual menus	倍長整数	7
qr view menubar	倍長整数	1
qr view operators toolbar	倍長整数	4
qr view standard toolbar	倍長整数	2
qr view style toolbar	倍長整数	3

QR Borders

関連コマンド：QR GET BORDERS

定数	タイプ	値
qr bottom border	倍長整数	8
qr inside horizontal border	倍長整数	32
qr inside vertical border	倍長整数	16
qr left border	倍長整数	1
qr right border	倍長整数	4
qr top border	倍長整数	2

QR Commands

関連コマンド：QR EXECUTE COMMAND、QR Get command status

定数	タイプ	値
qr cmd 4D View destination	倍長整数	2503
qr cmd add column	倍長整数	2608
qr cmd alt back color palette	倍長整数	1004
qr cmd automatic width	倍長整数	2605
qr cmd average	倍長整数	507
qr cmd back color palette	倍長整数	1003
qr cmd back colors toolbar	倍長整数	2052
qr cmd bold	倍長整数	500
qr cmd borders	倍長整数	2609
qr cmd center justified	倍長整数	504
qr cmd columns toolbar	倍長整数	2054
qr cmd count	倍長整数	510
qr cmd default justified	倍長整数	512
qr cmd delete column	倍長整数	2601
qr cmd disk file destination	倍長整数	2501
qr cmd edit column	倍長整数	2603
qr cmd font color palette	倍長整数	1002
qr cmd font dropdown	倍長整数	1000
qr cmd format	倍長整数	2606
qr cmd generate	倍長整数	2008
qr cmd graph destination	倍長整数	2502
qr cmd header and footer	倍長整数	2005
qr cmd hide column	倍長整数	2602
qr cmd hide line	倍長整数	2607
qr cmd HTML file destination	倍長整数	2504
qr cmd insert column	倍長整数	2600
qr cmd italic	倍長整数	501
qr cmd left justified	倍長整数	503
qr cmd max	倍長整数	509
qr cmd min	倍長整数	508
qr cmd move left	倍長整数	3002
qr cmd move right	倍長整数	3003
qr cmd new	倍長整数	2000
qr cmd open	倍長整数	2001
qr cmd operators toolbar	倍長整数	2051

定数	タイプ	値
qr cmd page setup	倍長整数	2006
qr cmd plain	倍長整数	511
qr cmd presentation	倍長整数	2611
qr cmd print preview	倍長整数	2007
qr cmd printer destination	倍長整数	2500
qr cmd repeated values	倍長整数	2604
qr cmd revert to save	倍長整数	2004
qr cmd right justified	倍長整数	505
qr cmd save	倍長整数	2002
qr cmd save as	倍長整数	2003
qr cmd standard toolbar	倍長整数	2053
qr cmd style toolbar	倍長整数	2050
qr cmd sum	倍長整数	506
qr cmd totals spacing	倍長整数	2610
qr cmd underline	倍長整数	502

QR Document Properties

関連コマンド：QR Get document property、QR SET DOCUMENT PROPERTY

定数	タイプ	値
qr printing dialog	倍長整数	1
qr unit	倍長整数	2

QR Operators

関連コマンド：QR GET TOTALS DATA、QR SET TOTALS DATA

定数	タイプ	値
qr average	倍長整数	2
qr count	倍長整数	16
qr max	倍長整数	8
qr min	倍長整数	4
qr sum	倍長整数	1

QR Output Destination

関連コマンド：QR GET DESTINATION、QR SET DESTINATION

定数	タイプ	値
qr 4D Chart area	倍長整数	4

定数	タイプ	値
qr 4D View area	倍長整数	3
qr HTML file	倍長整数	5
qr printer	倍長整数	1
qr text file	倍長整数	2

QR Report Types

関連コマンド：QR Get report kind、QR SET REPORT KIND

定数	タイプ	値
qr cross report	倍長整数	2
qr list report	倍長整数	1

QR Rows for Properties

関連コマンド：QR GET BORDERS、QR Get info row、QR SET BORDERS、QR SET INFO ROW、QR SET TEXT PROPERTY

定数	タイプ	値
qr detail	倍長整数	-2
qr footer	倍長整数	-5
qr grand total	倍長整数	-3
qr header	倍長整数	-4
qr title	倍長整数	-1

QR Text Properties

関連コマンド：QR Get text property、QR SET TEXT PROPERTY

定数	タイプ	値
qr alternate background color	倍長整数	9
qr background color	倍長整数	8
qr bold	倍長整数	3
qr font	倍長整数	1
qr font size	倍長整数	2
qr italic	倍長整数	4
qr justification	倍長整数	7
qr text color	倍長整数	6
qr underline	倍長整数	5

Query Destinations

関連コマンド：SET QUERY DESTINATION

定数	タイプ	値
Into current selection	倍長整数	0
Into named selection	倍長整数	2
Into set	倍長整数	1
Into variable	倍長整数	3

Relations

関連コマンド：GET FIELD RELATION、SET FIELD RELATION

定数	タイプ	値
Automatic	倍長整数	3
Do not modify	倍長整数	0
Manual	倍長整数	2
No relation	倍長整数	0
Structure configuration	倍長整数	1

Resources Properties

関連コマンド：Get resource properties、SET RESOURCE PROPERTIES

定数	タイプ	値
Changed resource bit	倍長整数	1
Changed resource mask	倍長整数	2
Locked resource bit	倍長整数	4
Locked resource mask	倍長整数	16
Preloaded resource bit	倍長整数	2
Preloaded resource mask	倍長整数	4
Protected resource bit	倍長整数	3
Protected resource mask	倍長整数	8
Purgeable resource bit	倍長整数	5
Purgeable resource mask	倍長整数	32
System heap resource bit	倍長整数	6
System heap resource mask	倍長整数	64

SCREEN DEPTH

関連コマンド：SCREEN DEPTH、SET SCREEN DEPTH

定数	タイプ	値
Black and white	倍長整数	0
Four colors	倍長整数	2
Is color	倍長整数	1
Is gray scale	倍長整数	0
Millions of colors 24 bit	倍長整数	24
Millions of colors 32 bit	倍長整数	32
Sixteen colors	倍長整数	4
Thousands of colors	倍長整数	16
Two fifty six colors	倍長整数	8

SET RGB COLOR

関連コマンド：SET RGB COLORS

定数	タイプ	値
Default background color	倍長整数	-2
Default dark shadow color	倍長整数	-3
Default foreground color	倍長整数	-1
Default light shadow color	倍長整数	-4

Standard System Signatures

関連コマンド：APPEND TO CLIPBOARD、GET CLIPBOARD、Get resource properties、SET RESOURCE PROPERTIES、Test clipboard

定数	タイプ	値
Picture Document	文字列	PICT
QT Animation compressor	文字列	rle
QT Compact video compressor	文字列	cdvc
QT Graphics compressor	文字列	smc
QT Photo compressor	文字列	jpeg
QT Raw compressor	文字列	raw
QT Video compressor	文字列	rpza
Text Document	文字列	TEXT
Windows MIDI Document	文字列	MI D
Windows Sound Document	文字列	WAV
Windows Video Document	文字列	AVI

System Documents

関連コマンド：Open document、Test path name

定数	タイプ	値
Get Pathname	倍長整数	3
Is a directory	倍長整数	0
Is a document	倍長整数	1
Read and Write	倍長整数	0
Read Mode	倍長整数	2
Write Mode	倍長整数	1

System Folder

関連コマンド：System folder

定数	タイプ	値
Apple or Start Menu_All	倍長整数	8
Apple or Start Menu_User	倍長整数	9
Desktop Win	倍長整数	15
Favorites Win	倍長整数	14
Fonts	倍長整数	1
Mac Control Panels	倍長整数	11
Mac Extensions	倍長整数	10
Mac Shutdown Items_All	倍長整数	6
Mac Shutdown Items_User	倍長整数	7
Preferences or Profiles_All	倍長整数	2
Preferences or Profiles_User	倍長整数	3
Program Files Win	倍長整数	16
Startup Items_All	倍長整数	4
Startup Items_User	倍長整数	5
System	倍長整数	0
System Win	倍長整数	12
System32 Win	倍長整数	13

TCP Port Numbers

関連コマンド：Get database parameter、SET DATABASE PARAMETER

定数	タイプ	値
TCP Authentication	倍長整数	113
TCP DNS	倍長整数	53
TCP Finger	倍長整数	79
TCP FTP Control	倍長整数	21
TCP FTP Data	倍長整数	20
TCP Gopher	倍長整数	70
TCP HTTP WWW	倍長整数	80
TCP IMAP3	倍長整数	220
TCP Kerberos	倍長整数	88
TCP KLogin	倍長整数	543
TCP Nickname	倍長整数	43
TCP NNTP	倍長整数	119
TCP NTalk	倍長整数	518
TCP NTP	倍長整数	123
TCP PMCP	倍長整数	1643
TCP PMD	倍長整数	1642
TCP POP3	倍長整数	110
TCP Printer	倍長整数	515
TCP RADACCT	倍長整数	1646
TCP RADIUS	倍長整数	1645
TCP Remote Cmd	倍長整数	514
TCP Remote Exec	倍長整数	512
TCP Remote Login	倍長整数	513
TCP Router	倍長整数	520
TCP SMTP	倍長整数	25
TCP SNMP	倍長整数	161
TCP SNMPTRAP	倍長整数	162
TCP SUN RPC	倍長整数	111
TCP Talk	倍長整数	517
TCP Telnet	倍長整数	23
TCP TFTP	倍長整数	69
TCP UUCP	倍長整数	540
TCP UUCP RLOGIN	倍長整数	541

Time Display Formats

関連コマンド：SET FORMAT、String

定数	タイプ	値
HH MM	倍長整数	2
HH MM AM PM	倍長整数	5
HH MM SS	倍長整数	1
Hour Min	倍長整数	4
Hour Min Sec	倍長整数	3

Web Services(Client)

関連コマンド：CALL WEB SERVICE、Get Web Service error info

定数	タイプ	値
Web Service Detailed Message	倍長整数	1
Web Service Dynamic	倍長整数	0
Web Service Error Code	倍長整数	0
Web Service Fault Actor	倍長整数	3
Web Service HTTP Error code	倍長整数	2
Web Service Manual	倍長整数	3
Web Service Manual In	倍長整数	1
Web Service Manual Out	倍長整数	2

Web Services(Server)

関連コマンド：Get SOAP info、SEND SOAP FAULT、SOAP DECLARATION

定数	タイプ	値
SOAP Client Fault	倍長整数	1
SOAP Input	倍長整数	1
SOAP Method Name	倍長整数	1
SOAP Output	倍長整数	2
SOAP Server Fault	倍長整数	2
SOAP Service Name	倍長整数	2

Window kind

関連コマンド：Window kind

定数	タイプ	値
External window	倍長整数	5
Floating window	倍長整数	14
Modal dialog	倍長整数	9
Regular window	倍長整数	8

Windows Log Events

関連コマンド：LOG EVENT

定数	タイプ	値
Error Message	倍長整数	2
Information Message	倍長整数	0
Warning Message	倍長整数	1

XML

関連コマンド：Parse XML information

定数	タイプ	値
DOCTYPE Name	倍長整数	3
Encoding	倍長整数	4
PUBLIC ID	倍長整数	1
SYSTEM ID	倍長整数	2
URI Document	倍長整数	6
Version	倍長整数	5

1954

コマンド索引

A

ABORT ・・・

ARRAY INTEGER(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・186
ARRAY LONGINT(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・187
ARRAY PICTURE(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・195
ARRAY POINTER(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・197
ARRAY REAL(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・・188
ARRAY STRING(長さ ; 配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・189
ARRAY TEXT(配列名 ; サイズ1 ; {サイズ2}) ・・・・・・・・・・・・・・・191
ARRAY TO LIST(配列 ; リスト ; {項目参照}) ・・・・・・・・・・・・・・・213
ARRAY TO SELECTION(配列1 ; フィールド1{ ;...; 配列N ; フィールドN}) ・220
ARRAY TO STRING LIST(ストリング ; リソースID {; リソースファイル}) ・1183
Ascii(文字) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・1284
AUTHENTICATE WEB SERVICE(名前; パスワード) ・・・・・・・・・・・1731
Average(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・・・854

B

BACKUP ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・227
BEEP ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1479
Before selection {(テーブル)} → ブール ・・・・・・・・・・・・・・・・1231
Before subselection(サブテーブル) → ブール ・・・・・・・・・・・・・1344
Before → ブール ・・・・・・・・・・・・・・・・・・・・・・・・・・・558
BEST OBJECT SIZE({*;}オブジェクト; 最適幅; 最適高さ{; 最大幅}) ・・・・845
BLOB PROPERTIES(blob ; 圧縮 {;サイズ {現在サイズ}} }) ・・・・・・・・244
BLOB size(blob) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・239
BLOB TO DOCUMENT(ドキュメント ; blob {; *}) ・・・・・・・・・・・・・248
BLOB to integer(blob ; バイト順序 {;オフセット}) → 数値 ・・・・・・・・270
BLOB to list(blob {;オフセット}) → 数値 ・・・・・・・・・・・・・・・・256
BLOB to longint(blob ; バイト順序 {;オフセット}) → 数値 ・・・・・・・・272
BLOB TO PICTURE(ピクチャBlob;ピクチャ) ・・・・・・・・・・・・・・・932
BLOB to real(blob ; 実数形式 {;オフセット})→実数 ・・・・・・・・・・・274
BLOB to text(blob ; テキスト形式 {;オフセット {;テキストの長さ}}) → テキスト
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・276
BLOB TO USERS (ユーザ) ・・・・・・・・・・・・・・・・・・・・・・・1529
BLOB TO VARIABLE(blob ; 変数 {;オフセット}) ・・・・・・・・・・・・・253
BOOLEAN ARRAY FROM SET(ブール配列{; セット }) ・・・・・・・・・・・225
BREAK LEVEL(レベル {; ページブレーク}) ・・・・・・・・・・・・・・・891
BRING TO FRONT(プロセス) ・・・・・・・・・・・・・・・・・・・・・・965
BUILD APPLICATION{(プロジェクト名)} ・・・・・・・・・・・・・・・・1437

BUTTON TEXT({*;} オブジェクト ; ボタンテキスト) ・・・・・・・・・・・815

C

CALL PROCESS(プロセス) ・・・・・・・・・・・・・・・・・・・・・・・・953
CALL WEB SERVICE(アクセスURL; soapAction; メソッド名; 名前空間
{; 複合型}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1724
CANCEL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・500
CANCEL TRANSACTION ・・・・・・・・・・・・・・・・・・・・・・・1451
Caps lock down → ブール ・・・・・・・・・・・・・・・・・・・・・・1494
CHANGE CURRENT USER({ユーザ名}{;パスワード}) ・・・・・・・・・・1516
CHANGE LICENSES ・・・・・・・・・・・・・・・・・・・・・・・・・1539
CHANGE PASSWORD(パスワード) ・・・・・・・・・・・・・・・・・・1519
Change string(元の文字列 ; 修正文字列 ; 修正位置) → 文字列 ・・・・・・1291
Char(ASCIIコード) → 文字列(1バイト) ・・・・・・・・・・・・・・・1286
CLEAR CLIPBOARD ・・・・・・・・・・・・・・・・・・・・・・・・・300
CLEAR LIST(リスト {; *}) ・・・・・・・・・・・・・・・・・・・・・608
CLEAR NAMED SELECTION(名前) ・・・・・・・・・・・・・・・・・・803
CLEAR SEMAPHORE(セマフォ) ・・・・・・・・・・・・・・・・・・・951
CLEAR SET(セット) ・・・・・・・・・・・・・・・・・・・・・・・・1261
CLEAR VARIABLE (変数) ・・・・・・・・・・・・・・・・・・・・・・1546
CLOSE DOCUMENT(ドキュメントファイル参照番号) ・・・・・・・・・・1367
CLOSE RESOURCE FILE(リソースファイル) ・・・・・・・・・・・・・1177
CLOSE WINDOW {(ウインドウ)} ・・・・・・・・・・・・・・・・・・1758
Command name(コマンド番号)→ 文字列 ・・・・・・・・・・・・・・・699
Compiled application→プール ・・・・・・・・・・・・・・・・・・・135
COMPRESS BLOB(blob {;圧縮}) ・・・・・・・・・・・・・・・・・・240
COMPRESS PICTURE FILE(ドキュメント参照番号 ; 方法 ; 品質) ・・・・926
COMPRESS PICTURE(ピクチャ ; 方法 ; 品質) ・・・・・・・・・・・・924
CONFIRM(メッセージ {; OKボタンタイトル{; キャンセルボタンタイトル}}) ・785
Contextual click → ブール値 ・・・・・・・・・・・・・・・・・・・568
COPY ARRAY(コピー元 ; コピー先) ・・・・・・・・・・・・・・・・・211
COPY BLOB(コピー元BLOB ; コピー先BLOB ; コピー元オフセット ;
コピー先オフセット ; コピー数) ・・・・・・・・・・・・・・・・・・281
COPY DOCUMENT(コピー元 ; コピー先 {; *}) ・・・・・・・・・・・・1368
Copy list(リスト)→ 数値 ・・・・・・・・・・・・・・・・・・・・・607
COPY NAMED SELECTION({テーブル ;} 名前) ・・・・・・・・・・・・799
COPY SET(コピー元 ; コピー先) ・・・・・・・・・・・・・・・・・・1273

Cos(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・746
Count fields(テーブル番号 ¦ テーブルポインタ) → 数値 ・・・・・・・・・1305
Count in array(配列; 値) →倍長整数 ・・・・・・・・・・・・・・・・・・207
Count list item(リスト{; *}) → 数値 ・・・・・・・・・・・・・・・・・・610
Count menu items(メニュー {; プロセス}) → 数値 ・・・・・・・・・・・・766
Count menus({プロセス}) → 数値 ・・・・・・・・・・・・・・・・・・・766
Count parameters → 数値 ・・・・・・・・・・・・・・・・・・・・・・・687
Count screens → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・1406
Count tables → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・1304
Count tasks → 整数 ・・・・・・・・・・・・・・・・・・・・・・・・・992
Count user processes → 整数 ・・・・・・・・・・・・・・・・・・・・993
Count users → 整数 ・・・・・・・・・・・・・・・・・・・・・・・・・992
CREATE ALIAS(ターゲットパス; エイリアスパス) ・・・・・・・・・・・1383
CREATE DATA FILE(アクセスパス) ・・・・・・・・・・・・・・・・・・150
Create document(ドキュメント {; タイプ}) → ドキュメントファイル参照番号
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1363
CREATE EMPTY SET({テーブル ;} セット) ・・・・・・・・・・・・・・・1256
CREATE FOLDER(パス名) ・・・・・・・・・・・・・・・・・・・・・・1378
CREATE RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・1110
CREATE RELATED ONE(フィールド) ・・・・・・・・・・・・・・・・・1148
Create resource file(リソースファイル名 {; ファイルタイプ }) → DocRef ・1174
CREATE SELECTION FROM ARRAY(テーブル;レコード配列
{;セレクション名}) ・・・・・・・・・・・・・・・・・・・・・・・・・804
CREATE SET FROM ARRAY(テーブル ; レコード配列 ; {セット名}) ・・1258
CREATE SET({テーブル ;} セット) ・・・・・・・・・・・・・・・・・・1257
CREATE SUBRECORD(サブテーブル) ・・・・・・・・・・・・・・・・・1333
CREATE THUMBNAIL(ピクチャ; サムネール{; 幅{; 高さ{; モード{; 深度}}}}) ・938
CREATE USER FORM (テーブル; フォーム; ユーザフォーム) ・・・・・1475
Current date {(*)} → 日付 ・・・・・・・・・・・・・・・・・・・・・・427
Current default table → ポインタ ・・・・・・・・・・・・・・・・・・1431
Current form page → 数値 ・・・・・・・・・・・・・・・・・・・・・・582
Current form table → ポインタ ・・・・・・・・・・・・・・・・・・・1432
Current form window → ウインドウ参照番号 ・・・・・・・・・・・・・1782
Current machine owner→ 文字列 ・・・・・・・・・・・・・・・・・・1421
Current machine → 文字列 ・・・・・・・・・・・・・・・・・・・・・1420
Current method name → 文字列 ・・・・・・・・・・・・・・・・・・・702
Current process → プロセス参照番号 ・・・・・・・・・・・・・・・・984

Current time {(*)} → 時間 ・・・・・・・・・・・・・・・・・・・・・・・・・438
Current user → 文字列 ・・・・・・・・・・・・・・・・・・・・・・・・1520
CUT NAMED SELECTION({テーブル ;} 名前) ・・・・・・・・・・・・・・・801
C_BLOB({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・・390
C_BOOLEAN({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・391
C_DATE({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・392
C_GRAPH({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・393
C_INTEGER({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・394
C_LONGINT({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・395
C_PICTURE({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・396
C_POINTER({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・397
C_REAL({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・398
C_STRING({メソッド ;}サイズ; 変数 {; 変数2 ; …; 変数N}) ・・・・・・・・399
C_TEXT({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・400
C_TIME({メソッド ;}変数 {; 変数2 ; …; 変数N}) ・・・・・・・・・・・・401

D

Data file({セグメント})→文字列 ・・・・・・・・・・・・・・・・・・・・139
DATA SEGMENT LIST({セグメント}) ・・・・・・・・・・・・・・・・・・145
Database event → 数値 ・・・・・・・・・・・・・・・・・・・・・・・1466
Date(日付文字列) → 日付 ・・・・・・・・・・・・・・・・・・・・・・437
Day number(日付) → 数値 ・・・・・・・・・・・・・・・・・・・・・・434
Day of(日付) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・430
Deactivated → ブール ・・・・・・・・・・・・・・・・・・・・・・・・562
Dec(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・737
DECODE(blob) ・・・・・・・・・・・・・・・・・・・・・・・・・・・1440
DECRYPT BLOB(データ; 送信公開鍵{; 受信秘密鍵}) ・・・・・・・・・・・288
DEFAULT TABLE(テーブル) ・・・・・・・・・・・・・・・・・・・・・1429
DELAY PROCESS(プロセス ; 遅延時間) ・・・・・・・・・・・・・・・・980
DELETE DOCUMENT(ドキュメント) ・・・・・・・・・・・・・・・・・1370
DELETE ELEMENT(配列 ; 削除位置 ; {要素数}) ・・・・・・・・・・・・209
DELETE FOLDER(フォルダ) ・・・・・・・・・・・・・・・・・・・・・1382
DELETE FROM BLOB(blob ; オフセット ; 削除数) ・・・・・・・・・・・・280
DELETE LIST ITEM(リスト ; 項目参照番号 ¦ * {; *}) ・・・・・・・・・・644
DELETE LISTBOX COLUMN ({*; }オブジェクト; 位置{; 列数}) ・・・・・・715
DELETE LISTBOX ROW ({*; }オブジェクト; 位置) ・・・・・・・・・・・723
DELETE MENU ITEM(メニュー ; メニュー項目 {; プロセス}) ・・・・・・780

DELETE RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・・・1117
DELETE RESOURCE(リソースタイプ ; リソースID {; リソースファイル}) ・1208
DELETE SELECTION {(テーブル)} ・・・・・・・・・・・・・・・・・・・1221
Delete string(元の文字列 ; 削除位置 ; 文字数) → 文字列 ・・・・・・・・・1293
DELETE SUBRECORD(サブテーブル) ・・・・・・・・・・・・・・・・・1335
DELETE USER FORM (テーブル; フォーム; ユーザフォーム) ・・・・・1477
DELETE USER(ユーザID) ・・・・・・・・・・・・・・・・・・・・・・1523
DIALOG({テーブル ; }フォーム) ・・・・・・・・・・・・・・・・・・・・421
DIFFERENCE(セット1 ; セット2 ; 結果セット) ・・・・・・・・・・・・・1267
DISABLE BUTTON({*;} オブジェクト) ・・・・・・・・・・・・・・・・・813
DISABLE MENU ITEM(メニュー番号 ; メニュー項目 {; プロセス}) ・・・・775
DISPLAY NOTIFICATION (タイトル; テキスト{; 表示時間}) ・・・・・・・796
DISPLAY RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・1109
DISPLAY SELECTION({テーブル} {; 選択モード}{; リスト更新}{; *{; *}}) ・1235
Displayed line number→ 倍長整数 ・・・・・・・・・・・・・・・・・・1227
DISTINCT VALUES(フィールド ; 配列) ・・・・・・・・・・・・・・・・・222
Document creator (ドキュメント) → 文字列 ・・・・・・・・・・・・・・1359
DOCUMENT LIST(パス名 ; ドキュメント) ・・・・・・・・・・・・・・・1391
DOCUMENT TO BLOB(ドキュメント ; blob {; *}) ・・・・・・・・・・・・246
Document type (ドキュメント) → 文字列 ・・・・・・・・・・・・・・・1357
DOM CLOSE XML(elementRef) ・・・・・・・・・・・・・・・・・・・・1827
DOM Count XML attributes(elementRef) → 倍長整数 ・・・・・・・・・・1818
DOM Count XML elements(elementRef; elementName) → 倍長整数 ・・・1803
DOM Create XML element(elementRef; xPath{; 属性名; 属性値}...
{; 属性名X; 属性値X}) → 文字列 ・・・・・・・・・・・・・・・・・・・1798
DOM Create XML ref(ルート{; 名前空間{; 名前空間名; 名前空間値}...
{; 名前空間名X; 名前空間値X}}) → 文字列 ・・・・・・・・・・・・・・1791
DOM EXPORT TO FILE(elementRef; filePath) ・・・・・・・・・・・・・1825
DOM EXPORT TO VAR(elementRef; vXmlVar) ・・・・・・・・・・・・・1826
DOM Find XML element(elementRef; xPath) → 文字列 ・・・・・・・・・1801
DOM Get First Child XML element(elementRef{; childElemName
{; childElemValue}}) → 文字列 ・・・・・・・・・・・・・・・・・・・1806
DOM Get last child XML element (elementRef{;childElemName
{;childElemValue}})→文字列 ・・・・・・・・・・・・・・・・・・・・1811
DOM Get next sibling XML element(elementRef{; siblingElemName
{; siblingElemValue}}) → 文字列 ・・・・・・・・・・・・・・・・・・・1808

DOM Get parent XML element（elementRef{; parentElemName {; parentElemValue}}）→文字列 ・・・・・・・・・・・・・・・・・・・・・・1805
DOM Get previous sibling XML element (elementRef {; siblingElemName {; siblingValue}})→文字列 ・・・・・・・・・・・・・・・・・・・・・・1810
DOM GET XML ATTRIBUTE BY INDEX(elementRef; attribIndex; attribName {; attribValue}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1821
DOM GET XML ATTRIBUTE BY NAME(elementRef; attribName; attribValue) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1822
DOM GET XML ELEMENT NAME(elementRef; elementName) ・・・・・1814
DOM GET XML ELEMENT VALUE(elementRef; elementValue{; cDATA}) ・1816
DOM Get XML element(elementRef; elementName; index; elementValue) → 文字列 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1804
DOM Get XML information(elementRef; xmlInfo) → 文字列 ・・・・・・・1824
DOM Parse XML source(document{; validation{; dtd}}) → 文字列 ・・・・1794
DOM Parse XML variable(変数{; validation{; dtd}}) → 文字列 ・・・・・・1796
DOM REMOVE XML ELEMENT(elementRef) ・・・・・・・・・・・・・1817
DOM SET XML ATTRIBUTE(elementRef; attribName; attribValue... {; attribName X; attribValue X}) ・・・・・・・・・・・・・・・・・・・・1820
DOM SET XML ELEMENT NAME(elementRef; elementName) ・・・・・1812
DOM SET XML ELEMENT VALUE(elementRef; elementValue) ・・・・・1815
DOM SET XML OPTIONS(elementRef;エンコード{; スタンドアロン {; インデント}}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・1793
DRAG AND DROP PROPERTIES(対象オブジェクト;対象要素;対象プロセス) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・492
DRAG WINDOW ・・・・・・・・・・・・・・・・・・・・・・・・1761
Drop position → 整数 ・・・・・・・・・・・・・・・・・・・・・・491
DUPLICATE RECORD {(テーブル)} ・・・・・・・・・・・・・・・・1111
During → ブール ・・・・・・・・・・・・・・・・・・・・・・・・558

E

EDIT ACCESS ・・・・・・・・・・・・・・・・・・・・・・・・・1515
EDIT FORM (テーブル; フォーム{; ユーザフォーム{; ライブラリ}}) ・・・1473
EDIT FORMULA (テーブル; フォーミュラ) ・・・・・・・・・・・・・・601
EDIT ITEM ({*; } オブジェクト{; 項目}) ・・・・・・・・・・・・・・516
ENABLE BUTTON({*;} オブジェクト) ・・・・・・・・・・・・・・・・811
ENABLE MENU ITEM(メニュー番号 ; メニュー項目 {; プロセス}) ・・・・776
ENABLE MENU ITEM(メニュー番号 ; メニュー項目番号) ・・・・・・・・776

ENCODE(blob) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1439
ENCRYPT BLOB(データ; 送信秘密鍵{; 受信公開鍵}) ・・・・・・・・・・・・282
End selection{(テーブル)} → ブール ・・・・・・・・・・・・・・・・・・1233
End subselection(サブテーブル) → ブール ・・・・・・・・・・・・・・・1345
ERASE WINDOW {(ウインドウ)} ・・・・・・・・・・・・・・・・・・・・1759
Euro converter(値;元の通貨;変換通貨) → 実数 ・・・・・・・・・・・・752
EXECUTE ON CLIENT(クライアント名 ; メソッド名 {; 引数 1; ...; 引数 N }) ・994
Execute on server(メソッド ; スタック {; 名前 {; パラメータ {; パラメータ2 ;...;
パラメータN}{; *}}}) → 数値 ・・・・・・・・・・・・・・・・・・・・・975
EXECUTE(ステートメント) ・・・・・・・・・・・・・・・・・・・・・・・698
Exp(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・744
EXPAND BLOB(blob) ・・・・・・・・・・・・・・・・・・・・・・・・・242
EXPORT DATA(ファイル名 {; プロジェクト {; *}}) ・・・・・・・・・・・671
EXPORT DIF({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・・667
EXPORT SYLK({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・・663
EXPORT TEXT({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・・659

F

False → ブール値(False) ・・・・・・・・・・・・・・・・・・・・・・・290
Field name((テーブル番号 ; フィールド番号) ¦ フィールドポインタ) → 文字列
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1307
Field(テーブル番号 ¦ フィールドポインタ {; フィールド番号) → 数値 ¦ ポインタ
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1310
FILTER EVENT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・678
FILTER KEYSTROKE(フィルタ文字) ・・・・・・・・・・・・・・・・・・・507
Find in array(配列 ; 値 ; {開始}) → 数値 ・・・・・・・・・・・・・・・・205
Find index key(インデックスフィールド ; 値) → 倍長整数 ・・・・・・・・1023
Find window(左 ; 上 { ; ウインドウ部分}) → ウインドウ参照番号 ・・・・1778
FIRST PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・580
FIRST RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・・1226
FIRST SUBRECORD(サブテーブル) ・・・・・・・・・・・・・・・・・・1340
FLUSH BUFFERS ・・・・・・・・・・・・・・・・・・・・・・・・・・・・148
Focus object → ポインタ ・・・・・・・・・・・・・・・・・・・・・・1512
FOLDER LIST(パス名 ; ディレクトリ) ・・・・・・・・・・・・・・・・・1390
Font number(フォント名) → 数値 ・・・・・・・・・・・・・・・・・・1413
FONT LIST(フォント) ・・・・・・・・・・・・・・・・・・・・・・・・1411
Font name(フォント番号) → 文字列 ・・・・・・・・・・・・・・・・・1412

FONT SIZE({*;} オブジェクト ; フォントサイズ) ・・・・・・・・・・・・・808
FONT STYLE({*;} オブジェクト ; フォントスタイル) ・・・・・・・・・・・809
FONT({*;} オブジェクト ; フォント) ・・・・・・・・・・・・・・・・・・807
Form event → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・537
Frontmost process {(*)} → 整数 ・・・・・・・・・・・・・・・・・・・・966
Frontmost window {(*)} → ウインドウ参照番号 ・・・・・・・・・・・・・1777

G

GENERATE CERTIFICATE REQUEST(秘密鍵; 証明書リクエスト; コード配列; 名称配列) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1215
GENERATE ENCRYPTION KEYPAIR(秘密鍵; 公開鍵 {; 長さ}) ・・・・・1213
Gestalt(セレクタ ; 値)→ 文字列 ・・・・・・・・・・・・・・・・・・・・1422
GET MOUSE(水平 ; 垂直 : マウスボタン { ; *}) ・・・・・・・・・・・・・1499
Get 4D folder{(folder)}→文字列 ・・・・・・・・・・・・・・・・・・・・142
Get alignment({*; }オブジェクト) → 数値 ・・・・・・・・・・・・・・・847
GET ALLOWED METHODS (メソッド配列) ・・・・・・・・・・・・・・・・600
GET AUTOMATIC RELATIONS (1対1; n) ・・・・・・・・・・・・・・・・1135
GET BACKUP INFORMATION(セレクタ; 情報1; 情報2) ・・・・・・・・・・228
GET CLIPBOARD(データタイプ ; データ) ・・・・・・・・・・・・・・・・301
Get component resource ID(コンポーネント名; タイプ; オリジナル番号) →数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1211
Get current printer→ 文字列 ・・・・・・・・・・・・・・・・・・・・・890
Get database parameter({テーブル;} セレクタ) →倍長整数 ・・・・・・・・1319
Get default user→ 数値 ・・・・・・・・・・・・・・・・・・・・・・・・1521
GET DOCUMENT ICON(ドキュメントパス; アイコン{; サイズ}) ・・・・・1401
Get document position(ドキュメント参照番号) → 数値 ・・・・・・・・・1403
GET DOCUMENT PROPERTIES(ドキュメント ; ロック ; 非表示 ; 作成日 ; 作成時間 ; 更新日 ; 更新時間) ・・・・・・・・・・・・・・・・・・・・・・1394
Get document size(ドキュメント {; *}) → 数値 ・・・・・・・・・・・・・1402
Get edited text → Text ・・・・・・・・・・・・・・・・・・・・・・・・563
GET FIELD ENTRY PROPERTIES(フィールドポインタ ¦ テーブル番号 {; フィールド番号}; リスト{; 必須入力{; 表示のみ{; 修正不可}}}) ・・・・・1313
GET FIELD PROPERTIES(フィールドポインタ ; タイプ {; 長さ {;インデックス}}) または(フィールドポインタ ¦ テーブル番号 {; フィールド番号}; タイプ {; 長さ {;インデックス{;ユニーク{;非表示}}}}) ・・・・・・・・・・・・・・・・・・1311
GET FIELD RELATION (始点フィールド; 1対1; 1対n{; *}) ・・・・・・・・1138
GET FIELD TITLES（テーブル;フィールド名;フィールド番号） ・・・・1492

GET FORM OBJECTS (オブジェクト配列{; 変数配列{; ページ配列}}{; *}) ・577
GET FORM PROPERTIES({テーブル ;} フォーム名 ; 幅 ; 高さ {; ページ数
{; 幅変更 {; 高さ変更 {; タイトル }}}}) ・・・・・・・・・・・・・・・・・・・・・569
Get format ({*; } オブジェクト) → 文字列 ・・・・・・・・・・・・・・・・817
GET GROUP LIST(グループ名 ; グループID) ・・・・・・・・・・・・・・1534
GET GROUP PROPERTIES(グループID ; グループ名 ; オーナー {; メンバーID})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1535
GET HIGHLIGHT(エリア ; 先頭 ; 最終) ・・・・・・・・・・・・・・・・・1508
GET HIGHLIGHTED RECORDS({テーブル; }セット名) ・・・・・・・・・1248
GET HTTP BODY(ボディ) ・・・・・・・・・・・・・・・・・・・・・・・1705
GET HTTP HEADER(ヘッダ¦フィールド配列{;値配列}) ・・・・・・・・1702
GET ICON RESOURCE(リソースID ; リソースデータ {; リソースファイル})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1193
Get indexed string(リソースID ; ストリングID {; リソースファイル}) → 文字列
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1186
GET LIST ITEM PROPERTIES(リスト; 項目参照¦ *; 入力可 {;スタイル
{;アイコン{; カラー}}}) ・・・・・・・・・・・・・・・・・・・・・・・・639
GET LIST PROPERTIES(リスト;表示形式{;アイコン{;線の高さ{;ダブルクリック
{; 複数選択可{; 編集可}}}}}) ・・・・・・・・・・・・・・・・・・・・・・623
GET LISTBOX ARRAYS ({*; }オブジェクト; 列名配列; ヘッダ名配列;
列変数配列; ヘッダ変数配列; 表示配列; スタイル配列) ・・・・・・・・・728
Get listbox column width ({*; }オブジェクト) →整数 ・・・・・・・・・・・719
Get listbox information ({*; }オブジェクト; 情報タイプ) →倍長整数 ・・・・730
Get listbox rows height ({*; }オブジェクト) → 整数 ・・・・・・・・・・・726
Get menu item key(メニュー ; メニュー項目 {; プロセス}) → 数値 ・・・・773
Get menu item mark(メニュー ; メニュー項目 {; プロセス}) → 文字列 ・・・771
Get menu item style(メニュー ; メニュー項目 {; プロセス}) → 数値 ・・・・769
Get menu item(メニュー ; メニュー項目 {; プロセス}) → 文字列 ・・・・・767
Get menu title(メニュー {; プロセス}) → 文字列 ・・・・・・・・・・・・767
Get number of listbox columns ({*; }オブジェクト) → 倍長整数 ・・・・・・716
Get number of listbox rows ({*; }オブジェクト) → 倍長整数 ・・・・・・・724
GET OBJECT RECT({*;} オブジェクト ; 左 ; 上 ; 右 ; 下) ・・・・・・・・842
GET PICTURE FROM CLIPBOARD(ピクチャ) ・・・・・・・・・・・・・・303
GET PICTURE FROM LIBRARY(ピクチャ参照番号¦ ピクチャ名;ピクチャ) 943
GET PICTURE RESOURCE(リソースID ; リソースデータ {; リソースファイル})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1191
Get platform interface → 数値 ・・・・・・・・・・・・・・・・・・・・・1482

Get plugin access(プラグイン) → 文字列 ・・・・・・・・・・・・・・・・・1541
GET PLUGIN LIST(番号配列：名前配列) ・・・・・・・・・・・・・・・・・1542
Get pointer(変数名) → ポインタ ・・・・・・・・・・・・・・・・・・・・・697
Get print marker(マーカー番号) → 数値 ・・・・・・・・・・・・・・・・・907
GET PRINT OPTION(オプション; 値1{; 値2}) ・・・・・・・・・・・・・・・896
GET PRINTABLE AREA(高さ{; 幅}) ・・・・・・・・・・・・・・・・・・・・918
GET PRINTABLE MARGIN(左; 上; 右; 下) ・・・・・・・・・・・・・・・・915
Get printed height → 数値 ・・・・・・・・・・・・・・・・・・・・・・・919
GET PROCESS VARIABLE(プロセス ; 送信元変数 ; 送信先変数 { ; 送信元変数2;
送信先変数2 ; ...; 送信元変数N ; 送信先変数N}) ・・・・・・・・・・・・・955
GET REGISTERED CLIENTS(クライアントリスト ; メソッド) ・・・・・1000
GET RELATION PROPERTIES(フィールドポインタ ¦ テーブル番号
{; フィールド番号}; 1テーブル; 1フィールド{; 選択フィールド{; 自動1
{; 自動n}}}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1315
Get resource name(リソースタイプ ; リソースID {; リソースファイル}) → 文字列
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1200
Get resource properties(リソースタイプ ; リソースID {; リソースファイル})
→ 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1203
GET RESOURCE(リソースタイプ ; リソースID ; リソースデータ
{; リソースファイル}) ・・・・・・・・・・・・・・・・・・・・・・・・1195
GET RESTORE INFORMATION(セレクタ; 情報1; 情報2) ・・・・・・・・・230
GET SERIAL INFORMATION(キー; ユーザ名; 会社名; 接続ユーザ; 最大ユーザ)
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・155
GET SERIAL PORT MAPPING(番号配列;名前配列) ・・・・・・・・・・・319
Get SOAP info(情報番号) → 文字列 ・・・・・・・・・・・・・・・・・・1741
Get string resource(リソースID {; リソースファイル}) → 文字列 ・・・・1187
GET TABLE PROPERTIES(テーブルポインタ ¦ テーブル番号; 非表示
{; 新規保存トリガ{; 既存保存トリガ{; 削除トリガ{; 読込みトリガ}}}}) ・・1309
GET TABLE TITLES（テーブル名;テーブル番号） ・・・・・・・・・・・1489
Get text from clipboard → 文字列 ・・・・・・・・・・・・・・・・・・・304
Get text resource(リソースID {; リソースファイル}) → テキスト ・・・・1189
GET USER LIST(ユーザ名 ; ユーザID) ・・・・・・・・・・・・・・・・・1525
GET USER PROPERTIES(ユーザID; ユーザ名; 起動メソッド; パスワード;
ログイン回数; 前回ログイン{; メンバー{; グループオーナ}) ・・・・・・・1526
GET WEB FORM VARIABLES(名前配列;値配列) ・・・・・・・・・・・・1697
Get Web Service error info(情報タイプ) → 文字列 ・・・・・・・・・・・1732
GET WEB SERVICE RESULT(戻り値{; パラメータ名}{; *}}) ・・・・・・1729

GET WINDOW RECT(左 ; 上 ; 右 ; 下 {; ウインドウ}) ・・・・・・・・・・1774
Get window title {(ウインドウ)} → 文字列 ・・・・・・・・・・・・・・・1768
GET XML ERROR(elementRef; errorText{; row{; column}}) ・・・・・・・・1848
GET XSLT ERROR(エラーテキスト{; 行{; 桁位置}}) ・・・・・・・・・・・1853
GOTO AREA({*;}オブジェクト) ・・・・・・・・・・・・・・・・・・・・・513
GOTO PAGE(ページ番号) ・・・・・・・・・・・・・・・・・・・・・・・・578
GOTO RECORD({テーブル ;} レコード番号) ・・・・・・・・・・・・・・1120
GOTO SELECTED RECORD({テーブル ;} レコード位置番号) ・・・・・・1224
GOTO XY(x ; y) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・794
GRAPH SETTINGS(グラフ ; x最小値 ; x最大値 ; y最小値 ; y最大値 ; xプロップ;
xグリッド ; yグリッド ; タイトル1{; タイトル2 ; ... ; タイトルN}) ・・・・595
GRAPH TABLE {(テーブル)} または({テーブル ;} グラフ番号 ; xフィールド ;
yフィールド1{ ;...; yフィールド8}) ・・・・・・・・・・・・・・・・・・・597
GRAPH(グラフエリア ; グラフ番号 ; xラベル ; y要素1 { ;...;y要素8}) ・・・590

H

HIDE MENU BAR → 数値 ・・・・・・・・・・・・・・・・・・・・・・・761
HIDE PROCESS(プロセス) ・・・・・・・・・・・・・・・・・・・・・・963
HIDE TOOL BAR ・・・・・・・・・・・・・・・・・・・・・・・・・・・1435
HIDE WINDOW {(ウインドウ)} ・・・・・・・・・・・・・・・・・・・・1763
HIGHLIGHT RECORDS({{テーブル; }セット名}{;* })) ・・・・・・・・・1246
HIGHLIGHT TEXT(エリア; 先頭 ; 最終) ・・・・・・・・・・・・・・・・1510

I

IDLE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・402
IMPORT DATA(ファイル名 {; プロジェクト {; *}}) ・・・・・・・・・・・669
IMPORT DIF({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・・665
IMPORT SYLK({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・661
IMPORT TEXT({テーブル ;} ドキュメント) ・・・・・・・・・・・・・・657
In break → ブール ・・・・・・・・・・・・・・・・・・・・・・・・・・560
In footer → ブール ・・・・・・・・・・・・・・・・・・・・・・・・・・560
In header → ブール ・・・・・・・・・・・・・・・・・・・・・・・・・559
In transaction → ブール ・・・・・・・・・・・・・・・・・・・・・・1452
INPUT FORM ({テーブル; }フォーム{; ユーザフォーム}{; *}) ・・・・・・584
INSERT ELEMENT(配列 ; 挿入位置 ; {要素数}) ・・・・・・・・・・・・208
INSERT IN BLOB(blob ; オフセット ; 挿入数 {;フィル}) ・・・・・・・・279
INSERT LIST ITEM(リスト ; 前項目参照番号 ¦ * ; 項目テキスト ; 項目参照番号
{ ; サブリスト { ; 展開}}) ・・・・・・・・・・・・・・・・・・・・・・635

INSERT LISTBOX COLUMN ({*; }オブジェクト; 列位置; 列名; 列変数; ヘッダ名; ヘッダ変数) ・・・

Load list(リスト名) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・603
LOAD RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・・・1104
LOAD SET({テーブル ;} セット ; ドキュメント) ・・・・・・・・・・・・・1266
LOAD VARIABLES (ドキュメント ; 変数1{ ;...; 変数N}) ・・・・・・・・・1545
LOCKED ATTRIBUTES({テーブル ;} プロセス ; ユーザ ; マシン ; プロセス名)
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1108
Locked {(テーブル)} → ブール ・・・・・・・・・・・・・・・・・・・・1107
LOG EVENT(メッセージ {; 重要度 }) ・・・・・・・・・・・・・・・・・・1423
Log(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・743
LONGINT ARRAY FROM SELECTION(テーブル;レコード番号配列
{;セレクション}) ・・・・・・・・・・・・・・・・・・・・・・・・・・224
LONGINT TO BLOB(倍長整数 ; blob ; バイト順序 {;オフセット¦ *}) ・・・・261
Lowercase(文字列) → 文字列 ・・・・・・・・・・・・・・・・・・・・・1290

M

Mac to ISO(テキスト)→ 文字列 ・・・・・・・・・・・・・・・・・・・・1297
Mac to Win(テキスト)→ 文字列 ・・・・・・・・・・・・・・・・・・・・1295
Macintosh command down → ブール ・・・・・・・・・・・・・・・・・・1496
Macintosh control down → ブール ・・・・・・・・・・・・・・・・・・・1498
Macintosh option down → ブール ・・・・・・・・・・・・・・・・・・・1497
MAP FILE TYPES(Macintosh ; Windows ; コンテキスト) ・・・・・・・・・1392
Max(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・・・・・856
MAXIMIZE WINDOW {(ウインドウ)} ・・・・・・・・・・・・・・・・・・1765
Menu bar height → 数値 ・・・・・・・・・・・・・・・・・・・・・・・1411
Menu bar screen → 数値 ・・・・・・・・・・・・・・・・・・・・・・・1410
MENU BAR(メニューバー番号 {; プロセス} {; *}) ・・・・・・・・・・・・759
Menu selected → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・764
MESSAGE(メッセージ) ・・・・・・・・・・・・・・・・・・・・・・・・790
MESSAGES OFF ・・・・・・・・・・・・・・・・・・・・・・・・・・・781
MESSAGES ON ・・・・・・・・・・・・・・・・・・・・・・・・・・・・781
Method called on error → 文字列 ・・・・・・・・・・・・・・・・・・・683
Method called on event → 文字列 ・・・・・・・・・・・・・・・・・・・677
Milliseconds → 倍長整数 ・・・・・・・・・・・・・・・・・・・・・・・442
Min(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・・・・・855
MINIMIZE WINDOW {(ウインドウ参照番号)} ・・・・・・・・・・・・・・1767
Mod(数値1 ; 数値2) → 数値 ・・・・・・・・・・・・・・・・・・・・・741
Modified record {(テーブル)} → ブール ・・・・・・・・・・・・・・・・1113

Modified(フィールド) → ブール ・・・・・・・・・・・・・・・・・・・・・・・423
MODIFY RECORD({テーブル} {; }{*}) ・・・・・・・・・・・・・・・・・・・・416
MODIFY SELECTION({テーブル} {; 選択モード}{; リスト更新} {; *{; *}}) ・1239
MODIFY SUBRECORD(サブテーブル ; フォーム ; {*}) ・・・・・・・・・・・420
Month of(日付) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・431
MOVE DOCUMENT(移動元パス名 ; 移動先パス名) ・・・・・・・・・・・・1369
MOVE OBJECT({*;} オブジェクト ; 水平移動 ; 垂直移動 ;{; 水平リサイズ
{; 垂直リサイズ}}{; *}) ・・・・・・・・・・・・・・・・・・・・・・・・・・843
MOVED LISTBOX COLUMN NUMBER ({*; }オブジェクト; 旧位置; 新位置) 720
MOVED LISTBOX ROW NUMBER ({*; }オブジェクト; 旧位置; 新位置) ・・727
MULTI SORT ARRAY(ポインタ配列名; ソート配列名) ・・・・・・・・・・202
MULTI SORT ARRAY(配列{; 順序}{; 配列2; 順序2; ...; 配列N; 順序N}) ・・・202

N

New list → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・606
New process(メソッド ; スタック {; プロセス{; パラメータ{; パラメータ2; ...;
パラメータN}{; *}}}) → 数値 ・・・・・・・・・・・・・・・・・・・・・・972
NEXT PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・581
NEXT RECORD({テーブル}) ・・・・・・・・・・・・・・・・・・・・・・1228
NEXT SUBRECORD(サブテーブル) ・・・・・・・・・・・・・・・・・・・1342
Next window {(ウインドウ)} → 数値 ・・・・・・・・・・・・・・・・・1777
Nil(ポインタ) → ブール ・・・・・・・・・・・・・・・・・・・・・・・・695
NO TRACE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・705
Not(ブール値) → ブール値 ・・・・・・・・・・・・・・・・・・・・・・291
Num(表現式) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・1278

O

ODBC CANCEL LOAD ・・・・・・・・・・・・・・・・・・・・・・・・528
ODBC End selection→ブール ・・・・・・・・・・・・・・・・・・・・・526
ODBC EXECUTE(sql命令 {;バインドオブジェクト1; ...;バインドオブジェクト_n})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・524
ODBC EXPORT(ソーステーブル{; プロジェクト}{; *}) ・・・・・・・・・535
ODBC GET LAST ERROR(エラーコード;エラーテキスト;ODBCエラー;
SQL Serverエラー) ・・・・・・・・・・・・・・・・・・・・・・・・・・532
ODBC GET OPTION(オプション; 値) ・・・・・・・・・・・・・・・・・・523
ODBC IMPORT(ソーステーブル{; プロジェクト}{; *}) ・・・・・・・・・・533
ODBC LOAD RECORD{(レコード数)} ・・・・・・・・・・・・・・・・・・527
ODBC LOGIN({データエントリ{; ユーザ名{; パスワード}}}) ・・・・・・・519

ODBC LOGOUT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・521
ODBC SET OPTION (オプション; 値) ・・・・・・・・・・・・・・・・・・・522
ODBC SET PARAMETER(オブジェクト; パラメータタイプ) ・・・・・・・529
OLD RELATED MANY(フィールド) ・・・・・・・・・・・・・・・・・・・1151
OLD RELATED ONE(フィールド) ・・・・・・・・・・・・・・・・・・・・1150
Old(フィールド) → 文字列、数値、日付、ブール、時間 ・・・・・・・・・425
ON ERR CALL(エラーメソッド) ・・・・・・・・・・・・・・・・・・・・679
ON EVENT CALL(イベントメソッド {; プロセス}) ・・・・・・・・・・・673
ONE RECORD SELECT {(テーブル)} ・・・・・・・・・・・・・・・・・1245
OPEN 4D PREFERENCES(セレクタ) ・・・・・・・・・・・・・・・・・・159
OPEN DATA FILE(アクセスパス) ・・・・・・・・・・・・・・・・・・・149
Open document (ドキュメント{; ファイルタイプ {;モード}}) →
ドキュメントファイル参照番号 ・・・・・・・・・・・・・・・・・・・・1360
Open external window(左 ; 上 ; 右 ; 下 ; タイプ ; タイトル ; プラグインエリア)
→ 整数 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1756
Open form window({テーブル ;} フォーム名 {; タイプ {; 水平位置 {; 垂直位置
{;*}}}})→ ウインドウ参照番号 ・・・・・・・・・・・・・・・・・・・・1779
Open resource file(リソースファイル名 {; ファイルタイプ }) → DocRef ・1170
OPEN WEB URL(url {; *}) ・・・・・・・・・・・・・・・・・・・・・・1716
Open window(左 ; 上 ; 右 ; 下 {;タイプ} {; タイトル} {; コントロールメニュー
(Macintosh版では、クローズ) ボックス}}}) → {ウインドウ参照番号} ・・・1752
ORDER BY FORMULA(テーブル {; ソート条件式} {; ソート種別1}{ ;...;
ソート条件式N ; ソート種別N }) ・・・・・・・・・・・・・・・・・・・1029
ORDER BY({テーブル}{;フィールド1}{; ソート種別1} {;...;フィールドN ;
ソート種別N }{;*}) ・・・・・・・・・・・・・・・・・・・・・・・・・1024
ORDER SUBRECORD BY(サブテーブル;サブフィールド1{;ソート種別1}
{; ... ;サブフィールドN ;{ソート種別N }}) ・・・・・・・・・・・・・・1346
OUTPUT FORM({テーブル; }フォーム{; ユーザフォーム}) ・・・・・・・・586
Outside call → ブール ・・・・・・・・・・・・・・・・・・・・・・・562

P

PAGE BREAK {(* ¦ >)} ・・・・・・・・・・・・・・・・・・・・・・884
PAGE SETUP({テーブル ;} フォーム) ・・・・・・・・・・・・・・・・・・905
PAUSE PROCESS(プロセス) ・・・・・・・・・・・・・・・・・・・・・・981
PICTURE LIBRARY LIST(ピクチャ参照番号 ; ピクチャ名) ・・・・・・・・940
PICTURE PROPERTIES(ピクチャ; 幅 ; 高さ {; 水平オフセット
{; 垂直オフセット {; モード}}}) ・・・・・・・・・・・・・・・・・・・937

Picture size(ピクチャ) → 整数 ・・・・・・・・・・・・・・・・・・・・・・・・937
PICTURE TO BLOB(ピクチャ;ピクチャBlob;フォーマット) ・・・・・・・・931
PICTURE TO GIF(ピクチャ; GIF用BLOB) ・・・・・・・・・・・・・・・・928
PICTURE TYPE LIST(フォーマット配列{; フォーマット名配列}) ・・・・・936
PLATFORM PROPERTIES(プラットフォーム{; システム{; マシン}}) ・・1414
PLAY(サウンド名 {; チャネル}) ・・・・・・・・・・・・・・・・・・・・1480
POP RECORD({テーブル}) ・・・・・・・・・・・・・・・・・・・・・・1128
Pop up menu(項目テキスト{; デフォルト{; X座標; Y座標}}) → 数値 ・・・1500
Position(検索文字列 ; 文字列) → 数値 ・・・・・・・・・・・・・・・・1280
POST CLICK(マウスX ; マウスY {; プロセス}{; *}}) ・・・・・・・・・・1504
POST EVENT(タイプ ; メッセージ ; 時間 ; マウスX ; マウスY ; モディファイア
{; プロセス}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1505
POST KEY(コード {; モディファイア {; プロセス}}) ・・・・・・・・・・1503
PREVIOUS PAGE ・・・・・・・・・・・・・・・・・・・・・・・・・・・581
PREVIOUS RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・1230
PREVIOUS SUBRECORD(サブテーブル) ・・・・・・・・・・・・・・・1343
Print form({table;} form{; area1{; area2}})→数値 ・・・・・・・・・・・881
PRINT LABEL({テーブル} { ; ラベルドキュメント}{; *}) ・・・・・・・・・875
PRINT OPTION VALUES(オプション; 名前配列{; 情報1配列; 情報2配列}) ・898
PRINT RECORD({テーブル} {;*}) ・・・・・・・・・・・・・・・・・・・885
PRINT SELECTION({テーブル} {;} {*}) ・・・・・・・・・・・・・・・・879
PRINT SETTINGS ・・・・・・・・・・・・・・・・・・・・・・・・・・・908
PRINTERS LIST(名前配列{; 場所配列{; モデル配列}}) ・・・・・・・・・・888
Printing page → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・887
Process aborted → ブール ・・・・・・・・・・・・・・・・・・・・・・983
PROCESS HTML TAGS(入力データ; 出力データ) ・・・・・・・・・・・・1695
Process number(プロセス名{; *})→ 数値 ・・・・・・・・・・・・・・・990
PROCESS PROPERTIES(プロセス ; 名前 ; ステータス ; 時間 {; 表示
{;ユニークID{; 派生元 }}}) ・・・・・・・・・・・・・・・・・・・・・987
Process state(プロセス) → 数値 ・・・・・・・・・・・・・・・・・・・985
PUSH RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・1127

Q

QR BLOB TO REPORT(エリア;blob) ・・・・・・・・・・・・・・・・・1036
QR Count columns(エリア)→倍長整数 ・・・・・・・・・・・・・・・・1087
QR DELETE COLUMN(エリア;カラム番号) ・・・・・・・・・・・・・・1088
QR DELETE OFFSCREEN AREA(エリア) ・・・・・・・・・・・・・・・・1037

QR EXECUTE COMMAND(エリア;コマンド) ・・・・・・・・・・・・・・・1054
QR Find column(エリア; 表現式) → 倍長整数 ・・・・・・・・・・・・・・・1057
QR Get area property(エリア; プロパティ)→ 倍長整数 ・・・・・・・・・・1046
QR GET BORDERS(エリア; カラム; 行; 罫線; 線{;色}) ・・・・・・・・・・1066
QR Get command status(エリア; コマンド{; 値}) →倍長整数 ・・・・・・・1055
QR GET DESTINATION(エリア; 出力先; {詳細}) ・・・・・・・・・・・・・1040
QR Get document property(エリア; プロパティ) → 倍長整数 ・・・・・・・1042
QR Get drop column(エリア)→倍長整数 ・・・・・・・・・・・・・・・・・1086
QR GET HEADER AND FOOTER(エリア; セレクタ; 左タイトル; 中央タイトル;
右タイトル; 高さ{; ピクチャ}{; ピクチャ整列}) ・・・・・・・・・・・・・1062
QR Get HTML template(エリア)→テキスト ・・・・・・・・・・・・・・・1091
QR GET INFO COLUMN(エリア; カラム番号; タイトル; オブジェクト; 非表示;
サイズ; 同一値; 表示フォーマット) ・・・・・・・・・・・・・・・・・・1071
QR Get info row(エリア; 行) → 倍長整数 ・・・・・・・・・・・・・・・・1075
QR Get report kind(エリア) → 倍長整数 ・・・・・・・・・・・・・・・・1044
QR Get report table(エリア)→倍長整数 ・・・・・・・・・・・・・・・・1048
QR GET SELECTION(エリア; 左; 上{; 右}{; 下}) ・・・・・・・・・・・・1059
QR GET SORTS(エリア;カラム配列;{並び替え順配列}) ・・・・・・・・・1077
QR Get text property(エリア;カラム番号; 行番号; プロパティ) → 倍長整数 1051
QR GET TOTALS DATA(エリア;カラム番号; ブレーク番号; 演算子; テキスト)
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1081
QR GET TOTALS SPACING(エリア; 小計番号;値) ・・・・・・・・・・・1084
QR INSERT COLUMN(エリア;カラム番号; オブジェクト) ・・・・・・・1085
QR New offscreen area → 倍長整数 ・・・・・・・・・・・・・・・・・・1037
QR ON COMMAND(エリア;4D代替メソッド) ・・・・・・・・・・・・・・1056
QR REPORT TO BLOB(エリア;blob) ・・・・・・・・・・・・・・・・・1035
QR REPORT({テーブル; }ドキュメント{; 階層型}{; ウィザード}{; 検索}{;*}) ・1031
QR RUN(エリア) ・・・・・・・・・・・・・・・・・・・・・・・・・・1053
QR SET AREA PROPERTY(エリア;プロパティ;値) ・・・・・・・・・・・1045
QR SET BORDERS(エリア ; カラム ; 行; 罫線; 線{ ;色}) ・・・・・・・・1064
QR SET DESTINATION(エリア; 出力先; {詳細}) ・・・・・・・・・・・・1038
QR SET DOCUMENT PROPERTY(エリア;プロパティ;値) ・・・・・・・1041
QR SET HEADER AND FOOTER(エリア; セレクタ; 左タイトル; 中央タイトル;
右タイトル; 高さ{; ピクチャ}{; ピクチャ整列}) ・・・・・・・・・・・・1060
QR SET HTML TEMPLATE(エリア; テンプレート) ・・・・・・・・・・・1089
QR SET INFO COLUMN(エリア; カラム番号; タイトル; オブジェクト; 非表示;
サイズ; 同一値; 表示フォーマット) ・・・・・・・・・・・・・・・・・・1068

QR SET INFO ROW(エリア; 行; 非表示) ・・・・・・・・・・・・・・・・1074
QR SET REPORT KIND(エリア;タイプ) ・・・・・・・・・・・・・・・・1043
QR SET REPORT TABLE(エリア; テーブル) ・・・・・・・・・・・・・・1047
QR SET SELECTION(エリア;左;上;右;下) ・・・・・・・・・・・・・・1058
QR SET SORTS(エリア;カラム配列;{並び替え順配列}) ・・・・・・・・・1076
QR SET TEXT PROPERTY(エリア;カラム番号; 行番号; プロパティ; 値) ・1049
QR SET TOTALS DATA(エリア;カラム番号; ブレーク番号; 演算子) ・・・1078
QR SET TOTALS SPACING(エリア; 小計番号;値) ・・・・・・・・・・・1083
QUERY BY EXAMPLE({テーブル} {;*}) ・・・・・・・・・・・・・・・・1001
QUERY BY FORMULA({テーブル} {;} {検索用フォーミュラ}) ・・・・・・1011
QUERY SELECTION BY FORMULA({テーブル} {;} {;検索用フォーミュラ}) 1013
QUERY SELECTION({テーブル}{; 検索条件} {; *}) ・・・・・・・・・・1009
QUERY SUBRECORDS(サブテーブル ; 検索条件式) ・・・・・・・・・・1347
QUERY WITH ARRAY(インデックスフィールド; 配列) ・・・・・・・・・1014
QUERY({テーブル} {;検索条件} {; *}) ・・・・・・・・・・・・・・・1003
QUIT 4D{(time)} ・・・・・・・・・・・・・・・・・・・・・・・・・151

R

Random → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・740
Read only state {(テーブル)} → ブール ・・・・・・・・・・・・・・・1103
READ ONLY {(テーブル ¦ *)} ・・・・・・・・・・・・・・・・・・・1102
READ PICTURE FILE(ファイル名; ピクチャ) ・・・・・・・・・・・・・934
READ WRITE {(テーブル ¦ *)} ・・・・・・・・・・・・・・・・・・・1101
REAL TO BLOB(実数 ; blob ; 実数形式 {;オフセット ¦ *}) ・・・・・・・・264
RECEIVE BUFFER(受信変数) ・・・・・・・・・・・・・・・・・・・・325
RECEIVE PACKET({ドキュメントファイル参照番号 ;} 受信変数 ; 文字数 ¦
終了文字) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・322
RECEIVE RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・330
RECEIVE VARIABLE(変数) ・・・・・・・・・・・・・・・・・・・・・328
Record number {(テーブル)} → 数値 ・・・・・・・・・・・・・・・・1119
Records in selection{(テーブル)} → 数値 ・・・・・・・・・・・・・・1220
Records in set(セット) → 数値 ・・・・・・・・・・・・・・・・・・1263
Records in subselection(サブテーブル) → 数値 ・・・・・・・・・・・1338
Records in table {(テーブル)} → 数値 ・・・・・・・・・・・・・・・1118
REDRAW LIST(リスト) ・・・・・・・・・・・・・・・・・・・・・・・613
REDRAW WINDOW {(ウインドウ参照番号)} ・・・・・・・・・・・・・・1760
REDRAW(オブジェクト) ・・・・・・・・・・・・・・・・・・・・・・1513

REDUCE SELECTION({テーブル ;} レコード数) ・・・・・・・・・・・・1242
REGISTER CLIENT(クライアント名 {; ピリオド } {; *}) ・・・・・・・・・・996
REJECT {(フィールド)} ・・・・・・・・・・・・・・・・・・・・・・・・514
RELATE MANY SELECTION(フィールド) ・・・・・・・・・・・・・・・1153
RELATE MANY(1テーブル ¦ 1フィールド) ・・・・・・・・・・・・・・1146
RELATE ONE SELECTION(nテーブル ; 1テーブル) ・・・・・・・・・・・1152
RELATE ONE(nテーブル ¦ nフィールド {; 選択フィールド}) ・・・・・・・1142
REMOVE FROM SET({テーブル ;} セット) ・・・・・・・・・・・・・・・1260
REMOVE PICTURE FROM LIBRARY(ピクチャ参照番号) ・・・・・・・・・947
Replace string(元の文字列 ; 対象文字列 ; 置き換え文字列 {; 回数}) → 文字列
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1294
Request(メッセージ {; デフォルト応答 {; OKボタンタイトル
{; キャンセルボタンタイトル}}}) → 文字列 ・・・・・・・・・・・・・・787
RESIZE FORM WINDOW (幅; 高さ) ・・・・・・・・・・・・・・・・・・1783
RESOLVE ALIAS(エイリアスパス; ターゲットパス) ・・・・・・・・・・・1385
RESOLVE POINTER(ポインタ ; 変数名 ; テーブル番号 ; フィールド番号) ・693
RESOURCE LIST(リソースタイプ ; リソースID ; リソース名
{; リソースファイル}) ・・・・・・・・・・・・・・・・・・・・・・・1180
RESOURCE TYPE LIST(リソースタイプ {; リソースファイル}) ・・・・・1178
RESTORE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・229
RESUME PROCESS(プロセス) ・・・・・・・・・・・・・・・・・・・・982
Right click → ブール値 ・・・・・・・・・・・・・・・・・・・・・・567
Round(数値 ; 桁位置) → 数値 ・・・・・・・・・・・・・・・・・・・738

S

SAVE LIST(リスト ; リスト名) ・・・・・・・・・・・・・・・・・・・・605
SAVE PICTURE TO FILE(ドキュメント参照番号 ; ピクチャ) ・・・・・・927
SAVE RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・・1115
SAVE RELATED ONE(フィールド) ・・・・・・・・・・・・・・・・・・1149
SAVE SET(セット ; ドキュメント) ・・・・・・・・・・・・・・・・・1264
SAVE VARIABLES (ドキュメント ; 変数1{ ;...; 変数N}) ・・・・・・・・・1543
SAX ADD PROCESSING INSTRUCTION(document; statement) ・・・・1845
SAX ADD XML CDATA(document; data) ・・・・・・・・・・・・・・・1842
SAX ADD XML COMMENT(document; comment) ・・・・・・・・・・・1832
SAX ADD XML DOCTYPE(document; docType) ・・・・・・・・・・・1830
SAX ADD XML ELEMENT VALUE(document;data) ・・・・・・・・・・1840
SAX CLOSE XML ELEMENT(document) ・・・・・・・・・・・・・・・1837

SAX GET XML CDATA(document; value) ・・・・・・・・・・・・・・・1844
SAX GET XML COMMENT(document; comment) ・・・・・・・・・・・・1833
SAX GET XML DOCUMENT VALUES(document; encoding; version, standalone)
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1829
SAX GET XML ELEMENT (document; name; prefix; attrNames; attrValues) 1838
SAX GET XML ELEMENT VALUE(document;value) ・・・・・・・・・・・1841
SAX GET XML ENTITY(document; name; value) ・・・・・・・・・・・1847
SAX Get XML node(document) → 倍長整数 ・・・・・・・・・・・・・1831
SAX GET XML PROCESSING INSTRUCTION(document; name; value) ・1846
SAX OPEN XML ELEMENT ARRAYS(document; tag{; attribNamesArray;
attribValuesArray}{;attribNamesArray2; attribValuesArray2; ...;
attribNamesArrayN; attribValuesArrayN}) ・・・・・・・・・・・・・・・1835
SAX OPEN XML ELEMENT(document; tag{; attribName; attribValue}...
{;attribNameX;attribValueX}) ・・・・・・・・・・・・・・・・・・・・1834
SAX SET XML OPTIONS(document; encoding{; standalone{; indentation}}) 1828
SCAN INDEX(フィールド ; レコード数 {; 方向}) ・・・・・・・・・・・・1244
SCREEN COORDINATES(左 ; 上 ; 右 ; 下 { ; 画面}) ・・・・・・・・・・1407
SCREEN DEPTH(深さ ; カラー { ; 画面}) ・・・・・・・・・・・・・・1408
Screen height {(*)} → 数値 ・・・・・・・・・・・・・・・・・・・・1405
Screen width {(*)} → 数値 ・・・・・・・・・・・・・・・・・・・・1406
SCROLL LINES({*; }オブジェクト{; 位置}{;* }) ・・・・・・・・・・・1507
Secured Web connection →ブール ・・・・・・・・・・・・・・・・・1713
Select document(ディレクトリ; ファイルタイプ; タイトル; オプション
{; 選択項目})→文字列 ・・・・・・・・・・・・・・・・・・・・・・・・1371
Select folder({メッセージ }) → 文字列 ・・・・・・・・・・・・・・・1379
SELECT LIST ITEMS BY POSITION(リスト; 項目位置{; 位置配列}) ・・・・652
SELECT LIST ITEMS BY REFERENCE(リスト; 項目参照{; 参照配列}) ・・655
SELECT LISTBOX ROW ({*; }オブジェクト; 位置{; アクション}) ・・・・・721
SELECT LOG FILE(ログファイル¦ *) ・・・・・・・・・・・・・・・・・153
Selected list items(リスト{; 項目配列}{; *}) → 倍長整数 ・・・・・・・649
Selected record number {(テーブル)} → 数値 ・・・・・・・・・・・・1223
SELECTION RANGE TO ARRAY(開始 ; 終了 ; フィールドまたはテーブル ;
配列 ; {フィールド2またはテーブル2 ; 配列2...;フィールドNまたはテーブルN ;
配列N}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・217
SELECTION TO ARRAY(フィールド¦ テーブル; 配列{; フィールド2¦ テーブル2;
配列2; ...; フィールドN¦ テーブルN; 配列N}) ・・・・・・・・・・・・・215
Self → ポインタ ・・・・・・・・・・・・・・・・・・・・・・・・・・692

Semaphore(セマフォ{;待ち時間}) → ブール ・・・・・・・・・・・・・・・・949
SEND HTML BLOB(blob;タイプ {; 非コンテキスト}) ・・・・・・・・・・1691
SEND HTML FILE(HTMLファイル) ・・・・・・・・・・・・・・・・・1688
SEND HTML TEXT(htmlテキスト{;非コンテキスト}) ・・・・・・・・・・1694
SEND HTTP RAW DATA(データ{;*}) ・・・・・・・・・・・・・・・・1707
SEND HTTP REDIRECT(url {; *}) ・・・・・・・・・・・・・・・・・1710
SEND PACKET({ドキュメントファイル参照番号 ;} パケット) ・・・・・・320
SEND RECORD {(テーブル)} ・・・・・・・・・・・・・・・・・・・329
SEND SOAP FAULT(faultタイプ; 説明) ・・・・・・・・・・・・・・・1739
SEND VARIABLE(変数) ・・・・・・・・・・・・・・・・・・・・・・327
Sequence number {(テーブル)} → 数値 ・・・・・・・・・・・・・・・1121
SET ABOUT(アイテム ; メソッド) ・・・・・・・・・・・・・・・・・763
SET ALIGNMENT({*; }オブジェクト) → 数値 ・・・・・・・・・・・・849
SET ALLOWED METHODS (メソッド配列) ・・・・・・・・・・・・・・599
SET AUTOMATIC RELATIONS(1リレート {; nリレート}) ・・・・・・・1134
SET BLOB SIZE(blob ; サイズ {;フィラー}) ・・・・・・・・・・・・238
SET CGI EXECUTABLE(url1{; url2}) ・・・・・・・・・・・・・・・・1714
SET CHANNEL(ポート ¦ 処理 {;セットアップ ¦ ドキュメント}) ・・・・・・311
SET CHOICE LIST({*;} オブジェクト ; リスト) ・・・・・・・・・・・830
SET COLOR({*; }オブジェクト; カラー{; 代替色}) ・・・・・・・・・・836
SET CURRENT PRINTER(プリンタ名) ・・・・・・・・・・・・・・・・889
SET CURSOR {(カーソル)} ・・・・・・・・・・・・・・・・・・・・1511
SET DATABASE PARAMETER({テーブル;} セレクタ;値) ・・・・・・・1321
SET DEFAULT CENTURY(世紀 { ; ピボット年度}) ・・・・・・・・・・443
SET DICTIONARY(辞書) ・・・・・・・・・・・・・・・・・・・・・1442
SET DOCUMENT CREATOR (ドキュメント ; ファイルクリエータ) ・・・1359
SET DOCUMENT POSITION(ドキュメント参照番号 ; オフセット {; アンカー})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1404
SET DOCUMENT PROPERTIES(ドキュメント ; ロック ; 非表示 ; 作成日 ;
作成時間 ; 更新日 ; 更新時間) ・・・・・・・・・・・・・・・・・・1400
SET DOCUMENT SIZE(ドキュメント ; サイズ) ・・・・・・・・・・・・1403
SET DOCUMENT TYPE (ドキュメント ; ファイルタイプ) ・・・・・・・1358
SET ENTERABLE({*;} 入力エリア; 入力可) ・・・・・・・・・・・・・831
SET ENVIRONMENT VARIABLE(変数名; 変数値) ・・・・・・・・・・・1425
SET FIELD RELATION (nテーブル ¦ 始点フィールド; 1対1; 1対n}) ・・・1136
SET FIELD TITLES(テーブル ¦ サブテーブル ; フィールドタイトル ;
フィールド番号) ・・・・・・・・・・・・・・・・・・・・・・・・1490

SET FILTER({*;} オブジェクト ; 入力フィルタ) ・・・・・・・・・・・・・828
SET FORM HORIZONTAL RESIZING (リサイズ{; 最小幅{; 最大幅}}) ・・・575
SET FORM SIZE ({オブジェクト; }水平; 垂直{;* }) ・・・・・・・・・・・571
SET FORM VERTICAL RESIZING (リサイズ{; 最小高さ{; 最大高さ}}) ・・・576
SET FORMAT({*;} オブジェクト ; 表示フォーマット) ・・・・・・・・・・819
Set group properties(グループID ; グループ名 ; オーナー {; メンバーID}) → 数値
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1537
SET HOME PAGE(ホームページ) ・・・・・・・・・・・・・・・・・・・1687
SET HTML ROOT(パス名HTML) ・・・・・・・・・・・・・・・・・・・1683
SET HTTP HEADER(ヘッダー ¦ フィールド{; 数値}) ・・・・・・・・・・1700
SET INDEX(フィールド; インデックス{; モード}{; *}) ・・・・・・・・・1317
SET LIST ITEM PROPERTIES(リスト; 項目参照 ¦ *; 入力可; スタイル; アイコン
{; カラー}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・636
SET LIST ITEM(リスト; 項目参照 ¦ *; 新規項目テキスト; 新規項目参照
{; サブリスト; 展開}) ・・・・・・・・・・・・・・・・・・・・・・・647
SET LIST PROPERTIES(リスト;表示形式{;アイコン{;線の高さ{;ダブルクリック
{; 複数選択可{; 編集可}}}}}) ・・・・・・・・・・・・・・・・・・・・614
SET LISTBOX COLUMN WIDTH ({*; }オブジェクト; 幅) ・・・・・・・・・718
SET LISTBOX GRID COLOR{*; }オブジェクト; カラー; 横線; 縦線) ・・・・732
SET LISTBOX ROWS HEIGHT({*; }オブジェクト; 高さ) ・・・・・・・・・725
SET MENU ITEM KEY(メニュー ; メニュー項目 ; キー {; プロセス}) ・・・774
SET MENU ITEM MARK(メニュー ; メニュー項目 ; マーク {; プロセス}) ・772
SET MENU ITEM STYLE(メニュー ; メニュー項目 ; スタイル{; プロセス}) 770
SET MENU ITEM(メニュー番号 ; メニュー項目番号 ; アイテムテキスト
{; プロセス}) ・・・・・・・・・・・・・・・・・・・・・・・・・・・768
SET PICTURE RESOURCE(リソースID ; リソースデータ {; リソースファイル})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1192
SET PICTURE TO CLIPBOARD(ピクチャ) ・・・・・・・・・・・・・・・306
SET PICTURE TO LIBRARY(ピクチャ ; ピクチャ参照番号 ;ピクチャ名) ・944
SET PLATFORM INTERFACE(インタフェース) ・・・・・・・・・・・・1483
SET PLUGIN ACCESS (プラグイン; グループ) ・・・・・・・・・・・・1540
SET PRINT MARKER(マーカー番号; 位置{;*}) ・・・・・・・・・・・・910
SET PRINT OPTION(オプション; 値1{; 値2}) ・・・・・・・・・・・・892
SET PRINT PREVIEW(プレビュー) ・・・・・・・・・・・・・・・・・909
SET PRINTABLE MARGIN(左; 上; 右; 下) ・・・・・・・・・・・・・・917
SET PROCESS VARIABLE(プロセス ; 送信先変数 ; ソース式 { ; 送信先変数2 ;
ソース式2 ; ...; 送信先変数N ; ソース式N}) ・・・・・・・・・・・・・958

SET QUERY DESTINATION(配置先タイプ {; 配置先オブジェクト}) ・・・1015
SET QUERY LIMIT(制限) ・・・・・・・・・・・・・・・・・・・・・・・1021
SET REAL COMPARISON LEVEL(イプシロン値) ・・・・・・・・・・・・748
SET RESOURCE NAME(リソースタイプ ; リソースID ; リソース名
{; リソースファイル}) ・・・・・・・・・・・・・・・・・・・・・・・1202
SET RESOURCE PROPERTIES(リソースタイプ ; リソースID ; リソース属性
{; リソースファイル}) ・・・・・・・・・・・・・・・・・・・・・・・1204
SET RESOURCE(リソースタイプ ; リソースID ; リソースデータ
{; リソースファイル}) ・・・・・・・・・・・・・・・・・・・・・・・1197
SET RGB COLORS({*; }オブジェクト; 描画色; 背景色{; 代替背景色}) ・・・838
SET SCREEN DEPTH(深さ ; カラー { ; 画面}) ・・・・・・・・・・・・1410
SET STRING RESOURCE(リソースID ; リソースデータ {; リソースファイル})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1188
SET TABLE TITLES(テーブルタイトル ; テーブル番号) ・・・・・・・・1485
SET TEXT RESOURCE(リソースID ; リソースデータ {; リソースファイル})
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1190
SET TEXT TO CLIPBOARD(テキスト) ・・・・・・・・・・・・・・・・・307
SET TIMEOUT(秒) ・・・・・・・・・・・・・・・・・・・・・・・・・316
SET TIMER(チック数) ・・・・・・・・・・・・・・・・・・・・・・・565
Set user properties(ユーザID; ユーザ名; 起動メソッド; パスワード;
ログイン回数; 前回ログイン{; メンバー{; グループオーナ}}) →数値 ・・・1531
SET VISIBLE({* ;} オブジェクト ; 表示) ・・・・・・・・・・・・・・・833
SET WEB DISPLAY LIMIT(レコード数 { ; ページ数 { ; ピクチャ参照番号}}）
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・1684
SET WEB SERVICE OPTION(オプション; 値) ・・・・・・・・・・・・1722
SET WEB SERVICE PARAMETER(名前; 値{; soapタイプ}) ・・・・・・1720
SET WEB TIMEOUT(タイムアウト) ・・・・・・・・・・・・・・・・・1682
SET WINDOW RECT(左 ; 上 ; 右 ; 下 {; ウインドウ}) ・・・・・・・・・1775
SET WINDOW TITLE(タイトル {; ウインドウ}) ・・・・・・・・・・・・1769
SET XSLT PARAMETER(パラメータ名; パラメータ値) ・・・・・・・・・1851
Shift down → ブール ・・・・・・・・・・・・・・・・・・・・・・・1493
SHOW LISTBOX GRID ({*; }オブジェクト; 横線; 縦線) ・・・・・・・・・731
SHOW LISTBOX SCROLLBAR({*; }オブジェクト; 横; 縦) ・・・・・・・・733
SHOW MENU BAR → 数値 ・・・・・・・・・・・・・・・・・・・・・・762
SHOW ON DISK(パス名{; *}) ・・・・・・・・・・・・・・・・・・・・1376
SHOW PROCESS(プロセス) ・・・・・・・・・・・・・・・・・・・・・964
SHOW TOOL BAR ・・・・・・・・・・・・・・・・・・・・・・・・・・1435

SHOW WINDOW {(ウインドウ)} ・・・・・・・・・・・・・・・・・・・・・・1764
Sin(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・745
Size of array(配列) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・199
SOAP DECLARATION(変数; タイプ; 入出力{; エイリアス}) ・・・・・・・・1736
SORT ARRAY(配列1{ ;...; 配列N} ; {ソート種別}) ・・・・・・・・・・・・200
SORT LIST(リスト {; > または <}) ・・・・・・・・・・・・・・・・・・・625
SORT LISTBOX COLUMNS ({*; }オブジェクト; 列変数; 順序{; 列変数2; 順序2}...
{; 列変数X; 順序X) ・・・・・・・・・・・・・・・・・・・・・・・・・・・717
SPELL CHECKING ・・・・・・・・・・・・・・・・・・・・・・・・・・・・1441
Square root(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・742
START TRANSACTION ・・・・・・・・・・・・・・・・・・・・・・・・・1450
START WEB SERVER ・・・・・・・・・・・・・・・・・・・・・・・・・1681
Std deviation(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・857
STOP WEB SERVER ・・・・・・・・・・・・・・・・・・・・・・・・・・1682
STRING LIST TO ARRAY(リソースID ; ストリング {; リソースファイル}) 1182
String(表現式 {; フォーマット}) → 文字列 ・・・・・・・・・・・・・・・1275
Structure file→文字列 ・・・・・・・・・・・・・・・・・・・・・・・・・137
Substring(文字列 ; 先頭文字位置 {; 文字数}) → 文字列 ・・・・・・・・・・1281
Subtotal(データ {; ページブレーク}) → 数値 ・・・・・・・・・・・・・・901
Sum squares(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・・859
Sum(数値フィールド) → 数値 ・・・・・・・・・・・・・・・・・・・・・・853
System folder→文字列 ・・・・・・・・・・・・・・・・・・・・・・・・1418

T

Table name(テーブル番号 ¦ テーブルポインタ) → 文字列 ・・・・・・・・1306
Table(テーブル番号 ¦ ポインタ) → ポインタ ¦ 数値 ・・・・・・・・・・・1308
Tan(数値) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・746
Temporary folder→ 文字列 ・・・・・・・・・・・・・・・・・・・・・・1419
Test clipboard(データタイプ) → 数値 ・・・・・・・・・・・・・・・・・・308
Test path name(パス名) → 数値 ・・・・・・・・・・・・・・・・・・・・1375
Test semaphore(セマフォ) → ブール ・・・・・・・・・・・・・・・・・・952
TEXT TO BLOB(テキスト ; blob ; テキスト形式 {;オフセット ¦ *}) ・・・・267
Tickcount → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・441
Time string(秒) → 文字列 ・・・・・・・・・・・・・・・・・・・・・・・439
Time(時間文字列) → 時間 ・・・・・・・・・・・・・・・・・・・・・・・440
TRACE ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・703
Trigger level → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・1468

TRIGGER PROPERTIES(トリガレベル ; イベント ; テーブル番号 ; レコード番号)
・・・

Y

Year of(日付) → 数値 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・433

その他

文字列参照[[文字位置]] → 文字列(1バイト) ・・・・・・・・・・・・・・・・1287

1982