

4th Dimension

ランゲージリファレンス
Windows® and Mac™ OS



ACI

4th Dimension ランゲージリファレンス Windows® and Mac™ OS

Copyright© 1994 - 2000 ACI 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ACI 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、ACIロゴ、およびその他のACI製品の名称は、ACI 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。

序章	29
	マニュアル全般について	30
	このマニュアルについて	31
	表記方法について	31
第1章	はじめに	33
	プログラミング言語とは何か	33
	プログラミング言語の使用目的	34
	データ制御の実施	35
	従来のコンピュータ言語との相違点	36
	メソッドはプログラミング言語の入口	37
	オブジェクトメソッド入門	37
	フォームメソッドを使ってフォームを制御する	40
	トリガを使ってデータベースの規則を制御する	41
	データベース全体でプロジェクトメソッドを使用する	41
	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	41
	データベースを作成する	42
	アプリケーションを構築する	43
第2章	プログラミング言語の構成要素	45
	データタイプ	46
	演算子	47
	式	48
	変数	51
	変数の作成	52
	変数にデータを代入する	52
	ローカル変数、プロセス変数、インタープロセス変数	53
	フォーム上のオブジェクトと変数	55
	システム変数	55
第3章	メソッドの使用方法	57
	メソッドの種類	58
	4th Dimensionバージョン3との互換性	59
	メソッドの例と用語	61
	制御フロー	63
	シーケンス構造	63

	分岐構造	63
	If...Else...End if構造	64
	case of...Else...End case構造	65
	ループ構造	66
	While...End whileループ	67
	Repeat...Untilループ	68
	For...End forループ	68
第4章	フォームメソッドとオブジェクトメソッド	75
	フォームを制御する	76
	オブジェクトメソッドを使用する	77
	オブジェクトメソッドとデータ入力	78
	オブジェクトメソッドとインタフェースオブジェクト ...	79
	ボタン	80
	スクロールエリアとドロップダウンリストボックス	81
	サーモメータ、ルーラ、ダイアル	84
	グラフエリア	84
	プラグインエリア	84
	オブジェクトメソッドとレポート出力	85
第5章	フォームイベント	87
	フォームイベントのチェック	88
	フォームイベントの一般的な規則	89
	フォームイベント	90
	データ入力の場合	90
	テーブルのサブフォーム	91
	サブテーブルのサブフォーム	92
	「ユーザ」モードのレコード表示	93
	MODIFY SELECTIONコマンドとDISPLAY SELECTIONコマンド...	93
	フォームによる書き出し実行時	94
	フォームによる読み込み実行時	94
	フォームレポート	94
第6章	プロジェクトメソッドとデータベースメソッド..	97
	プロジェクトメソッド	97
	メニューメソッド	98
	サブルーチン（メソッドから起動されるメソッド）	99
	再帰プロジェクトメソッド	103
	データベースメソッド	105
	「On Startup」データベースメソッド	106
	4th Dimensionバージョン3との互換性	106
	「On Exit」データベースメソッド	107
第7章	データベースアプリケーション	113
	カスタムメニューの例	115
	ユーザの認識	115

メニュー表示の裏側	118
「ユーザ」モードとアプリケーションの比較	119
「ユーザ」モードにおけるデータベースの使用	120
組み込みエディタを使ったアプリケーションの使用	121
高度に自動化されたアプリケーション	123
完全に自動化されたアプリケーション	123
「ユーザ」モードメニューと同等のコマンド	124
アプリケーション開発用ツール	125
開発用ツール	125
4D プラグイン	125

第8章	デバッグ	127
	なぜデバッグを使用するか?	127
	入力エラー	127
	シンタックス (構文) エラー	127
	環境エラー	128
	設計エラーとロジックエラー	129
	実行時エラー	129
	エラーが発生した場合の対応	130
	「シンタックスエラー」ウィンドウ	131
	デバッグ	132
	「実行コントロール」ツールバーボタン	134
	「実行コントロール」ツールバーについて	137
	「デバッグ」ウィンドウの各エリア	137
	「デフォルト表現式 / 値」エリア	139
	「メソッド連鎖」エリア	144
	「カスタム表現式 / 値」エリア	145
	「ソースコード」エリア	148
	ブレークポイント	151
	ブレークポイントの編集	152
	ブレークリスト	153
	コマンドの中断	155
	「ブレークポイント」エリア	159
	Tips	160
	デバッグのショートカット	161
	表示されないプロセスやコードを実行していないプロセス のトレース (4th Dimensionのみ)	163

第9章	配列とポインタ	165
	配列	165
	配列を使用する	165
	2次元配列を使用する	166
	ローカル配列、プロセス配列、インタープロセス配列 ..	166
	配列を表示する	167
	グループ化したスクロールエリアを使用する	170

	ポインタ	172
	ポインタの使用例	173
	ボタンに対するポインタの使用	174
	MyTableに対するポインタの使用	175
	フィールドに対するポインタの使用	175
	配列要素に対するポインタの使用	175
	配列に対するポインタの使用	176
	ポインタ配列の使用	176
	ポインタを使用したボタンの設定	177
	メソッドに対するポインタの受け渡し	178
	ポインタに対するポインタ	179
第10章	プロセス	181
	プロセスの作成と消去	182
	プロセスの要素	183
	インタフェース要素	183
	データ要素	183
	ランゲージ要素	184
	ユーザプロセス	185
	4th Dimensionによって作成されるプロセス	185
	グローバルプロセスとローカルプロセス	186
	プロセス間のレコードのロック	186
第11章	プログラミング言語の定義	187
	識別子	187
	テーブル	187
	フィールド	188
	サブテーブル	188
	サブフィールド	189
	インタープロセス変数	189
	プロセス変数	189
	ローカル変数	190
	配列	190
	フォーム	192
	メソッド	193
	プラグインコマンド (プラグイン、関数、エリア)	194
	セット	194
	命名セレクション	195
	プロセス	196
	名前規則のまとめ	197
	名前が重複する場合	198
	データタイプ	199
	文字列	199
	数値	200
	日付	200
	時間	201

ブール	201
ピクチャ	201
ポインタ	201
BLOB	202
配列	202
サブテーブル	202
未定義	202
データタイプの変換	202
定数	204
定義済定数	204
リテラル定数	205
演算子	208
優先順位	208
代入演算子	208
文字列演算子	209
数値演算子	209
日付演算子	210
時間演算子	210
比較演算子	211
論理演算子	216
ピクチャ演算子	217
ビットワイズ演算子	219

第12章	コマンドと引数	223
	コマンド定義	223
	シンタックス (構文)	225
	引数 (パラメータ)	225
	引数 (パラメータ) の指定	226

第13章	4D環境コマンド	227
	QUIT 4D	228
	FLUSH BUFFERS	229
	SELECT LOG FILE	230
	ADD DATA SEGMENT	231
	PLATFORM PROPERTIES	232
	ACI folder	236
	Structure file	238
	Data file	239
	Application file	241
	Compiled application	242
	Application version	243
	Application type	244
	Version type	245
	DATA SEGMENT LIST	245

第14章	配列コマンド	247
	配列を作成する	248
	配列および4D言語のその他のエリア	249
	配列とフォームオブジェクト	250
	例：ドロップダウンリストを作成する	251
	フォーム上で「arSalaries」ドロップダウンリストを作成する	251
	配列を初期化する	251
	選択した値の[従業員]給与フィールドへのレポート	253
	配列のサイズを取得する	254
	配列の要素を並べ替える	254
	要素の追加と削除	254
	配列内のクリックの処理：選択した要素のテスト	255
	選択した要素を設定する	256
	配列内の値を検索する	256
	コンボボックスの扱い	257
	グループ化されたスクロールエリア	258
	配列と4D言語	261
	ローカル配列、プロセス配列、インタープロセス配列	261
	配列の引数（パラメータ）としての受け渡し	262
	配列の他の配列への割り当て	262
	配列とポインタ	263
	配列の要素ゼロを使用する	265
	2次元配列	267
	配列とメモリ	268
	配列コマンド	270
	ARRAY BOOLEAN	271
	ARRAY DATE	271
	ARRAY INTEGER	271
	ARRAY LONGINT	271
	ARRAY PICTURE	271
	ARRAY POINTER	271
	ARRAY REAL	271
	ARRAY TEXT	271
	ARRAY STRING	271
	SORT ARRAY	273
	COPY ARRAY	275
	INSERT ELEMENT	276
	DELETE ELEMENT	277
	Find in array	278
	Size of array	279
	LIST TO ARRAY	280
	ARRAY TO LIST	281
	SELECTION TO ARRAY	282
	ARRAY TO SELECTION	283
	DISTINCT VALUES	284

	SELECTION RANGE TO ARRAY	286
第15章	BLOBコマンド	289
	BLOBの定義	289
	BLOBとメモリ	289
	BLOBの表示	290
	BLOBフィールド	290
	引数 (パラメータ) の受け渡し、ポインタおよび関数の結果 ..	290
	BLOBの割り当て	291
	BLOBの内容のアドレス指定	291
	BLOBの4th Dimensionコマンド	291
	SET BLOB SIZE	294
	BLOB size	295
	COMPRESS BLOB	295
	EXPAND BLOB	297
	BLOB PROPERTIES	299
	DOCUMENT TO BLOB	301
	BLOB TO DOCUMENT	303
	VARIABLE TO BLOB	305
	BLOB TO VARIABLE	308
	LIST TO BLOB	309
	BLOB to list	310
	INTEGER TO BLOB	312
	LONGINT TO BLOB	314
	REAL TO BLOB	316
	TEXT TO BLOB	318
	BLOB to integer	320
	BLOB to longint	322
	BLOB to real	323
	BLOB to text	325
	INSERT IN BLOB	327
	DELETE FROM BLOB	328
	COPY BLOB	328
	BOOLEAN ARRAY FROM SET	329
	LONGINT ARRAY FROM SELECTION	329
第16章	ブールコマンド	331
	True	332
	False	332
	Not	332
第17章	クリップボードコマンド	333
	APPEND TO CLIPBOARD	334
	CLEAR CLIPBOARD	340
	GET CLIPBOARD	341
	GET PICTURE FROM CLIPBOARD	343
	Get text from clipboard	344

	SET PICTURE TO CLIPBOARD	345
	SET TEXT TO CLIPBOARD	346
	Test clipboard	347
第18章	通信コマンド	349
	SET CHANNEL	350
	SEND PACKET	355
	RECEIVE PACKET	357
	SET TIMEOUT	360
	RECEIVE BUFFER	361
	SEND RECORD	363
	RECEIVE RECORD	364
	SEND VARIABLE	368
	RECEIVE VARIABLE	369
	USE ASCII MAP	370
第19章	コンパイラコマンド	371
	コンパイルするコードの記述に関する一般規則	372
	C_BLOB	374
	C_BOOLEAN	375
	C_DATE	376
	C_GRAPH	377
	C_INTEGER	378
	C_LONGINT	379
	C_PICTURE	380
	C_POINTER	381
	C_REAL	382
	C_STRING	383
	C_TEXT	384
	C_TIME	385
	IDLE	386
第20章	データ入力コマンド	387
	データ入力時にカレントレコードを変更する	388
	ADD RECORD	389
	MODIFY RECORD	391
	ADD SUBRECORD	392
	MODIFY SUBRECORD	393
	DIALOG	394
	Modified	396
	Old	398
第21章	日付関数と時間関数	399
	Current date	400
	Day of	402
	Month of	403

	Year of	405
	Day number	405
	Add to date	407
	Date	408
	SET DEFAULT CENTURY	409
	Current time	410
	Time string	411
	Time	412
	Tickcount	412
	Milliseconds	413
第22章	ドラッグ&ドロップコマンド	415
	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 ..	415
	ドラッグ&ドロップのユーザインタフェース処理	416
	ドラッグ&ドロップのプログラムによる処理	419
	ドラッグ&ドロップのコマンド	421
	Drop position	423
	DRAG AND DROP PROPERTIES	424
第23章	入力制御コマンド	431
	ACCEPT	431
	CANCEL	432
	Keystroke	433
	FILTER KEYSTROKE	438
	GOTO AREA	445
	REJECT	446
第24章	フォームイベント関数	449
	From event	450
	イベントとメソッド	451
	バージョン6とバージョン3の間の互換性について	457
	Before	469
	During	469
	After	470
	In header	470
	In break	471
	In footer	471
	Activated	472
	Deactivate	473
	Outside call	473
	SET TIMER	474
	Get edited text	475

第25章	フォームページコマンド	477
	FIRST PAGE	478
	LAST PAGE	478
	NEXT PAGE	479
	PREVIOUS PAGE	479
	GOTO PAGE	480
	Current from page	481
第26章	グラフコマンド	483
	GRAPH	484
	GRAPH SETTINGS	488
	GRAPH TABLE	489
第27章	階層リストコマンド	493
	Load list	494
	SAVE LIST	495
	New list	496
	Copy list	497
	CLEAR LIST	497
	Count list items	499
	Is a list	500
	REDRAW LIST	501
	SET LIST PROPERTIES	502
	GET LIST PROPERTIES	510
	SORT LIST	512
	APPEND TO LIST	514
	INSERT LIST ITEM	521
	SET LIST PROPERTIES	522
	GET LIST PROPERTIES	524
	List item position	525
	List item parent	526
	DELETE LIST ITEM	528
	GET LIST ITEM	529
	SET LIST ITEM	531
	Selected list item	532
	SELECT LIST ITEM	534
	SELECT LIST ITEM BY PEFERENCE	536
第28章	データ読み込みとデータ書き出しコマンド	537
	IMPORT TEXT	537
	EXPORT TEXT	539
	IMPORT SYLK	541
	EXPORT SYLK	543
	IMPORT DIF	545

	EXPORT DIF	547
	IMPORT DATA	549
	EXPORT DATA	551
第29章	割り込みコマンド	553
	ON EVENT CALL	553
	FILTER EVENT	557
	ON ERR CALL	558
	ABORT	561
第30章	ランゲージコマンド	563
	TRACE	564
	NO TRACE	566
	Count parameters	567
	Type	569
	Self	572
	RESOLVE POINTER	573
	Nil	575
	Is a variable	575
	Get pointer	576
	EXECUTE	577
	Command name	578
	コマンド名によるコマンド	581
	コマンド番号によるコマンド	595
第31章	算術関数	609
	Abs	610
	Int	610
	Dec	611
	Round	612
	Trunc	613
	Random	614
	Mod	615
	Square root	616
	Log	617
	Exp	618
	三角関数	619
	Sin	619
	Cos	620
	Tan	620
	Arctan	621
	SET REAL COMPARISON LEVEL	622
	Euro converter	623
	実数の表示について	625

第32章	メニューコマンド	627
	メニューの管理	627
	メニューバー	627
	メニュー番号とメニューコマンド番号	628
	関連メニューバー	629
	プログラムからメニューアイテムを修正する	629
	MENU BAR	631
	HIDE MENU BAR	633
	SET ABOUT	634
	Menu selectioned	635
	Count menus	637
	Count menu items	637
	Get menu title	638
	Get menu item	638
	SET MENU ITEM	639
	Get menu item style	640
	SET MENU ITEM STYLE	641
	Get menu item mark	642
	SET MENU ITEM MARK	643
	Get menu item key	644
	SET MENU ITEM KEY	645
	DISABLE MENU ITEM	646
	ENABLE MENU ITEM	646
	APPEND MENU ITEM	647
	INSERT MENU ITEM	649
	DELETE MENU ITEM	650
第33章	メッセージコマンド	651
	MESSAGES ON	652
	MESSAGES OFF	652
	ALERT	653
	CONFIRM	655
	Request	657
	MESSAGE	659
	GOTO XY	663
第34章	命名セレクションコマンド	665
	命名セレクションとセット	666
	命名セレクションの例	667
	COPY NAMED SELECTION	668
	CUT NAMED SELECTION	669
	USE NAMED SELECTION	670
	CLEAR NAMED SELECTION	671
	CREATE SELECTION FROM ARRAY	671

第35章	オブジェクトプロパティコマンド	673
	FONT	675
	FONT SIZE	676
	FONT STYLE	677
	ENABLE BUTTON	678
	DISABLE BUTTON	679
	BUTTON TEXT	680
	SET FORMAT	682
	SET FILTER	684
	SET CHOICE LIST	686
	SET ENTERABLE	687
	SET VISIBLE	688
	SET COLOR	690
	SET RGB COLORS	692
	GET OBJECT RECT	696
	MOVE OBJECT	697
第36章	リスト(統計)上の関数	699
	フィールドを使用する	699
	Sum	701
	Average	702
	Min	703
	Max	704
	Std deviation	705
	Variance	706
	Sum squares	707
第37章	ピクチャコマンド	709
	サポートされるピクチャのフォーマット	710
	Apple QuickTime圧縮	711
	QuickTimeの圧縮タイプ	711
	画像圧縮エラー	711
	Windows上でApple QuickTimeを使用する	712
	COMPRESS PICTURE	713
	LOAD COMPRESS PICTURE FROM FILE	714
	COMPRESS PICTURE FILE	715
	SAVE PICTURE TO FILE	716
	PICT TO GIF	717
	Picture size	719
	PICTURE PROPERTIES	720
	PICTURE LIBRARY LIST	721
	GET PICTURE FROM LIBRARY	723
	SET PICTURE TO LIBRARY	724
	REMOVE PICTURE FROM LIBRARY	727

第38章	印刷コマンド	729
	フォームレポートにおけるブレーク処理の生成	731
	Subtotal関数を使用したブレーク処理	731
	BREAK LEVELコマンドおよびACCUMULATEコマンド を使用したブレーク処理	731
	2つの方法の比較	732
	REPORT	733
	PRINT LABEL	735
	PRINT SELECTION	738
	Printing page	740
	BREAK LEVEL	741
	ACCUMULATE	742
	Subtotal	743
	Level	744
	PRINT RECORD	746
	PAGE SETUP	747
	PRINT SETTINGS	748
	SET PRINT PREVIEW	749
	PRINT FORM	750
	PAGE BREAK	752
第39章	プロセス（通信）コマンド	753
	Semaphore	754
	CLEAR SEMAPHORE	756
	Test semaphore	757
	CALL PROCESS	758
	GET PROCESS VARIABLE	759
	SET PROCESS VARIABLE	762
	VARIABLE TO VARIABLE	765
第40章	プロセス（ユーザインタフェース）コマンド ..	767
	HIDE PROCESS	768
	SHOW PROCESS	769
	BRING TO FRONT	769
	Frontmost process	770
第41章	プロセスコマンド	771
	New process	771
	Execute on server	774
	DELAY PROCESS	778
	PAUSE PROCESS	780
	RESUME PROCESS	782
	Process aborted	783
	Current process	784
	Process state	784

	PROCESS PROPERTIES	785
	Process number	788
	Count users	790
	Count tasks	790
	Count user processes	791
	EXECUTE ON CLIENT	791
	REGISTER CLIENT	793
	UNREGISTER CLIENT	796
	GET REGISTERD CLIENTS	797
第42章	検索とソートコマンド	799
	検索	800
	ソート	801
	QUERY BY EXAMPLE	802
	QUERY	803
	検索条件の指定方法	805
	QUERY SELECTION	810
	QUERY BY FORMULA	812
	QUERY SELECTION BY FORMULA	814
	SET QUERY DESTINATION	815
	SET QUERY LIMIT	820
	Find index key	821
	ORDER BY	822
	ORDER BY FORMULA	827
	QUERY WITH ARRAY QUERY WITH ARRAY	829
第43章	レコードロックコマンド	831
	レコードのロック	832
	リードオンリー状態とリードライト状態	832
	リードオンリー状態	832
	リードライト状態	833
	テーブルステータスの変更	833
	レコードのロード、更新、アンロード	834
	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 ..	835
	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 ..	837
	READ WRITE	838
	READ ONLY	839
	Read only state	840
	LOAD RECORD	841
	UNLOAD RECORD	842
	Locked	843
	LOCKED ATTRIBUTES	844

第44章	レコードコマンド	845
	DISPLAY RECORD	846
	CREATE RECORD	847
	DUPLICATE RECORD	848
	Is new record	849
	Modified record	850
	Is record loaded	851
	SAVE RECORD	852
	DELETE RECORD	853
	Records in table	854
	Record number	855
	GOTO RECORD	856
	Sequence number	857
	レコードに付けられた番号の使用	858
	レコードに付けられた番号の例	859
	PUSH RECORD	862
	POP RECORD	863
	レコードスタックの使用	864
第45章	リレートコマンド	865
	コマンドを使用したテーブルの自動リレート	866
	テーブルリレートを実行するコマンド	867
	AUTOMATIC RELATIONS	869
	RELATE ONE	870
	RELATE MANY	873
	CREATE RELATED ONE	875
	SAVE RELATED ONE	876
	修正されたデータの管理	877
	OLD RELATED ONE	877
	SAVE OLD RELATED ONE	878
	OLD RELATED MANY	878
	RELATE ONE SELECTION	879
	RELATE MANY SELECTION	880
第46章	リソースコマンド	881
	リソースの概念	881
	データフォークとリソースフォーク	881
	4D Transporter	881
	リソースファイル	882
	ユーザ独自のリソースファイルの作成	883
	リソースファイル連鎖	883
	リソースタイプ	885
	リソース名とリソースID番号	886
	リソースプロパティ	888
	リソース内容の取り扱い	888

4Dコマンドとリソース	889
リソースと4D Insider：例題	890
Open resource file	897
Create resource file	901
CLOSE RESOURCE FOLE	904
RESOURCE TYPE LIST	905
RESOURCE LIST	907
STRING LIST TO ARRAY	909
ARRAY TO STRING LIST	910
Get indexed string	912
Get string resource	913
SET STRING RESOURCE	914
Get text resource	915
SET TEXT RESOURCE	916
GET PICTURE RESOURCE	917
SET PICTURE RESOURCE	918
GET ICON RESOURCE	919
GET RESOURCE	921
SET RESOURCE	922
Get resource name	925
SET RESOURCE NAME	927
Get resource properties	928
SET RESOURCE PROPERTIES	929
リソース属性とその及ぼす影響について	930
DELETE RESOURCE	932

第47章 カレントセクションコマンド 935

ALL RECORDS	936
Records in selection	936
DELETE SELECTION	937
Selected record number	939
GOTO SELECTED RECORD	940
FIRST RECORD	942
NEXT RECORD	942
LAST RECORD	943
PREVIOUS RECORD	943
Before selection	944
End selection	946
DISPLAY SELECTION	948
MODIFY SELECTION	951
APPLY TO SELECTION	952
REDUCE SELECTION	953
SCAN INDEX	954
ONE RECORD SELECT	955
HIGHLIGHT RECORDS	956

第48章	セットコマンド	957
	セットとカレントセクション	958
	プロセスセットとインタープロセスセット	960
	セットとトランザクション	960
	セットの例	961
	システムセット : UserSet	962
	システムセット : LockedSet	962
	CREATE EMPTY SET	963
	CREATE SET	964
	CREATE SET FROM ARRAY	965
	USE SET	966
	ADD TO SET	967
	REMOVE FROM SET	967
	CLEAR SET	968
	Is in set	968
	Records in set	969
	SAVE SET	969
	LOAD SET	971
	DIFFERENCE	972
	INTERSECTION	973
	UNION	974
	COPY SET	976
第49章	文字列関数	977
	文字参照記号	978
	String	980
	Num	982
	Position	984
	Substring	985
	Length	986
	Ascii	987
	Char	989
	Uppercase	990
	Lowercase	990
	Change string	991
	Insert string	992
	Delete string	993
	Replace string	994
	Convert case	995
	Mac to Win	997
	Win to Mac	998
	Mac to ISO	999
	ISO to Mac	1002

第50章	ストラクチャアクセスコマンド	1005
	Count tables	1006
	Count fields	1006
	Table name	1007
	Field name	1008
	Table	1009
	Field	1010
	GET FIELD PROPERTIES	1011
	SET INDEX	1013
	Get database parameter	1015
	SET DATADABE PARAMETER	1016
第51章	サブレコードコマンド	1019
	CREATE SUBRECORD	1020
	DELETE SUBRECORD	1021
	ALL SUBRECORDS	1022
	Records in subselection	1022
	APPLY TO SUBSELECTION	1023
	FIRST SUBRECORD	1024
	LAST SUBRECORD	1024
	NEXT SUBRECORD	1025
	PREVIOUS SUBRECORD	1026
	Before subselection	1026
	End subselection	1027
	ORDER SUBRECORD BY	1028
	QUERY SUBRECORDS	1029
第52章	システムドキュメントコマンド	1031
	システムドキュメントの概要	1031
	ドキュメントファイルのタイプとクリエータ	1033
	DocRef：ドキュメント参照番号	1033
	I/Oエラーの処理	1034
	Documentシステム変数	1034
	ドキュメント名やドキュメントパス名の指定	1034
	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	1035
	Open document	1038
	Create document	1042
	Append document	1046
	CLOSE DOCUMENT	1047
	COPY DOCUMENT	1049
	MOVE DOCUMENT	1050
	DELETE DOCUMENT	1051
	Test path name	1052

CREATE FOLDER	1053
VOLUME LIST	1054
VOLUME ATTRIBUTES	1055
FOLDER LIST	1057
DOCUMENT LIST	1058
Document type	1059
SET DOCUMENT	1060
MAP FILE TYPES	1061
Document creator	1063
SET DOCUMENT CREATOR	1064
GET DOCUMENT PROPERTIES	1065
SET DOCUMENT PROPERTIES	1071
Get document size	1072
SET DOCUMENT SIZE	1072
Get document position	1073
SET DOCUMENT POSITION	1074
Select folder	1075

第53章 システム環境コマンド..... 1079

Screen height	1080
Screen width	1080
Count screens	1081
SCREEN COORDINATES	1081
SCREEN DEPTH	1082
SET SCREEN DEPTH	1083
Menu bar screen	1084
Menu bar height	1084
FONT LIST	1085
Font name	1085
Font number	1086
System folder	1087
Temporary folder	1087
Current machine	1088
Current machine owner	1088
Gestalt	1089
LOG EVENT	1090

第54章 テーブルコマンド..... 1093

DEFAULT TABLE	1093
Current default table	1095
フォームを指定する	1096
INPUT FORM	1097
OUTPUT FORM	1098
Current form table	1099

第55章	ツールバーコマンド	1101
	HIDE TOOL BAR	1101
	SHOW TOOL BAR	1101
第56章	トランザクションコマンド	1103
	トランザクションを使用する	1103
	トランザクションの例	1104
	START TRANSACTION	1108
	VALIDATE TRANSACTION	1108
	CANCEL TRANSACTION	1108
	In transaction	1109
第57章	トリガコマンド	1111
	トリガについて	1111
	4Dの旧バージョンとの互換性	1112
	トリガのアクティブ化と作成	1113
	1. トリガをアクティブにする	1113
	2. トリガを作成する	1115
	データベースイベント	1116
	トリガと関数	1116
	トリガと4Dアーキテクチャ	1119
	トリガとトランザクション	1121
	トリガのカスケード	1121
	トリガ内での通し（シーケンス）番号の使用	1123
	Database event	1124
	Trigger level	1126
	TRIGGER PROPERTIES	1127
第58章	ユーザインタフェースコマンド	1129
	BEEP	1130
	PLAY	1131
	Get platform interface	1133
	SET PLATFORM INTERFACE	1133
	SET TABLE TITLES	1135
	SET FIELD TITLES	1139
	Shift down	1140
	Caps lock down	1141
	Windows Ctrl down	1142
	Windows Alt down	1142
	Macintosh command down	1143
	Macintosh option down	1143
	Macintosh control down	1144
	GET MOUSE	1145
	Pop up menu	1146

	POST KEY	1149
	POST CLICK	1150
	POST EVENT	1151
	GET HIGHLIGHT	1152
	HIGHLIGHT TEXT	1154
	SET CURSOR	1155
	Last object	1155
	REDRAW	1156
	INVERT BACKGROUND	1157
第59章	ユーザ&グループコマンド.....	1159
	EDIT ACCESS	1160
	CHANGE ACCESS	1160
	CHANGE PASSWORD	1161
	Validata password	1162
	Current user	1163
	User in group	1163
	DELETE USER	1164
	Is user deleted	1165
	GET USER LIST	1166
	GET USER PROPERTIES	1167
	Set user properties	1168
	GET GROUP LIST	1171
	GET GROUP PROPERTIES	1172
	Set group properties	1174
	CHANGE LICENSE	1176
第60章	変数コマンド.....	1179
	SAVE VARIABLES	1179
	LOAD VARIABLES	1180
	CLEAR VARIABLE	1181
	Undefined	1182
第61章	Webサーバコマンド	1185
	Webサービス：概要	1185
	次は何をするのか?	1188
	Webサービス：システム設定	1189
	4th DimensionとWeb	1189
	4D ServerとWeb	1190
	Web上での4Dデータベースのサービス	1190
	4D Webサービスの開始	1192
	Web上に公開された4Dデータベースへの接続	1193
	TCPポート番号を特定の値に設定する	1194
	Webサービス：入門編（パートI）	1196

Web接続の開始	1199
1つになったデータベースとWebサーバ	1201
Webサービス：入門編（パートII）	1203
データベースにHTMLを追加する	1203
URLのリンク	1206
ボタン	1207
実行する4Dメソッドの指定	1208
プロジェクトメソッド	1209
次は何をするのか？	1210
Webサービス：接続セキュリティ	1210
Webアクセス用のパスワードマネージメントシステム ..	1210
4DWebサーバのアクセスシステムのオーバービュー ..	1211
ロボットに関するセキュリティ注意事項	1213
一般Webユーザ	1214
Webパスワードシステムとの相互作用	1215
デフォルトHTMLルートフォルダを定義する	1215
データベースのプロパティとSET HTML ROOT	1217
On Web Authenticationデータベースメソッド	1217
Webサービス：「Web接続」プロセス	1222
「Webサーバ」プロセス	1222
「Web接続」プロセス	1223
Web接続のコンテキスト管理：コンテキストモード ...	1224
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	1225
「Web接続」プロセスとWebセッション	1229
Web接続タイムアウト	1230
「On Web Connection」データベースメソッド	1231
URLエクストラデータ	1231
HTTPリクエストのヘッダ	1232
On Web Connectionデータベースメソッドの呼び出し ..	1233
例：クライアントのローカルホームページの実装	1235
Webサービス：Webサーバセッティング	1239
デフォルトのホームページを定義します	1239
データ入力制御のためにJavascriptを使用する	1241
サーバがHTTPクエリを受けるIPアドレスを定義する ..	1241
スタティックなページでの、4DVARのコメントを使用する ..	1242
新規のコンテキスト参照モード	1242
拡張ASCII文字を直接送る	1243
4Dで文字セットの変換を修正する	1243
スタティックページのキャッシュ	1244
Webサービス：非コンテキストモード	1246
Webサービス：HTMLサポート	1257
Webサービス：HTMLとJavaScriptのカプセル化	1264
はじめに	1264
HTMLオブジェクトと4Dメソッドとのバインド設定 ...	1265
HTMLオブジェクトと4D変数とのバインド設定（パート1）...	1265

JavaScriptカプセル化	1268
HTMLオブジェクトと4D変数とのバインド設定 (パート2)...	1269
HTMLオブジェクトと4D変数とのバインド設定 (パート3)...	1274
ファイル参照とURL (コンテキストモード).....	1275
URLでコールした4Dメソッドに渡されたテキスト引数 ..	1276
コンパイルモードにおけるランタイムエラー	1276
URLエクストラデータをを用いた作業	1279
URLエクストラデータのダイナミックな設定方法	1279
Webサービス：Webサイトに関する情報	1280
接続ログファイル	1281
START WEB SERVER	1283
STOP WEB SERVER	1284
SET WEB TIMEOUT	1284
SET WEB DISPLAY LIMIT	1285
SET HOME PAGE	1288
SEND HTML FILE	1289
SEND HTML BLOB	1292
Web Context	1294
SET HTML ROOT	1295
WEB CACHE STATISTICS	1298
SET HTTP HEADER	1299
SEND HTTP REDIRECT	1300
OPEN WEB URL	1302

第62章 ウインドウコマンド 1305

ウインドウについて	1306
ウインドウのタイプ	1307
Open window	1313
Open external window	1317
SHOW WINDOW	1318
HIDE WINDOW	1318
CLOSE WINDOW	1320
ERASE WINDOW	1321
REDRAW WINDOW	1322
DRAG WINDOW	1322
Get window title	1324
SET WINDOW TITLE	1325
WINDOW LIST	1326
Window kind	1327
Window process	1328
GET WINDOW RECT	1328
SET WINDOW RECT	1329
Frontmost window	1330
Next window	1330
Find window	1331
MAXIMIZE WINDOW	1332

	MINIMIZE WINDOW	1333
	Open form window	1335
	GET FORM PROPERTIES	1337
付録A	システム変数	1339
	OK	1339
	Document	1340
	FldDelimit	1340
	RecDelimit	1340
	Error	1341
	MouseDown、MouseX、MouseY、KeyCode、Modifiers、 MouseProc	1341
付録B	ASCIIコード	1343
	ASCIIコード0から127	1343
	拡張ASCIIコード（128から255）	1346
	ASCIIコードと4th Dimensionの理解	1351
	ファンクションキー	1352
付録C	エラーメッセージ	1353
	シンタックスエラー -	1353
	データベースエンジンエラー	1356
	ネットワークコンポーネントエラーコード	1359
	システムエラーコード	1360
	File Managerのエラ -	1360
	Memory Managerのエラー	1361
	プリントエラーコード	1362
	Resource Managerのエラ -	1362
	Macintosh SANE NaN Errorのコード	1362
	Sound Managerのエラー	1363
	シリアルポートのエラーコード	1363
	システムエラー	1363
	データファイルのロック状態のテスト	1364
付録D	定義済み定数	1367
	4D Environment	1367
	ASCII Codes	1367
	BLOB	1369
	Clipboard	1369
	Colors	1369
	Communications	1370
	Database Engine	1370
	Database Events	1371
	Database Parameters	1371
	Date Display Formats	1371

Days and Months	1372
Events (Modifiers)	1373
Events (What)	1373
Expressions	1374
Field and Variable Types	1374
Find window	1375
Font Styles	1375
Form Events	1376
Function Keys	1377
Hierarchical Lists	1377
ISO Latin Character Entities	1378
Math	1380
Open window	1380
Picture Display Formats	1380
Platform Interfaces	1380
Platform Properties	1381
Process state	1381
Process Type	1381
Query Destinations	1382
Resources Properties	1382
SCREEN DEPTH	1383
SET RGB COLOR	1383
Standard System Signatures	1383
TCP Port Numbers	1384
Time Display Formats	1385
Window kind	1385
Window Position	1385
Window NT Log Events	1385

コマンド索引	1387
---------------------	-------------

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」モードで利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。

『4th Dimension ユーティリティガイド』は、4D Tools、Customizer Plus、4D Transporterといった4th Dimensionに提供されるユーティリティの手引書です。

『4th Dimension 用語集』は、バージョン6で変更された4th Dimension用語を掲載しています。

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。

なお、このマニュアルは、4th Dimensionおよび4D ServerのWindowsとMacintoshユーザの両方を対象にしたクロスプラットフォームのマニュアルです。WindowsとMacintoshにおいて明らかに内容が異なる（画面、キーボード操作等）場合にのみ両方の説明を併記しています。それ以外はWindows版を中心に本文は記述されています。また、本文中で使用されているWindows版のスクリーンショットは「Windows 95」を基に作成しています。そのため、WindowsNT上で使用している場合、本文中の画面と多少異なる箇所がありますが、あらかじめご了承ください。

表記方法について

このマニュアルを含め、パッケージ中の全マニュアルではより内容を理解できるように一定のマークを使用しています。

次のようなマークが使用されています。

注：4th Dimensionをより豊富に使用出来るように、このような強調文で注釈やショートカットを提供します。

4D Server：マニュアルを通して、4th Dimension、4D Server / 4D Clientは単に4th Dimensionと呼びます。2つの製品の操作の違いは、この4D Serverマークの中で説明されています。4D Serverマークは4D Server / 4D Client の使用方法に関する情報を提供しています。この情報は、4D Server / 4D Clientの操作が4th Dimensionと異なる部分のみ提供します。

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 >

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る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

4th Dimensionにプログラミング言語はほとんど必要ありません。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？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」メニューコマンド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利用できる最も先進的で柔軟性のある言語のひとつです。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ひとつとなります。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、その後でひとつまたは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が、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

「テーブルメソッド/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利用できるメソッド

「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。

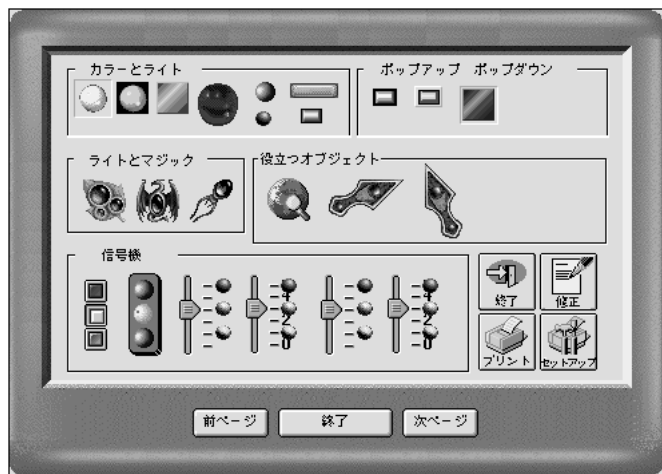
フィールドやサブフィールド等のデータの入力、表示、格納のためのアクティブオブジェクト

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。

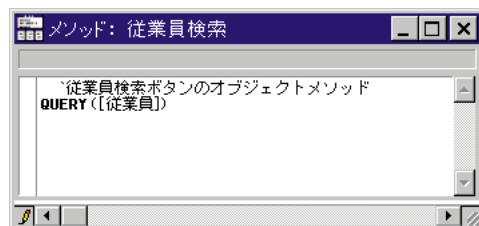
想像力が許す限りのグラフィックを使用したフォームを作成することもできます。



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のに気がきます。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着するのと同様に、フォームメソッドはフォームに付着します。ひとつのフォームには、ひと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、第5章および第24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、後で複雑にしていけることもできます。

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。

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ひとつの方法として、メニューコマンドへの割り当てがあります。メニューコマンドに割り当てられたメソッドは、メニューコマンドを選択した時に実行されます。メニューコマンドは、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、第6章の「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があります。

「実行」：「デザイン」モードで設計したものを実行する。

「テスト」：「ユーザ」モードで設計したものをテストしたり、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、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つのメニューコマンドを選択するだけで、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ではありませんが、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。アプリケーション構築における操作手順に関する詳細は、第7章を参照してください。

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：データベースでのデータの分類

変数：データ用のメモリ内に入れられる一時的な記憶領域

演算子：2つの値の間で計算をする記号

式：値を求める構成要素の組み合わせ

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

メソッド：プログラミング言語を使用して作成した命令の集まり

この章では、データタイプ、演算子、式について説明します。

次の章でメソッドの各タイプについて説明します。

また、残りの章で、プログラミング言語の各コマンドについて説明します。

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、複数のフィールドタイプを伴う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、別のタイプとしてそれを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つのデータからひと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ひと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...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“独立型”であることはほとんどありませんが、独立して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、デバッグ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11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!	日付	これは、日付定数!97.07.21! (1997年7月21日) 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。
!!2:03:04!!+!!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、ポインタを[従業員]名前というフィールドに返します。
Table (1)	ポインタ	これは、最初のテーブルにポインタを返すコマンドです。

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： $\pm 1.7e \pm 308$ の数字（15バイト）

日付：100.1.1から32767.12.31

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、レポートの印刷を実行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、ダイヤル）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、**Current date**関数等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、データの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（グローバル）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、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、ユーザからひと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ための良い例です。

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、それを基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、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、ひと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 사용할 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インスタンスが存在します。

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、単に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、データベースのどこでも使用できるプロセス変数です。

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。

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ひとつです。

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行のことでひと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の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、メニューコマンドと関連付けるというもうひと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コマンドに関連付けると、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。これでメニューコマンド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に関する詳細は、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

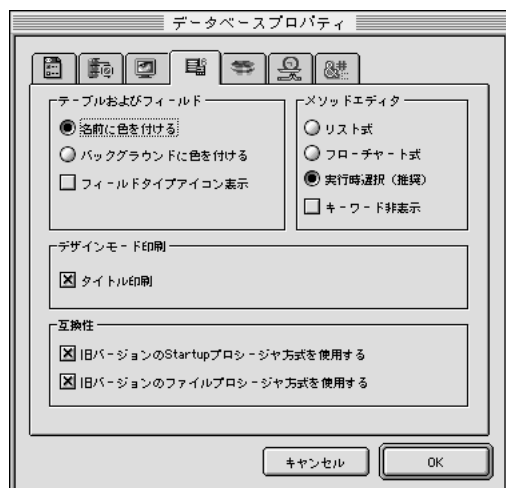
4th Dimensionのバージョン6で新しく作成されたデータベースで作業している場合は、この互換性に関する注記は読み飛ばしてかまいません。

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) はひとつだけでした。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なります。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ひと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太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、理解するまで見直してください。

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。

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種類のうちひと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、別のパス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つのパス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、これは多くのパスの中のひとつだけにプログラムの流れを導きます。

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)
  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
  ステートメント
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、メソッドを複数の動作から選択できるようにします。**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 “ケース” と呼びます。例えば、以下の行はケースです。

```
¥ (番号=1)
```

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。“True (真)” になるケースがない場合には、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、最後のケースの後に**Else**文を含むことができます。すべてのケースが “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のどれもない場合
ALERT ("1、2、3のどれもない。")	` どれ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。

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。

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)” 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。

Tip：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 ($vElem;1;Size of array(anArray))
  `配列要素を用いて処理を実行する
  anArray{$vElem}:=...
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 ($vChar;1;Length(vtSomeText))
  `TABの場合その桁を用いて処理を実行する
  If (Ascii(vtSomeText[$vChar])=Char(Tab))
    ...
  End if
End for
```

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

```
FIRST RECORD([テーブル1])
For ($vRecord;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 ; 1 ; 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 (\$vElem;1;**Size of array**(anArray);1;-1)

` 配列要素を用いて処理を実行する

anArray{\$vElem}:=...

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 (\$vChar;1;**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[[\$vChar]])=**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 (\$vRecord;1;**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 (\$vElem;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{\$vElem}:=...

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、他の条件が真になった場合はループから抜き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($vCounter) ` 倍長整数タイプのローカル変数を使用
For ($vCounter;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 ($vElem;1;Size of array(anArray))
  ` ...
  ` この行を用いて処理を実行する
  ` ...
  For ($vSubElem;1;Size of array(anArray{$vElem}))
    ` この要素を用いて処理を実行する
    anArray{$vElem}{$vSubElem}:=...
  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)
$viElem:=0
For ($viTable;1;Count table)
  For($viField;1;Count fields($viTable))
    $vpField:=Field($viTable;$viField)
    If (Type($vpField->)=Is Date)
      $viElem:=$viElem+1
      INSERT ELEMENT($apDateFields;$viElem)
      $apDateFields{$viElem}:=$vpField
    End if
  End for
End for
```


フォームは、4th Dimensionの最も特徴的な部分です。4th Dimensionが提供するツールを使用することにより、自由にさまざまなフォームを作成することができます。入力フォームに優れたユーザインタフェースを利用することができるだけでなく、出力フォームにおいても読みやすいレポートを作成することができます。

4th Dimensionは、プログラミング言語を使用しないでフォームオブジェクトを管理できる組み込みツールを備えています。このツールには、データの属性チェック、入力フィルタ、フォーマット、データの範囲チェック、選択リスト、デフォルト値の設定、ボタンの属性設定等が用意されています。また、プログラミング言語は組み込みツールの機能を拡張し、複雑なフォームの制御や範囲チェックを実現することができます。だからと言って、馴れ親しんできた組み込みツールによる制御をあきらめる必要はありません。実際、これらのツールを最大限利用すべきです。プログラミング言語は、単にユーザが既に会得したツールの機能を拡張するために使用します。

以下の2種類のメソッドは、フォームを制御するために使用されます。

オブジェクトメソッド
フォームメソッド

オブジェクトメソッドは、フォーム制御のための最も一般的なメソッドです。フォームは、すべてのオブジェクトに対してオブジェクトメソッドを持つことができ、データの入出力時に使用します。

フォームメソッドは、特定の入力フォームまたは出力フォームで使用します。

フォームを制御する

フォームの制御は、データベースをカスタマイズする上で最も難しい部分のひとつです。プロジェクトメソッドを実行している場合は、フォームを制御する必要はありません。メソッドは指示されたことだけを実行し、それに影響を与える外的要素はほとんどありません。しかし、フォームを使用すると、自分のメソッドが取り扱わなければならないユーザというイベントの新しいソースが作成されます。

ユーザは、自分で実行することを決定します。例えば、ボタンのクリックや、フィールド間の移動、別のフォームページへの移動、メニューの選択等、ユーザは一般にメソッドを混乱させるような行動を取ります。

さらに、フォーム自体を使用することから生じる特別なイベントもあります。フォームが最初に表示された時点で、初期化が必要となるイベントです。レコードが受け入れられた時点で、特別な処理が必要な場合もあります。レポートを印刷する場合は、レポートのさまざまな部分に対していくつもの制御が必要となります。

これらのイベントは、どのように管理するのでしょうか？メソッドが多くの起こり得るイベントをそれぞれチェックし、応答しなければならないとしたら、メソッドで何かを実行することは、ほとんど不可能になります。これでは、イベントを判定するだけで終わってしまいます。しかし、4th Dimensionにはユーザに代わってイベントを管理し、それを的確にメソッドに知らせる手段が用意されています。

イベントの管理は、オブジェクトメソッドとフォームイベントの2つの手段を用いて行われます。オブジェクトメソッドはオブジェクト指向のイベントマネージャです。このイベントマネージャは、フォームオブジェクトで発生したイベントに対して応答します。また、フォームイベントも重要なイベントマネージャです。これは、フォームに発生するすべてのイベントをチェックします。

オブジェクトメソッドを使用する

オブジェクトメソッドは、フォーム内のオブジェクトに付着するメソッドです。この役割は、オブジェクトに対して非常に特殊な存在であり、付着したオブジェクトだけを管理します。

オブジェクトメソッドは、以下のような場合に実行されます。

- データ入力の実行
- 画面上でのレコード表示
- フォームを使ったレポートの印刷
- フォームを使ったデータの読み込みと書き出し

オブジェクトメソッドは必須のものではありませんが、各オブジェクトはひとつのオブジェクトメソッドを持つことができます。

オブジェクトは、入力（例えば、フィールドや変数等）やインタフェース（例えば、ボタン、ドロップダウンリスト、サーモメータ等）の一部として使用します。オブジェクトメソッドはオブジェクトが操作されるたびに実行されます。例えば、ボタンのオブジェクトメソッドは、ボタンがクリックされた時に実行されます。フィールドのオブジェクトメソッドは、そのフィールド上でデータが入力または修正された場合に実行されます。

また、レコードを画面上に表示、あるいは印刷する場合にもオブジェクトのオブジェクトメソッドを実行します。この場合のオブジェクトメソッドはオブジェクトのフォーマットや外観に影響を与えます。

オブジェクトメソッドは数行程度の短いものがほとんどです。4th Dimensionデータベースでは、オブジェクトメソッドが唯一のメソッドになっているのがほとんどです。オブジェクトメソッドがオブジェクトの中に作成されると、オブジェクトを切り取り、コピー、複製した場合でも、オブジェクトメソッドは属するオブジェクトに従います。つまり、オブジェクトのライブラリからスクラップブックに貼り付けて、別のフォームに貼り付けることができます。オブジェクトメソッドでプロジェクトメソッドを呼び出せる点に注目してください。プロジェクトメソッドでは、オブジェクトメソッドによってコピーされたり、貼り付けられたりすることはありません。

オブジェクトメソッドとデータ入力

フィールドや変数等のオブジェクトメソッドは、以下のような操作に対して実行されます。

入力されたデータの属性チェック

変数に対する値の割り当て

フィールドの連結や小文字から大文字への変換等、文字列の操作

合計、平均、カウントの計算等、算術および日付計算の実行

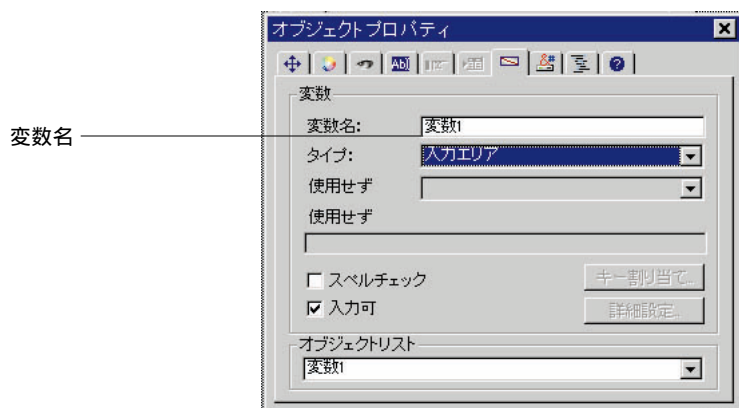
リレートーブルのデータ管理

データがデータ入力中のフィールドや変数によって変更されると、そのフィールドや変数に対するオブジェクトメソッドが実行されます。このオブジェクトメソッドは、フォームメソッドの前に実行されます。

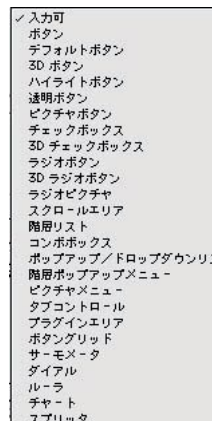
オブジェクトメソッドとインタフェースオブジェクト

フォームには、ユーザとのインタフェースを司る広範囲のオブジェクトを含むことがあります。これらのオブジェクトは、ボタンの自動動作等の組み込みツールによって完全に制御することができます。オブジェクトの中には非常に柔軟性のあるものやプログラミング言語がチェックや制御に必要なものもあります。この節では、オブジェクトとプログラミング言語間の相互作用について説明します。

以下の図は“変数”ページを示しています。オブジェクト（変数）は、ここで定義します（オブジェクトの作成とその使用に関する詳細は、『4th Dimension デザインリファレンス』を参照してください）。オブジェクトの名前は、4th Dimensionが自動的に作成する変数の名前でもあります。この変数は、オブジェクトの存在をチェックする場合によく使用されます。



以下の図は、28種類のオブジェクトタイプを定義するために使用するドロップダウンリストボックスを示しています。



オブジェクトメソッドは、オブジェクトの取り扱いを簡単にします。オブジェクトにオブジェクトメソッドを組み合わせて使用することによって、ユーザはメソッドと対話することができます。次節では、オブジェクトタイプ別のオブジェクトメソッドの使用方法について説明します。

ボタン

ボタンは、フォームの中で最もユーザインタフェースに優れたオブジェクトです。ボタンによるオブジェクトメソッドの扱いは非常に簡単です。つまり、ボタンがクリックされた時点で「On Clicked」イベントがチェックされていると、オブジェクトメソッドは実行されます。そうでない場合には、オブジェクトメソッドはフォームイベントに従って実行されます。

ボタンの名前は、変数の名前でもあります。変数は自動的に作成され、ボタンに付随します。ボタンがクリックされると、変数に1が代入されます。その他の場合には、変数の値は0となります。ボタンの自動制御はオブジェクトメソッドが実行された後で行われます。下記は、6つのボタンタイプとそのボタンがクリックされた際の結果です。

動作なし
キャンセル
登録
次レコード
前レコード
先頭レコード
最終レコード
レコード削除
次ページ
前ページ
先頭ページ
最終ページ
サブレコードを開く
サブレコードを削除
サブレコードを追加

注：ボタンのオブジェクトメソッドは、ボタン変数が設定された後で実行されることを覚えておいてください。

「ボタン」：ボタン変数に1を代入します。

「ラジオボタン」：ラジオボタンは、ボタン変数名の頭文字でグループ化され、同じ頭文字を持つボタンがひとつのグループになります。ラジオボタンがクリックされると、ボタン変数に1を代入し、ボタンを反転表示します。他のすべてのラジオボタンはそれぞれの変数に0を代入し、通常表示されます。

「チェックボックス」：ボックスがチェックされると1、チェックされないと0が設定されます。

「透明ボタン」：ボタン変数に1が代入されます（このボタンは、クリックされても反転表示しません）。

「ハイライトボタン」：ボタン変数に1が代入されます（このボタンは、クリックされると反転表示します）。

「ラジオピクチャ」：ラジオピクチャは、ラジオボタンと同じように機能するハイライトボタンです。ラジオピクチャはボタン変数名の頭文字でグループ化され、同じ頭文字を持つボタンがひとつのグループになります。ラジオピクチャがクリックされるとボタン変数に1を代入し、ボタンは反転表示されたままとなります。他のすべてのラジオピクチャはそれぞれの変数に0を代入し、通常表示します。

スクロールエリアとドロップダウンリストボックス

スクロールエリアとドロップダウンリストボックスは、プログラミングの観点から見ると同じように機能します。両方とも項目リスト（配列要素）を含み、これらの項目の中からユーザが要素を選択できるようにします。項目リストは配列です。配列の名前はオブジェクトの名前と同じです。

注：配列は、配列コマンドで作成されます。

ドロップダウンリストボックスとスクロールエリアは、以下の4つのデータタイプを表示することができます。

文字列

数値

日付

時間

特に、スクロールエリアでは、表示エリアとブ - ル配列変更を自由にすることができます。

これらは本質的に柔軟であるため、他のオブジェクトの場合よりもプログラミング言語との相互作用が一層必要になります。例えば、以下のようなことが必要です。

スクロールエリアまたはドロップダウンリストボックスに項目を格納する。

ユーザが項目を選択した場合に応答する。

必要に応じてリストを変更する。

ドロップダウンリストボックスとスクロールエリアを使用する例は、第9章を参照してください。

スクロールエリアまたはドロップダウンリストボックスにデータを格納する

配列は、任意の方法で格納できます。「デザイン」モードで作成したリストやレコードのセクションからも格納できます。例えば、リストから配列へ格納するには、以下のように行います。

LIST TO ARRAY ("Values" ; arValList)

配列の格納は常に可能です。通常、以下の3つの場所で格納されます。

「On Startup」データベースメソッド

フォームが表示される前のプロジェクトメソッド

フォームイベントの On Load イベント

しかし、配列を格納するタイミングはエリアの使用方法によって異なります。配列要素が決して変わらない場合は「On Startup」データベースメソッドで、配列要素が各レコードごとに変わらない場合はプロジェクトメソッド、そして配列要素がレコードごとに変わる場合にはフォームメソッドまたはオブジェクトメソッドの On Load イベントで実行するのが一般的です。

ユーザが項目を選択した場合の応答

ユーザがドロップダウンリストボックス、またはスクロールエリアの項目を選択すると、以下のようなイベントが発生します。

1. 選択された項目番号（配列要素）を配列の名前に設定する。
2. On Clicked イベントが発生する。
3. オブジェクトメソッドが実行される。

配列の名前は、ユーザが選択した項目を表す正の数値に設定されます。以下の図のドロップダウンリストボックスとスクロールエリアを参照してください。表示されている配列変数を “MyArr” とすると、ユーザが4番目の要素を選択した場合（図で示された通り）に、配列変数 “MyArr” に4を設定します。

要素 1		要素 1	↑
要素 2		要素 2	
要素 3		要素 3	
要素 4		要素 4	
要素 5		要素 5	↓

ユーザがスクロールエリアの空のエリアをクリック、または配列要素を全く選択しなかった場合は配列変数に0を設定します。この場合には、適切な値を得るための特別な判断を加えたり、ドロップダウンリストボックスを変更する必要があります。

項目が常に選択されているようにする場合は、最後に選択した項目の数を保存し、それをその項目に戻します。

例えば、以下の例はユーザが何も選択しなかった場合に、項目を前の値に戻します。

Case of

¥ (On Load)

MyItem:=1

項目の初期化

MyArr:= 1

¥ (MyArr=0)

何も選択されなかった場合...

MyArr:=項目

前に選択された項目をセットする

Else

項目:=MyArr

それ以外の場合は、選択された項目を保存する

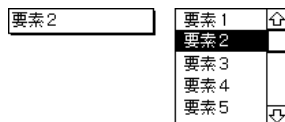
End case

項目が選択されているかどうかを調べるために、オブジェクトのオブジェクトメソッドを使用します。オブジェクトメソッドが実行されると、オブジェクトが選択されたことを意味します。

スクロールエリアとドロップダウンリストボックスの項目変更

配列をどのような方法で変更しても、4th Dimensionは自動的にそれを認識し、その変更を反映させるためにオブジェクトを更新します。変更には、配列要素の変更、要素の削除と追加、配列全体の削除があります。

項目を“選択”するために、選択する項目に対して配列変数を設定します。この項目はスクロールエリアで反転表示されるか、あるいはドロップダウンリスト（ポップアップメニュー）の表示メニューアイテムとして表示されます。配列変数の設定には、代入演算子を使用します。これを使用しないと、配列の変更が認識されません。下図は2番目の要素が選択されたドロップダウンリストボックスとスクロールエリアを示しています。



このドロップダウンリストボックスとスクロールエリアが配列“MyArr”の場合、以下のように“MyArr”に選択した項目の値を代入します。

MyArr:=2

サーモメータ、ルーラ、ダイヤル

サーモメータ、ルーラ、ダイヤルはエリアの割合を数値として表示します。オブジェクトに属する変数は画面の状態を反映するために変更され、反対に画面は変数の値を反映するように変更されます。つまり、ユーザがオブジェクト上をドラッグした場合は変数の値を変更し、変数の値を変更した場合には新しい値を反映するために画面を変更します。

オブジェクトのオブジェクトメソッドは、一般にOn Loadイベントでオブジェクトを初期化します。

グラフエリア

グラフエリアは、**GRAPH**コマンドで設定します。

プラグインエリア

プラグインエリアは、4th Dimensionのプログラミング言語以外の言語で作成されたメソッドで制御するエリアです。プラグインエリアの処理に制限はありませんが、これを選択すると、4th Dimensionに制御を戻すまで、プラグインがすべてを制御、管理します。4th Dimensionに対して制御を返す場合、On Plug In Areaイベントが発生します。

オブジェクトメソッドとレポート出力

フィールドや変数に付着するオブジェクトメソッドは、それらを含むフォームが印刷される時点で実行されます。また、オブジェクトのフォームエリアが印刷された場合にのみ実行されます。

例えば、下図で示したフォームには、B0ブレイクエリアに“vTotal”という名前の変数があります。

The screenshot shows a form window titled 'フォーム: [給与]Form1'. Inside, there's a table with 5 columns: '部門' (Department), '名字' (Surname), '名前' (Name), '入社年月日' (Joining Date), and '給与' (Salary). Below the table is a '合計部門' (Total Department) field. To the right of the table, in the B0 break area, there's a variable 'vTotal'.

このオブジェクトメソッドを持つ変数は、レベル0のブレイクで印刷されます。このフォームにおける、“vTotal”のオブジェクトメソッドは以下の通りです。

vTotal:= **Subtotal** (給与)

このオブジェクトメソッドは、変数“vTotal”に給与フィールドの小計を代入します。

フォームメソッドとオブジェクトメソッドは、フォームイベントに従って実行されます。フォームイベントは、メソッドを実行するタイミングを決定します。

注：この章では、バージョン6で新たに追加されたフォームイベントの説明は行っていません。バージョン6で新たに追加されたフォームイベントに関する詳細は、第24章を参照してください。

フォームメソッドは、フォームやプロセスで何か起こった場合に開始します。また、4th Dimensionのマルチプロセス環境では、あるプロセスに別のプロセスを呼び出させます。データ入力、表示、およびプロセス間通信で使用するフォームの基本的なフォームイベントをいくつか次に示します。

Beforeフェーズ

Duringフェーズ

Afterフェーズ

Outside Callフェーズ

Activatedフェーズ

Deactivatedフェーズ

フォームイベントの各部は、“フェーズ”として参照されます。フォームのメソッドは、各フェーズで実行されます。Before、During、Afterの各フェーズは最も重要なもので、頻繁に使用されます。Outside Callフェーズは、呼び出しのプロセスが別のプロセスから作成された場合に実行されます。ActivatedフェーズとDeactivatedフェーズは、ウィンドウがアクティブかアクティブでない時に呼び出されます。この2つのフェーズは、オブジェクトメソッドの中では発生しません。

また、印刷時には、以下のようなフォームイベントがあります。

In Headerフェーズ

Beforeフェーズ

Duringフェーズ

In Breakフェーズ

In Footerフェーズ

フォームイベントは、フォームに付着するすべてのメソッドに影響を与えます。

フォームイベントはフォーム処理の中核ですが、ほとんどの場合において、それを意識する必要はありません。必要な場合にのみ使用します。しかし、フォームイベントの各フェーズを使用することにより、さまざまなデータやオブジェクトの管理を適切なタイミングで制御することができます。

オブジェクトに対してオブジェクトメソッドを作成する場合、フォームイベントを考慮する必要はありません。オブジェクトメソッドは、その属するオブジェクトが操作されるたびに実行されるのでフォームイベントを必要としません。これにより、オブジェクトメソッドは適切なタイミングで実行されます。しかし、特定のフェーズだけに処理を制限する場合には、オブジェクトメソッド内でフォームイベントのフェーズを判定する必要があります。

フォームイベントのチェック

プログラミング言語は、フォームイベントを判定するための関数を備えています。各関数は、フェーズが発生している場合に “ True ” を返し、他の場合は “ False ” を返します。

次に、各関数の名前とそれに対応するフェーズについて説明します。

「 Before 」 : Beforeフェーズは、フォームが表示、印刷される前に各レコードに対して発生します。

「 During 」 : Duringフェーズは、レコードの表示やオブジェクトの修正やクリック、あるいはメニュー - の選択の各時点で発生します。

「 After 」 : Afterフェーズは、データ入力時にレコードが受け入れられた後にだけ発生します。

「 Before&During 」 : これは特別なフェーズです。レコードが画面上に表示された時点で発生します。これは2つのフェーズが同時に “ True ” を返した場合のみです。

「 Outside Call 」 : Outside Callフェーズは、呼び出しプロセスが別のプロセスから作成された場合に発生します。

「 Activated 」 : Activatedフェーズは、ウインドウがアクティブになった場合に発生します。

「 Deactivated 」 : Deactivatedフェーズは、ウインドウがアクティブでない場合に発生します。

「 In header 」 : In headerフェーズは、フォームのヘッダエリアが印刷される前か、あるいは画面上に表示される前に発生します。ひとつのヘッダが各ページに印刷されます。ヘッダが複数の場合には、各ブレイクレベルが印刷されるたびに発生します。最初のヘッダであるかどうかは、 **Before selection** 関数を使用して判定します。

「In break」：In Breakフェーズは、ブレイク処理中に発生します。ブレイクエリアはそれから印刷されます。印刷されるブレイクレベルは、**Level**関数を使用して判定することができます。

「In footer」：In Footerフェーズは、フッタエリアの印刷が開始される前にのみ発生します。ひとつのフッタが各ページに印刷されます。**End selection**関数では、最後のフッタであるかどうかを判定することができます。

メソッドは一般にフォームイベントのフェーズを判定するために、**Case**構文を使用します。次はデータ入力におけるフォームメソッドの一般的な構造を示しています。

Case of

¥ (Before)

　` 変数の初期化

¥ (During)

　` 入力データの表示

¥ (After)

　` レコードの保存

End case

フォームイベントの一般的な規則

ここでは、フォームイベントに関連したいくつかの一般的な規則を示します。

メソッドのフォームイベントの各フェーズは、以下の順序で実行されます。

1. オブジェクトメソッドの実行
2. フォームメソッドの実行

データ入力に使用するオブジェクトのオブジェクトメソッドは、データの入力順序順で実行されます。

データ入力で使用されないオブジェクトのオブジェクトメソッドは実行されません。オブジェクトメソッドはデータの入力順序とは無関係に実行されます。

レコードが画面上に表示または印刷される場合は、オブジェクトメソッドが“修正不可”でないオブジェクトのオブジェクトメソッドだけが実行されます。例えば、ヘッダの変数は、ヘッダが表示された場合にのみ、(In Headerフェーズ中で)そのオブジェクトメソッドが実行されます。

フォームイベント

以下の節は、各フォームの使用ごとに、フォームイベントを説明しています。

データ入力の場合

ユーザがフォームにデータを入力する場合のフォームイベントには、以下のフェーズがあります。

「Before」：このフェーズは、フォームが表示される前に発生します。これは、新規レコードや修正レコードのチェックに使用します。Beforeフェーズは、変数の初期化やフィールドのデフォルト値の表示のために使用します。

「During」：このフェーズは、データが修正されたり、アクティブオブジェクトが使用されると、データ入力時に発生します。

「After」：このフェーズは、レコードが受け入れられた後にだけ発生します。

「Outside Call」：このフェーズは、呼び出しが別のプロセスから作成された場合のデータ入力時に発生します。

「Deactivated」：このフェーズは、フォームやオブジェクトメソッドを含んだウィンドウが背後にある場合や最前面のウィンドウがない場合のデータ入力時に発生します。Deactivatedフェーズは、警告メッセージやデバッグが表示されている場合にも発生します。

「Activated」：このフェーズは、フォームやオブジェクトメソッドを含んだウィンドウが最前面に配置されるたびにデータ入力時に発生します。

Duringフェーズは、ユーザがフォームに対して何かを行った場合に発生します。ユーザの動作から、Duringフェーズを発生させるものとしては、以下のようなものがあります。

フィールドや変数のデータを変更してから、そのフィールドや変数を離れる

ボタンをクリックする

サーモメータ、ルーラー、ダイヤルを操作する

メニューからメニューコマンドを選択する

ドロップダウンリストボックス（ポップアップメニュー）から項目を選択する

スクロールエリアから項目を選択する

ショートカットキーを押す

プラグインエリアをクリックする

組み込みエリアを使用する

Duringフェーズは、データの属性チェック、入力データのフォーマット、リレートテーブルのデータ管理、制御選択のチェックや応答、フォームオブジェクトの特定の処理等に使用します。

Afterフェーズは、ユーザがレコードを受け入れた後で発生します。しかし、レコードをキャンセルした場合は発生しません。

テーブルのサブフォーム

サブフォーム内のレコード表示のフォームイベントには、以下のフェーズがあります。

「Before」：このフェーズは、新しいレコードがサブテーブルに追加された場合に発生します。

「During」：このフェーズは、サブテーブルのレコードが最初に表示された場合と、その修正時に発生します。親レコードのDuringフェーズは、サブフォームのDuringフェーズの後に発生します。

「After」：このフェーズは、サブテーブルの各レコードが変更され、受け入れられた場合に発生します。

「Deactivated」：このフェーズは、フルページフォームを含んだウィンドウが背後に配置されている場合に、リレートされたレコードのフルページフォーム内でのみ発生します。

「Activated」：このフェーズは、フルページフォームが最前面に配置されて、リレートされたレコードのフルページフォーム内でのみ発生します。

「Outside Call」：このフェーズは、プロセスの呼び出しが最前面のウィンドウ内のフルページフォームを含んだプロセスで作成された場合に、リレートされたレコードのフルページフォーム内でのみ発生します。

フォームイベントの各フェーズに対して、オブジェクトメソッド、フォームメソッドが実行されます。

サブテーブルのサブフォーム

サブフォーム内のサブレコード表示のフォームイベントには、以下のフェーズがあります。

「Before」：このフェーズは、親フォームのBeforeフェーズの実行前に新しいサブレコードが追加された場合に発生します。

「During」：このフェーズは、各サブレコードが最初に表示された場合と、その修正時に発生します。親レコードのDuringフェーズは、サブフォームのDuringフェーズの後に発生します。

「After」：このフェーズは、親レコードのAfterフェーズの前にそれぞれのサブレコードに対して発生します。

「Deactivated」：このフェーズは、フルページフォームを含んだウインドウが背後に配置されている場合に、サブレコードのフルページフォーム内でのみ発生します。

「Activated」：このフェーズは、フルページフォームが最前面になった場合に、サブレコードのフルページフォーム内でのみ発生します。

「Outside Call」：このフェーズは、呼び出しが最前面のウインドウ内のフルページフォームを含んだプロセスで作成された場合に、サブレコードのフルページフォーム内でのみ発生します。

サブレコードのフルページフォームが受け入れられた時点では、Afterフェーズはありません。

フォームイベントの各フェーズに対して、オブジェクトメソッド、フォームメソッドが実行されます。

注：ご覧のように、リレートされたレコードとサブフォーム内のサブレコードのフォームイベントは同一のものではありません。

「ユーザ」モードのレコード表示

「ユーザ」モードのレコード表示のフォームイベントには以下の4つのフェーズがあります。

「In Header」：このフェーズは、ヘッダエリアが表示される前に発生します。タイトルの表示やレコードのサマリー（小計）情報を生成する場合にも使用します。

「Before&During」：このフェーズは、BeforeフェーズとDuringフェーズが同時に“True（真）”である場合に発生します。レコードが表示されるたびに1回発生します。

「During」：このフェーズは、“リスト更新”時にレコードのフィールドが修正された場合にのみ発生します。

「After」：このフェーズは、“リスト更新”時にレコードの修正を受け入れた場合にのみ発生します。

フォームイベントの各フェーズごとに、フォームメソッドが実行されます。

MODIFY SELECTIONコマンドとDISPLAY SELECTIONコマンド

MODIFY SELECTIONコマンドまたは**DISPLAY SELECTION**コマンドを使用してレコードを表示すると、フォームイベント中に7つのフェーズを使用することができます。レコードをダブルクリックすると、データ入力の規則に従って入力フォームメソッドを実行します。

「Before」：このフェーズは、レコード表示前に一度だけ発生します。

「In Header」：このフェーズは、ヘッダエリア表示前に一度だけ発生します。タイトルやサマリー（小計）情報を表示する場合に使用します。

「Before&During」：このフェーズは、BeforeフェーズとDuringフェーズが同時に“True”である場合に発生します。各レコードが表示されるたびに発生します。

「During」：このフェーズは、Beforeフェーズが“False”でDuringフェーズが“True”の場合に発生します。メニューの選択、ボタンのクリック、レコードのダブルクリックを行った場合に発生します。

「Deactivated」：このフェーズは、レコードリストを含んだウインドウが背後に配置されるたびに発生します。

「Activated」：このフェーズは、レコードリストを含んだウインドウが最前面に配置されるたびに発生します。

「Outside Call」：このフェーズは、別のプロセスから出力フォームに含まれているプロセスの呼び出しがある場合に発生します。

フォームイベントの各フェーズごとに、フォームメソッドが実行されます。

フォームによる書き出し実行時

フォームによるレコードの書き出しの場合は、フォームイベント中のフェーズがひとつだけあります。

「Before」：このフェーズは、レコードが書き出される前に一度だけ発生します。

フォームイベント中にオブジェクトメソッド、フォームメソッドを実行します。このフォームイベントを使用して、書き出す前のデータを修正することができます。例えば、複数のフィールドを連結する、または固定長データから埋め込み文字のスペースを取り除くといった使用方法があります。

フォームによる読み込み実行時

フォームによるレコードの読み込みの場合は、フォームイベント中のフェーズがひとつだけあります。

「After」：このフェーズは、レコードが読み込みディスクに読み込まれる前に一度だけ発生します。

フォームイベント中にオブジェクトメソッド、フォームメソッドを実行します。このフォームイベントを使用して、読み込んだデータを修正することができます。例えば、固定長データから埋め込み文字のスペースを取り除くといった使用方法があります。

フォームレポート

レポートを印刷する場合には、以下のようなフォームイベントのフェーズがあります。

「In Header」：このフェーズは、ページごとにフォームヘッダエリアまたはブレイクヘッダエリアが、印刷時に発生します。ひとつのページにひとつのヘッダと複数のブレイクヘッダがあります。複数のブレイクヘッダが存在する場合は、**Level**関数を使用して、どのブレイクヘッダが印刷されるのかを判定することができます。また、**Before selection**関数を使用すると、レポートの最初のヘッダを判定することができます。

「Before」：このフェーズは、ディテイルエリアにおいてレコードの印刷ごとに一度発生します。

「During」：このフェーズは、ディテイルエリアのレコードの印刷ごとに一度発生します（Beforeフェーズに準じます）。

「In Break」：このフェーズは、ブレイクエリアでのブレイクの印刷ごとに発生します。Level関数を使用してブレイクレベルを判定することができます。

「In Footer」：このフェーズは、ページごとのフッタエリアの印刷時に発生します。**End selection**関数を使用してレポートの最後のフッタを判定することができます。

フォームイベントの各フェーズごとに、フォームメソッドを実行します。例えば、ヘッダエリアに変数を設定すると、そのオブジェクトメソッドはヘッダが印刷される場合にのみ実行されます。

レポート処理を制御するようなオブジェクトメソッドをなるべく使用することをお勧めします。それぞれの印刷エリアにオブジェクトメソッドを持ったフォームオブジェクトを設定するだけで適切なタイミングでオブジェクトメソッドを実行します。

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はなく、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コマンド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コマンドを選択すると、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コマンドに割り当てます。メニューコマンドが選択されると、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ひとつです。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。

カスタムメニューコマンドは、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コマンドは、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ひとつです。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箇所でも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。サブルーチンと呼び出す側は、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。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)

ALERT([人事]名字)

` [人事]名字の内容が “williams” であるものとする

`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 “処理” に “処理” を行わせるだけでなく、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:=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
\$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)
      $vIQueryResult:=Records in selection([友人関係])
      If ($vIQueryResult > 0)
        $vtTheWholeStory:=[友人関係]名前+"の子供である "+$vtTheWholeStory
        $vsName:=[友人関係]名前
      End if
    Until ($vIQueryResult=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是以下の通りです。

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）。

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、以下のとおりです。

1. 「エクスプローラ」ウインドウを開く。
2. 「メソッド」タブを選択する。
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. 「編集」ボタンをクリックするか、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 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 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

4D Compilerコンパイルし、4D Engine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。

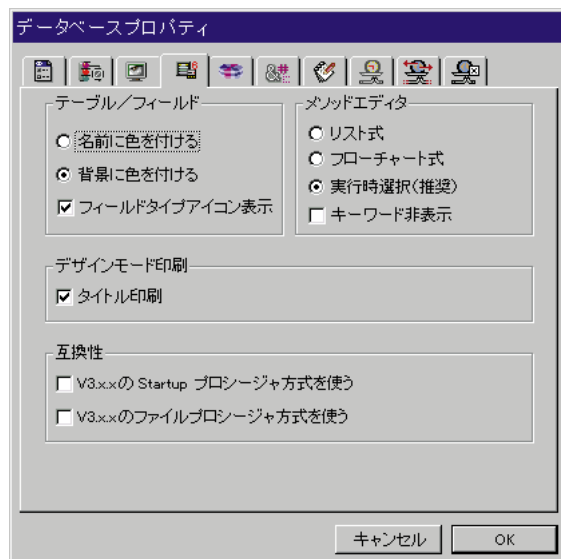
条件が一致しない（システムリソースがない等の）場合には、明示的にQUIT 4Dコマンドを呼び出して、データベースを開かないようにする。

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

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、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

4D Compilerでコンパイルし、4D Engine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」メニューコマンドを選択した場合

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ひとつずつアポートします。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ことを忘はいけません。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 ; \$vIState ;

 vIElapsedTime ; \$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 (\$vIProcess ; 1 ; **Count tasks**)

PROCESS PROPERTIES (\$vIProcess ; \$vsProcessName ; \$vIState
 ; \$vIElapsedTime ; \$vbVisible)

If ((((\$vsProcessName="ML_@") | (\$vsProcessName="M_@"))) & (\$vIState>=0))

 \$vbDone:=**False**

 <>vtIPMessage:="QUIT"

BRING TO FRONT (\$vIProcess)

CALL PROCESS (\$vIProcess)

 \$vhStart=**Current time**

Repeat

DELAY PROCESS(**Current process** ; 60)

Until((**Process state**(\$vIProcess)<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コマンド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」メニューコマンド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。

アプリケーションとは、特定の要求を満たすために作成する一種のデータベースです。アプリケーションは、その使用を容易にするためのユーザインターフェースを備えています。4th Dimensionの機能は、そのアプリケーションの機能に依ります。しかし、4th Dimensionで作成したアプリケーションは、従来のプログラミング言語で作成されたものに比べて無駄なく簡単に操作することができます。4th Dimensionは、以下のようなアプリケーションを作成することができます。

請求書発行システム

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

インターネットまたはイントラネット経由の共有データベース

ひとつのアプリケーションがこれらのシステムすべてを含むことも可能です。これらのようなアプリケーションは、データベースを使用するのが普通です。加えて、4th Dimensionが持つツールにより、以下のような先進的なアプリケーションを作成することができます。

ドキュメント管理システム

グラフィックイメージ管理システム

カタログ発行アプリケーション

シリアルデバイス制御と監視システム

電子メールシステム

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、ミュージックコレクションのようなリスト

アプリケーションは、一般に「ユーザ」モードで使用するデータベースとして作成されます。カスタマイズするにしたがって、データベースからアプリケーションへと“拡張”させることができます。アプリケーションが通常のデータベースと異なる点は、データ

ベースを管理するのに必要なプロセスはユーザから見えないようになっていることです。データベース管理は自動化され、ユーザはメニューを使用して処理を実行します。

「ユーザ」モードのデータベースで何らかの処理を実行する場合には、その手順を記憶させる必要があります。アプリケーションは「カスタム」モードで起動します。「カスタム」モードでは、「ユーザ」モードで自動化される機能を管理する必要があります。これには、以下のようなものがあります。

「テーブル操作」：「テーブル/フォーム選択」ダイアログボックスと“テーブルリスト”ウインドウは使用することができません。テーブル間の操作を制御するには、メニューコマンドやオブジェクトメソッドを用います。

「メニュー」：「カスタム」モードには、デフォルトでは「終了」メニューコマンドのある「ファイル」メニューと「編集」メニュー、「ヘルプ」メニュー（Windowsのみ）、「アップル」メニュー（Macintoshのみ）だけが用意されています。アプリケーションに多くのメニューを用いる場合は、4Dメソッドを使用してメニューを作成し、管理しなければなりません。

「エディタ」：「クエリ」エディタや「並び替え」エディタ等のエディタが、「カスタム」モードでは自動的に使用することはできません。「カスタム」モードでこれらのエディタを用いたい場合は、4Dメソッドを使用して呼び出す必要があります。

次の節では、プログラミング言語によるデータベース使用の自動化について説明します。

カスタムメニューの例

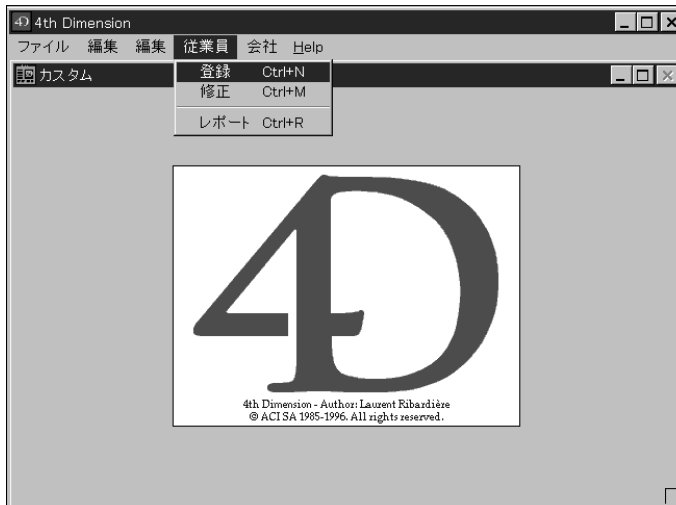
カスタムメニューは、データベースの主要なインタフェースです。これは、データベースの習得や使用を容易にします。カスタムメニューは、「メニュー」エディタを使用して、各メニューコマンドにメソッド名を連結するだけで簡単に作成することができます。

「ユーザの認識」の節では、ユーザがメニューコマンドを選択したときに起こる事象について説明します。次の節「メニュー表示の裏側」では、この作業を実現するための設計について説明します。事例はとても簡単ですが、カスタムメニューを用いると、データベースの使用や習得がいかに楽になるかがわかります。ユーザは、「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、ユーザの必要に応じた作業だけに目を向ければいいのです。

ユーザの認識

ユーザは新しい従業員を追加するために、“登録”というカスタムのメニューコマンドを選択します。

注：ここで使用している図の中には、バージョン3の4th Dimensionをもとに作成されたものが一部含まれています。



従業員テーブルの入力フォームが表示されます。

カスタム

登録

3

削除

キャンセル

個人情報

名字

名前

会社名

都道府

区市郡

町村

郵便番号

ユ - ザは、従業員の名字を入力し、以下のフィールドへ移動します。

カスタム

登録

3

削除

キャンセル

個人情報

名字 John

名前

会社名

都道府

区市郡

町村

郵便番号

ユ - ザは、従業員の名前を入力し、以下のフィールドへ移動します。

カスタム

登録

3

削除

キャンセル

個人情報

名字 John

名前 Smith

会社名

都道府

区市郡

町村

郵便番号

ユ - ザは、名前が大文字に変換されたことを確認します。

カスタム

登録

個人情報

名字 John

名前 SMITH

会社名

都道府

区市郡

町村

郵便番号

削除

キャンセル

ユ - ザは、レコ - ドの入力を終了すると、「登録」ボタンをクリックします。

カスタム

登録

個人情報

名字 John

名前 SMITH

会社名 刀根エンタープライズ

都道府 東京都

区市郡 豊島区

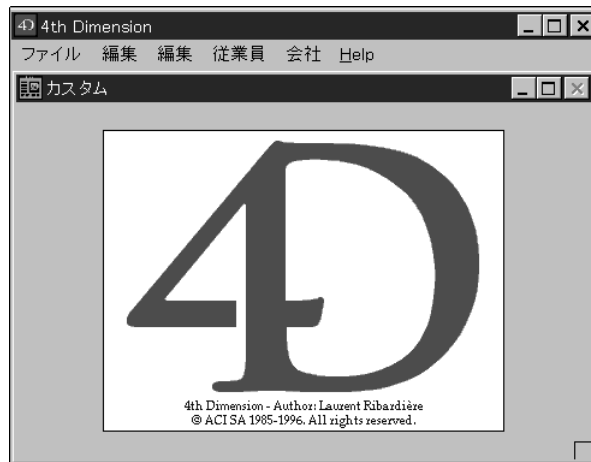
町村 南大塚

郵便番号 170

削除

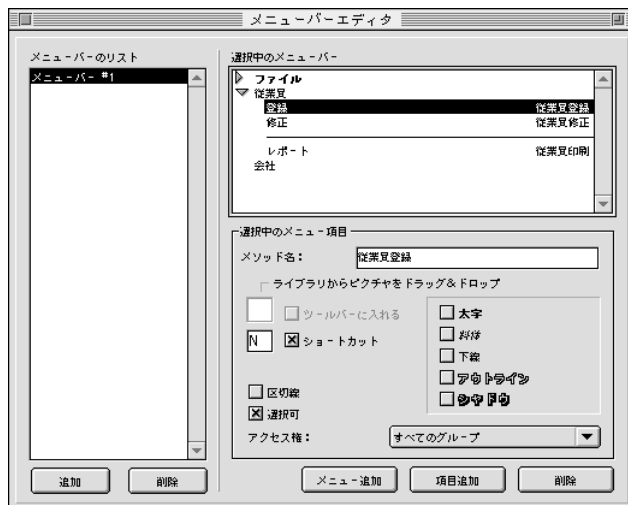
キャンセル

空の入力フォームが表示されるため、ユーザは「キャンセル」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。



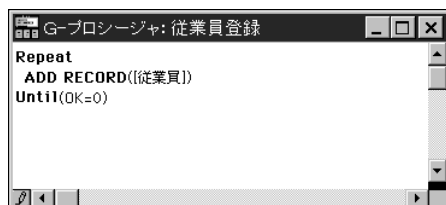
メニュー表示の裏側

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



メニューコマンドの“登録”には、“従業員登録”という名前のプロジェクトメソッドが付随しています。このメソッドは、「デザイン」モードの「メソッド」エディタで作成されたものです。

ユーザがこのメニューアイテムを選択すると、メソッド“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」メニューコマンドと同様の動作を行います。これにより、入力フォームがユーザに表示され、ユーザが新規レコードを追加できるようになります。ユーザがレコードを保存すると、空の入力フォームが表示されます。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、以下のことが起こります。

「名前」フィールドにはオブジェクトメソッドがないため、何も実行されません。

「名字」フィールドにはオブジェクトメソッドがあるため、「フォーム」エディタと「メソッド」エディタを使用して、このオブジェクトメソッドが「デザイン」モードに作成されています。メソッドは以下を実行します。

名前:=Uppercase (名前)

これは、名前のフィールドを大文字に変換します。

レコードが入力され、ユーザが以下のフォームで「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。

それ以上実行するステートメントがないため、“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」モードとアプリケーションの比較

「ユーザ」モードで処理を実行する方法と、プログラミング言語で同じ処理を実行する方法とを比較してみましょう。実行する処理は、両方ともレコードの検索、ソート、およびレポートの印刷です。この章の例では、「ユーザ」モードと「カスタム」モードの両方で4th Dimensionの組み込みエディタを使用します。この場合、プログラミング言語は処理を部分的に自動化します。

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処理を示します。また“組み込みエディタを使用したアプリケーション”の節では、アプリケーションで実行される同じ処理を表示します。

どちらも同じ処理を実行しますが、2番目の節の行程ではプログラミング言語の使用により自動化されている点に注目してください。

クエリ		
すべて表示		⌘G
サブセットを表示		⌘H
クエリ...		
フォームによるクエリ...		⌘S
クエリ&修正...		⌘L
フォーミュラによるクエリ...		
並び替え...		⌘T

「ユーザ」モードにおけるデータベースの使用

ユーザは、「クエリ」メニューから「クエリ...」を選択します。
「クエリ」エディタが表示されます。



クエリ		
すべて表示		⌘G
サブセットを表示		⌘H
クエリ...		
フォームによるクエリ...		⌘S
クエリ&修正...		⌘L
フォーミュラによるクエリ...		
並び替え...		⌘T

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



ユーザは、ソート条件を入力し「並び替え」ボタンをクリックします。
最後にユーザは、「ファイル」メニューから「プリント...」を選択します。
「プリントフォーム選択」ダイアログボックスが表示されます。

この場合、ユーザはどのフォームを選択したかを覚えておく必要があります。フォームを選択し、「OK」ボタンをクリックします。

「用紙設定」ダイアログボックスが表示されます。

ユーザが「OK」ボタンをクリックすると、レポートが印刷されます。

組み込みエディタを使ったアプリケーションの使用

前の作業がカスタムモードではどのように行われるかを見てみましょう。

ユーザは、カスタムメニューから該当するメニューコマンド「レポート」を選択します。

この時点でも、ユーザにとってアプリケーションの使用はより簡単になっています。ユーザは、最初の手順が検索であることを認識している必要はありません。

メソッド“従業員印刷”は、メニューコマンドに付随しています。これは以下のようなメソッドです。

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“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、ソートが以下のステップであることを知っておく必要がないことに注目してください。

「並び替え」エディタが表示されます。



ユーザがソート条件を入力し「並び替え」ボタンをクリックすると、ソートが実行されます。

次にメソッド“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、次に実行すべきことを記憶しておく必要がありません。処理手順は、既に定義されています。

最後にメソッド“従業員印刷”の最終行が実行されます。

PRINT SELECTION ([従業員])

「用紙設定」ダイアログボックスが表示されます。

ユーザが「OK」ボタンをクリックすると、レポートが印刷されます。

高度に自動化されたアプリケーション

前述の比較で使用したものと同一コマンドで、より高度に自動化されたアプリケーションを作成することができます。

以下の比較では、最初に前例のアプリケーションで組み込みエディタを使用したものを示します。その次に、プログラミング言語で完全に処理を自動化した例を示します。この場合、ユーザが唯一知っておかなければならないことはレポートを作成するために「レポート」メニューを選択しなければならないということだけです。必要な動作をより完全に記述するために同一コマンドがどのように使用されているか注目してください。

完全に自動化されたアプリケーション

ユーザは、メソッドを開始するカスタムのメニューコマンドを選択します。

メソッド“従業員印刷”は、メニューコマンドに付随しています。このメソッドは、以下のようになっています。

```
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**コマンドに指定した検索条件が実行されます。ユーザは、何もする必要がありません。

メソッド“従業員印刷”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; >)
```

この場合も「並べ替え」ダイアログボックスは表示されずにソートがすぐに実行されます。ここでもユーザは何もする必要がありません。

メソッド“従業員印刷”の最終行が実行されます。

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

PRINT SELECTIONコマンドは、「用紙設定」ダイアログボックスを表示しません。引数にアスタリスク(*)が指定されているため、出力フォームの作成時点に指定した値を用いて印刷します。

「ユーザ」モードメニューと同等のコマンド

これまでの例でもわかるように、プログラミング言語には、「ユーザ」モードのメニューコマンドと同じ動作を実行するコマンドが数多く存在します。これらのコマンドは、データベースをカスタマイズをする上で便利な機能を提供します。

各メニューコマンドは、動作を実行させたり、エディタやダイアログボックスを表示します。コマンドを使用して、エディタと動作を“結合”したカスタムシーケンスで、自動的に反復処理を実行することができます。また、これらのコマンドは、エディタ等からユーザによる割り込みを受けなくても、引数で指定されたとおりに検索等の動作を自動的に実行することができます。

以下の表は、「ユーザ」モードのメニューコマンドと、それに相当するコマンドと使用例を示したものです。

メニューコマンド	コマンド	使用例
フォーミュラで更新	APPLY TO SELECTION	APPLY TO SELECTION ([従業員];[従業員]給与:= [従業員]給与*1.05)
テーブル/フォーム選択	DEFAULT TABLE	DEFAULT TABLE ([従業員])
	INPUT FORM	INPUT FORM ([従業員];入力)
	OUTPUT FORM	OUTPUT FORM ([従業員];出力)
ASCIIテーブル編集	USE ASCII MAP	USE ASCII MAP ("テーブル名")
データ書き出し	EXPORT TEXT	EXPORT TEXT ([従業員];"")
チャート	GRAPH TABLE	GRAPH TABLE ([従業員])
データ読み込み	IMPORT TEXT	IMPORT TEXT ([従業員];"")
ラベル	PRINT LABEL	PRINT LABEL ([従業員];"")
レコード修正	MODIFY RECORD	MODIFY RECORD ([従業員])
新規レコード	ADD RECORD	ADD RECORD ([従業員])
プリント	PRINT SELECTION	PRINT SELECTION ([従業員])
クイックレポート	REPORT	REPORT ([従業員];"")
フォーミュラによるクエリ	QUERY BY FORMULA	QUERY BY FORMULA ([従業員]年齢=20)
フォームによるクエリ	QUERY BY EXAMPLE	QUERY BY EXAMPLE([従業員];"検索")
クエリ	QUERY	QUERY ([従業員])
すべて表示	ALL RECORDS	ALL RECORDS ([従業員])
	MODIFY SELECTION	MODIFY SELECTION ([従業員])
並び替え	ORDER BY	ORDER BY ([従業員])

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、はじめは気付かなかったさまざまな機能を発見することでしょう。4D開発環境にその他ツールやプラグインを追加し、標準の4Dの機能を拡張することができます。

開発用ツール

ACI社からアプリケーション開発に使用できる各種ツールが提供されています。これらのツールを使用して、別のデータベースからファイルやフィールド等のオブジェクトを移動したり、データベースのシンタックスチェックを行うことができます。次のようなツールがあります。

「4D Insider」：このツールは、別のデータベースからテーブル、フィールド、メソッド、メニューバー、リスト、外部モジュール等を移動することができます。また、4th Dimensionデータベースのクロスリファレンスを作成することもできます。このツールはプロシージャや変数、コマンド、プラグイン、ストラクチャ、リスト、フォーム等を表示したり、印刷する場合に使用します。

「4D Compiler」：このツールは、アセンブラ命令でメソッドやオブジェクトメソッドを解析します。このツールはデータベースの処理速度を高速にし、コードの整合性をチェックします。また、論理的やシンタックス上の矛盾を発見したり、データベースを保護することができます。

4D プラグイン

4D アプリケーションの機能は、4D開発環境に専門のプラグインを追加することにより拡張することができます。

ACIは、以下のプロダクティビティプラグインを提供しています。

4D Calc：スプレッドシート

4D Draw：グラフィカル描画プログラム

4D Write：ワードプロセッサ

ACIは、また以下のコネクティビティプラグインも提供しています。

4D ODBC：ODBC経由の接続

4D for Oracle：ORACLEデータベースとの接続

4D SQL Server : SYBASE SQL ServerおよびMicrosoft SQL Serverとの接続

4D Open : 分散4D情報システムを構築するための4D間接続

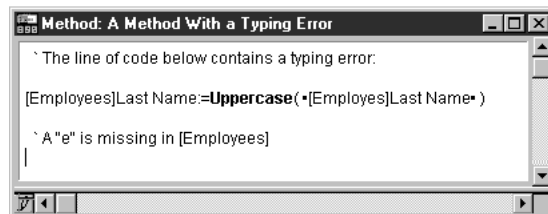
なぜデバッグを使用するか？

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ありがちなエラーには、入力エラー、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、実行時エラー等、いくつかの種類があります。

入力エラー

入力エラーは、「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、入力エラーを表しています。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。



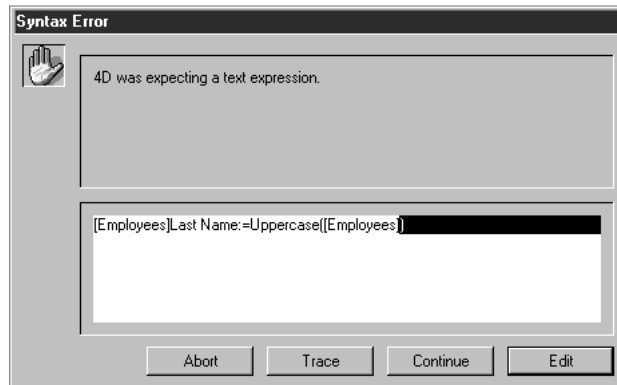
注：コメントは、このマニュアルで説明するために手動で入力しています。4Dがエラーの発生位置で挿入するのは（・）だけです。

このようになった場合には、入力エラーを修正し、（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。

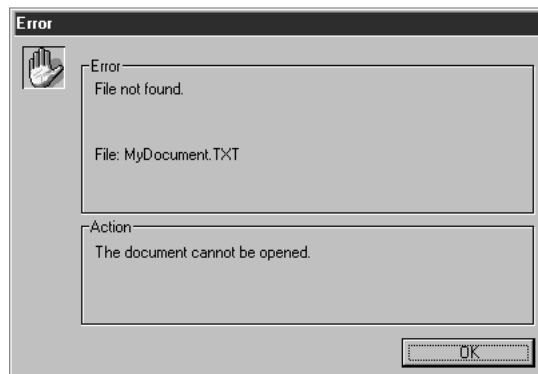


このウインドウでのエラーは、テキスト式が対象となっている**Uppercase**関数にテーブル名が渡されていることです。このウインドウとボタンの詳細については、後述の「「シNTAXエラー」ウインドウ」の節を参照してください。

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開か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**コマンドの説明箇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uggを使用します。

これまでに説明しているエラーは、入力エラーを除いて、「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。

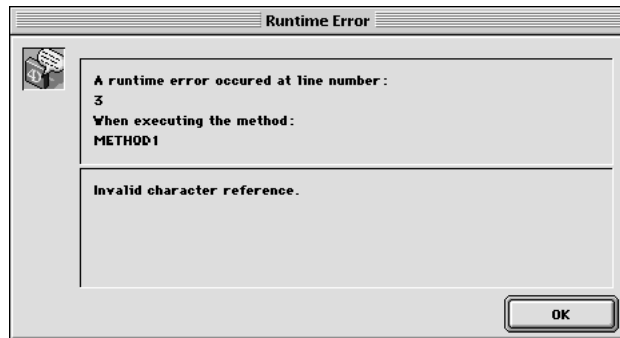
設計やロジックのエラーには、次のような場合もあります。

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。

実行時エラー

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、日常的に発生します。相当行数（数百行程度）の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の状態です。

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になります。

デバッガを使用するもうひとつの理由は、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、実利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。

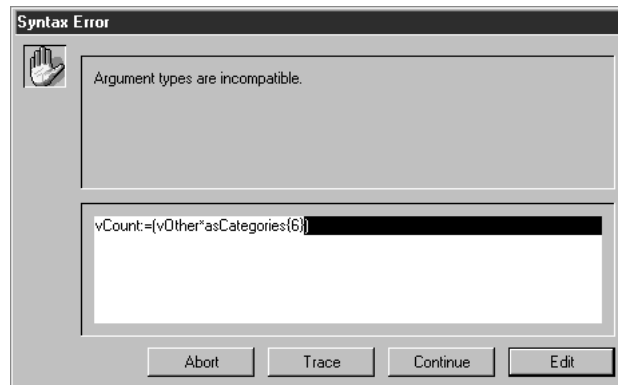
「シンタックスエラー」ウインドウ

「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/デバッグ」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」エディタで開かれ、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。

デバグガ

デバグガという用語は、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、デバグガを使用します。デバグガでは、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。

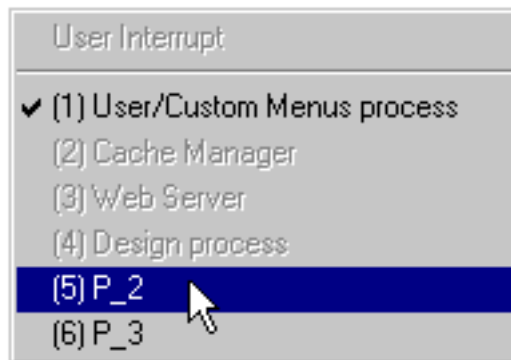
次のような方法を使用して、「デバグ」ウインドウでメソッドを表示し、トレースすることができます。

「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、デバグボタンをクリックした場合（ユーザーモード）。

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。

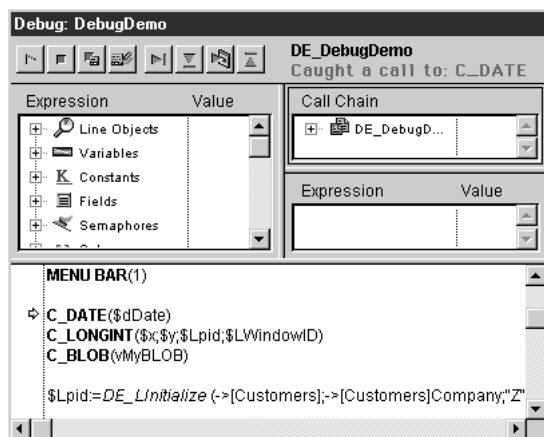


ランタイムエクスプローラのプロセスページを選択した後、トレースボタンをクリックした場合（後述の「表示されないプロセスまたはコードを実行していないプロセスのトレース」の節を参照してください）。

「メゾットエディタ」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。



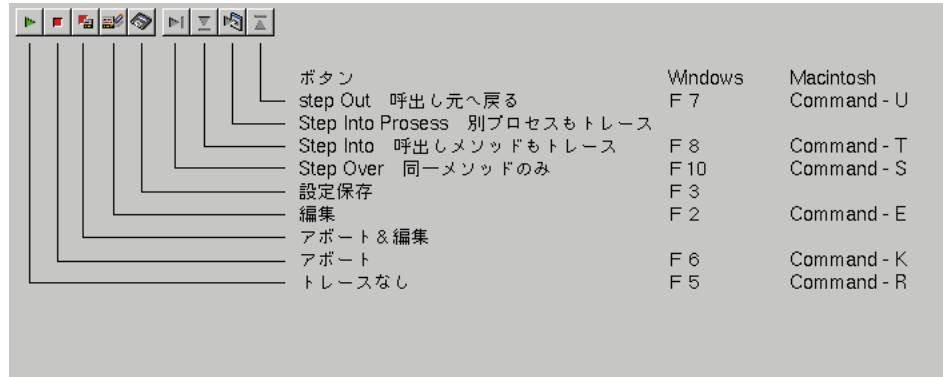
必要に応じて「デバッグ」ウインドウを移動、またはその内部にあるウインドウ枠のサイズを変更することができます。

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「カスタム表現式/値」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ひとつの「デバッグ」ウインドウを開くことができます。

「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」ウインドウで開きます。

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、再び開か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「カスタム表現式/値」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ではなく、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 Process（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」ウインドウが開かれ、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。

「デフォルト表現式 / 値」エリア

「メソッド連鎖」エリア

「カスタム表現式 / 値」エリア

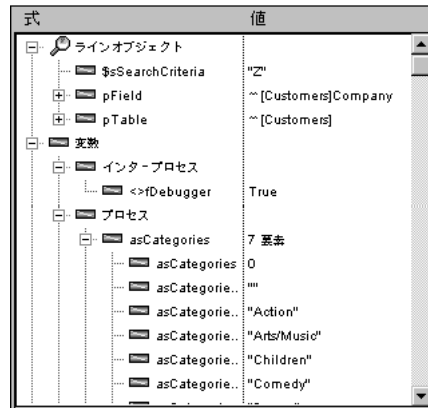
「ソースコード」エリア

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。

「デフォルト表現式／値」エリア

「デフォルト表現式／値」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。



「デフォルト表現式／値」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「カスタム表現式 / 値」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クションの利用に關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クションも一覧表示します。各命名セクションでは、「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）及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せんが（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。デバッグ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。デバッグ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「デフォルト表現式 / 値」エリアから「カスタム表現式 / 値」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）で使用されているか、または

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、ひと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「カスタム表現式/値」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。

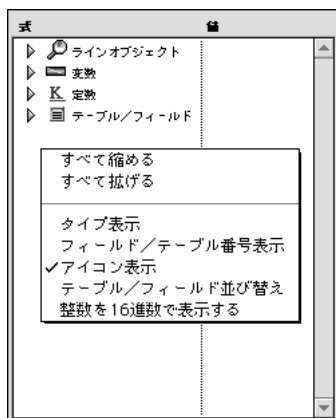
「スピード」メニュー

「デフォルト表現式 / 値」エリアの「スピード」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「デフォルト表現式 / 値」エリア内の任意の位置をクリックする。

Macintoshでは、「デフォルト表現式 / 値」エリアの任意の位置で「control+クリック」を実行する。

「デフォルト表現式 / 値」エリアの「スピード」メニューが、次のように表示されます。



「すべて縮める」：「デフォルト表現式 / 値」エリアの階層リストのすべてのレベルを縮小します。

「すべて広げる」：「デフォルト表現式 / 値」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「デフォルト表現式 / 値」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。

表現式	値
[Comp Platforms] : [10]	0 selected records
[Comp References] : [11]	0 selected records
[Companies] : [2]	1676 selected records
[Companies]City : [2]3 : Alpha(40)	"Santa Fe"
[Companies]Company ID : [2]1 : Long Integer	5
[Companies]Company Name : [2]2 : Alpha(40)	"ACUMEN, Inc."
[Companies]Company Profile : [2]9 : Text	"ACUMEN Inc. is the pu..."
[Companies]Consulting Info : [2]13 : Text	""
[Companies]Country : [2]6 : Alpha(20)	"USA"
[Companies]Expertise Info : [2]14 : Text	"ACUMEN's expertise is..."

「メソッド連鎖」エリア

ひと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な非常に便利です。「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。

メソッド連鎖	
M_BitTestDemo	未定義
DE_Initialize	0
\$0	~ [Customers]
\$1	~ [Customers]Company
\$2	Z
\$3	
DE_DebugDemo	

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_Initialize」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「カスタム表現式 / 値」エリアにドラッグ&ドロップすることができます。

「カスタム表現式 / 値」エリア

「メソッド連鎖」エリアのすぐ下にあるのは、「カスタム表現式 / 値」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、ひと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表示されています。

式	値
OK	1
pField	~[Customers]Company
[Customers]Company	""
Records in selection([Customers])	0
\$sSearchCriteria	"Z@"

新しい式の挿入

次のように「カスタム表現式 / 値」エリアに式を追加して、評価することができます。

「デフォルト表現式 / 値」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、「カスタム表現式 / 値」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。

「カスタム表現式 / 値」エリアの「スピード」メニュー

式を入力し、編集する場合には、「カスタム表現式 / 値」エリアの「スピード」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「スピード」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、「カスタム表現式 / 値」エリアの任意の位置をクリックする。

Macintoshでは、「カスタム表現式 / 値」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... コマンド挿入	▶
すべて削除 すべて縮める すべて拡げる	
タイプ表示 フィールド、テーブル番号表示 ✓アイコン表示 テーブル／フィールド並び替え 整数を16進数で表示する	

「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 / すべて広げる」：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。

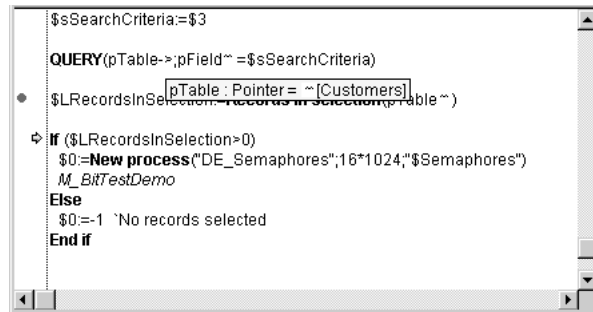
「ソースコード」エリア

「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「カスタム表現式 / 値」エリアにコピーされます。選択されたテキストは、「カスタム表現式/値」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「カスタム表現式 / 値」エリア内へドラッグ&ドロップします。

（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。

デバッガのブレイクポイントの設定

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。

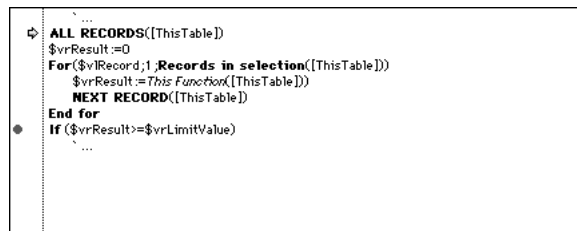
次に、プログラムカウンタを行「**ALL RECORDS** ([This Table])」に設定し、次のようなコードを実行していると想定してみます。

```
`...  
ALL RECORDS ([This Table])  
$vrResult:=0  
For ($vIRecord ; 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、現在の実行をアポ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ひとつの方法ですが、「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、後で再び開くと、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。

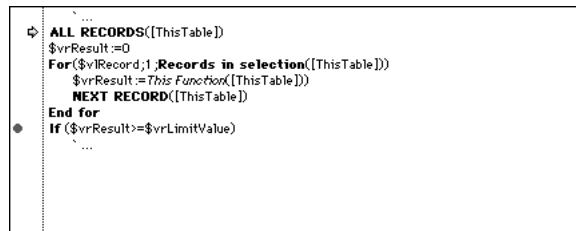
永続的ブレークポイントをまだ使用する場合には、これを残しておきたい場合もあります。永続的ブレークポイントを編集すると、一時的に使用不可能にすることができます。編集方法については、「ブレークポイント」の節で説明しています。

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がメソッドエディターウィ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（デバッグも同様に）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。

ブレイクポイントの編集

「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「ブレイクポイントプロパティ」ウインドウにアクセスできます。

既存のブレイクポイントをクリックすると、そのブレイクポイントについてのウインドウが表示されます。

ブレイクポイントが設定されていない行をクリックすると、デバッガはブレイクポイントを作成し、新しく作成されたブレイクポイントに関するウインドウを表示します。

「ブレイ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イ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、永続的ブレイクポイントを作成します。永続的ブレイクポイントは、「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。一時的ブレイ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。一時的ブレイクポイントは、メソッド中で一度だけブレイクしたい場合に役立ちます。一時的ブレイクポイントは、「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イクポイントを直接設定することもできます。

注：一時的ブレイ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り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に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。

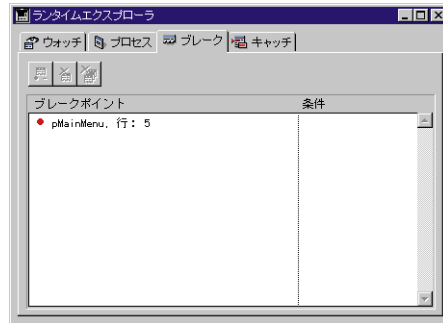
「デバッグ」ウインドウ内でブレークポイントを作成、または編集することができます。さらに「デザイン」モードの「ブレークリスト」ウインドウを使用して、既存のブレークポイントを編集することもできます。詳細については、次節を参照してください。

ブレークリスト

ブレークリストは「デバッグウインドウ」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します。



ブレークリストは2つの欄から構成されています。

- ・左の欄には、ブレークポイントの有効／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」ウインドウ又はメゾットエディタを使用）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ゾットが表示しているメゾットエディタウインドウをオープンします。

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ゾット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方法は：

1. 右の欄の任意の場所をクリックします
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）を入力します。

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します（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ます。

3.deleteキーを押す、または削除ボタンをクリックします。

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、またはスピードメニューから「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

コマンド中断リストは4Dコマンドを中断するブレークを付け加えることが出来る「ランタイムエクプローラ」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。

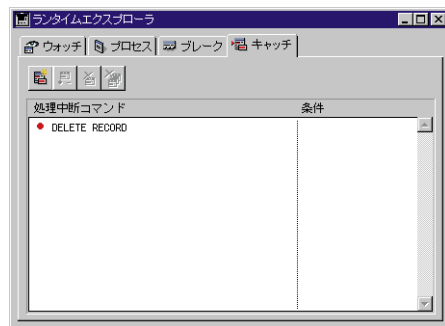
少し経験を積めば、ブレークポイントとコマンドの中断とを組み合わせ使用できるようになります。

中断コマンドページを開くには

1. デザインモードにします。

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します。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します（フローティングパレットは通常、前面に表示されたままになっています）。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。または、Windowsでは、Ctrlキーを押しながらShiftキーとF9キーを押します。Macintoshでは、commandキーを押しながらshiftキーとF9キーを押します。

3. コマンドリストで中断コマンドを表示し、中断タブコントロールをクリックします。



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。

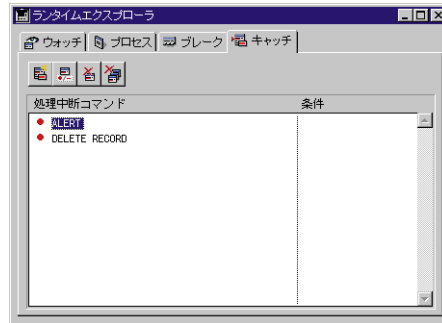
- ・左の欄には、中断コマンドの有効 / 無効状況と、コマンド名が表示されます。
- ・右の欄には、中断コマンドに関連する条件（ある場合）が表示されます。

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. 「ブレイクリスト」メニューから「新規ブレイクポイント追加」ボタンをクリックします。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）をダブルクリックします。

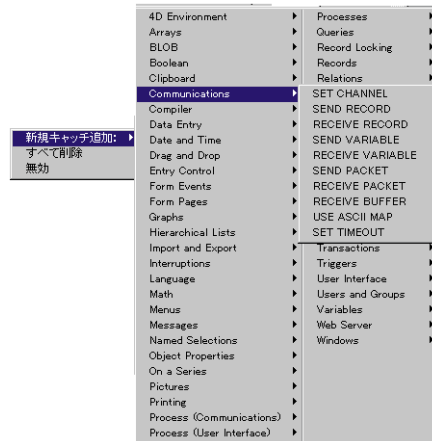
ともに、リストにコマンド名が新しく追加され、デフォルトとして**ALERT**コマンドと表示されます。このコマンド名は編集できる状態になっています。



2. 中断しようとするコマンド名を入力します。
3. enterキーまたはreturnキーを押し、選択を確定します。

または、

1. 右マウスボタンをクリック (Windows) または「control+クリック」(Macintosh) すると、スピード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します。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します (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します。

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します（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ます。
3. 「有効 / 無効」ボタンをクリックするか、またはスピードメニューから「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します（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ます。
3. Delete キーを押す、または削除ボタンをクリックします。

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、またはスピードメニューから「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）を入力します。

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します（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ます。
3. 「ブレークリスト」メニュー、またはスピードメニューから「有効 / 無効」を選択します。

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します（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ます。
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」を選択します。

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、またはスピードメニューから「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します。
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）を入力します。

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。

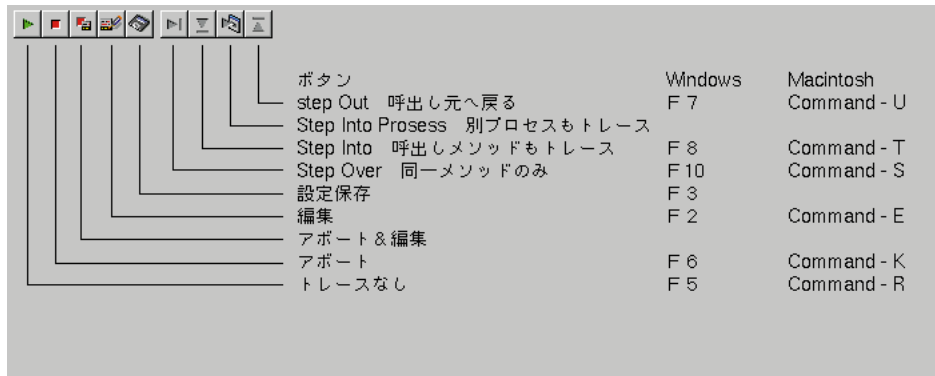
中断コマンドやブレークポイントを無効にすると、削除するのと同じ効果があります。実行中、デバッガはこの項目に対して時間をほとんど割きません。項目を無効にする利点は、再度その項目が必要となった場合に再作成しなくてもいいという点です。

デバッグのショートカット

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 Process (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。

「デフォルト表現式 / 値」エリアでのキーボードショートカット

「デフォルト表現式 / 値」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、「デフォルト表現式 / 値」エリアの「スピード」メニューがプルダウンされます。

「デフォルト表現式 / 値」エリア内の項目をダブルクリックすると、その項目が「カスタム表現式 / 値」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。

「カスタム表現式 / 値」エリアでのキーボードショートカット

「カスタム表現式 / 値」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、「カスタム表現式 / 値」エリアの「スピード」メニューがプルダウンされます。

「カスタム表現式 / 値」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、ブレイクポイントが設定される (永続的ブレイクポイントの場合) またはブレイ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、一時的ブレイ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、新しいブレイクポイントや既存のブレイクポイントの「ブレイクポイントプロパティ」ウインドウが表示されます。

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、それをカスタム表現式&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」メニューコマンドへのショートカットとして使用します。

表示されないプロセスやコードを実行していないプロセスのトレース（4th Dimensionのみ）

デバッガの「ステップ（新規プロセスもトレース）」ボタンを使用すると、**New Process** 関数を使用してプロセスを開始した時点から、そのプロセスをトレースすることができます。

また、プロセスが開始されてからかなり経過した後でプロセスをトレースする場合もあります。

プロセスで、表示されているウインドウが少なくとも1個あり、そのウインドウが最も手前に表示されている場合に、そのウインドウ内で「Alt+クリック（Windows）」または「option+クリック（Macintosh）」を押すと、そのプロセスのトレースモードが開始されます。

「Alt+クリック」または「option+クリック」は、プロセスが次のような状態の場合には、使用することが困難な場合もあります。

プロセスがコードを実行しているが、そのプロセスのウインドウが他のウインドウの後ろにあるため、それらを移動してプロセスにアクセスしたい場合

プロセスがコードを実行しているが、ユーザインタフェースがない（ウインドウがない）場合

プロセスがデータ入力の状態にあり、イベントを待っている場合（*）

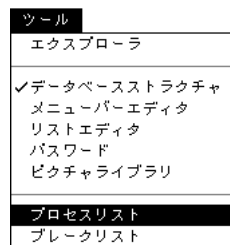
プロセスが現在一時停止されている場合（*）

プロセスが現在遅延されている場合（*）

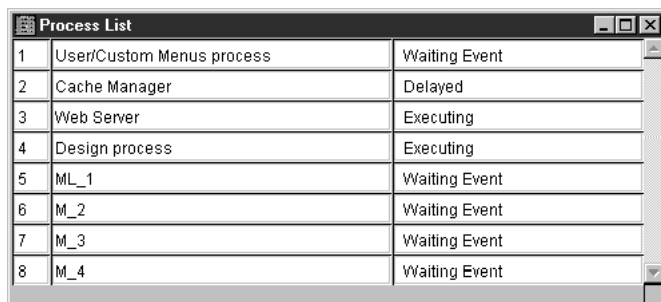
（*）は、プロセスが実行されているものの、コードを実行していない状態です。

4Dにはプロセスのトレースを開始する別の便利な方法もあります。この方法では、プロセスが表示されている必要もなく、プロセスがコードを実行している必要もありません。

1. まだ「デザイン」モードになっていない場合には、「デザイン」モードに切り替える。
2. 「ツール」メニューから「プロセスリスト」を選択する。

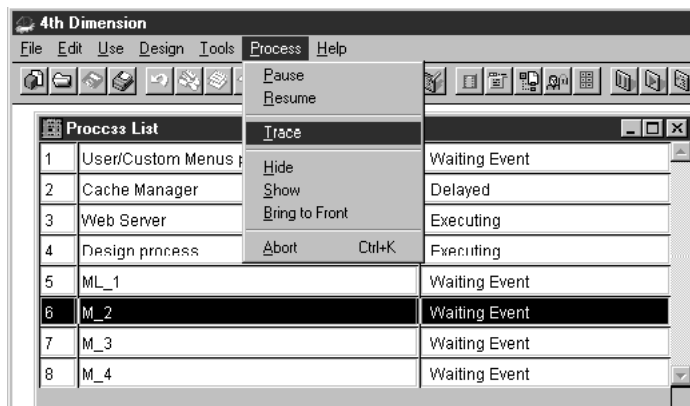


3. 「プロセスリスト」ウインドウが表示される。「プロセスリスト」ウインドウには、現在実行されているプロセスが（実際にコードを実行しているかどうかに関わらず）表示されます。



4. トレースしたいプロセスをクリックして選択する。

5. 「プロセス」メニューから「トレース」を選択する。



ここで注目すべき点は、4Dはトレースの要求を「記憶する」ということです。

プロセスが現在コードを実行している場合には、そのプロセスのデバッガがただちに表示されます。

プロセスが現在コードを実行していない場合（つまり、プロセスがデータ入力モードでイベント待機中）、デバッガはプロセスがコードの実行を再開した直後に表示されます。

Tips：ボタンをクリックした際に、そのボタンのオブジェクトメソッドをトレースする場合があります。「Alt+クリック（Windows）」や「option+クリック（Macintosh）」は、クリックの「速度」によって有効な場合も無効な場合もあります。このような場合には、「プロセス」メニューから「トレース」メニューコマンドを使用します。オブジェクトメソッドが開始されると、ただちにデバッガが表示されます。それ以外の場合には、メソッド内にTRACEコマンド呼び出しを設定することもできます。

この章では、配列について説明します。このマニュアルの前章で説明したものよりも高度な技法について説明します。

配列

配列は、関連するデータのグループをより効率良く記憶、アクセス、管理するために使用します。配列は番号付きの変数の集まりのようなものです。その各項目を配列の“要素”と呼びます。

配列は、変数と同じようにデータタイプを持っています。すべての要素は、同じデータタイプになります。また、一般にサイズは固定ですが、サイズを変えるためのコマンドも用意されています。

配列は、メモリに常駐しますので、コピー、ソート、検索等の処理を高速に実行することができます。ただし、大きな配列を作成する場合には、メモリの容量を上回らないように、随時使用しなければなりません。特に、配列にレコードを代入するコマンドを使用する場合には注意が必要です。レコード数が多いと、大きな配列が作成されて、メモリを浪費してしまうからです。

配列を使用する

後の章で説明する配列コマンドを使って配列を作成します。以下のステートメントは、配列“名字”に名字を5つ代入します。

ARRAY TEXT (名字 ; 5)

中カッコ ({ }) を使用して配列の要素を参照します。中カッコ内には数値を記述します。以下のステートメントは、5つの名字を配列“名字”に格納します。

```
名字{1}:="鳥居"  
名字{2}:="石井"  
名字{3}:="高橋"  
名字{4}:="吉野"  
名字{5}:="森"
```

SELECTION TO ARRAYコマンドと**LIST TO ARRAY**コマンドは、より効率良くデータを配列に格納するコマンドです。

配列要素の参照は、変数と同じように扱います。例えば、以下のステートメントは、変数 “ MyName ” に文字列 “ 高橋 ” を代入します。

MyName:=名字{3}

2次元配列を使用する

2次元配列も配列コマンドで作成します。以下のステートメントは、3行4列の配列を作成します。4th Dimensionは、2次元配列の行を分割した配列として取り扱います。

ARRAY TEXT (名前 ; 3 ; 4)

以下の図はデータを格納した配列を示しています。

名前{2}{3}は要素 名前{3}は配列	名前{1}	鳥居	石井	高橋	吉野
	名前{2}	勲	明彦	雄二	康幸
	名前{3}	タヌキ	モグラ	ネコ	イヌ

2つの中カッコを組み合わせで2次元配列の要素を参照します。例えば、3行目の2番目の要素を参照するには、以下のように記述します。

名前{3}{2}

上のデータを使用すると、このステートメントは文字列 “ モグラ ” を返します。

“ 名前 ” が各要素の行であると理解することが重要です。例えば、“ 名前{3} ” は3行目を参照します。この場合は、ニックネームを示します。

2次元配列の1 “ 次元 ” は他のすべての配列と同様に処理できます。例えば、3番目の配列 “ 名前 ” を別の配列 “ arLast ” にコピーするには、以下のコマンドを使用します。

COPY ARRAY (名前{3} ; arLast)

ローカル配列、プロセス配列、インタープロセス配列

配列には、ローカル配列、プロセス配列、およびインタープロセス配列の3種類があります。

ローカル配列は任意のメソッド内で使用できます。この配列はメソッドが終了すると消去されます。ローカル変数と同じように、なるべくこの種の配列を使用してください。

ローカル配列は、配列名の先頭にドル記号 (\$) を付けて宣言します。同じ名前のローカル配列が異なるメソッドにおいて、異なるタイプになっていても構いません。ローカル配列は実行サイクルに対しローカルなため、スクロールエリアやドロップダウンリストボックス (Macintosh版では、ポップアップメニュー) としてフォーム内にローカル配列を表示することはできません。

プロセス配列は任意のプロセス内で使用します。この配列はプロセスが終了すると消去されます。複数のプロセスが同じ配列を定義し、プロセスごとに異なった値を入れることができます。ただし、これらの配列はプロセス間で同じ型を持っていない限りなりません。

インタープロセス配列はすべてのプロセスで使用可能です。これはプロセス間でデータを共用する場合と情報を交換する場合にのみ使用します。インタープロセス配列は配列名の先頭にインタープロセス記号 "<>" を付けます。

以下の例は、整数型のプロセス配列を作成します。

ARRAY INTEGER (<>整数配列 ; 20)

配列を表示する

配列は、ドロップダウンリストボックス (ポップアップメニュー) やスクロールエリアを使用しているフォームに表示することができます。どちらの場合においても、ドロップダウンリストボックスやスクロールエリアに表示される配列の名前は、フォーム上のドロップダウンリストボックスとスクロールエリアの名前と同一になっていることを理解することが重要です。さらに、ドロップダウンリストボックスやスクロールエリアの名前は、ドロップダウンリストボックスやスクロールエリア内で選択された配列要素の番号を含む変数であることを理解することも重要です。

注：フォームに2次元配列の1列やローカル配列を表示することはできません。

表示される配列は、以下の4つの中のいずれかを使用して作成します。

LIST TO ARRAY コマンド：このコマンドは、「リスト」エディタで定義されたリストから配列を作成します。

SELECTION TO ARRAY コマンド：このコマンドは、レコードのセクションからひとつまたは複数の配列を作成します。

DISTINCT VALUES コマンド：このコマンドは、テーブルから個々の値の配列を作成します。

配列の各要素は、ひとつのメソッド内で個々に割り当てることができます。

注：実数、整数、倍長整数、日付、ブールタイプの配列は、『4th Dimension デザインリファレンス』の中で記述されている表示フォーマットを使って、フォーマットすることができます。

スクロールエリアの例

以下の例は、スクロールエリア内での配列の挿入および表示方法を示しています。配列“配列給与”は、**DISTINCT VALUES**コマンドを使用して作成されます。これには、会社の従業員の給与のすべてが含まれています。また、“配列給与”と定義されたスクロールエリアはフォーム上に描画されています。

ユーザがスクロールエリアから配列要素を選択すると、“給与”フィールドに「ユーザ」モードまたは「カスタム」モードで選択された値が割り当てられます。

上記の例を作成するには、まず、ある会社には異なる従業員の名前と給与を含んだデータベースがあることが前提となります。

1. 配列を初期化する。

プロジェクトメソッド内で、**DISTINCT VALUES**コマンドを使用して“配列給与”を初期化します。また、前述の4つの方法のいずれかで配列を初期化することができます。

DISTINCT VALUES ([従業員]給与 ; 配列給与)

2. フォーム上にスクロールエリア“配列給与”を作成する。

フォーム上にスクロールエリアを作成し、その名前を“配列給与”にする。スクロールエリアの名前は配列の名前とまったく同一です。

3. ユーザがスクロールエリアから選択した配列要素を確認するために、スクロールエリアにオブジェクトメソッドを作成する。

オブジェクトメソッドはスクロールエリアを初期化し、そしてスクロールエリア内で選択された値をフィールドや別の変数に割り当てます。スクロールエリアは変数であり、ユーザによって選択された値を自動的にレコードに保存しないことを理解することが重要です。

If (On Load | (配列給与=0))

 配列給与:=1

End if

[従業員]給与:=配列給与{配列給与}

On Loadイベントでは、現在の要素に配列の最初の要素が代入されます。ユーザがスクロールエリアから配列要素を選択すると、[従業員]給与フィールドに選択された値が代入されます。これはユーザによって選択された値を保存します。

ドロップダウンリストボックス（ポップアップメニュー）の例

以下の例は、ドロップダウンリストボックス内での配列の挿入および表示方法を示しています。例では、ユーザはドロップダウンリストボックスから国を選択しています。**LIST TO ARRAY**コマンドは配列“配列国名”の初期化に使用されます。

1. 配列を初期化する。

プロジェクトメソッド内で、**LIST TO ARRAY**コマンドを使用して“配列国名”を初期化します。

LIST TO ARRAY ("国名"; 配列国名)

2. フォーム上にドロップダウンリストボックス“配列国名”を作成する。

フォーム上にポップアップメニューを作成し、その名前を“配列国名”にする。ドロップダウンリストボックスの名前は配列の名前とまったく同一です。

3. ユーザがドロップダウンリストボックスから選択した配列要素を確認するために、ドロップダウンリストボックスにオブジェクトメソッドを作成する。

オブジェクトメソッドはドロップダウンリストボックスを初期化し、そしてドロップダウンリストボックス内で選択された値をフィールドや別の変数に割り当てます。ドロップダウンリストボックスは変数であり、ユーザによって選択された値を自動的にレコードに保存しないことを理解することが重要です。

If (On Load | (配列国名=0))

 配列国名:=1

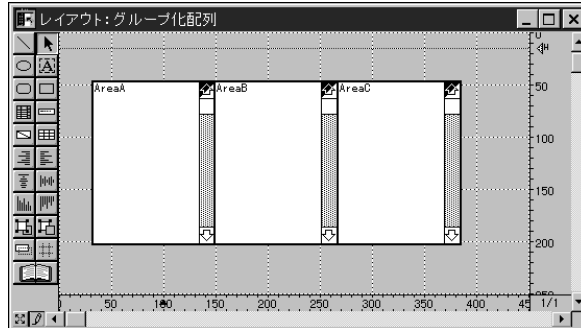
End if

[従業員]国:=配列国名{配列国名}

On Loadイベントでは、現在の要素に配列の最初の要素が代入されます。ユーザがドロップダウンリストボックスから配列要素を選択すると、[従業員]国フィールドに選択された値が代入されます。これはユーザによって選択された値を保存します。

グループ化したスクロールエリアを使用する

フォーム上のスクロールエリアをグループ化することができます。スクロールエリアをグループ化すると、それらはまるでひとつのスクロールエリアであるかのように動作します。また、スクロールエリアごとに、フォントやスタイルを設定することができます。データ入力時に、最前面にあるスクロールエリアのみがスクロールバーを表示します。下図は「デザイン」モード内のグループ化された3つのスクロールエリアのフォームです。



下記に、グループ化したスクロールエリアを作成する際の注意点を記します。

- 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1番上と1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。

- 1番右側のスクロールエリアを前面に移動する。実行時には、最前面のスクロールエリアのスクロールバーだけを操作可能にする。

- 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。

- グループの周りにくぼみをつける（グループを選択し、“Ctrl（Macintosh版では、command）”キーを押したまま“4”を押す）。

以下のステートメントは、グループ化されたスクロールエリアで示した3つの配列にデータを格納します。これは、[従業員]テーブルのフィールドのデータと[部門]テーブルのフィールドのデータを使用します。[部門]テーブルは、[従業員]テーブルからリレートされているため使用することができます。

SELECTION TO ARRAY ([従業員]名字 ; エリアA ; [従業員]役職 ; エリアB ; [部門]名称;エリアC)

以下の図は、先ほどの実行結果の画面を示しています。

深山	課長	総務部
岩橋	指導職	第2 営業部
中村	主任	第1 開発部
小田	部長	総務部
刀根	課長	第2 開発部
嶋田	課長	第3 開発部
魚住	主任	技術開発部
藤上	一般職	第2 開発部

ひとつのスクロールバーしか表示されない点に注目してください。これは、最前面にあるスクロールエリアのスクロールバーだけが表示されるからです。3つの配列がまるでひとつであるかのようにスクロールします。

ユーザが任意行をクリックすると、3つのエリアが同時に反転表示されます。また、各スクロールエリアに付随する変数に、ユーザがクリックした行の番号が代入され、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“小田”という名前をクリックすると、配列に付随する変数“エリアA”、“エリアB”、“エリアC”のすべてに3が代入されます。しかし、“エリアA”に対するオブジェクトメソッドだけが実行されます。

ひとつの配列に付随する変数に値が代入されると、他の配列に付随する変数も自動的に同じ値がセットされ、スクロールエリアの該当する行が反転表示されます。

配列は、以下のステートメントでソートすることができます。

SORT ARRAY (エリアA ; エリアB ; エリアC ; >)

以下の図は、ソートの結果を示しています。

岩橋	指導職	第2 営業部
魚住	主任	技術開発部
小田	部長	総務部
深山	課長	総務部
中村	主任	第1 開発部
刀根	課長	第2 開発部
嶋田	課長	第3 開発部
藤上	一般職	第2 開発部

配列が**SORT ARRAY**コマンドの最初の引数を基準にソートされる点に注目してください。他の2つの配列はその行との同調を維持するために設定されます。

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することができます。

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。実際に友人が新しい車に買い替えても、“君の車”は正しくなっています。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、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:=> [テーブル]	DEFAULT TABLE (My Table->)	不使用
フィールド	My Field:=> [テーブル]フィールド	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 (E版にはない vp Arr)	COPY ARRAY(My Arr-> ; B)	COPY ARRAY(B;My Arr->)
配列要素	My Erem:=> 配列{1} (E版にはない vp Erem)	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ができます。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 (→) をポインタの後に付けると、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（プログラミング言語の観点から）は、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 - 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 - 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)
```

MyTableに対するポインタの使用

プログラミング言語でMyTableをアクセスする場合には、ポインタを使用して、MyTableを参照することができます。MyTableへのポインタは、主にMyTable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、MyTable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MyTable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MyTable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MyFieldをアクセスする場合には、ポインタを使用して、MyField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、MyField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MyField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")

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)

要素p:=->配列{1}

` 配列の作成

` 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、ひと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

¥ ([初期設定]設定=1)

ボタン1:=1

¥ ([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

- ` “設定”フィールドを判定する
- ` “設定”フィールドが “ True ” の場合...
- ` ラジオボタンをオンにする

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、そのメソッドを示します。

```
` $1 - 文字列のポインタで、大文字に変更する
` $2 - 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“私の変数”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、“My Var”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。

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。

メニューコマンド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、任意のメニューコマンドを選択して「新規プロセス開始」チェックボックスをクリックする。メニューコマンド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」メニューコマンドを選択した場合。

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。

「ひとつまたは複数のウインドウ」：各プロセスは、ひと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。

「ひとつのアクティブ（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クション」：各プロセスは、個々にカレントセ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」メニューコマンド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」モードで開かれる場合には、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ィ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せんが、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。

ここでは、4th Dimensionのプログラミング言語を構成する次のような構成要素を定義します。

識別子
データタイプ
定数
演算子

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	<i>Capitalize</i> ([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	<i>Capitalize</i> ([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。

注：この構文は、WindowsおよびMacintosh上で使用できます。

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。

注：この構文は、WindowsおよびMacintosh上で使用できます。

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。

プロセス配列名	ステートメント内のプロセス配列名
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。\${viParam} viParam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:= <i>SP New offscreen area</i>
DR PUBLISH	<i>DR PUBLISH</i> (エリア ; 0 ; \$Temp)

セット

セットには、2種類あります。

インタ - 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 - 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。

注：この構文は、WindowsおよびMacintosh上で使用できます。

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
前の先頭につけません)。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 - 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。

注：この構文は、WindowsおよびMacintosh上で使用できます。

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 - バルプロセス

ロ - 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" ; 32768 ; "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	3 1	[テーブル1]
フィールド	3 1	[テーブル1]フィールド 1
サブテーブル	3 1	[テーブル1]サブファイル
サブフィールド	3 1	[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	<> + 3 1	<>インタープロセス変数
プロセス変数	3 1	変数
ローカル変数	\$ + 3 1	\$ローカル変数
フォーム	3 1	"フォーム"
インタープロセス配列	<> + 3 1	<>インタープロセス配列
プロセス配列	3 1	プロセス配列
ローカル配列	\$ + 3 1	\$ローカル配列
メソッド	3 1	メソッド
プラグインルーチン	3 1	CT GET DOCUMENT SIZE
インタープロセスセット	<> + 8 0	"<>インタープロセスセット"
プロセスセット	8 0	"プロセスセット"
クライアントセット	\$ + 8 0	"\$クライアントセット"
ローカルプロセス	\$ + 3 1	\$ローカルプロセス
グローバルプロセス	3 1	グローバルプロセス
命名セクション	8 0	"命名セクション"
インタープロセス命名セクション	<> + 8 0	"<>インター命名セクション"

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。

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	x		
BLOB（備考3参照）			x
配列（備考4参照）	x		x
サブテーブル		x	x
未定義	x		

備考

- 1. 文字列には、英数字フィールド、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、第15章「BLOBコマンド」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、第14章「配列コマンド」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（そ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ひと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、 $\pm 1.7e \pm 308$ （15桁）です。

整数データタイプ（2バイト整数）の範囲は、-32,768から32,766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、別の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1900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』マニュアルとサブレコードのテーマのコマンド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（少なくともひと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、データタイプの検証を強化しています。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NTAX（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	Date	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。

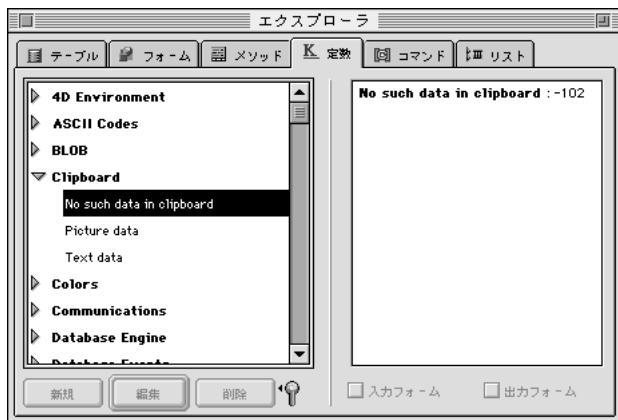
注：この表に示すデータ変換の他に、演算子と他のコマンドを組み合わせることで、より洗練されたデータ変換を実行することができます。

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。

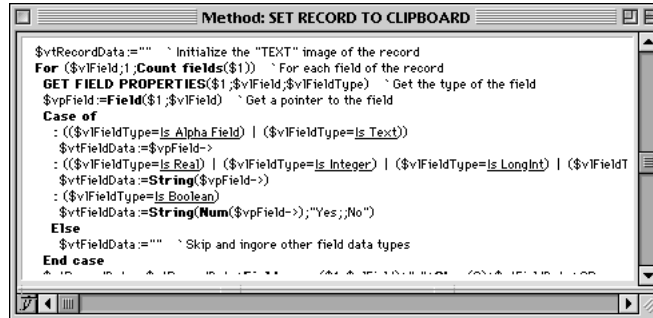
「メソッドエディタ」ウインドウに定数の名前を直接入力します。

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@記号（アットマーク）を使用すると、定数名すべてを入力しなくても済みます。例えば、「No such da@」と入力してからreturnキーまたはenterキーを押すと、4th Dimensionはコードの行の正当性を検査するときその行に「No such data in clipboard」という定数を入れます。

注：このマニュアルには定義済み定数（約500種類）がテーマ別に一覧されています。詳細は、序章の「このマニュアルについて」の節を参照してください。必要に応じて、コマンド説明においても定義済み定数が一覧されています。

定義済定数は、「メソッドエディタ」ウインドウや「デバッグ」ウインドウ内では、次のように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。文字列定数の例をつぎに示します。

"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、1900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。

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、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。剰余が整数でないからです。

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 (-2³¹から(2³¹)-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!
日付の引算 -	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
モジュール %	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、\$vISeconds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。

```
$vISeconds:=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+!!00:01:00!!
```

ただし、アプリケーションを開発する際、秒数で表現され時間値に追加される遅延が数値としてしか利用できない状況に遭遇することがあります。

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!! # !!01.02.04!! “ 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	!!01.02.03!! >= !!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	!!01.02.03!! <= !!01.02.03!! “ True(真) ” !!01.02.04!! <= !!01.02.03!! “ False(偽) ”

```
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) ” vPtrA = vPtrC “ False(偽) ”
不等号(#)	ポインタ # ポインタ	ブール	vPtrA # vPtrC “ True(真) ” vPtrA # vPtrB “ 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"

"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。

"abcefg hij"="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@"="abcefg hij"

ワイルドカードは、"ひと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
$0:=""
For ($vChar ; 1 ; Length ($1))
  If (Ascii ($1[[$vChar]]) # 64)
    $0:=$0+$1[[$vChar]]
  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）” のときに “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の右側に移動する
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の下に移動する
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。

次の表はピクチャ演算子の例を表しています。結果はトランケート（中央合わせ）とバックグラウンドの両方で示されています。

演算子	例	ピクチャ	バックグラウンド	トランケート
水平移動	ピクチャ1 + ピクチャ2			
垂直連結	ピクチャ1 / ピクチャ2			
XOR重ね	ピクチャ1 & ピクチャ2			
OR重ね	ピクチャ1 ピクチャ2			
右へ水平移動	ピクチャ1 + 5			
左へ水平移動	ピクチャ1 + (-5)			
下へ垂直移動	ピクチャ1 / 5			
上へ垂直移動	ピクチャ1 / (-5)			
サイズを拡大	ピクチャ * 2			
サイズを縮小	ピクチャ * 0.5			
水平スケール拡大	ピクチャ * + 2			
水平スケール縮小	ピクチャ * + 0.5			
垂直スケール拡大	ピクチャ */ 2			
垂直スケール縮小	ピクチャ */ 0.5			

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 (包含)		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。もうひとつの値を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	<p>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。</p> <table><tr><td>1 & 1</td><td>1</td></tr><tr><td>0 & 1</td><td>0</td></tr><tr><td>1 & 0</td><td>0</td></tr><tr><td>0 & 0</td><td>0</td></tr></table> <p>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。</p>	1 & 1	1	0 & 1	0	1 & 0	0	0 & 0	0
1 & 1	1								
0 & 1	0								
1 & 0	0								
0 & 0	0								
Bitwise OR (包含)	<p>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。</p> <table><tr><td>1 1</td><td>1</td></tr><tr><td>0 1</td><td>1</td></tr><tr><td>1 0</td><td>1</td></tr><tr><td>0 0</td><td>0</td></tr></table> <p>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。</p>	1 1	1	0 1	1	1 0	1	0 0	0
1 1	1								
0 1	1								
1 0	1								
0 0	0								
Bitwise OR (排他)	<p>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。</p> <table><tr><td>1 ^ 1</td><td>0</td></tr><tr><td>0 ^ 1</td><td>1</td></tr><tr><td>1 ^ 0</td><td>1</td></tr><tr><td>0 ^ 0</td><td>0</td></tr></table> <p>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。</p>	1 ^ 1	0	0 ^ 1	1	1 ^ 0	1	0 ^ 0	0
1 ^ 1	0								
0 ^ 1	1								
1 ^ 0	1								
0 ^ 0	0								
Left Bit Shift	<p>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。</p> <p>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。</p>								
Right Bit Shift	<p>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。</p> <p>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。</p>								

- Bitセット

最初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 (包含)	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:=( $vlIntA<<16) | $vlIntB ` 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。
```

```
$vlIntB:=$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。

この章では、コマンドの説明に使用する表記方法について説明します。また、コマンドに引数を渡す際の規則についても説明します。

コマンド定義

コマンドの説明は、次の5つの部分で構成されています。

見出し

シンタックス

引数

機能説明

使用例

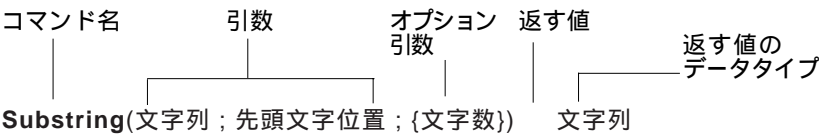
また、コマンドの説明には、警告、注釈、および4D Serverでのコマンドの動作に関する説明も含まれています。

以下の図は、コマンド定義を示しています。

		通信コマンド		18
コマンド		RECEIVE VARIABLE		
シンタックス		RECEIVE VARIABLE (変数)		
引数		引数 変数	タイプ 変数 →	説明 受信する変数
説明		<p>説明</p> <p>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。</p> <p>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。</p> <p>注：</p> <ol style="list-style-type: none">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く必要があります。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。このコマンドでは配列変数はサポートされません。シリアルポート経由でドキュメントファイルに対して配列の送受信を行いたい場合、BLOB コマンドを使用します。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モードにより、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。 <p>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。</p> <p>▼ 以下の例は、シリアルポートから変数を受信します。</p> <pre>SET CHANNEL(1;10+3072+16384+0) SET TIMEOUT(20) RECEIVE VARIABLE(私の変数)</pre> <p>、エラー による割り込みの例え</p> <p>参照</p> <p>参照なし</p>		
使用例				
参照				

シンタックス（構文）

シンタックスは、コマンドの文法です。各コマンドの最初には、コマンドの文法を示します。コマンド名の後には引数が続きます。コマンドが関数である場合、引数の後には矢印が続き、コマンドが返す値のデータタイプを示します。以下の図に、シンタックスダイアグラムの例を示します。



引数（パラメータ）

引数（パラメータ）は、コマンドに渡すデータで、“< >”で囲みます。

引数表は、表形式で各引数のデータタイプと短い説明文を併記します。以下の図に、引数の記述方法を示すシンタックスダイアグラムと引数表を示します。

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）

引数がオプションの場合は、引数の名前を、シンタックスダイアグラム中で中カッコ（{...}）で囲みます（先ほどのシンタックスダイアグラムの“文字数”引数を参照してください）。

引数を繰り返し使用する場合（常にオプション）は、引数の後に省略記号（...）をつけて示します。省略記号の後に、引数名と引数が繰り返すことのできる回数を示す数字をつけます。

引数が無限に繰り返すことができる場合、数字はNになります。例えば、次のようになります。

SAVE VARIABLES (ドキュメントファイル；変数1{;...; 変数N})

引数（パラメータ）の指定

引数（パラメータ）をコマンドに対して指定する際に、守らなければならない規則がいくつかあります。その規則を次に示します。

引数は、カッコで囲みます。例えば、**ADD RECORD**コマンドに[MyTable]を引数として指定する場合は、次のように記述します。

ADD RECORD ([My Table])

コマンドが複数の引数を持つ場合は、引数間をセミicolon（;）で区切ります。引数を繰り返す場合も同様です。例えば、**INPUT FORM**コマンドに[MyTable]と"入力フォーム"を引数として指定する場合は、次のように記述します。

INPUT FORM ([My Table] ; "入力フォーム")

オプションの引数を省略する場合は、付属のセミicolonも省略します。例えば、**Request**関数に対する2番目の引数はオプションです。2番目の引数を指定しない場合は、次のように記述します。

Result := **Request** (文字列1)

2番目の引数を指定する場合は2つの引数間をセミicolon（;）で区切り、次のように記述します。

Result := **Request** (文字列1 ; 文字列2)

引数がない場合は、カッコも省略します。引数を持たないコマンドも引数を省略したコマンドも同様です。例えば、**ADD RECORD**コマンドの引数に[テーブル]を指定する場合は、以下のように記述します。

ADD RECORD ([テーブル])

テーブルの引数を省略した場合には、カッコも省略し、以下のように記述します。

ADD RECORD

この章では、「ルーチン」エディタの「4D Environment」テーマ内にある4D環境コマンドについて説明します。

QUIT 4D	ACI folder	Application version
FLUSH BUFFERS	Structure file	Application type
SELECT LOG FILE	Data file	Version Type
ADD DATA SEGMENT	Application file	DATA SEGMENT LIST
PLATFORM PROPERTIES	Compiled application	

QUIT 4D

QUIT 4D

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

QUIT 4Dコマンドは、4th Dimensionを終了してデスクトップに戻ります。**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。したがって、このデータベースメソッドを使用して、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が必要です。「**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つずつアポートします。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。

以下の例は、「ファイル」メニューに「終了」メニューコマンド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
なし

FLUSH BUFFERS

FLUSH BUFFERS (ログファイル *)

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。「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「データ保存：」プロパティで、通常使用の場合にデータバッファの内容を保存する間隔を指定できます。

参照
なし

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。

重要：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。これはデータベースに4D Backupモジュールがインストールされている場合にのみ使用することができます。

<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。

注：4D Serverでは、このコマンドは動作しません。このコマンドに関する詳細は、4D Backupモジュールに付属のマニュアルを参照してください。

参照

なし

ADD DATA SEGMENT

ADD DATA SEGMENT

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。

既存のデータセグメントがすべて満杯になると、4th Dimensionまたは4D Serverにエラーコード 9999が発生します。そして、ディスクの空きがないことを示すエラーメッセージがユーザに表示されます。

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

なし

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依存
マシン	数値	動作している種類に依存

説明

PLATFORM PROPERTIES コマンドは、実行しているプラットフォーム、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の種類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なのか、それともPower Macintosh、またはWindowsなのかを示します。この引数は、次のようにあらかじめ定義されている定数の1つを返します。

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> の中に返される情報は、動作している4th Dimensionの種類によって異なります。

Macintosh (68KとPowerPC)

Macintosh版の4th Dimensionを実行している場合、残りの2つの引数には以下の情報が返されます。

引数<システム>には32ビット(倍長整数)値が返され、上位16ビットは未使用で、下位16ビットは以下のような構造になっています。

上位バイトには、主要なバージョン番号が入っています。

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：System7.5.1は\$0751のようにコード化されるため、10進数では1873になります。

注：4Dで、これらの値は%(モジューロ)と/(整数値を返す除算)を使った数値計算で求められます。また、バージョン6から追加された「ビットワイズ」演算子を使っても求められます。

引数<マシン>には、Macintoshのモデルを示す一意(重複しない)ID番号が返されます。

注：このID番号は、Appleコンピュータ社から発行されているデベロッパまたはテクニカルドキュメントに記載されています。新しいモデルのMacintoshがリリースされた場合は、新たに値を追加する必要があります。

Windows版

Windows版の4th Dimensionを実行している場合、引数<システム> と <マシン> には、次で説明する情報が返されます。

引数<システム>からは32ビット(倍長整数)値が返され、そのビットとバイトは以下のような構造になっています。

上位レベルのビットが1になっている場合、Windows NTの特色を持つシステムで実行していることを示します。ビットが0になっている場合は、Windows 3.1またはWindows 95のものと実行しているということを意味します。

注：上位レベルのビットは倍長整数の符号に使われています。したがって、4Dでは、値が正か負かを調べるだけです。正ならWindows NTが動作していることになります。

下位バイトには、Windowsの主要なバージョン番号が入っています。3が返された場合、Windows版3.xもしくはWindows NTが動作中であることを示します。4が返された場合、Windows 95または新しいシェルでWindows NTが動作中であることを示します。以下の下位バイトにはWindowsのバージョン番号の枝番が入っています。

注：4Dで、これらの値は%（モジュール）と//（整数値を返す除算）を利用した数値計算で求められます。また、バージョン6から追加された「ビットワイズ」演算子を使っても求められます。

引数<マシン>は、次のようにあらかじめ定義されている定数の1つを返します。

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 601l	倍長整数	601
PowerPc 603	倍長整数	603
PowerPc 604	倍長整数	604

注：Windows3.1.xの場合、Pentiumプロセッサが搭載されていても“486”が返されます。

以下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。

・「SHOW OS VERSION」プロジェクトメソッド

PLATFORM PROPERTIES (\$vIPlatform ; \$vISystem ; \$vIMachine)

If ((\$vIPlatform<1) | (3<\$vIPlatform))

 \$vsPlatformOS:=""

Else

If (\$vIPlatform=3)

 \$vsPlatformOS:=""

If (\$vISystem<0)

 \$winMajVers:=((2^31)+\$vISystem)%256

 \$winMinVers:=(((2^31)+\$vISystem)¥256)%256

If (\$winMajVers>=4)

 \$vsPlatformOS:="Windows™ 95"

Else

 \$vsPlatformOS:="Windows™ (with Win32s)"

End if

Else

 \$winMajVers:=\$vISystem%256

 \$winMinVers:=(\$vISystem//256)%256

 \$vsPlatformOS:="Windows™ NT"

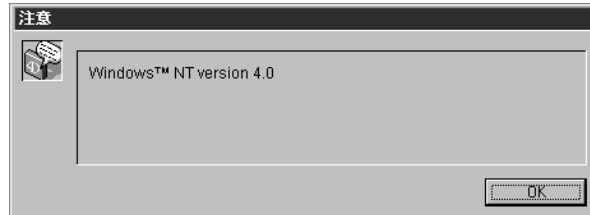
End if

```

$vsPlatformOS:=$vsPlatformOS+"バージョン"+String ($winMajVers)+". "
+String ($winMinVers)
Else
$vsPlatformOS:="MacOS™バージョン "+String ($vSystem//256)+". "
、
+String(($vSystem//16)%16)+((" "+String($vSystem%16))*Num(($vSystem%16) # 0))
End if
End if
ALERT($vsPlatformOS)

```

Windows上では、次の図のような「警告」ボックスを表示します。



Macintosh上では、次の図のような「警告」ボックスを表示します。



参照
なし

ACI folder

ACI 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	ACIフォルダのパス名
-----	-----	-------------

説明

ACI folder関数は、稼働しているシステムフォルダまたはディレクトリの中に配置されたACIフォルダのパス名を返します。

Windows 上：

“ACI”と名付けられた「ACI」フォルダは、稼働している「WINDOWS」ディレクトリ（通常、C:\WINDOWS）の中に配置されます。したがって、**ACI folder**関数は通常、「C:\WINDOWS\ACI」というパス名を返します。しかし、PCコンピュータはマルチブートコンフィギュレーション（システム定義）を使って、セットアップを実行することができ、配置場所と稼働中のWINDOWSディレクトリの名前をインストール作業中にカスタマイズすることができます。そのため、もし、「ACI」フォルダの中にユーザ自身のファイル（書類）を保存したい場合、**ACI folde**関数を使って「ACI」フォルダへの実際のパス名を取得することができます。

Macintosh 上：

“ACI”と名付けられた「ACI」フォルダは、アクティブなシステムフォルダの「初期設定」フォルダの中に配置されます。4Dを初めてインストールした直後なら、**ACI folder**関数は通常、「Macintosh HD:システムフォルダ:初期設定:ACI:」というパス名を返します。しかし、Macintoshユーザはディスク名やフォルダ名を変更することができるため、この「ACI」フォルダのパス名は変わることがあります。したがって、「ACI」フォルダの中に独自のファイル（書類）を保存したい場合、**ACI folde**関数を使って「ACI」フォルダへの実際のパス名を取得することができます。

プラットフォーム独立性と国際性：ACIフォルダへの実際のパス名を取得するために**ACI folde**関数を使用することにより、あるローカライズされたシステムで動作している任意のプラットフォーム上での機能を保証します。

4D環境は、次の情報を格納するために「ACI」フォルダを使用します。

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

4D Client / 4D Serverまたはインターネット/イントラネットのネットワークコンポーネント（ただし、Windows上では...\ACI\NETWORKディレクトリの中）

4D Serverからダウンロードされたリソースを格納するために4D Clientによって作成された「.rex」ファイルと「.res」ファイル

4D Serverからダウンロードされた4D Extensions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

警告：ACIフォルダの中には、任意のファイルまたはドキュメントを自由に格納できますが、4D環境自身によって作成されたファイルの移動および変更は避けた方が賢明です。

以下の例は、シングルユーザデータベースの起動中に、ACI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" ; "初期設定ファイル")

`「PREF」Macintoshファイルタイプを「PRF」Windowsファイル拡張子にマップする

` \$vsPrefDocName:=**ACI folder**+"MyPrefs" `初期設定ファイルにパス名を作成する

` そのファイルが存在するかチェックする

If (Test pathname (\$vsPrefDocName+(".PRF"*Num(On Windows)))#Is a document)

 \$vtPrefDocRef:=**Create document**(\$vsPrefDocName ; "PREF") `ファイルを作成する

Else

 \$vtPrefDocRef:=**Open document**(\$vsPrefDocName ; "PREF") `ファイルを開く

End if

If (OK=1)

 `ドキュメント内容を処理する

CLOSE DOCUMENT(\$vtPrefDocRef)

Else

 `エラーの取り扱い

End if

参照

System folder、Temporary folder、Test path name

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。

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。

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」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

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ひと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 file**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。

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。

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。

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

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。

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

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確認します。

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 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、ACIの内部バージョンを意味する
2-3-4	ACI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：

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

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 Engine	倍長整数	1
4D Runtime	倍長整数	2
4D Runtime Classic	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

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

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

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。

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

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)

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

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、配列の要素といえます。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。

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 ($vElem ; 1 ; 5)
  ALERT ("配列要素 #" + String($vElem) + "は" + atNames{$vElem} + "です。")
End for
```

atNames{\$vElem}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

その他に配列で使用できる4Dコマンドがあります。詳細については、以下のコマンドの説明を参照してください。

配列とレコードセレクションを操作するには、**SUBSELECTION RANGE TO ARRAY**、**SELECTION TO ARRAY**、**ARRAY TO SELECTION**、**DISTINCT VALUES**コマンドを使用します。

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。

バージョン6の新規コマンドは、配列を一度の呼び出しで作成します。このようなコマンドには、**FONT LIST**、**WINDOW LIST**、**VOLUME LIST**、**FOLDER LIST**、**DOCUMENT LIST**コマンドがあります。

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

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（「オブジェクトプロパティ」ウインドウの「デフォルト値」ボタンを使用して）あらかじめ定義することができますが、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。すなわち、「オブジェクトプロパティ」ウインドウの「変数」ページで変数オブジェクトを指定します。



名前（この例ではatName）は、配列の作成と処理に使用する配列の名前です。

注：2次元配列やポインタ配列は表示できません。

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。

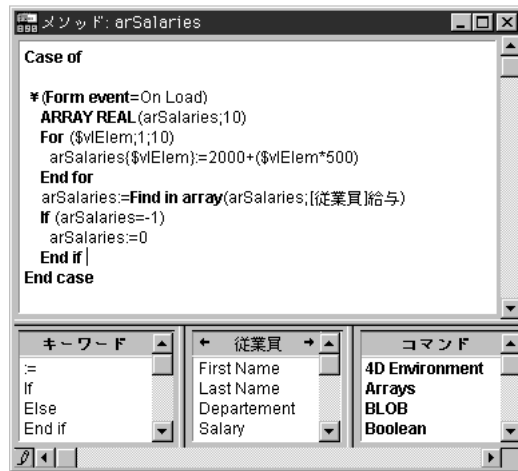


配列を初期化する

「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。これには、「オブジェクトプロパティ」ウィンドウでそのイベントを有効にします。



「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($vElem ; 1 ; 10)
    arSalaries{$vElem}:=2000+($vElem*500)
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」または「On Data Change」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($vIElem ; 1 ; 10)
```

```
arSalaries{$vIElem}:=2000+($vIElem*500)
```

```
End for
```

```
arSalaries:=Find in array(arSalaries ; [従業員]給与)
```

```
If (arSalaries=-1)
```

```
arSalaries:=0
```

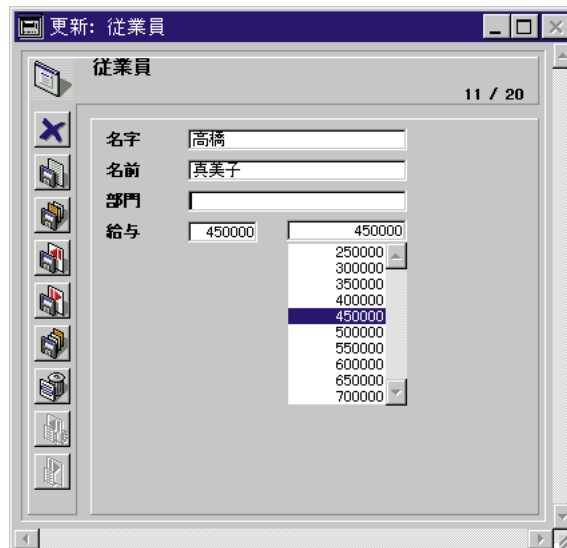
```
End if
```

¥ (Form even=On Data Change)

```
[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

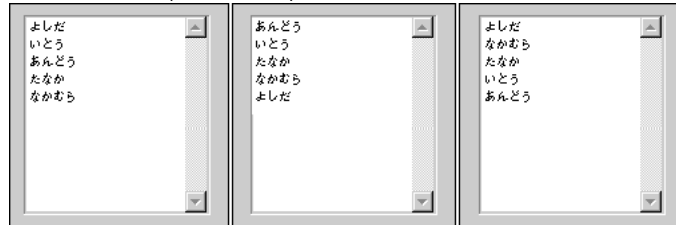
配列の要素は、**SORT ARRAY**コマンドを使用して並べ替えることができます。前述の例を使用し、配列がスクロールエリアに表示されているとします。

- まず、エリアは左のリストのように表示されています。
- 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 ; >)
```

- 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

要素の追加、挿入、削除は、**INSERT ELEMENT**コマンドおよび**DELETE ELEMENT**コマンドを使用して行います。

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、

CLEAR VARIABLE(atNames)

¥ (Form event=On Clicked)

If (atNames#0)

　　vtInfo:="atNames{atNames} + "上をクリックしました。"

End if

¥ (Form event=On Double Clicked)

If (atNames#0)

ALERT (atNames{atNames}+ "上をダブルクリックしました。")

End if

End case

注：各イベントは、「オブジェクトプロパティ」ウインドウでアクティブにする必要があります。

シンタックス「atNames{\$viElem)」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 (配列が空の場合)
atNames:=1

` 最後の要素を選択する (配列が空の場合)
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、選択した要素のひと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)
    $vElem:=Find in array (atNames ; $vsName)
    If ($vElem>0)
        atNames:=$vElem
    Else
        ALERT ($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、更に処理を

、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ひとつではない場合

、次回のために、そのリストに値を追加する

\$vElem:=Size of array (asColors)+1

INSERT ELEMENT(asColors;\$vElem)

asColors{\$vElem}:=asColors{0}

Else

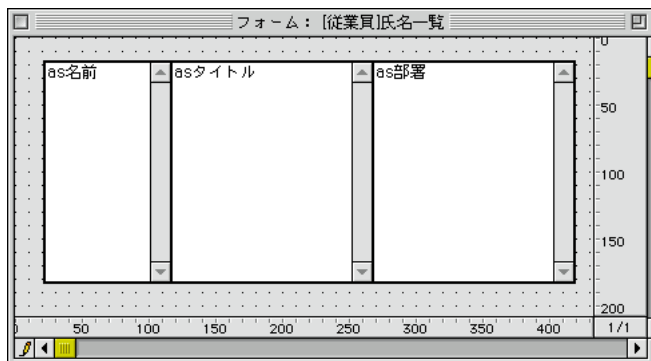
、入力した値がオブジェクトに付属の値の中にある場合

End if

End case

グループ化されたスクロールエリア

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、それらはひと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」モードにおいてひと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現れるためである。

- 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。

- グループの周りにグループよりも1ピクセル大きい境界線を引く（グループを選択し、「Ctrl（Macintosh版では、command）」キーを押したまま「1」を押す）。右端の境界線を左に1ピクセル移動（ショートカットでは、「Ctrl（command）+」キー）すると、見やすくなる。

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。

従業員	
名字	
名前	
入社年月日	
給与	2 ⁶
役職名	
部門コード	

部門	
部門コード	
部門名	
管理者	
部門予算	
合計給与	

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。

名前	役職名	部門名
菊地芳枝		経理
佐々木信一	常務取締役	役員
高橋紘大	課長	営業
田中利明	取締役部長	役員
鎌田俊大	部長	販売

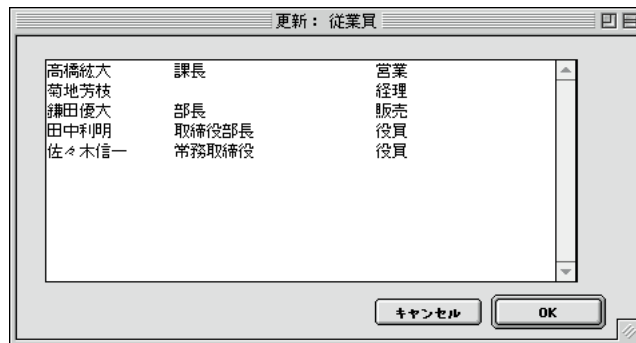
キャンセル OK

スクロールバーがひと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ひと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、ユーザを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「Bentley」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、プロセスごとにひと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<>（WindowsおよびMacintosh）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。

注：プロセス配列やインタープロセス配列を引数として渡すことはできますが、ローカル配列は渡せません。

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)
$0:=0
For ($vElem ; 1 ; Size of array ($1->))
    $0:=$0+$1->{$vElem}
End for
```

次に、以下のように記述します。

```
$vSum:= Array sum (->arSalaries)
` ...
$vSum:= Array sum (->aiDefectCounts)
` ..
$vSum:= 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 ($vElem ; 1 ; Size of array($1->))
    If ($1->{$vElem}# "")
        $1->{$vElem}:=Uppercase ($1->{$vElem}[[1]])+ Lowercase (Substring ( $1->{$vElem} ; 2))
    End if
End for
```

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ひとつは、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。もうひとつの方法は、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 ($vChar ; 1 ; Length ($1))
    $1[[vChar]]:=Char (<>aiCustomOutMap{Ascii ($1[[vChar]])})
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:=$1
For ($vChar ; 1 ; Length ($1))
    $0[[vChar]]:=Char (<>aiCustomInMap{Ascii ($0[[vChar]])})
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 (\$vITable ; 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{\$vITable} ; 1 ; **Count tables** (\$vITable))

`要素の値を設定する

For (\$vIField ; 1 ; **Size of array**(<>apFields{\$vITable}))

<>apFields{\$vITable}{\$vIField}:=Field(\$vITable;\$vIField)

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)};$apTheFieldslamWorkingOn)
`ブールと日付フィールドを初期化する
For ($vElem ; 1 ; Size of array ($apTheFieldslamWorkingOn))
  Case of
    ¥ (Type ($apTheFieldslamWorkingOn{$vElem}->)=Is Date)
      $apTheFieldslamWorkingOn{$vElem}->:=Current date
    ¥ (Type ($apTheFieldslamWorkingOn{$vElem}->)=Is Boolean)
      $apTheFieldslamWorkingOn{$vElem}->:=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	$(31 + \text{配列の要素の数}) // 8$
日付	$(1 + \text{配列の要素の数}) * 6$
文字列	$(1 + \text{配列の要素の数}) * \text{指定した文字列の長さ} (+1 \text{ 奇数の場合}, +2 \text{ 偶数の場合}) * 2$
整数	$(1 + \text{配列の要素の数}) * 2$
倍長整数	$(1 + \text{配列の要素の数}) * 4$
ピクチャ	$(1 + \text{配列の要素の数}) * 4 + \text{各ピクチャサイズの合計}$
実数	$(1 + \text{配列の要素の数}) * 16$
テキスト	$((1 + \text{配列の要素の数}) * 6) + \text{各テキストサイズの合計}$
2次元配列	$(1 + \text{配列の要素の数}) * 12 + \text{各配列サイズの合計}$

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。
`夜中にエラーが発生する危険を犯さないように、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

```

配列コマンド

この節では、この章では、「ルーチン」エディタの「Arrays」テーマ内にある配列コマンドについて説明します。本節のコマンドを使用すると、配列の作成、配列のソート、配列要素の検索、テーブルに対するデータの出し入れ、およびその他の処理を実行することができます。配列は一般にスクロールエリアやポップアップメニューに表示します。また、配列はメモリ上でデータを扱うためにも使用します。次のコマンドは配列処理に適しています。

ARRAY BOOLEAN	ARRAY STRING	Find in array
ARRAY DATE	ARRAY TEXT	LIST TO ARRAY
ARRAY INTEGER	ARRAY TO LIST	INSERT ELEMENT
ARRAY LONGINT	ARRAY TO SELECTION	Size of array
ARRAY PICTURE	COPY ARRAY	SELECTION TO ARRAY
ARRAY POINTER	DELETE ELEMENT	SELECTION RANGE TO ARRAY
ARRAY REAL	DISTINCT VALUES	SORT ARRAY

- ARRAY BOOLEAN
- ARRAY DATE
- ARRAY INTEGER
- ARRAY LONGINT
- ARRAY PICTURE
- ARRAY POINTER
- ARRAY REAL
- ARRAY TEXT
- ARRAY STRING

ARRAY BOOLEAN (配列名 ; サイズ1 ; {サイズ2})
ARRAY DATE (配列名 ; サイズ1 ; {サイズ2})
ARRAY INTEGER (配列名 ; サイズ1 ; {サイズ2})
ARRAY LONGINT (配列名 ; サイズ1 ; {サイズ2})
ARRAY PICTURE (配列名 ; サイズ1 ; {サイズ2})
ARRAY POINTER (配列名 ; サイズ1 ; {サイズ2})
ARRAY REAL (配列名 ; サイズ1 ; {サイズ2})
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、配列の数
サイズ2	数値	2次元配列の要素の数

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
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、配列の数
サイズ2	数値	2次元配列の要素の数

説明

これらの配列コマンドは、すべてメモリ上に配列を作成します。

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>に配列の数を、<サイズ2>には各配列の要素の数を指定します。

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、この節の他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。

引数<長さ>は、文字列の配列にのみ使用します。これは、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。文字列の配列は、テキストの配列よりも高速に処理を実行します。

0番目の要素（配列名{0}）は、常にひとつの配列として作成されます。配列タイプに応じた空の値が代入されます。配列の要素数は、**Size of array**関数を使用して求めます。次の行は配列の定義はそのまま残しますが、（0番目の要素を除く）すべての配列要素を削除します。

ARRAY TEXT (Mine ; 0)

配列を作成した時点の配列の各要素は、空の値になります。数値の配列は0、文字列とテキストは""、日付の配列は!00.00.00!、ブールの配列は“ False ”です。

配列の要素は、中カッコ（{ }）を使用して参照します。例えば、配列“ MyArr ”の2番目の要素を参照する場合は、“ MyArr{2} ”と記述します。

2次元配列の要素は、2組の中カッコを使用して参照します。例えば、“ MyArr{3}{5} ”と記述すると、3番目の配列の5番目の要素を参照することができます。

この節のコマンドを使用して配列の大きさを変更することができます。これらのコマンドを既存の配列に使用して、要素を追加、または削除することができます。例えば、4つの要素を持つテキストの配列“ Mine ”に対して次のステートメントを実行すると、2つの要素が削除されます。

ARRAY TEXT (Mine ; 2)

そして、以下のステートメントを実行すると、既存の要素に影響を与えないで4つの要素が追加されます。

ARRAY TEXT (Mine ; 6)

以下の例は、文字列の配列を作成し、サブテーブルの情報を配列に格納します。

```
ALL SUBRECORDS ([従業員]子供)
ARRAY STRING (15 ; a名前 ; Records in subselection ([従業員]子供))
For ($i;1;Size of array (a名前))
  a名前{$i}:=[従業員]子供'名前'
  NEXT SUBRECORDS ([従業員]子供)
End for

参照
なし
```

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。

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、表には現れない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;allIDs)  
  SORT ARRAY(asNames;allIDs;>)  
¥ (Form event=On Unload)  
  CLEAR VARIABLE (asNames)  
  CLEAR VARIABLE (allIDs)  
¥ (Form event=On Clicked)  
  If (asNames#0)  
    ` 正しいレコードを取得するために配列allIDsを使用する  
    QUERY([従業員];[従業員]ID番号=allIDs{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;allIDs;>)
```

`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;allIDs;<)

参照
なし

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同じ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。

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
COPY ARRAY (arCompany ; arData)

参照
なし

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 > にひと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。

以下の例は、配列 “リスト” の10番目の要素の位置に新しい15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。

\$vElem:=**Size of array** (anArray)+1

INSERT ELEMENT (anArray;\$vElem)

anArray{\$vElem}:=...

参照

なし

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、ひと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。

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。  
$vIElem:=Size of array (anArray)  
If ($vIElem>0)  
    DELETE ELEMENT (anArray;$vIElem)  
End if
```

参照
なし

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

 \$viElem:=**Find in array** (\$1->,"")

If (\$viElem>0)

DELETE ELEMENT (\$1->,\$viElem)

End if

Until (\$viElem<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
なし
```

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
なし

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てなくなっています。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。

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

なし

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てなくなっています。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が（＜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。

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

なし

SELECTION TO ARRAY

SELECTION TO ARRAY (フィールド1 ; 配列1{ ;...; フィールドN ; 配列N})

引数	タイプ	説明
フィールド	フィールド	データを使用するフィールド
配列	配列	フィールドのデータを格納する配列

SELECTION TO ARRAY (テーブル ; 配列1{ ; ... ; フィールドN ; 配列N})

引数	タイプ	説明
テーブル	テーブル	レコ - ド番号を使用するテーブル
配列	配列	レコ - ド番号を格納する配列

説明

第1のシンタックスでは、**SELECTION TO ARRAY**コマンドは、ひとつまたは複数の配列を作成し、フィールドのデータやカレントセクションのフィールドをその配列にコピーします。最初に指定したフィールドの属するテーブルのカレントセクションを指定します。自動リレートが設定されていれば、別のテーブルのフィールドも使用することができます。配列の大きさ（要素の数）は、セクションのレコード数と一致します。

配列は、対応する各フィールドのタイプと同じになります。ただし、既存の文字列の配列に、テキストフィールドのデータをコピーした場合は、文字列の配列のままです。また、時間フィールドを配列にコピーした場合は、倍長整数型の配列になります。

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。

SELECTION TO ARRAYコマンドは、セ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。

以下の例は、自動リレートに関連している[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
なし

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、ひと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。

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
なし

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、配列のサイズはセクションの中から見つけれ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。

以下の例は、カレントセ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。

```
([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

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ために選択されたレコード番号
終了	数値	データ抽出を終了するために選択されたレコード番号
フィールド／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コマンドはひとつまたは複数の配列を作成し、その配列にカレントセ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

れた選択レコードの範囲に適用されます。

ユーザは、以下の式 ($1 \leq \text{開始} \leq \text{終了} \leq \text{Records in selection}([\dots])$) を満たしている <開始> と <終了> の選択レコード番号を **SUBSELECTION TO ARRAY** コマンドに受け渡します。

もし、以下の式 ($1 \leq \text{開始} \leq \text{終了} \leq \text{Records in selection}([\dots])$) を満たす値を受け渡すと、選択されたレコードが (<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、間違った選択レコード番号を受け渡すと、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選択レコードから最終の選択レコードに値を返します。

(<開始> > <終了>) の場合は、選択されたレコードが <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

n 対1自動リレートまたはマニュアル (手動) の n 対1リレートを自動リレートにするために **AUTOMATIC RELATION** コマンドをコールして提供されるリレートフィールドから値をロードする (どちらの場合も、値は n 対1リレートのいくつかのレベルを通してテーブルからロードされます)。

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。

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

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、ひとつのBLOBは連続した可変長バイトであり、ひと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。ひとつの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。ひとつの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。

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、第19章「コンパイルコマンド」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、第17章「クリップボードコマンド」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、第46章「リソースコマンド」を参照してください。

この章では、「ルーチン」エディタの「BLOB」テーマ内にあるBLOBコマンドについて説明します。

DOCUMENT TO BLOB	BLOB to integer	LIST TO BLOB
BLB TO DOCUMENT	LONGINT TO BLOB	BLOB to list
VARIABLE TO BLOB	BLOB to longint	COPY BLOB
BLOB TO VARIABLE	REAL TO BLOB	INSERT IN BLOB
COMPRESS BLOB	BLOB to real	DELEE FROM BLOB
EXPAND BLOB	TEXT TO BLOB	BLOB size
BLOB PROPERTIES	BLOB to text	SET BLOB SIZE
INTEGER TO BLOB	BOOLEAN ARRAY FROM SET	
LONGINT AARRAY FROM SELECTION		

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がひとつ作成されます。

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。

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。

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 ; $vICompressed ; $vIExpandedSize ; $vICurrentSize)
If ($vICompressed=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。

```
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、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。

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 ; $vICompressed ; $vIExpandedSize ; $vICurrentSize)
If ($vICompressed # 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;$vICompressed;$vIExpandedSize;$vICurrentSize)
        If ($vICompressed#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 ; $vICompressed ; $vIExpandedSize ; $vICurrentSize)
BLOB PROPERTIES ($1-> ; $vICompressed ; $vIExpandedSize ; $vICurrentSize)
If ($vIExpandedSize=0)
    $0:=0
    If (Count parameters>=2)
        $2->:=0
    End if
Else
    $0:=100-((($vICurrentSize/$vIExpandedSize)*100)
    If (Count parameters>=2)
        $2->:=$vIExpandedSize-$vICurrentSize
    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)
$vIPercent:= Space saved by compression (->vxBlob ; ->vIBlobSize)
ALERT ("圧縮サイズと割合: "+String (vIBlobSize)+"バイト"+String ($vIPercent;"#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、MacOS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。

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、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、MacOS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。

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、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 {; オフセット | *})

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
オフセット *	文字	BLOB内でのオフセット（バイト単位）値を追加する場合には* * でなければ、書き込み後の新規オフセット

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 (${1})
C_LONGINT ($viParam)
SET BLOB SIZE ($1-> ; 0)
For ($viParam ; 2 ; Count parameters)
    VARIABLE TO BLOB (${viParam}-> ; $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 (${1})
C_LONGINT ($viParam ; $vIOffset)
$vIOffset:=0
For ($viParam ; 2 ; Count parameters)
    BLOB TO VARIABLE ($1-> ; ${viParam}-> ; $vIOffset)
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。

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。

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の中へ書き込む整数の値
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)

vOffset:=50

INTEGER TO BLOB (518 ; vxBlob ; Macintosh byte ordering ; vOffset)

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vOffset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の中へ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) へ書き込まれます。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)

viOffset:=50

LONGINT TO BLOB (0x01020304 ; vxBlob ; Macintosh byte ordering ; viOffset)

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viOffset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の中へ書き込む実数の値
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)

PCおよびPower Macintoshでは、vxBlobのサイズは8バイトです。

68KのMacintoshでは、vxBlobのサイズは10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)

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 := ...

viOffset:=50

REAL TO BLOB (518 ; vxBlob ; Windows Double real format ; viOffset)

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viOffset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の中へ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。

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。

```
$viOffset:=0x200
For ($viLoop ; 0 ;19)
    $viValue:=BLOB to integer (vxSomeBlob ; PC byte ordering ; $viOffset)
    ` $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。

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。

```
$viOffset:=0x200
For ($viLoop ; 0 ;19)
    $viValue:=BLOB to longint (vxSomeBlob ; PC byte ordering ; $viOffset)
    ` $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})

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。

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。

```
$viOffset:=0x200
```

```
For ($viLoop ; 0 ;19)
```

```
    $viValue:=BLOB to real(vxSomeBlob ; PC byte ordering ; $viOffset)
```

```
    ` $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。

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)
viSize:=BLOB Size (vxResData)
If (viSize>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
  viOffset:=0
  viNbEntries:=BLOB to integer (vxResData ; Macintosh Byte Ordering ; viOffset)
  `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 (viOffset<viSize)
      vsEntry:=BLOB to text (vxResData ; Pascal string ; viOffset)
      ``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 {; フィルタ})

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
フィルタ	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 < フィルタ > 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、< フィルタ > 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 < フィルタ > 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

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。

参照

なし

LONGINT ARRAY FROM SELECTION

LONGINT ARRAY FROM SELECTION ({テーブル;}レコード番号配列{;セクション名})

引数	タイプ	説明
テーブル	テーブル	テーブル この引数を省略した場合は現在のデフォルトテーブル
レコード番号配列	倍長整数配列	レコード番号の配列
セクション名	文字列	命名セクション この引数を省略した場合は現在のカレントセクションの名前

説明

このコマンドは、セ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クションを使用します。この場合は最初の引数としてテーブルを指定しなければいけません。

注：配列要素0は-1に初期化されます。

参照

INSERT IN BLOB

この章では、3種類のブール関数について説明します。ブールコマンドは、「ルーチン」エディタの「Boolean」テーマ内にあります。これらの関数は、論理計算に用いられます。

True	False	Not
------	-------	-----

以下の例は、ボタン “ MyButton ” がクリックされた時に “ True ” を返し、クリックされなかった時には “ False ” を返します。MyButtonがクリックされると、MyButton変数に1を代入します。この例は、ボタンの値をもとにMyButton変数をセットします。

If (MyButton=1)	` MyButtonがクリックされたら...
MyButton変数:= True	` MyButton変数にTrueがセットされる
Else	` MyButtonがクリックされなかったら...
MyButton変数:= False	` MyButton変数にFalseがセットされる
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。

```
MyButton変数:= (MyButton=1)
```

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。

以下の例は、v変数にTrueを代入します。

v変数:=True

参照

なし

False

False ブール値 (False)

説明

False関数は、ブール値のFalseを返します。

以下の例は、v変数にFalseを代入します。

v変数:=False

参照

なし

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。

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

参照

なし

この章では、「ルーチン」エディタの「Clipboard」テーマ内にあるクリップボードコマンドについて説明します。

APPEND TO CLIPBOARD

CLEAR CLIPBOARD

GET CLIPBOARD

Get text from clipborad

GET PICTURE FROM CLIPBOARD

SET PICTURE TO CLIPBOARD

SET TEXT TO CLIPBOARD

Test clipborad

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、たったひと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ひと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 ; $vIField ; $vIFieldType)
C_POINTER ($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) | (Record number ($vpTable->)= -3))
` 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
$vtRecordData:=""
` レコードの各フィールドに対してループする
For ($vIField ; 1 ; Count fields ($1))
` フィールドタイプを取得する
GET FIELD PROPERTIES ($1 ; $vIField ; $vIFieldType)
` フィールドへのポインタを取得する
$vpField:=Field ($1 ; $vIField)
` フィールドのタイプに従って、適切な方法でそのデータをコピーする(またはしない)
Case of
  ¥ (($vIFieldType=Is Alpha field ) | ($vIFieldType=Is Text ))
    $vtFieldData:=$vpField->
  ¥ (($vIFieldType=Is Real ) | ($vIFieldType=Is Integer ) | ($vIFieldType=Is LongInt )
    | ($vIFieldType=Is Date ) | ($vIFieldType=Is Time ))
    $vtFieldData:=String ($vpField->)
  ¥ ($vIFieldType=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 ; $vField)+":"+Char (9)+$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。

従業員

社員ID: 1

名字: ジェーン

名前: ロバート

生年月日: 1968/10/10

住所: 駒沢1-17-17-4F

市町村: 世田谷区

都道府県: 東京都

郵便番号: 154-0012

給与: 350000

子供: クリスティーナ
シルベスタ
アーノルド

入社年月日: 2/5/61

出勤時間: 08:00:00

フルタイム: ☒ はい ☐ いいえ

「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。

クリップボード

社員ID:1
名字:ジェーン
名前:ロバート
生年月日:1968/10/10
住所:駒沢1-17-17-4F
市町村:世田谷区
都道府県:東京都
郵便番号:154-0012
給与:350000
分類:
入社年月日:2/5/61
出勤時間:08:00:00
フルタイム:はい
画像:
子供:

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 ; $vIField ; $vIFieldType ; $vIPosCR ; $vIPosColon)
C_POINTER ($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) | (Record number ($vpTable->)=3))
  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
      $vsDocName:=Temporary folder+"スクラップ"+String (1+(Random%99))
      `スクラップファイルが存在する場合には削除する(エラーはここでテストする)
      DELETE DOCUMENT ($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
      $vIField:=0
      Repeat
        `テキスト内の以下のフィールド行を探す
        $vIPosCR:=Position (CR ; $vtClipboardData)
        If ($vIPosCR>0)
          `フィールド行を取り出す
          $vtFieldData:=Substring ($vtClipboardData ; 1 ; $vIPosCR-1)
          `コロン"."があるか？
```

```

$viPosColon:=Position (":" ; $vtFieldData)
If ($viPosColon>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 ; $viPosColon+2)
End if
  `フィールド番号をインクリメントする
  $viField:=$viField+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 ($viField<=Count fields ($vpTable))
  `フィールドタイプを取得する
  GET FIELD PROPERTIES ($1 ; $viField ; $viFieldType)
  `フィールドポインタを取得する
  $vpField:=Field ($1 ; $viField)
  `フィールドのタイプに従って、適切な方法でそのデータを
  `コピーする(またはしない)
  Case of
    ¥ (($viFieldType=Is Alpha field ) | ($viFieldType=Is Text))
      $vpField->:=$vtFieldData
    ¥ (($viFieldType=Is Real ) | ($viFieldType=Is Integer ) |
      ($viFieldType=Is LongInt))
      $vpField->:=Num ($vtFieldData)
    ¥ ($viFieldType=Is Date )
      $vpField->:=Date ($vtFieldData)
    ¥ ($viFieldType=Is Time )
      $vpField->:=Time ($vtFieldData)
    ¥ ($viFieldType=Is 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 ; $viPosCR+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PEND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。

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' ; \$vxSykData) `Syk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。

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。

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。

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。

```
$vSize:=Test clipboard ("TEXT")
```

Case of

```
¥ ($vSize<=0)
```

```
  ALERT ("クリップボードにテキストはありません。")
```

```
¥ ($vSize<=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
```

```
¥ ($vSize>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。

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くテキスト>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。

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、以下のいずれかひと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ひと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が、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

この章では、「ルーチン」エディタの「Communications」テーマ内にある通信コマンドについて説明します。この章のコマンドは、ドキュメントファイルやシリアルポートから、データの書き出しと読み込みを行います。これらのコマンドのほとんどがディスク上のドキュメントと共に作動します。ドキュメント上での動作に関する詳細は、第52章の「システムドキュメントコマンド」を参照してください。

RECEIVE BUFFER

RECEIVE PACKET

RECEIVE RECORD

RECEIVE VARIABLE

SEND PACKET

SEND RECORD

SEND VARIABLE

SET CHANNEL

SET TIMEOUT

USE ASCII MAP

SET CHANNEL

SET CHANNEL (ポート ; セットアップ)

引数	タイプ	説明
ポート	数値	使用するシリアルポート番号
セットアップ	数値	シリアルポートの設定

SET CHANNEL (処理 ; {ドキュメント})

引数	タイプ	説明
処理	数値	ファイル処理コード
ドキュメント	文字列	処理対象のドキュメントファイル名

説明

SET CHANNEL コマンドには、2つの形式があります。第1の形式は、シリアルポートを開きます。第2の形式は、ドキュメントファイルを開きます。このコマンドは、同時にひとつのポートまたはひと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。

SET CHANNEL コマンドの第1の形式は、シリアルポートを開きます。同時にプロトコルや、ポートに対するその他の情報を設定します。

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**とともに使用できます）。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。

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> 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（一度にひとつ）。以下の表はポートの値を示します。

範囲	説明
0	プロトコルを使用しないプリンタポート（Macintosh） またはCOM2（PC）
1	プロトコルを使用しないモデムポート（Macintosh） またはCOM1（PC）
20	XON / XOFF等のソフトウェアプロトコルのプリンタポート （Macintosh）またはCOM2（PC）
21	XON / XOFF等のソフトウェアプロトコルのモデムポート （Macintosh）またはCOM1（PC）
30	RTS / CTS等のハードウェアプロトコルのプリンタポート （Macintosh）またはCOM2（PC）
31	RTS / CTS等のハードウェアプロトコルのモデムポート（Macintosh） またはCOM1（PC）
100から199	プロトコルを使用しないのシリアル通信
201から299	XON / XOFF等のソフトウェアプロトコルのシリアル通信
301から399	RTS / CTS等のハードウェアプロトコルのシリアル通信

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

標準的な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するために追加することができます。元は、4th Dimensionは2つの標準のシリアルポートのみアドレスし、追加したポートのサポートを実行するのは遅れていました。両立する理由で、両方をアドレスするシステムは保留でした。

標準のシリアルポート（プリンター / COM2またはモデム / COM1）をアドレスするには、ポートのパラメータの値を0、1、20、21、30、31のどれかににすることが出来ます（前のアドレスメゾットに相当します）。または、100以上の値にします（以下の説明を参照してください）。

追加したシリアルポートをアドレスするには、値をN+100にする必要があります（Nはポートのアドレスの値です）。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) プロトコルを使用しないプリンタ / COM2ポートを使用する場合は、以下のシンタックスのひとつを使用します。

SET CHANNEL (0;param)

または、

SET CHANNEL (102;param)

(2) XON / XOFF等のプロトコルのモデム / 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) RTS / CTS等のプロトコルの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<セットアップ>の値を加算して指定します。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
	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

Tip：加算、または引数<ポート>や<セットアップ>に渡す各種数値（COM1...COM99の値は含まれない）は、「デザイン」モードでエクスプローラウィンドウの定数タブ内の「Communications」テーマ内に表示されます。COM1...COM99については、数値そのものを使用します。

注：転送速度の算出方法は、以下の式を使用します。

セットアップ=(57600 / 転送速度) * 2) - 2

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、同時にひと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。ドキュメントコマンドに関する詳細は、第45章を参照してください。

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。次ページの表に、<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。

通常のドキュメントファイルの処理には、ドキュメントファイルコマンドを使用する必要があります。SEND RECORDコマンド、RECEIVE RECORDコマンド、SEND VARIABLEコマンド、RECEIVE VARIABLEコマンドのいずれかを使用する場合にのみ、ドキュメントファイルに対してSET CHANNELコマンドを使用します。ドキュメントファイルコマンドでは、これらのコマンドを操作することができません。

SEND PACKETコマンドとRECEIVE PACKETコマンドは、SET CHANNELコマンドや第52章で説明されているシステムドキュメントコマンドを用いれば、作成またはオープンされたドキュメントから使用可能です。

以下の例は、ImageWriterIIに接続されたプリンタポートを開き、“こんにちは”という単語を出力し、フォームフィードを行って、ポートを閉じます。

```
SET CHANNEL (0 ; 10+3072+16834+0)    `プリンタポートを開く
$単語:="こんにちは"
SEND PACKET ($単語)                    `プリンタに“こんにちは”という単語を送信する
SEND PACKET (Char (12))                `フォームフィードを実行する
SET CHANNEL (11)                        `ポートを閉じる
SET CHANNELコマンドの第2の形式の例を下記に記します。
```

引数に10を指定し、ファイル拡張子が.TXT以外の場合、プログラムでは拡張子とは無関係に既存のファイルをオープンできますが、ファイル拡張子は無視され、ファイルが存在しない場合は.TXTファイルが作られます。例えば、SET CHANNEL (10;"Archive.BAK")によって、Archive.BAKファイルが存在している場合はオープンされますが、ファイルが存在しない場合は、Archive.TXTが作られます。以下のコードは、ドキュメントが存在していてもいなくても.BAKを強制的に作ります。

```
` この例では以下のような想定に基づいています
` MAP FILE TYPES("BACK"; "BAK"; "バックアップドキュメント") がコールされている
$DocRef:=Create document (""; "BACK")
If(OK=1)
  ` SEND/RECEIVE RECORD/VARIABLE は Create/Open/Append document
  ` でオープンされたドキュメントを処理しない
  CLOSE DOCUMENT ($DocRef)
  SET CHANNEL (10 ; Document)
  If (OK=1)
    ` ここでは SEND/RECEIVE RECORD/VARIABLE が使用できる
    SET CHANNEL (11)
  End if
End if
```

参照
なし

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 ファイルまたは 参照番号	時間	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 - 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**コマンドでそのドキュメントファイルの内容がすべて消去されます。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。

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
なし

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参照番号	時間	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。

RECEIVE PACKETコマンドには、2つの形式があります。

第1の形式は、受信する文字数（<文字数>）を指定します。RECEIVE PACKETコマンドは、<受信変数>に指定した文字数分のデータを格納します。

第2の形式は、指定した文字（<終了文字>）が現われるまでデータを読み込みます。RECEIVE PACKETコマンドは、最初の<終了文字>を発見するまで<受信変数>にデータを格納します。RECEIVE PACKETコマンドは、<終了文字>を<受信変数>に格納しません。<受信変数>は、文字またはテキストタイプです。

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で読み込みを終了します。しかし、文字列変数は固定長ですが、テキスト変数は32000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。

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。以下の**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み込む際に、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）。

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“読み込20”に格納します。

RECEIVE PACKET (読み込20 ; 20)

以下の例は、変数“ドキュメント”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“vデータ”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 (ドキュメント ; vデータ ; Char (13))

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、サブルーチン呼び出しで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。

```
ドキュメント:=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 (ドキュメント ; $Var1 ; 15)    ` 名字のデータを15バイト読み込む
  RECEIVE PACKET (ドキュメント ; $Var2 ; 15)    ` 名前のデータを15バイト読み込む
  If (OK=1)                                       ` まだドキュメントファイルを最後まで読み込んでない場合
    CREATE RECORD ([従業員])                   ` 新しいレコードを作成
    [従業員]名字:=Strip ($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 (ドキュメント) ` 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。

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。

以下の表は、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

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 (MacOS 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

なし

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。

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

なし

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がテーブル定義の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ねばなりません。**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。

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 file ([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

なし

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がテーブル定義の応じて値を変換します。

注：

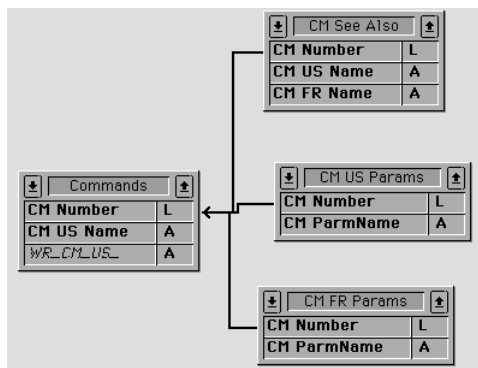
1. このコマンドを使用してドキュメントファイルにレコードを送信する場合、**SET CHANNEL** コマンドを使ってあらかじめドキュメントファイルを開いておかなければなりません。**Open document**、**Create document**、**Append document** で開かれたドキュメントファイルに対して、**RECEIVE RECORD** を使用することはできません。
2. **RECEIVE RECORD** コマンドの実行中に、「Ctrl + Alt + Shift」キー（Windows）または「command + option + shift」キー（Macintosh）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クションを使って作業を実行する。

SET CHANNEL (12;"") `チャンネルを開き、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
$viNbCmd:=Records in selection ([Commands])
SEND VARIABLE ($viNbCmd)
FIRST RECORD ([Commands])
` 各コマンドに対し
For ($viCmd;1;$viNbCmd)
` [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")
    $viNbParm:=Records in selection ([CM US Params])
  ¥ (BUILD_LANG="FR")
    $viNbParm:=Records in selection ([CM FR Params])
End case
SEND VARIABLE ($viNbParm)
` 引数レコードを送信（存在する場合）
For ($viParm;1;$viNbParm)
  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
$viNbSee:=Records in selection ([CM See Also])
SEND VARIABLE ($viNbSee)
` [See Also] レコードを送信（存在する場合）
For ($viSee;1;$viNbSee)
  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？
    ` タグを元に言語を取り出す
    $CurLang:=Substring ($vsTag;Length ($vsTag)-1)
    If ((($CurLang="US") | ($CurLang="FR"))) ` 有効な言語を取得したか？
      ` このドキュメントにはいくつのコマンドが存在するか？
      RECEIVE VARIABLE ($vINbCmd)
      If ($vINbCmd>0) ` 少なくともひとつ存在するなら
        For ($vICmd;1;$vINbCmd) ` アーカイブされた [Commands] 各レコードに対して
          ` レコードを受信
          RECEIVE RECORD ([Commands])
          ` 新規レコードの保存、または既存のレコードにその値をコピーする
          ` サブルーチン呼び出す
          CM_IMP_CMD ($CurLang)
          ` 引数の数を受信（ある場合）
          RECEIVE VARIABLE ($vINbParm)
          If ($vINbParm>=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呼び出す
            CM_IMP_PARM ($vINbParm;$CurLang)
          End if
          ` “参照” の数を受信（ある場合）
          RECEIVE VARIABLE ($vINbSee)
          If ($vINbSee>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呼び出す
            CM_IMP_SEEA ($vINbSee;$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ない")
  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

なし

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。シリアルポート経由でドキュメントファイルに対して配列の送受信を行いたい場合、**BLOB**コマンドを使用します。

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 (私の変数)

参照

なし

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。シリアルポート経由でドキュメントファイルに対して配列の送受信を行いたい場合、**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。

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)
```

参照

なし

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。

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テーブ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

なし

4D Compilerは、アセンブラ命令でデータベースアプリケーションを解析することができます。4D Compilerの利点には、以下のようなものがあります。

「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」：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。データベースをコンパイルすることにより、安全性を高めます。

「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」：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。

倍長整数変数および整数変数に対する数値演算は、通常、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なります。

コンパイルするコードの記述に関する一般規則

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、4D Compilerのドキュメントに記載されている通り、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。

4D Compiler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必要があります。

変数を明示的にタイプ定義するもうひとつの理由は、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。

例題

(1) 以下は、4D Compiler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 ($vIPacket)
For ($vIPacket ; 2 ; Count parameters)
    SEND PACKET ($1 ; ${$vIPacket})
End for
```

(5) 以下の例では、プロジェクトメソッド「COMPILER_Param_Predeclare28」は4D Compiler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}) ` ...
```

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_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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

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

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なっています。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。

参照

前節での例を参照してください。

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_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。

参照

前節での例を参照してください。

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。

参照

前節での例を参照してください。

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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

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_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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

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なっています。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。

参照

前節での例を参照してください。

IDLE

IDLE

説明

IDLEコマンドは、4D Compilerのため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

 ` 4Dコマンドを含まない処理を実行する

IDLE

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

なし

この章では、「ルーチン」エディタの「Data Entry」テーマ内にあるデータ入力コマンドについて説明します。カスタムデータベースを作成するときには、この章のコマンドを使用します。この章のコマンドを使用してデータの入力、画面表示、レポート印刷を行います。これらのコマンドは、フォームに関連します。

ADD RECORD	Modified	Old
ADD SUBRECORD	MODIFY RECORD	
DIALOG	MODIFY SUBRECORD	

データ入力は、データベースの基本的な機能のひとつです。ユーザが、データベースにデータを入力するための重要な処理手順です。**ADD RECORD**コマンドと**MODIFY RECORD**コマンドは、データ入力で最もよく使用されるコマンドです。これらのコマンドは、「ユーザ」モードの「新規レコード」や「レコード修正」メニューコマンドと同じ動作をします。

入力フォームは、データを入力するために使用します。入力フォームは、複数のページを持ち、それぞれのページに異なるデータまたは異なる方法で示された同じデータを持つことができます。入力フォームに関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

ADD SUBRECORDコマンドと**MODIFY SUBRECORD**コマンドはサブレコードを追加したり、修正します。それらはサブレコードに関して、**ADD RECORD**コマンドや**MODIFY RECORD**コマンドと同じ動作をします。

DIALOGコマンドは、ユーザにデータのやり取りを行わせる際によく使用されます。また、レコードに直接関係のないデータを入力する際にも使用されます。

Modified関数は、データ入力時に修正されたフィールドの場合に“True（真）”を返します。

Old関数は、カレントレコードの<フィールド>の修正する前の内容を返します。

データ入力時にカレントレコードを変更する

各テーブルは、カレントセクションとカレントレコードを持っています。この節では、カレントレコードについて説明することにします。**ADD RECORD**コマンド、**MODIFY RECORD**コマンド、**MODIFY SELECTION**コマンドのいずれかのコマンドを使用してレコードを表示する場合に、表示されたレコードがカレントレコードになります。**NEXT RECORD**コマンド等を使用してカレントレコードを変更した場合や、表示レコードへの変更をディスクに保存する必要がある場合は、**SAVE RECORD**コマンドを使用しなければなりません。ただし、「次レコード」ボタンをクリックしたり、その他の自動動作ボタンをクリックした場合には、レコードを自動的に保存するために**SAVE RECORD**コマンドを使用する必要はありません。レコードを自動的に保存することのできる自動動作ボタンには、「登録」、「次レコード」、「前レコード」、「先頭レコード」、「最終レコード」の各ボタンがあります。

コマンドでカレントレコードを変更しても、Beforeフェーズは発生しません。しかし、自動動作ボタンを使用して他のレコードに移動すると、Beforeフェーズが発生します。

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取り消し

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受け入れるか、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ピネーション（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。

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トが変更される前に**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ひとつを実行します。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。少なくともひと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

なし

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取り消し

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。

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トが変更される前に**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

なし

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取り消し

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。

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。

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
なし

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取り消し

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修正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。

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
なし

DIALOG

DIALOG ({テーブル;}フォーム)

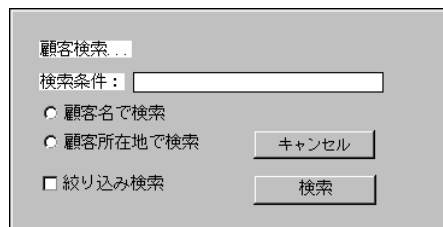
引数	タイプ	説明
テーブル	テーブル名	フォームの属するテーブルまたは省略した場合、デフォルトテーブル
フォーム	文字列	ダイアログとして表示するフォーム

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。



ダイアログにおいて、データの入力は変数を介してのみ実行することができます。フィールドには現在の値を表示することはできますが、入力することはできません。

Tip：フィールドデータ入力で提供されるような機能が必要な場合、ダイアログは、**ADD RECORD**コマンドのようなことができます。この場合に、フォームが受け入れられるとレコードはそのテーブルに追加されます。

Tip：逆に、**DIALOG**コマンドを使用して、データ入力を実行することができます。この場合は、ユーザはレコードを作成し、保存しなければなりません。**DIALOG**コマンドはレコードを操作しません。

データを表示したり、情報を得る際に**ALERT**コマンドや**CONFIRM**コマンド、**Request**関数の代わりに**DIALOG**コマンドを使用すると、これらのコマンドで処理するよりも複雑になります。

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。その代わりに「OK」ボタンと「キャンセル」ボタンが自動的に生成されます。任意のカスタムボタンを作成した場合には、「OK」ボタンと「キャンセル」ボタンのどちらも生成されません。

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、ダイアログが受け付けられます。

「キャンセル」ボタンをクリックされたり、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。

次の例は、検索条件を入力するための**DIALOG**コマンドの使用法を示しています。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
なし

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	判定に対するフィールド
戻り値	ブール	フィールドに新しい値が割り当てられた場合に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

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呼び出ししますが、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
それをするのも忘れないで
` ...
` 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
なし

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

なし

この章では、「ルーチン」エディタの「Date and Time」テーマ内にある日付と時間のコマンドおよび関数について説明します。日付と時間は、数値と同様に計算に用いることができます。西暦2000年に対応したコマンドが新たに追加されました。

以下のような式を用いて、時間、分、秒を計算することができます。

時間:=変数時間 // 3600	、時間を計算
分:=(変数時間 // 60) % 60	、分を計算
秒:=変数時間 % 60	、秒を計算

この章では、以下の関数について説明します。

Add to date	Day number	Time String
Current date	Day of	SET DEFAULT CENTURY
Current time	Month of	Year of
Date	Time	Tickcount
Milliseconds		

Current date

Current date {(*)} 日付

引数	タイプ	説明
*	*	サ - 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えます。

` 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;\$vIPos)

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
なし

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。

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

なし

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。

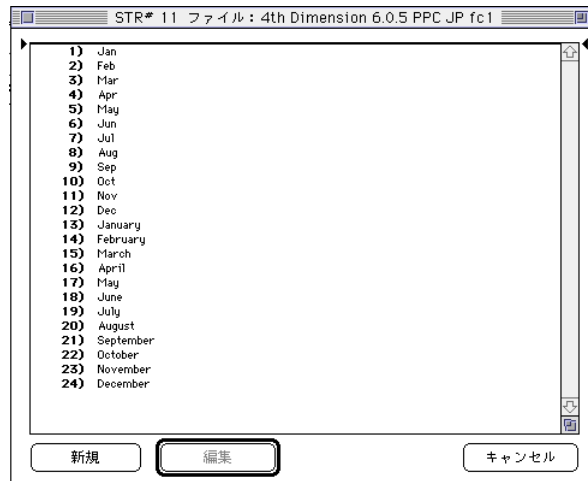
v結果:=**Month of** (!97.07.21!)

`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

なし

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。

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

なし

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。

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

なし

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ヶ月の日数やう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。

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

なし

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")
```

参照

なし

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。

デフォルトでは、4Dは世紀を20世紀にセットします。例えば、

97.01.25は、1997年1月25日を意味します。

07.01.25は、1907年1月25日を意味します。

このデフォルトを変更するには、マイナス1した新しいデフォルトの世紀を引数 < 世紀 > に渡します。例えば、以下のコードを呼び出すと、

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、オプション引数の < ピボット年度 > を指定すると、4Dは2桁年度の日付入力を以下のように解釈します。

入力した2桁年度がピボット年度と等しいかそれより大きい場合は、4Dは現在のデフォルト世紀を使用します。

入力した2桁年度がピボット年度より小さい場合は、4Dは現在のデフォルト世紀に1プラスした世紀を使用します。

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。

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
なし

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。

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
なし

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。

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

なし

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。

Milliseconds関数の例を参照してください。

参照

Current time、Milliseconds

Milliseconds

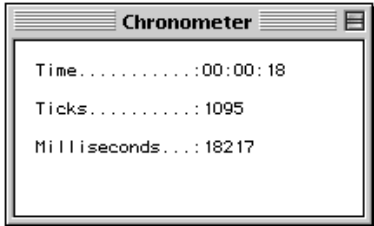
Milliseconds 数値

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。

注：Milliseconds関数は、実数型の値を戻します。
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

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。

DRAW AND DROP PROPERTIES

Drop position

4th Dimensionのバージョン6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ました。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。

バージョン6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ACI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。

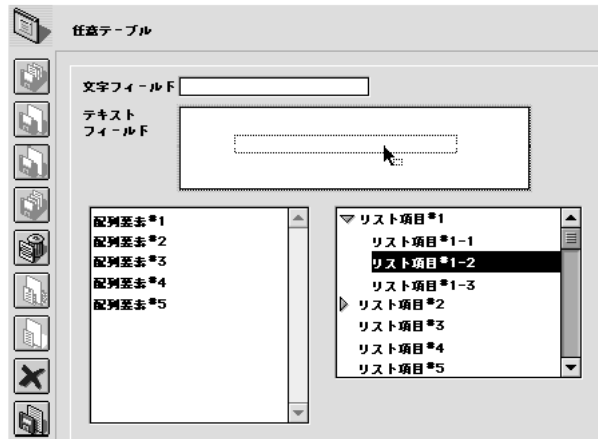
注：ドラッグとドロップの両方ができるオブジェクトは、ユーザが禁止しない限り、それ自体にもドロップできます。詳細については、以下の説明を参照してください。

以下の図は、選択したオブジェクトに対して「オブジェクトプロパティ」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。



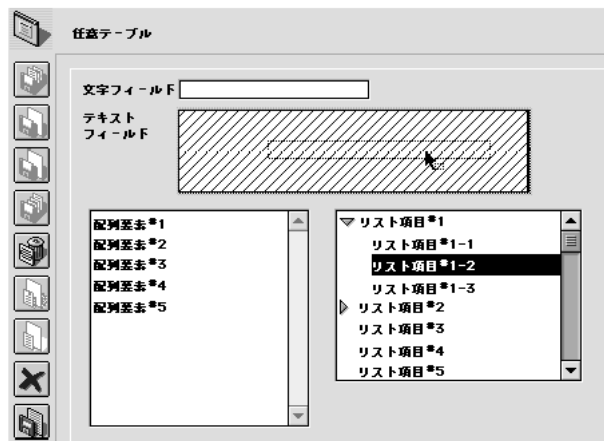
テキストフィールドエリアの周りに反転したグレーのフレームのハイライト（点滅）があることに注目してください。このハイライトは、送信先オブジェクト（上の図では、テキストフィールド）を示します。この箇所でマウスボタンを放すと、4th Dimensionは、ユーザがハイライトされた送信先オブジェクトにドラッグしたオブジェクトをドロップしたいものと想定します。

「データベースプロパティ」ダイアログボックスでは、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。

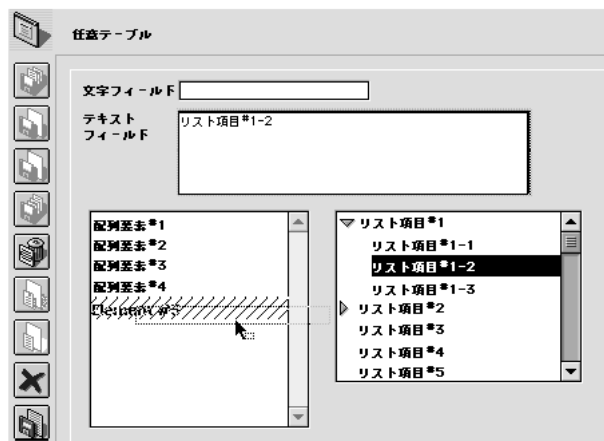


デフォルトのハイライト（点滅）は、「フレーム」です。これは、オブジェクトの周りを囲むグレーの反転した矩形です。色付きの背景または色付きのオブジェクトフレームを使用する場合、このハイライト（点滅）を使用するとわかりにくくなります。この場合は、別の手段として、「パターン」ハイライトを使用します。これは、以下に示すように送信先オブジェクトを斜線パターンで塗りつぶします。

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「オブジェクトプロパティ」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ことです。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&ドロップがひと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なく、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。

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ひとつ）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응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응答する必要があります。以下の例では、イベントに응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、フローティングウィ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、送信先ウィンドウ（送信先オブジェクトが存在するウィンドウ）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られない場合には、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 ; $vSrcElem ; $vPID)
  If ($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 ; $vSrcElem ; $vPID)
  `送信先の配列要素番号を取得する
  $vIDstElem:=Drop position
  `配列要素が配列要素自体にドロップされなかった場合
  If ($vIDstElem # $vSrcElem)
    `配列の要素0にドラッグされた要素を保存する
    $1->{0}:=$1->{$vSrcElem}
    `ドラッグされた配列要素を削除する
    DELETE ELEMENT ($1-> ; $vSrcElem)
    `送信先要素がドラッグされた要素を超える場合
    If ($vIDstElem>$vSrcElem)
      `送信先の配列要素番号から1引く
      $vIDstElem:=$vIDstElem-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 ($vIDstElem=-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
      $vIDstElem:=Size of array($1->)+1
    End if
    `この新しい配列要素を挿入する
    INSERT ELEMENT ($1-> ; $vIDstElem)
    `配列の要素ゼロに先ほど保存した値を設定する
    $1->{$vIDstElem}:=$1->{0}
```

```

`要素は新しく選択された配列の要素になる
$1->:=$vIDstElem
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;$vIDSrcElem;$vIPID)
  ` 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
    $vIDSrcType:=Type($vpSrcObj->)
    Case of
      ¥ ($vIDSrcType=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->
      ¥ ($vIDSrcType=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;$vsVarName;$vITableNum;$vIFieldNum)
`フィールドの場合
If (($vITableNum>0) & ($vIFieldNum>0))
` IP 変数へ値をコピー
    <>vtDraggedData:=$vpSrcObj->
End if
¥ ($vISrcType=Is String Var)
` 文字列変数はOK
    $0:=0
¥ (($vISrcType=String array) | ($vISrcType=Text array))
` 文字列配列およびテキスト配列はOK
    $0:=0
¥ (($vISrcType=Is LongInt) | ($vISrcType=Is Real))
If (Is a list($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;$vISrcElem;$vIPID)
    RESOLVE POINTER($vpSrcObj;$vsVarName;$vITableNum;$vIFieldNum)
    ` フィールドの場合
    If (($vITableNum>0) & ($vIFieldNum>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
        $vISrcType:=Type($vpSrcObj->)
    Case of
    ` 配列の場合
        ¥ (($vISrcType=String array) | ($vISrcType=Text array))
        If ($vIPID # Current process)
        ` この変数の対象プロセスのインスタンスから要素を読み込む
            GET PROCESS VARIABLE($vIPID;
                $vpSrcObj->{$vISrcElem};$vtDraggedData)
        Else
        ` 配列要素をコピー
            $vtDraggedData:=$vpSrcObj->{$vISrcElem}
        End if
        ¥ (($vISrcType=Is LongInt) | ($vISrcType=Is Real))
        ` 階層リストの場合

```

```

If (Is a list($vpSrcObj->))
  ` 別プロセスのリストの場合
    If ($vIPID # Current process)
      ` 別プロセスのリストリファレンスを取得
      GET PROCESS VARIABLE($vIPID;$vpSrcObj->;$vList)
    Else
      $vList:=$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($vList;$vISrcElem;$vItemRef;$vItemText)
    $vtDraggedData:=$vItemText
  End if
Else
  ` 文字列変数またはテキスト変数
    If ($vIPID # Current process)
      GET PROCESS VARIABLE($vIPID;$vpSrcObj->;$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もある)
  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

この章では、「ルーチン」エディタの「Entry Control」テーマ内にある入力制御コマンドについて説明します。この章のコマンドは、デ - タの入力を制御、またはフォームを閉じるために使用します。

ACCEPT	FILTER KEYSTROKE	Keystroke
CANCEL	GOTO AREA	REJECT

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、メニューコマンド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」メニューコマンド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、ひとつの**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
なし

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、レコードセクションを表示しているフォームを終了するため

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。フォームをキャンセルした後で、システム変数OKは0に設定されます。

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」メニューコマンド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、ひとつの**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。

参照
なし

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**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無効になってから、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。

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り分ける場合

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 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-> ; $vIStart ; $vIEnd)
`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 ; $vIStart-1-Num($vIStart=$vIEnd))
+Substring ($vtNewValue ; $vIEnd)
```

```
`受け付け可能な文字が入力された
¥ (Position (Keystroke ; "abcdefghijklmnopqrstuvwxyz -0123456789")>0)
If ($vIStart# $vIEnd)
`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 ; $vIStart-1)
+Keystroke+Substring ($vtNewValue ; $vIEnd)
```

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
¥ ($vIStart<=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
¥ ($vIStart>=Length ($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 ; $vIStart-1)+Keystroke
        +Substring ($vtNewValue ; $vIStart)
    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 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 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:="ひと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。

The screenshot shows a graphical user interface for searching cities. At the top, there is a label '都市の名前:' (City Name:). Below it is a text input field containing 'new h'. Underneath the input field is a status line that says '13 cities found.'. Below this is a list box containing the following city names: New Hamburg, New Hampton, New Hanover, New Harbor, New Harmony, New Hartford, New Haven, New Haven Heights, New Hebron, New Holland, New Hope, and New Hopewell. The list box has a vertical scrollbar on its right side.

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ィンドウで検索機能を提供し、そのウィ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

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 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、編集用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の有効になってから、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 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 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基にして、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トを渡します。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-> ; \$vIStart ; \$vIEnd)

`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;\$vIStart;\$vIEnd)

`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 ; \$vIStart ; \$vIEnd ; \$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 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;$2)
C_LONGINT ($vIStart;$vIEnd)
` 入力可能エリア内のテキスト選択範囲を取得
GET HIGHLIGHT ($1-> ; $vIStart ; $vIEnd)
    `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-> ; $vIStart ; $vIEnd)
    `検索すべきものがあるか？
    If ($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 ($vIStart=$vIEnd)
            $vIStart:=$vIStart-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+"@")
        `ひとつあった場合
        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-> ; $vIStart ; $vIEnd ; [辞書]見出し)
            `投影テキストを編集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。
            $vIEnd:=$vIStart+Length ([辞書]見出し)
            HIGHLIGHT TEXT (vsComments ; $vIEnd ; $vIEnd)
        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、フォームのアクティブオブジェクトとして<入力オブジェクト>を選択するために使用します。このコマンドは、ユーザがフィールドや変数の中をクリック、または“tab”キーを押すのと同じことを実行することができます。

このコマンドの構文は、引数を選択できるように変更されました。

フィールドまたは変数の参照（これまでと同じ）

例：**GOTO AREA** ([Personnel]Name)

オブジェクト名での参照

例：**GOTO AREA** (*;"MyVar")

パラメータオプションを指定するときは、オブジェクトにオブジェクト名（文字列）を付けます。パラメータオプションを省略するときは、フィールドかオブジェクト変数を付けます。その場合は、文字列の代わりにフィールドかリファレンス変数（フィールドかオブジェクト変数のみ）を付けます。オブジェクト名の説明は、第35章「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機能し、サブフォームのリストフォーム上のデータ入力エリアでは何も行いません。

例

(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
なし

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、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

この章では、「ルーチン」エディタの「Form event」テーマ内にあるフォームイベント関数について説明します。この章の関数は、フォームフォームイベントを判定するために使用します。しかし、できる限り、フォームメソッドのフォームイベントに対する判定の代わりにオブジェクトメソッドを使用してください。ここの関数は、フォームメソッドとオブジェクトメソッド以外では無効です。これらはより効果的なコマンドです。

Activated	During	In footer
After	Form event	In header
Before	Get edited text	Outside call
Deactivated	In break	SET TIMER

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は、前もって定義された以下のような定数を持っています。

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するとオブジェクト内の入力前のテキストを返します。 【Get edited text returns the object's text without this character】
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、最後に入力された文字をエリアの内容を返します。 【Get edited text returns the object's text including this character】
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す。
On Losing Focus	14	フォームオブジェクトがフォーカスを失います。
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す。
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 Menu Selected	18	メニューアイテムが選択されました。
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。

参照
なし

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。

4th Dimensionはまず、フォームのオブジェクトを順々にブラウズしていき、次に（イベント内に含まれている）対応するイベントプロパティが選択された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、対応するフォームのイベントプロパティが選択されている場合、4th Dimensionは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、特別な順序で呼び出されることはありません。大雑把に言えば、オブジェクトメソッドは常にフォームメソッドの前に呼び出されます。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。その後、4th Dimension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、4th Dimension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を上述と同じように使用します。

On LoadイベントとOn Unloadイベントを除き、フォームのイベントプロパティが与えられたイベント用に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呼び出しが妨げられることはありません。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。

ひとつのイベント内に含まれるオブジェクト数は、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（あらゆるページの）フォームのオブジェクトは、すべてこれらのオブジェクトメソッドを呼び出させます。そして、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、そのフォームは自身のフォームメソッドが呼び出されることを確認します。

On Activateイベント-オブジェクトメソッドは呼び出されません。なぜならこのイベントは特定のオブジェクトに適用されるのではなく、全体としてのフォームに適用されるからです。この結果、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、フォームだけがフォームメソッドが呼び出されることを確認します。

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ているドロップ可能なオブジェクトだけが、自身のオブジェクトメソッドが呼び出されることを確認します。フォームメソッドは呼び出されません。

警告：他のイベントとは異なり、On Drag Overイベント中は、オブジェクト用のオブジェクトメソッドは、ドラッグ&ドロップソース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&ドロップの送信先オブジェクトのプロセスコンテキスト内で実行されるものではありません。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。

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 (クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Double Clicked	はい (クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Before Keystroke	はい (キーボード入力可能な場合) (*)	はい	関連するオブジェクトのみ
On After Keystroke	はい (キーボード入力可能な場合) (*)	はい	関連するオブジェクトのみ
On Getting Focus	はい (タブ可能な場合) (*)	はい	関連するオブジェクトのみ
On Losing Focus	はい (タブ可能な場合) (*)	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 (ドロップ可能な場合) (*)	はい	関連するオブジェクトのみ
On Drag Over	はい (ドロップ可能な場合) (*)	なし	関連するオブジェクトのみ
On Menu Selected	なし	はい	なし
On Data Change	はい (変更可能な場合) (*)	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 Detail	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 Detail	はい	はい	全オブジェクト
On Open Detail	なし	はい	なし
On Close Detail	なし	はい	なし
On Resize	なし	はい	なし
On Timer	なし	はい	なし

(*)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。

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ください。「デザイン」モードで（「フォームプロパティ」および「オブジェクトプロパティ」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。

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

表示フォーマットが「バックグラウンド」に設定されているピクチャフィールドやピクチャ変数

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

スクロール可能エリア、階層リスト

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と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ひと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ますようになります。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。

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、以下の2つの例外があります。

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
- 階層リスト

これらオブジェクトのひとつに対し、On Before Keystrokeか又は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：これらの2つのイベントは、関連するオブジェクトメソッドでのみ使用できます。これらは**POST KEY**コマンドによって発生される事ができます。

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）

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ひとつに対し、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。

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、メニューコマンドやボタンのようなオブジェクトへの（Windowsでの）アクセラレータや（Macintoshでの）ショートカット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。

イベントカテゴリー

フォームイベントは、以下のカテゴリーに分類できます。

一般的なイベント

On Load、On Unload、On Validate、On Display Details、On Open Details、
On Close Details

フォームに適応したイベント

On Activate、On Deactivate、On Outside Call、On Close Box、On Menu Selected、
On Open Details、On Close Details、On Timer、On Resize

ユーザアクション（動作）に関連したイベント

On Clicked、On Double Clicked、On Keystroke、On Getting Focus、On losing Focus
On Data Change、On Plug in Area、On Before Keystroke、On After Keystroke

ドラッグ&ドロップのイベント

On Drop、On Drag Over

印刷用のイベント

On Header、On Printing Details、On Printing Break、On Printing Footer

バージョン6とバージョン3の間の互換性について

以下に示すのは、バージョン6のフォームイベントとバージョン3のレイアウト実行サイクルとの対応表です。

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Load	Beforeフェーズ	Before
On Unload	同等の実行サイクルはありません	なし
On Validate	Afterフェーズ	After
On Clicked	一般的なDuringフェーズ	During
On Double Clicked	一般的なDuringフェーズ	During
On Before Keystroke	同等の実行サイクルはありません	なし
On After Keystroke	同等の実行サイクルはありません	なし
On Getting Focus	同等の実行サイクルはありません	なし
On Losing Focus	同等の実行サイクルはありません	なし
On Activate	Activatedフェーズ	Activated
On Deactivate	Deactivatedフェーズ	Deactivated
On Outside Call	Outside Callフェーズ	Outside Call
On Drop	同等の実行サイクルはありません	なし
On Drag Over	同等の実行サイクルはありません	なし
On Menu Selected	一般的なDuringフェーズ	DuringとMenu selected

On Data Change	一般的なDuringフェーズ	During
On Plug in Area	一般的なDuringフェーズ	During
On Header	印刷時のヘッダフェーズ	In header
On Printing Detail	一般的なDuringフェーズ	During
On Printing Break	印刷時のブレイクフェーズ	In break
On Printing Footer	印刷時のフッタフェーズ	In footer
On Close Box	同等の実行サイクルはありません	OPEN WINDOW (クローズボックス付き)
On Display Detail	BeforeおよびDuringフェーズ	BeforeとDuring
On Open Detail	一般的なDuringフェーズ	During
On Close Detail	一般的なDuringフェーズ	During

ユーザがバージョン6の4th Dimensionを使用してバージョン3のデータベースを開くと、プログラムは2つのオペレーションを実行します。

ストラクチャファイルの新規フォーマットへの変換

データファイルの新規フォーマットへの変換

変換後のデータベースを使用する際、「デザイン」モードでフォームが編集または変更されていない場合、フォームはバージョン3の時と同じ状態でストラクチャファイル内にまだ記憶されていることとなります。ユーザのバージョン3の既存アプリケーションとの互換性を確実にするために、フォームとオブジェクトのイベントプロパティは自動的に設定され、その設定は「ala V1」に反映します。つまり、バージョン6のイベントプロパティが自動的に選択され、そして「古いV1コマンド」はバージョン1の時と同じように機能します。

オブジェクト（フィールドまたは変数）にバージョン3の「変更時のみ実行」オプションが選択されている場合、そのイベントプロパティはバージョン3のデータ入力中に発生する**During**実行サイクルに対応するプロパティに縮小されます。

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Clicked	一般的なDuringフェーズ	During
On Double Clicked	一般的なDuringフェーズ	During
On Data Change	一般的なDuringフェーズ	During
On Plug in Area	一般的なDuringフェーズ	During

バージョン6でフォームとそのオブジェクトの編集を開始すると、デフォルトではフォームとオブジェクトのイベントプロパティは同じ方式で設定されます。バージョン6の新規イベントを利用するには、「デザイン」モードでフォームとオブジェクト用のイベントプロパティを選択し、それから**Form event**関数を使用してフォームメソッドとオブジェクトメソッドを変更します。

バージョン3の実行サイクルに対応しないバージョン6の新規イベントは、以下の通りです。

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Unload	同等の実行サイクルはありません	なし
On Keystroke	同等の実行サイクルはありません	なし
On Getting Focus	同等の実行サイクルはありません	なし
On Losing Focus	同等の実行サイクルはありません	なし
On Drop	同等の実行サイクルはありません	なし
On Drag Over	同等の実行サイクルはありません	なし
On Close Box	同等の実行サイクルはありません	OPEN WINDOW (クローズボックス付き)

以下の新規イベントを使用することで、よりイベントの性質に適合する動作を実行できます。

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Clicked	一般的なDuringフェーズ	During
On Double Clicked	一般的なDuringフェーズ	During
On Menu Selected	一般的なDuringフェーズ	DuringとMenu selected
On Data Change	一般的なDuringフェーズ	During
On Plug in Area	一般的なDuringフェーズ	During
On Printing Detail	一般的なDuringフェーズ	During
On Display Detail	BeforeとDuringフェーズ	BeforeとDuring
On Open Detail	一般的なDuringフェーズ	During
On Close Detail	一般的なDuringフェーズ	During

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}+" のハンバーガーをひとつ!")

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あることに注意して

`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を注意してください。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 (つまり同じウィンドウと

`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->

Else

`違う場合、ソースオブジェクトは変数ですか？

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してください。

```

        GET PROCESS VARIABLE ($!SrcProcess ; $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。
¥ ($vDataType=Picture Array)
`ソースオブジェクトは同じプロセス（つまり同じウィンドウと
`フォームから）のものでしょうか？
If ($!SrcProcess=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:=$vpSrcObject->{$vSrcElement}
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 ($!SrcProcess ; $vpSrcObject
    ->{$vSrcElement} ; $vgDraggedPict)
    [テーブル]ピクチャ:=$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->))

`最初のブレイクヘッダ用のコードはここです。

¥ (Level =1)

`ブレイクヘッダレベル1用のコードはここです。

¥ (Level =2)

`ブレイクヘッダレベル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)

`ブレイクが印刷されようとしています。

Case of

¥ (Level =0)

`ブレイクレベル0用のコードはここです。

¥ (Level =1)

`ブレイクレベル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 Detail**)

\$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:="メニューアイテム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 Detail**)

\$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 Detail**)

\$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:="メニューアイテムが選択されました。"

¥ (**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 even=**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 (Last object ; \$vsVarName ; \$vITableNum ; \$vIFieldNum)

If ((\$vITableNum#0) & (\$vIFieldNum#0))

Case of

¥ (\$vIFieldNum=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。"

...

¥ (\$vIFieldNum=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、クローズウィ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 プール

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 プール

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

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

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ィ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

SET TIMER

SET TIMER (チック数)

引数	タイプ	説明
チック数	倍長整数	設定するチック数

説明

このコマンドは、現在のプロセスで、指定されたTick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。

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
End if
```

参照

On Timerフォームイベント

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。

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icolon、アポストロフィー等の区切り）。

参照

なし

この章では、「ルーチン」エディタの「Form Pages」テーマ内にあるフォームページコマンドについて説明します。この章のコマンドは、フォームページを操作します。

FIRST PAGE、**LAST PAGE**、**NEXT PAGE**、**PREVIOUS PAGE**の各コマンドと同じ処理を実行する自動動作ボタンが4th Dimension内部に用意されています。さらに、バージョン6からはと同等の自動動作が導入され、タブコントロールやドロップダウンリストボックス等のオブジェクトに割り当てることができます。適宜、これらのコマンドの代わりに自動動作ボタンを使用できます。

Current form page	GOTO PAGE	NEXT PAGE
FIRST PAGE	LAST PAGE	PREVIOUS PAGE

ページコマンドは、入力フォームやダイアログに表示されるフォームで使用することができます。出力フォームは先頭のページしか使用しません。フォームは常に、最低でも先頭の1ページが存在します。フォームのページ数に関係なく、ひとつのフォームにはひとつのフォームメソッドしか存在しません。

表示されているページを調べるには、**Current form page**コマンドを使用します。

注：フォームを設計する際に、1ページからNページまでだけでなく、0ページも作成することができます。0ページには、すべてのページに表示されるオブジェクトを配置します。フォームを使用し、フォームページコマンドを呼び出すときには、1からNまでのページを使って作業します。0ページは、表示されているページと自動的に組み合わせられます。

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。

以下の例は、メニューコマンドから呼び出される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。

以下の例は、メニューコマンドから呼び出される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以下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、**NEXT PAGE**コマンドは何も実行しません。

以下の例は、メニューコマンドから呼び出される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。

以下の例は、メニューコマンドから呼び出される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

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。

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

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。

フォームにおいて、メニューバーから任意のメニューコマンドを選択、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

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

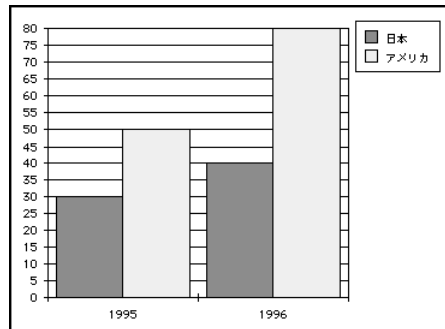
1. 以下の例は、配列を使用してグラフを作成します。このコードは、フォームメソッドやオブジェクトメソッドで使用します。データが定数のため、あまり現実的ではありません。

```

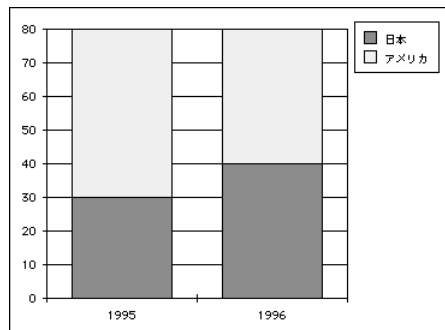
ARRAY STRING (4 ; X ; 2)           ` X軸の配列作成
X{1}:="1995"                          ` X Label #1
X{2}:="1996"                          ` X Label #2
ARRAY REAL (A ; 2)                  ` Y軸の配列作成
A{1}:=30                              ` データを挿入
A{2}:=40
ARRAY REAL (B ; 2)                  ` Y軸の配列作成
B{1}:=50                              ` 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。

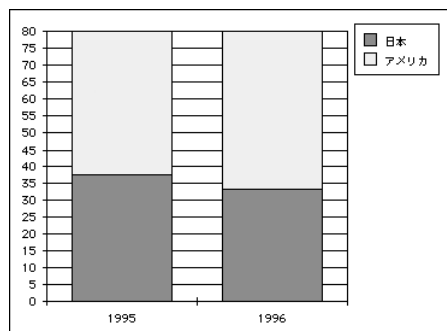
「vタイプ」を1にすると、「棒グラフ」で表示します。



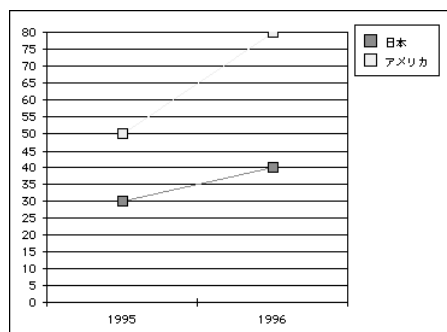
「vタイプ」を2にすると、「棒・比率グラフ」で表示します。



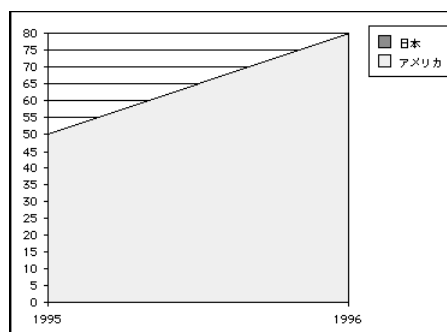
「vタイプ」を3にすると、「棒・累積グラフ」で表示します。



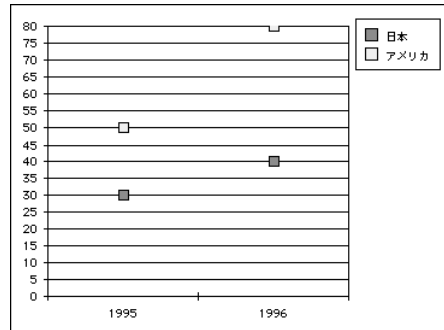
「vタイプ」を4にすると、「線グラフ」で表示します。



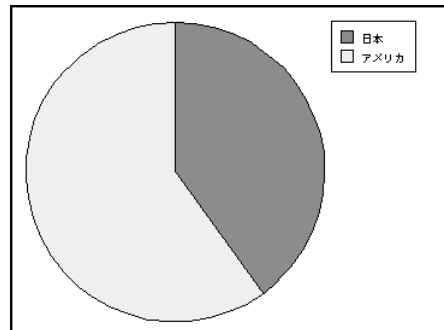
「vタイプ」を5にすると、「面グラフ」で表示します。



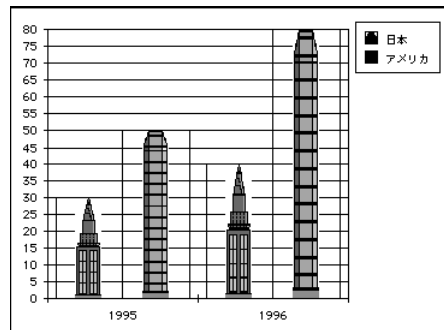
「vタイプ」を6にすると、「点グラフ」で表示します。



「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}}

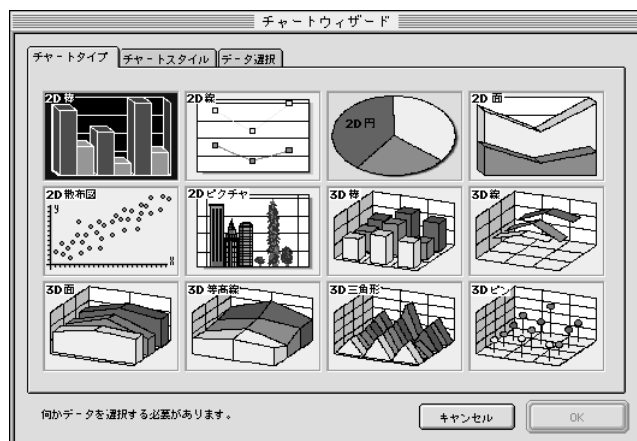
または
GRAPH TABLE({テーブル ;} グラフ番号 ; xフィールド ; yフィールド1{ ;...;
yフィールド8})

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」メニューコマンドを選択して、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

この章では、「ルーチン」エディタの「Hierarchical Lists」テーマ内にある階層リストコマンドについて説明します。

Load list

SAVE LIST

New list

Copy list

CLEAR LIST

Count list items

Is a list

REDRAW LIST

SET LIST PROPERTIES

GET LIST PROPERTIES

SORT LIST

APPEND TO LIST

INSERT LIST ITEM

SET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES

List item position

List item parent

DELETE LIST ITEM

GET LIST ITEM

SET LIST ITEM

Selected list item

SELECT LIST ITEM

SELECT LIST ITEM BY REFERENCE

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、その変更を永久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 LIST ITEM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	コピーされる数値
戻り値	数値	複製したリストの数値

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、新しいリストの数値を返します。

新しいリストを使い終わったら、**CLEAR LIST**コマンドを呼び出して削除します。

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、オプション引数の<*>を付けて、リストの項目またはサブ項目にサブリストが添付されている場合、それも一緒に廃棄します。

フォームオブジェクトに添付されているリストを「オブジェクトプロパティ」ウィンドウによって消去する必要はありません。4Dがリストを自動的にロードして消去します。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。

フォームに現在表示されているもうひとつのリストの項目（あらゆるレベルの）に添付されているサブリストを消去するには、以下のように行います。

1. 親項目で**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リストから切り離す。
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計算する。

例題

1. もはや必要のないオブジェクトとデータ（例えば、ウインドウが終了し、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、フォームにおけるユーザの動作によっては、既に消去した階層リストを最後に消去することがあるでしょう。**Is a list**関数を使用して、必要な場合にだけリストを消去します。

・「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 items (リスト) 倍長整数

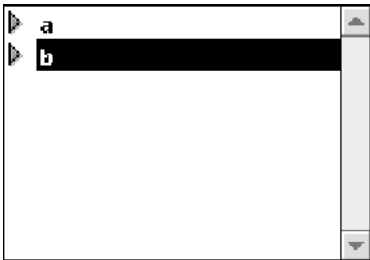
引数	タイプ	説明
リスト	数値	数値
戻り値	倍長整数	拡張されたリストの中にある項目数

説明
Count list items関数は、引数<リスト>によって渡された参照番号を持つリストにおいて、現在“表示されている”項目の数を返します。

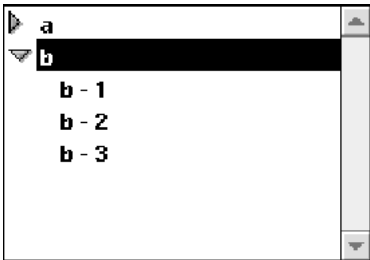
Count list items関数は、リストにある項目の総数を返すわけではありません。これは、リストおよびサブリストの現在の展開 / 縮小の状態に応じて、表示されている項目の数を返します。

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。

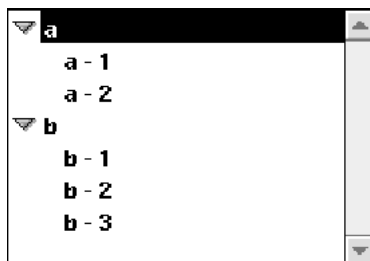
例題
次は、「ユーザ」モードで表示された「hList」という名前のリストです。



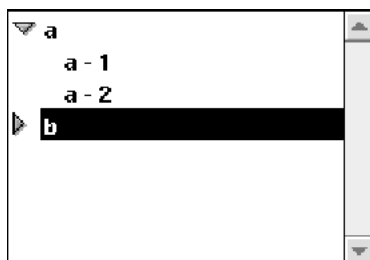
\$vINbItems:=**Count list items** (hList) `この時点で、「\$vINbItems」変数は2を取得する



\$vINbItems:=**Count list items** (hList) `この時点で、「\$vINbItems」変数は5を取得する



\$vINblItems:=**Count list items** (hList) `この時点で、「\$vINblItems」変数は7を取得する



\$vINblItems:=**Count list items** (hList) `この時点で、「\$vINblItems」変数は4を取得する

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. **DRAW 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、リストのひと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、

変更が行われた後でリストを再計算する
REDRAW LIST (hList) ` 良い例
` ...

\$vList:=hList
` ...
変更が行われた後でリストを再計算する
REDRAW LIST (\$vList) ` 悪い例
` ...

参照
なし

SET LIST PROPERTIES

SET LIST PROPERTIES (リスト ; 表示形式 {; アイコン {; 線の高さ}})

引数	タイプ	説明
リスト	数値	数値
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	'icn' MacOSベースのリソースIDまたは デフォルトプラットフォームのノード アイコンの場合は0
線の高さ	数値	線の高さの最小値（ピクセル単位）

説明

SET LIST PROPERTIESコマンドは、引数<リスト>に渡されたリスト参照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>は、4th Dimensionが提供する以下の定義済定数のいずれかです。

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。ひと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、もうひとつのアイコンはサブ項目を持たないことを示します。以下は、Windows表示のデフォルトの階層リストです。

展開されたノード

繋がれている点線

縮小されたノード

サブ項目を持っていないノード

United States

New York

San Francisco

Los Angeles

Seattle

Boston

Chicago

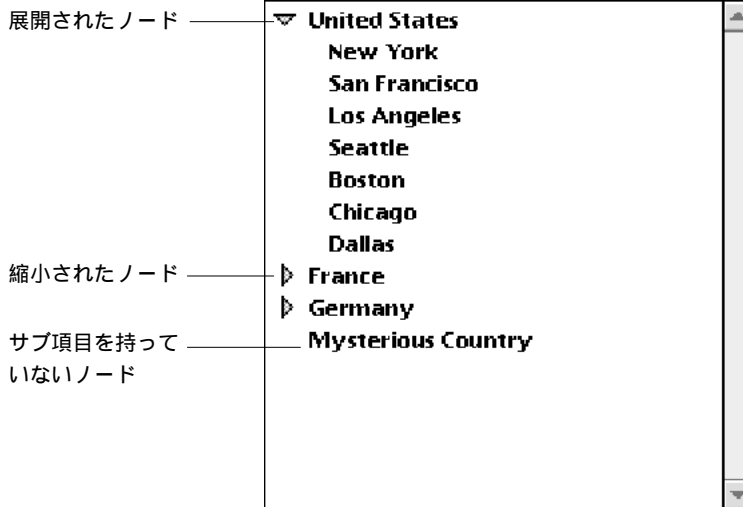
Dallas

France

Germany

Mysterious Country

Macintosh表示では、リストは点線で繋がれていません。ひと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>に渡される値は縮小されたノードのためのアイコンを設定し、<アイコン>+1は展開されたノードのためのアイコン、<アイコン>+2はサブ項目を持たないノード（Windows表示に設定されている場合）を設定します。

例えば、15000を渡した場合、カラーアイコン'cicn' ID=15000が縮小されたノード用に表示され、カラーアイコン'cicn' ID=15001は展開されたノード用に、カラーアイコン'cicn' ID=15002はサブ項目のないノード用に表示されます。

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
140	Windows表示のサブ項目のないノード

引数<アイコン>を渡さない場合、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。

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")

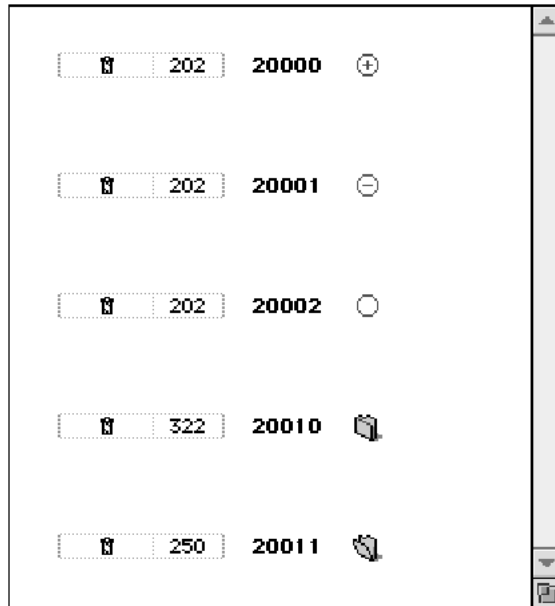
SET LIST PROPERTIES(hlCities ; vlAppearance ; vlIcon)

¥ (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 ; 20000)

階層リストは、以下のように表示されます。



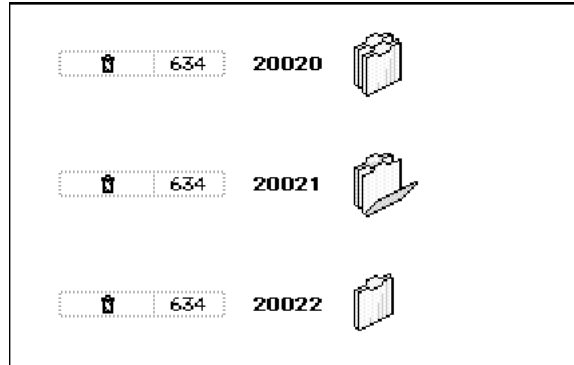
5. 以下の行により、

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。



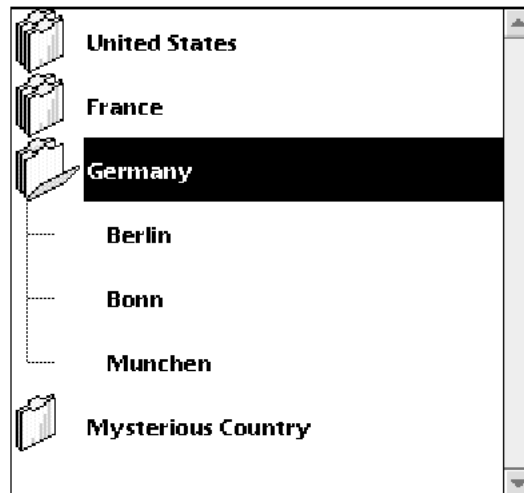
次に、以下に示す'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに追加されます。



6. 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}})

引数	タイプ	説明
リスト	数値	数値
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	'icn' MacOSベースのリソースID
線の高さ	数値	線の高さの最小値（ピクセル単位）

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。

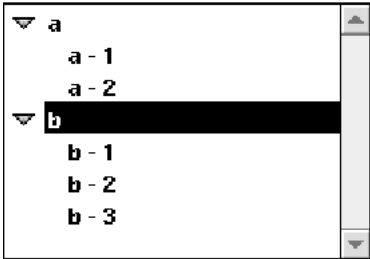
オプション引数<線の高さ>は、線の高さの最小値を返します。

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 (Macintosh表示)。



`「bMacOrWin」 ボタンのオブジェクトメソッド

GET LIST PROPERTIES (hList ; \$vAppearance ; \$vIcon ; \$vLH)

If (\$vAppearance=Ala Macintosh)

 \$vAppearance:=Ala Windows

 \$vIcon:=Windows node

 \$vLH:=20

Else

 \$vAppearance:=Ala Macintosh

 \$vIcon:=Macintosh node

 \$vLH:=0

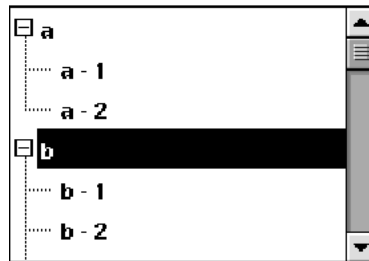
End if

SET LIST PROPERTIES (hList ; \$vAppearance ; \$vIcon ; \$vLH)

REDRAW LIST (hList) `REDRAW LISTコマンドを忘れずに呼び出します。

 `samaなければ、リストが更新されません。

上記のコードを実行すると、以下のようなWindows表示形式で表示されます。



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。

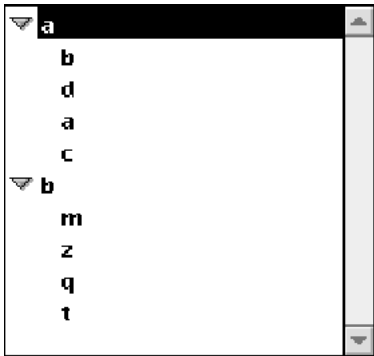
SORT LISTコマンドは、あらゆるレベルのリストをソートします。まずリストの項目をソートし、次にサブリストがあればそれらをソートし、サブリストの中の項目をソートするというようにリストのすべてのレベルに降りていきます。通常、**SORT LIST**コマンドをフォームに表示されているリストに適用するのはこのためです。サブリストのソートは、上位レベルを呼び出したときに順序が変更されるので、ほとんど意味がありません。

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/縮小状態は変更しません。ただし、選択された項目がソートによって移動することがあるため、ソートの前と後では**SORT LIST**コマンドから異なる位置が返される可能性があります。

選択されたリスト項目がソートの前後で異なる位置を返すことがあります。

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、

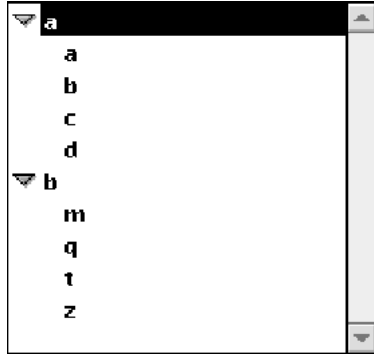
`リストとそのサブリストを昇順にソートする

SORT LIST (hList ; >)

REDRAW LIST (hList) `REDRAW LISTコマンドを忘れずに呼び出します。

`さもなければリストは更新されません。

リストは、以下ようになります。



以下のコードを実行すると、

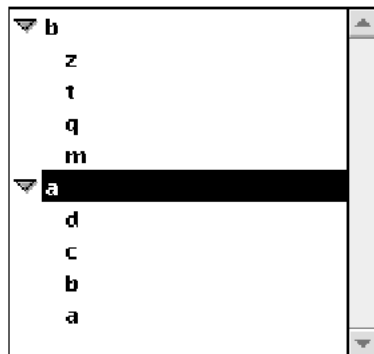
`リストとそのサブリストを降順にソートする

SORT LIST (hList ; <)

REDRAW LIST (hList) `REDRAW LISTコマンドを忘れずに呼び出します。

`さもなければリストは更新されません。

リストは、以下ようになります。



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	数値	数値
項目テキスト	文字列	新規リスト項目のテキスト(最大31文字)
項目参照番号	数値	新規リスト項目のユニークな参照番号
サブリスト	数値	新規リスト項目に添付するオプションのサブリスト
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

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。31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 / 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。この値は専用の値で4th Dimensionだけがそれを持ちます。以下では、この参照番号をどのように使用するかに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<項目参照番号>に0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削除するために（例えば、Qで始まるキーフィールドがひとつもない等）、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。いずれにせよ、重要なのは本質的には選択した項目（クリックまたはドラッグした項目）を扱うということです。関数**Selected list item**（クリック）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ります。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<項目参照番号>によって0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。

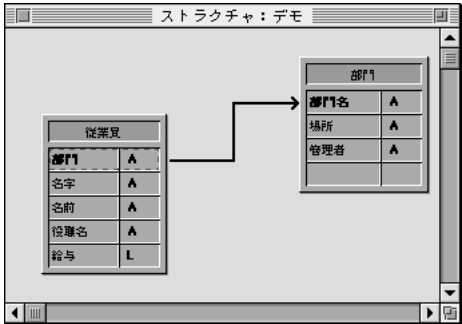
なぜ一意の参照番号が必要なのでしょう？

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 item**関数と**GET LIST ITEM**コマンドを使用すれば、現在選択されている項目を処理することができます。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
活動休眼中	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。

、 「hList」 階層リストのオブジェクトメソッド

Case of

¥ (Form event=On Load)

C_LONGINT (hList ; \$hSubList ; \$vIDepartment ; \$vIEmployee)

` 新たな空の階層リストを作成する

hList:=**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 (\$vIDepartment ; 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？

\$vINbEmployees:=**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 (\$vINbEmployees>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 (\$vIEmployee ; 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 (hList ; [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 (hList ; 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
$vlItemPos:=Selected list item (hlList)
`念のために位置を確認する
If ($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。

```

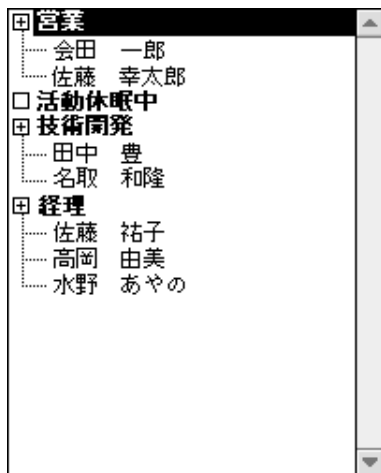
この例では、部門項目と従業員項目を区別する必要がある理由はひとつだけです。

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かります。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。

参照
なし

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	数値	数値
前項目参照番号 *	数値 *	項目の参照番号 *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31文字)
項目参照番号	数値	新規リスト項目のユニークな参照番号
サブリスト	数値	新規リスト項目に添付するオプション 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；項目参照番号；入力可；スタイル；アイコン)

引数	タイプ	説明
リスト	数値	数値
項目参照番号	数値	項目参照番号、または 0の場合、リストに最後に追加された項目
入力可	ブール	True=入力可、False=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	Macintoshベースの'cicn'リソースID、または Macintoshベースの65536+'PICT'リソースID、 または、131072+ピクチャ参照番号

説明

SET LIST ITEM 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を変更します。

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。

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ひとつ、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。

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ひとつを渡します。

N、NはMacOSベースの'cicn'リソースのリソースIDです。

Use PICT resource+N、NはMacOS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、4Dであらかじめ定義されている定数です。

APPEND TO LISTコマンドの例を参照してください。

参照

GET LIST ITEM PROPERTIES、SET LIST ITEM

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES (リスト ; 項目参照番号 ; 入力可 {; スタイル {; アイコン}))

引数	タイプ	説明
リスト	数値	数値
項目参照番号	数値	項目参照番号
入力可	ブール	True=入力可、False=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	Macintoshベースの'cicn'リソースID、または Macintoshベースの65536+'PICT'リソースID、 または、131072+ピクチャ参照番号

説明

GET LIST ITEM PROPERTIES 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プロパティを返します。

このコマンドを呼び出すと、以下のような結果が生じます。

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。

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。

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。

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	数値	数値
項目参照番号	数値	項目参照番号
戻り値	数値	展開されたリスト内の項目の位置

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 / 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 ; 項目参照番号) 数値

引数	タイプ	説明
リスト	数値	数値
項目参照番号	数値	項目参照番号
戻り値	数値	親項目の項目参照番号、親項目がない場合=0

説明

List item parent関数は、親項目の項目参照番号を返します。

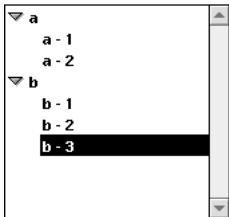
数値を引数<リスト>で、リスト項目の項目参照番号を引数<項目参照番号>で渡します。項目参照番号がリストの既存の項目を参照し、この項目がサブリストにある場合（つまり親項目を持つ）この結果として親項目の項目参照番号が取得されます。

渡した項目参照番号を持つ項目が存在しない場合、または項目が親を持たない場合、List item parent関数は0を返します。

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、「\$vParentItemRef」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。

```
$vItemPos:=Selected list item(hList)
```

```
GET LIST ITEM(hList ; $vItemPos ; $vItemRef ; $vItemText)
```

```
$vParentItemRef:=List item parent(hList ; $vItemRef) ` $vParentItemRefは200を取得する
```

項目 “a-1” が選択された場合、「\$vParentItemRef」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、「\$vParentItemRef」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	数値	数値
項目参照番号 *	数値 *	項目の参照番号 *の場合、現在選択されているリスト項目 指定した場合、メモリにサブリストがある 場合はそれを消去 省略した場合、サブリストがあっても消去 されません
*	*	

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。

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。

`さもなければ、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 ; 項目参照番号 ; 項目テキスト { ; サブリスト ; { ; 展開 } })

引数	タイプ	説明
リスト	数値	数値
項目位置	数値	展開されたリスト内の項目の位置
項目参照番号	数値	項目の参照番号
項目テキスト	文字列	リスト項目のテキスト
サブリスト	数値	サブ数値 (ある場合)
展開	ブール	サブリストがある場合、 サブリストが現在展開されている場合はTrue サブリストが現在縮小されている場合はFalse

説明

GET LIST ITEM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内の、<項目参照番号>によって渡された位置を持つ項目についての情報を返します。

位置は、リストとそのサブリストの展開 / 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コマンドは引数を変更せずに返します。

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

「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。

```
$vItemPos:=Selected list item(hList)
If ($vItemPos>0)
    GET LIST ITEM (hList;$vItemPos;$vItemRef;$vItemText;$hSublist;$vbExpanded)
    If (Is a list ($hSublist))
        SET LIST ITEM (hList ; $vItemRef ; $vItemText ; $vItemRef ; $hSublist ; Not ($vbExpanded))
        REDRAW LIST (hList)
    End if
End if
```

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 ; 項目参照番号 ; 新規項目テキスト ; 新規項目参照番号{ ; サブリスト ; { 展開 } })

引数	タイプ	説明
リスト	数値	数値
項目参照番号	数値	項目の参照番号 0の場合、リストに最後に追加された項目
新規項目テキスト	文字列	新規リスト項目のテキスト
新規項目参照番号	数値	新規項目の参照番号
サブリスト	数値	項目に付属する新規サブリスト 項目に添付されるサブリストまたは サブリストがない場合は0 (サブリストが あれば、現在のものを切り離す) または変更がない場合は-1
展開	ブール	サブリストが展開されるか縮小されるか を示す

説明

SET 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内の、< 項目参照番号 > によって渡される項目参照番号を持つ項目を変更します。

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。オプションで < 項目参照番号 > で0を渡して、**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。

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いよいでしょう。そうすれば、リストがなくなったときに**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で指定してください。

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。

```
$vListItemPos:=Selected list item (hList)
If ($vListItemPos>0)
  GET LIST ITEM (hList ; $vListItemPos ; $vListItemRef ; $vslItemText ; $hSublist ; $vbExpanded)
  $vbNewSubList:=Not (Is a list ($hSublist))
  If ($vbNewSubList)
    $hSublist:=New list
  End if
  vUniqueRef:=vUniqueRef+1
  APPEND TO LIST ($hSubList ; "新規項目" ; vUniqueRef)
  If ($vbNewSubList)
    SET LIST ITEM (hList ; $vListItemRef ; $vslItemText ; $vListItemRef ; $hSublist ; True)
  End if
  SELECT LIST ITEM BY REFERENCE (hList ; vUniqueRef)
  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 item

Selected list item (リスト) 倍長整数

引数	タイプ	説明
リスト	数値	数値
戻り値	倍長整数	展開されたリストの現在選択されている リスト項目の位置

説明

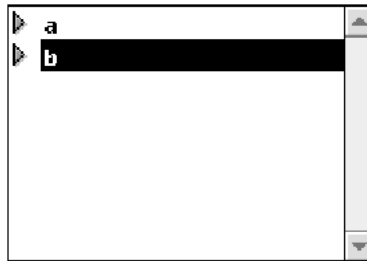
Selected list item関数は、引数<リスト>によって渡された参照番号を持つリストの中の選択された項目の位置を返します。

この関数は、フォームに表示されるリストに適用し、ユーザがどの項目を選択したかを検出します。

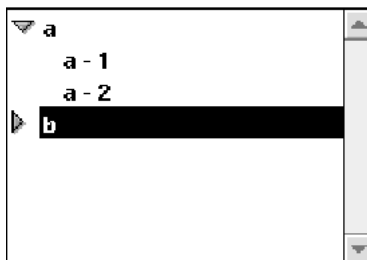
リストにサブリストがある場合、この関数はメインリスト（フォームで実際に定義されたリスト）のサブリストではなく、メインリストに適用します。位置は、リストとそのサブリストの現在の展開／縮小状態を使用して、メインリストの最初の項目と相対的に表現されます。

例題

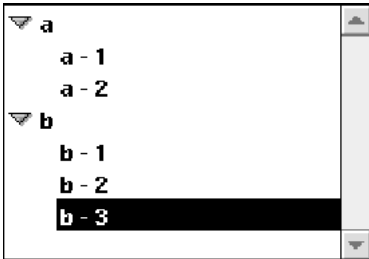
次は、「ユーザ」モードで表示された「hList」という名前のリストです。



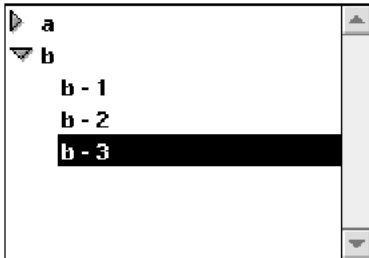
`$vItemPos:=Selected list item (hList)` ` この時点では`$vItemPos`は2を取得する



`$vItemPos:=Selected list item (hList)` ` この時点では`$vItemPos`は4を取得する



\$vllItemPos:=**Selected list item** (hList) ` この時点では\$vllItemPosは7を取得する



\$vllItemPos:=**Selected list item** (hList) ` この時点では\$vllItemPosは5を取得する

参照
SELECT LIST BY REFERENCE、SELECT LIST ITEM

SELECT LIST ITEM

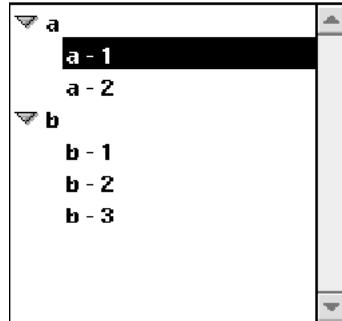
SELECT LIST ITEM (リスト ; 項目位置)

引数	タイプ	説明
リスト	数値	数値
項目位置	数値	展開されたリスト内の項目の位置

説明
SELECT LIST ITEM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内の、<項目位置>によって渡された位置にある項目を選択します。引数<項目位置>は、リストとそのサブリストの展開 / 縮小の現在の状態を使用して相対的に表される位置です。位置は、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、デフォルトとして最初の項目が選択されます。

例題

次は、「ユーザ」モードに表示された「hList」という名前のリストです。



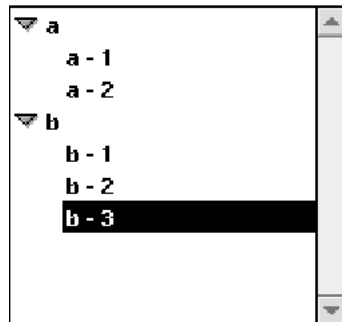
以下のコードを実行すると、

```
SELECT LIST ITEM (hList ; Count list items (hList))
```

```
REDRAW LIST 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。



参照

SELECT LIST ITEM BY REFERENCE、Selected list item

SELECT LIST ITEM BY REFERENCE

SELECT LIST ITEM BY REFERENCE (リスト ; 項目参照番号)

引数	タイプ	説明
リスト	数値	数値
項目参照番号	数値	項目参照番号

説明

SELECT LIST ITEM BY REFERENCE 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中の<項目参照番号>によって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。

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、このコマンドが必要なサブリストを展開して選択した項目を表示します。

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。

```
$vItemPos:=Selected list item (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 ; $vItemPos ; $vItemRef ; $vItemText)
`親項目が存在する場合は、親項目の項目参照番号を取得
$vParentItemRef:=List item parent (hList ; $vItemRef)
If ($vParentItemRef>0)
    `親項目を選択する
    SELECT LIST ITEM BY REFERENCE (hList ; List item parent (hList ; $vItemRef))
    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

この章では、「ルーチン」エディタの「Import and Export」テーマ内にあるデータ読み込み / データ書き出しコマンドについて説明します。この章のコマンドは、データを読み込んだり、書き出すために使用します。どちらも、フォームを介して処理します。

EXPORT DATA
EXPORT TEXT
IMPORT SYLK

EXPORT DIF
IMPORT DATA
IMPORT TEXT

EXPORT SYLK
IMPORT DIF

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ひとつの方法は、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

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ひと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。

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。

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ひとつの方法は、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ひと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。

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。

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、読み込み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ひとつの方法は、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、読み込みもうとする各フィールドに対してひと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。

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。

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は以下のフォーマットのデータを読み込むことができます：テキスト、固定長のテキスト、SYLK、DIF、DBF (dBase) および4th Dimension。

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。

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、パラメータのひと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。4Dは以下のフォーマットのデータを書き出すことができます：テキスト、固定長のテキスト、SYLK、DIF、DBF (dBase) および4th Dimension。

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ますようにします。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、パラメータのひと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0になります。

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。

この章では、「ルーチン」エディタの「Interruptions」テーマ内にある割り込みコマンドについて説明します。この章のコマンドは、メソッド実行の監視、エラーとイベントの管理を行います。

ON EVENT CALL

FILTER EVENT

ON ERR CALL

ABORT

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。いちどにひと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のメソッドがひと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。「エクスポーラ」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、モディ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、caps 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")
    $vINbRecords:=Records in selection ([従業員])
    For ($vIRecord;1;$vINbRecords)
        If (<>vbWeStop)
            ALERT ("レコード数 : "+ String ($vIRecord)+" / "
                +String (Records in selection ([従業員]))+"で印刷が中止されました")
            $vIRecord:=$vINbRecords+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差迂生しているためです。

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

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。

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ひと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 CALL 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 ($vErrCode)
Else
  ALERT ("ドキュメントが作成できません。 I/O エラー : "+String ($vErrCode))
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ひと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 ($vElem)
If (Count parameters>0)
  $ErrorMethod:=$1
Else
  $ErrorMethod:=""
End if
If ($ErrorMethod# "")
  C_LONGINT (gError)
  gError:=0
  $vElem:=1+Size of array (asErrorMethod)
  INSERT ELEMENT (asErrorMethod;$vElem)
  asErrorMethod{$vElem}:=$1
  ON ERR CALL ($1)
Else
  ON ERR CALL ("")
  $vElem:=Size of array (asErrorMethod)

```



```

If ($vElem>0)
  DELETE ELEMENT (asErrorMethod;$vElem)
If ($vElem>1)
  ON ERR CALL (asErrorMethod{$vElem-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

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。等、等。

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。

参照

なし

この章では、「ルーチン」エディタの「Language」テーマ内にあるランゲージコマンドについて説明します。この章のコマンドは、さまざまなデ - タベ - ス関数に使用されます。

Command name	Is a variable	Self
Count parameters	Nil	TRACE
EXECUTE	NO TRACE	Type
Get pointer	RESOLVE POINTER	

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のデバッグを起動します。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、デバッグ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

メントが無効になり、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Tip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グを終了します。「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

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したら、以下のように使用することができます。

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 (\$vParam;2;**Count parameters**)
 If (\${ \$vParam}>\$0)
 \$0:=\${ \$vParam}
 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したら、以下のように使用できます。

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

コンパイルコマンド、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 または変数	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うことができます。

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) に対して使う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. **DRAW 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 ($2;$vITypeFlags)
If (Count parameters>=2)
    $vITypeFlags:=$2
Else
    $vITypeFlags:=0xFFFFFFFF
End if
For ($vIField;1;Count fields ($1))
    $vpField:=Field (Table ($1);$vIField)
    $vIFieldType:=Type ($vpField->)
    If ( $vITypeFlags ?? $vIFieldType )
        Case of
            ¥ (($vIFieldType=Is Alpha Field)|($vIFieldType=Is Text))
                $vpField->:=""
            ¥ (($vIFieldType=Is Real)|($vIFieldType=Is Integer)|($vIFieldType=Is LongInt))
                $vpField->:=0
            ¥ ($vIFieldType=Is Date)
                $vpField->:=!00/00/00!
            ¥ ($vIFieldType=Is Time)
                $vpField->:=!!00:00:00!!
            ¥ ($vIFieldType=Is Boolean)
```

```

    $vpField->:=False
  ¥ ($vFieldType=Is Picture)
    C_PICTURE ($vgEmptyPicture)
    $vpField->:=$vgEmptyPicture
  ¥ ($vFieldType=Is Subtable)
    Repeat
      ALL SUBRECORDS ($vpField->)
      DELETE SUBRECORD ($vpField->)
    Until (Records in subselection ($vpField->)=0)
  ¥ ($vFieldType=Is BLOB)
    SET BLOB SIZE ($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。

- 入力可能な変数をひとつ作成し、“v”と名付ける。
- オブジェクトのプロパティを設定する。
- 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

- 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。
- 「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 ; $vITableNum ; $vIFieldNum)  
$vIVarNum:=Num (Substring ($vsVarName ; 2))
```

フォームをこのような方法で構築すると、100個の変数のメソッドを一度で記述することになる点に注意してください。DoSomething (1)、DoSomething (2) からDoSomething (100) までを書き込む必要はありません。

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 ; $vITableNum ; $vIFieldNum)  
  If (Not(($vITableNum>0)&($vIFieldNum=0)&($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**関数を使用する代わりに、**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**関数を使用する代わりに、**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：**Get pointer (\$tTabNom+ "{3}")** のような実行はできません。変数名のみ可能です。

例題

フォーム上で、5 X 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 ($vIVar ; 1 ; 50)
  $vpVar:=Get pointer ("v" + String ($vIVar)) ` 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4D Compiler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。例えば、4Dの日本語版（コマンド名は日本語版も英語版と同じです）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 DEFAULT ([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、エクスプローラウィンドウのその名前の所を選択します。コマンド番号は、ウィ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-> ; "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、以下のよう記述します。

```
STANDARD INPUT FORM (->[従業員])
```

```
STANDARD 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 / 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

コマンド名によるコマンド

次の表は、4Dコマンドを名前ごとに一覧表示し、一緒にコマンド番号も示しています。
コマンド番号は、**Command name**関数で使用する必要があります。

コマンド名	コマンド番号
A	
ABORT	156
Abs.....	99
ACCEPT	269
ACCUMULATE	303
ACI folder.....	485
Activated	346
ADD DATA SEGMENT	361
ADD RECORD.....	56
ADD SUBRECORD	202
Add to date	393
ADD TO SET	119
After	31
ALERT	41
ALL RECORDS	47
ALL SUBRECORDS	109
Append document.....	265
APPEND MENU ITEM	411
APPEND TO CLIPBOARD	403
APPEND TO LIST	376
Application file	491
Application type	494
Application version	493
APPLY TO SELECTION	70
APPLY TO SUBSELECTION	73
Arctan	20
ARRAY BOOLEAN	223
ARRAY DATE	224
ARRAY INTEGER	220
ARRAY LONGINT	221
ARRAY PICTURE	279
ARRAY POINTER	280
ARRAY REAL	219
ARRAY STRING	218
ARRAY TEXT	222
ARRAY TO LIST	287
ARRAY TO SELECTION	261

ARRAY TO STRING LIST	512
Ascii	91
AUTOMATIC RELATIONS	310
Average	2
B	
BEEP	151
Before	29
Before selection	198
Before subselection	199
BLOB PROPERTIES	536
BLOB size	605
BLOB TO DOCUMENT	526
BLOB to integer	549
BLOB to list	557
BLOB to longint	551
BLOB to real	553
BLOB to text	555
BLOB TO VARIABLE	533
BREAK LEVEL	302
BRING TO FRONT	326
BUTTON TEXT	194
C	
CALL PROCESS	329
CANCEL	270
CANCEL TRANSACTION	241
Caps lock down	547
CHANGE ACCESS	289
CHANGE PASSWORD	186
Change string	234
Char	90
CLEAR CLIPBOARD	402
CLEAR LIST	377
CLEAR NAMED SELECTION	333
CLEAR SEMAPHORE	144
CLEAR SET	117
CLEAR VARIABLE	89
CLOSE DOCUMENT	267
CLOSE RESOURCE FILE	498
CLOSE WINDOW	154
Command name	538
Compiled application	492
COMPRESS BLOB	534
COMPRESS PICTURE	355

COMPRESS PICTURE FILE	359
CONFIRM	162
COPY ARRAY	226
COPY BLOB	558
COPY DOCUMENT	541
Copy list	626
COPY NAMED SELECTION	331
COPY SET	600
Cos	18
Count fields	255
Count list items	380
Count menu items	405
Count menus	404
Count parameters	259
Count screens	437
Count tables	254
Count tasks	335
Count user processes	343
Count users	342
CREATE DIRECTORY	475
Create document	266
CREATE EMPTY SET	140
CREATE RECORD	68
CREATE RELATED ONE	65
Create resource file	496
CREATE SET	116
CREATE SUBRECORD	72
Current date	33
Current default table	363
Current form page	276
Current form table	627
Current machine	483
Current machine owner	484
Current process	322
Current time	178
Current user 182	
CUT NAMED SELECTION	334
C_BLOB	604
C_BOOLEAN	305
C_DATE	307
C_GRAPH	352
C_INTEGER	282
C_LONGINT	283

C_PICTURE	286
C_POINTER	301
C_REAL	285
C_STRING	293
C_TEXT	284
C_TIME	306
D	
Data file	490
DATA SEGMENT LIST	527
Database event	369
Date	102
Day number	114
Day of	23
Deactivated	347
Dec	9
DEFAULT TABLE	46
DELAY PROCESS	323
DELETE DOCUMENT	159
DELETE ELEMENT	228
DELETE FROM BLOB	560
DELETE LIST ITEM	624
DELETE MENU ITEM	413
DELETE RECORD	58
DELETE RESOURCE	501
DELETE SELECTION	66
Delete string	232
DELETE SUBRECORD	96
DELETE USER	615
DIALOG	40
DIFFERENCE	122
DISABLE BUTTON	193
DISABLE MENU ITEM	150
DISPLAY RECORD	105
DISPLAY SELECTION	59
DISTINCT VALUES	339
Document creator	529
DOCUMENT LIST	474
DOCUMENT TO BLOB	525
Document type	528
Drag and drop position	608
DRAW AND DROP PROPERTIES	607
DRAW WINDOW	452
DUPLICATE RECORD	225

During	30
E	
EDIT ACCESS	281
ENABLE BUTTON	192
ENABLE MENU ITEM	149
End selection	36
End subselection	37
ERASE WINDOW	160
EXECUTE	63
Execute on server	373
Exp	21
EXPAND BLOB	535
EXPORT DIF	84
EXPORT SYLK	85
EXPORT TEXT	167
F	
False	215
Field	253
Field name	257
FILTER EVENT	321
FILTER KEYSTROKE	389
Find in array	230
Find window	449
FIRST PAGE	250
FIRST RECORD	50
FIRST SUBRECORD	61
FLUSH BUFFERS	297
FOLDER LIST	473
FONT	164
FONT LIST	460
Font name	462
Font number	461
FONT SIZE	165
FONT STYLE	166
Form event	388
Frontmost process	327
Frontmost window	447
G	
Gestalt	488
GET CLIPBOARD	401
Get document position	481
GET DOCUMENT PROPERTIES	477
Get document size	479

GET FIELD PROPERTIES	258
GET GROUP LIST.....	610
GET GROUP PROPERTIES	613
GET HIGHLIGHT	209
GET ICON RESOURCE	517
Get indexed string	510
GET LIST ITEM	378
GET LIST ITEM PROPERTIES	631
GET LIST PROPERTIES	632
Get menu item	422
Get menu item key	424
Get menu item mark	428
Get menu item style	426
Get menu title	430
GET MOUSE	468
GET PICTURE FROM CLIPBOARD	522
GET PICTURE RESOURCE	502
Get platform interface	470
Get pointer	304
GET PROCESS VARIABLE	371
GET RESOURCE	508
Get resource name	513
Get resource properties	515
Get string resource	506
Get text from clipboard	524
Get text resource	504
GET USER LIST	609
GET USER PROPERTIES	611
GET WINDOW RECT	443
Get window title	450
GOTO AREA	206
GOTO PAGE	247
GOTO RECORD	242
GOTO SELECTED RECORD	245
GOTO XY	161
GRAPH	169
GRAPH SETTINGS	298
GRAPH TABLE	148
H	
HIDE MENU BAR	432
HIDE PROCESS	324
HIDE TOOLBAR	434
HIGHLIGHT TEXT	210

I	
IDLE	311
IMPORT DIF	86
IMPORT SYLK	87
IMPORT TEXT	168
In break	113
In footer	191
In header	112
In transaction	397
INPUT FORM	55
INSERT ELEMENT	227
INSERT IN BLOB	559
INSERT LIST ITEM	625
INSERT MENU ITEM	412
Insert string	231
Int	8
INTEGER TO BLOB	548
INTERSECTION	121
INVERT BACKGROUND	93
Is a list	621
Is a variable	294
Is in set	273
Is user deleted	616
ISO to Mac	520
K	
Keystroke	390
L	
Last object	278
LAST PAGE	251
LAST RECORD	200
LAST SUBRECORD	201
Length	16
Level	101
List item parent	633
List item position	629
LIST TO ARRAY	288
LIST TO BLOB	556
LOAD COMPRESS PICTURE FROM FILE	357
Load list	383
LOAD RECORD	52
LOAD SET	185
LOAD VARIABLES	74
Locked	147

LOCKED ATTRIBUTES	353
Log	22
LONGINT TO BLOB	550
Lowercase	14
M	
Mac to ISO	519
Mac to Win	463
Macintosh command down	546
Macintosh control down	544
Macintosh option down	545
MAP FILE TYPES	366
Max	3
MENU BAR	67
Menu bar height	440
Menu bar screen	441
Menu selected	152
MESSAGE	88
MESSAGES OFF	175
MESSAGES ON	181
Milliseconds	459
Min	4
Mod	98
Modified	32
Modified record	314
MODIFY RECORD	57
MODIFY SELECTION	204
MODIFY SUBRECORD	203
Month of	24
MOVE DOCUMENT	540
N	
New list	375
New process	317
NEXT PAGE	248
NEXT RECORD	51
NEXT SUBRECORD	62
Next window	448
Nil	315
NO TRACE	158
Not	34
Num	11
O	
Old	35
OLD RELATED MANY	263

OLD RELATED ONE	44
ON ERR CALL	155
ON EVENT CALL	190
ONE RECORD SELECT	189
Open document	264
Open external window	309
Open resource file	497
Open window	153
ORDER BY	49
ORDER BY FORMULA	300
ORDER SUBRECORDS BY	107
OUTPUT FORM	54
Outside call	328
P	
PAGE BREAK	6
PAGE SETUP	299
PAUSE PROCESS	319
PICTURE PROPERTIES	457
Picture size	356
PLATFORM PROPERTIES	365
PLAY	290
POP RECORD	177
Pop up menu	542
Position	15
POST CLICK	466
POST EVENT	467
POST KEY	465
PREVIOUS PAGE	249
PREVIOUS RECORD	110
PREVIOUS SUBRECORD	111
PRINT FORM	5
PRINT LABEL	39
PRINT RECORD	71
PRINT SELECTION	60
PRINT SETTINGS	106
Printing page	275
Process number	372
PROCESS PROPERTIES	336
Process state	330
PUSH RECORD	176
Q	
QUERY	277
QUERY BY EXAMPLE	292

QUERY BY FORMULA	48
QUERY SELECTION	341
QUERY SELECTION BY FORMULA	207
QUERY SUBRECORDS	108
QUIT 4D	291
R	
Random	100
READ ONLY	145
Read only state	362
READ WRITE	146
REAL TO BLOB	552
RECEIVE BUFFER	172
RECEIVE PACKET	104
RECEIVE RECORD	79
RECEIVE VARIABLE	81
Record number	243
Records in selection	76
Records in set	195
Records in subselection	7
Records in table	83
REDRAW.....	174
REDRAW LIST	382
REDRAW WINDOW	456
REDUCE SELECTION	351
REJECT	38
RELATE MANY	262
RELATE MANY SELECTION	340
RELATE ONE	42
RELATE ONE SELECTION	349
REMOVE FROM SET	561
Replace string	233
REPORT	197
Request	163
RESOLVE POINTER	394
RESOURCE LIST	500
RESOURCE TYPE LIST	499
RESUME PROCESS	320
Round	94
S	
SAVE LIST	384
SAVE OLD RELATED ONE	45
SAVE PICTURE TO FILE	358

SAVE RECORD	53
SAVE RELATED ONE	43
SAVE SET	184
SAVE VARIABLES	75
SCAN INDEX	350
SCREEN COORDINATES	438
SCREEN DEPTH	439
Screen height	188
Screen width	187
SEARCH BY INDEX	64
SELECT LIST ITEM	381
SELECT LIST ITEM BY REFERENCE	630
SELECT LOG FILE	345
Selected list item	379
Selected record number	246
SELECTION TO ARRAY	260
SELECTION RANGE TO ARRAY	368
Self	308
Semaphore	143
SEND HTML FILE	619
SEND PACKET	103
SEND RECORD	78
SEND VARIABLE	80
Sequence number	244
SET ABOUT	316
SET BLOB SIZE	606
SET CHANNEL	77
SET CHOICE LIST	237
SET COLOR	271
SET CURSOR	469
SET DEFAULT CENTURY	392
SET DOCUMENT CREATOR	531
SET DOCUMENT POSITION	482
SET DOCUMENT PROPERTIES	478
SET DOCUMENT SIZE	480
SET DOCUMENT TYPE	530
SET ENTERABLE	238
SET FIELD TITLES	602
SET FILTER	235
SET FORMAT	236
SET GROUP PROPERTIES	614
SET HTML ROOT	634
SET INDEX	344

SET LIST ITEM	385
SET LIST ITEM PROPERTIES	386
SET LIST PROPERTIES	387
SET MENU ITEM	348
SET MENU ITEM KEY	423
SET MENU ITEM MARK	208
SET MENU ITEM STYLE	425
SET PICTURE RESOURCE	503
SET PICTURE TO CLIPBOARD	521
SET PLATFORM INTERFACE	367
SET PRINT PREVIEW	364
SET PROCESS VARIABLE	370
SET QUERY DESTINATION	396
SET QUERY LIMIT	395
SET REAL COMPARISON LEVEL	623
SET RESOURCE	509
SET RESOURCE NAME	514
SET RESOURCE PROPERTIES	516
SET RGB COLOR	628
SET SCREEN DEPTH	537
SET STRING RESOURCE	507
SET TABLE TITLES	601
SET TEXT RESOURCE	505
SET TEXT TO CLIPBOARD	523
SET TIMEOUT	268
SET USER PROPERTIES	612
SET VISIBLE	603
SET WEB DISPLAY LIMIT	620
SET WEB TIMEOUT	622
SET WINDOW RECT	444
SET WINDOW TITLE	213
Shift down	543
SHOW MENU BAR	431
SHOW PROCESS	325
SHOW TOOLBAR	433
Sin	17
Size of array	274
SORT ARRAY	229
SORT BY INDEX	170
SORT LIST	391
Square root	539
START TRANSACTION	239
START WEB SERVER	617

Std deviation	26
STOP WEB SERVER	618
String	10
STRING LIST TO ARRAY	511
Structure file	489
Substring	12
Subtotal	97
Sum	1
Sum squares	28
System folder	487
T	
Table	252
Table name	256
Tan	19
Temporary folder	486
Test clipboard	400
Test path name	476
TEXT TO BLOB	554
Tickcount	458
Time	179
Time string	180
TRACE	157
Trigger level	398
TRIGGER PROPERTIES 399	99
True	214
Trunc	95
Type	295
U	
Undefined	82
UNION	120
UNLOAD RECORD	212
Uppercase	13
USE ASCII MAP	205
USE NAMED SELECTION	332
USE SET	118
User in group	338
V	
VALIDATE TRANSACTION	240
VARIABLE TO BLOB	532
VARRIABLE TO VARIABLE	635
Variance	27
Version type	495
VOLUME ATTRIBUTES	472

VOLUME LIST	471
W	
Win to Mac	464
Window kind	445
WINDOW LIST	442
Window process	446
Windows Alt down	563
Windows Ctrl down	562
Y	
Year of	5

参照

Command name、EXECUTE

コマンド番号によるコマンド

次の表は、4Dコマンドを番号ごとに一覧表示し、一緒にコマンドの名前も示しています。
コマンド番号は、**Command name**関数で使用する必要があります。

注：リストされていない番号は、現在使用されていません。

コマンド番号	コマンド名
1	Sum
2	Average
3	Max
4	Min
5	PRINT FORM
6	PAGE BREAK
7	Records in subselection
8	Int
9	Dec
10	String
11	Num
12	Substring
13	Uppercase
14	Lowercase
15	Position
16	Length
17	Sin
18	Cos
19	Tan
20	Arctan
21	Exp
22	Log
23	Day of
24	Month of
25	Year of
26	Std deviation
27	Variance
28	Sum squares
29	Before
30	During
31	After
32	Modified
33	Current date
34	Not
35	Old

36	End selection
37	End subselection
38	REJECT
39	PRINT LABEL
40	DIALOG
41	ALERT
42	RELATE ONE
43	SAVE RELATED ONE
44	OLD RELATED ONE
45	SAVE OLD RELATED ONE
46	DEFAULT TABLE
47	ALL RECORDS
48	QUERY BY FORMULA
49	ORDER BY
50	FIRST RECORD
51	NEXT RECORD
52	LOAD RECORD
53	SAVE RECORD
54	OUTPUT FORM
55	INPUT FORM
56	ADD RECORD
57	MODIFY RECORD
58	DELETE RECORD
59	DISPLAY SELECTION
60	PRINT SELECTION
61	FIRST SUBRECORD
62	NEXT SUBRECORD
63	EXECUTE
64	SEARCH BY INDEX
65	CREATE RELATED ONE
66	DELETE SELECTION
67	MENU BAR
68	CREATE RECORD
70	APPLY TO SELECTION
71	PRINT RECORD
72	CREATE SUBRECORD
73	APPLY TO SUBSELECTION
74	LOAD VARIABLES
75	SAVE VARIABLES
76	Records in selection
77	SET CHANNEL
78	SEND RECORD
79	RECEIVE RECORD

80	SEND VARIABLE
81	RECEIVE VARIABLE
82	Undefined
83	Records in table
84	EXPORT DIF
85	EXPORT SYLK
86	IMPORT DIF
87	IMPORT SYLK
88	MESSAGE
89	CLEAR VARIABLE
90	Char
91	Ascii
93	INVERT BACKGROUND
94	Round
95	Trunc
96	DELETE SUBRECORD
97	Subtotal
98	Mod
99	Abs
100	Random
101	Level
102	Date
103	SEND PACKET
104	RECEIVE PACKET
105	DISPLAY RECORD
106	PRINT SETTINGS
107	ORDER SUBRECORDS BY
108	QUERY SUBRECORDS
109	ALL SUBRECORDS
110	PREVIOUS RECORD
111	PREVIOUS SUBRECORD
112	In header
113	In break
114	Day number
116	CREATE SET
117	CLEAR SET
118	USE SET
119	ADD TO SET
120	UNION
121	INTERSECTION
122	DIFFERENCE
140	CREATE EMPTY SET
143	Semaphore

144	CLEAR SEMAPHORE
145	READ ONLY
146	READ WRITE
147	Locked
148	GRAPH TABLE
149	ENABLE MENU ITEM
150	DISABLE MENU ITEM
151	BEEP
152	Menu selected
153	Open window
154	CLOSE WINDOW
155	ON ERR CALL
156	ABORT
157	TRACE
158	NO TRACE
159	DELETE DOCUMENT
160	ERASE WINDOW
161	GOTO XY
162	CONFIRM
163	Request
164	FONT
165	FONT SIZE
166	FONT STYLE
167	EXPORT TEXT
168	IMPORT TEXT
169	GRAPH
170	SORT BY INDEX
172	RECEIVE BUFFER
174	REDRAW
175	MESSAGES OFF
176	PUSH RECORD
177	POP RECORD
178	Current time
179	Time
180	Time string
181	MESSAGES ON
182	Current user
184	SAVE SET
185	LOAD SET
186	CHANGE PASSWORD
187	Screen width
188	Screen height
189	ONE RECORD SELECT

190	ON EVENT CALL
191	In footer
192	ENABLE BUTTON
193	DISABLE BUTTON
194	BUTTON TEXT
195	Records in set
197	REPORT
198	Before selection
199	Before subselection
200	LAST RECORD
201	LAST SUBRECORD
202	ADD SUBRECORD
203	MODIFY SUBRECORD
204	MODIFY SELECTION
205	USE ASCII MAP
206	GOTO AREA
207	QUERY SELECTION BY FORMULA
208	SET MENU ITEM MARK
209	GET HIGHLIGHT
210	HIGHLIGHT TEXT
212	UNLOAD RECORD
213	SET WINDOW TITLE
214	True
215	False
218	ARRAY STRING
219	ARRAY REAL
220	ARRAY INTEGER
221	ARRAY LONGINT
222	ARRAY TEXT
223	ARRAY BOOLEAN
224	ARRAY DATE
225	DUPLICATE RECORD
226	COPY ARRAY
227	INSERT ELEMENT
228	DELETE ELEMENT
229	SORT ARRAY
230	Find in array
231	Insert string
232	Delete string
233	Replace string
234	Change string
235	SET FILTER
236	SET FORMAT

237	SET CHOICE LIST
238	SET ENTERABLE
239	START TRANSACTION
240	VALIDATE TRANSACTION
241	CANCEL TRANSACTION
242	GOTO RECORD
243	Record number
244	Sequence number
245	GOTO SELECTED RECORD
246	Selected record number
247	GOTO PAGE
248	NEXT PAGE
249	PREVIOUS PAGE
250	FIRST PAGE
251	LAST PAGE
252	Table
253	Field
254	Count tables
255	Count fields
256	Table name
257	Field name
258	GET FIELD PROPERTIES
259	Count parameters
260	SELECTION TO ARRAY
261	ARRAY TO SELECTION
262	RELATE MANY
263	OLD RELATED MANY
264	Open document
265	Append document
266	Create document
267	CLOSE DOCUMENT
268	SET TIMEOUT
269	ACCEPT
270	CANCEL
271	SET COLOR
273	Is in set
274	Size of array
275	Printing page
276	Current form page
277	QUERY
278	Last object
279	ARRAY PICTURE
280	ARRAY POINTER

281	EDIT ACCESS
282	C_INTEGER
283	C_LONGINT
284	C_TEXT
285	C_REAL
286	C_PICTURE
287	ARRAY TO LIST
288	LIST TO ARRAY
289	CHANGE ACCESS
290	PLAY
291	QUIT 4D
292	QUERY BY EXAMPLE
293	C_STRING
294	Is a variable
295	Type
297	FLUSH BUFFERS
298	GRAPH SETTINGS
299	PAGE SETUP
300	ORDER BY FORMULA
301	C_POINTER
302	BREAK LEVEL
303	ACCUMULATE
304	Get pointer
305	C_BOOLEAN
306	C_TIME
307	C_DATE
308	Self
309	Open external window
310	AUTOMATIC RELATIONS
311	IDLE
314	Modified record
315	Nil
316	SET ABOUT
317	New process
319	PAUSE PROCESS
320	RESUME PROCESS
321	FILTER EVENT
322	Current process
323	DELAY PROCESS
324	HIDE PROCESS
325	SHOW PROCESS
326	BRING TO FRONT
327	Frontmost process

328	Outside call
329	CALL PROCESS
330	Process state
331	COPY NAMED SELECTION
332	USE NAMED SELECTION
333	CLEAR NAMED SELECTION
334	CUT NAMED SELECTION
335	Count tasks
336	PROCESS PROPERTIES
338	User in group
339	DISTINCT VALUES
340	RELATE MANY SELECTION
341	QUERY SELECTION
342	Count users
343	Count user processes
344	SET INDEX
345	SELECT LOG FILE
346	Activated
347	Deactivated
348	SET MENU ITEM
349	RELATE ONE SELECTION
350	SCAN INDEX
351	REDUCE SELECTION
352	C_GRAPH
353	LOCKED ATTRIBUTES
355	COMPRESS PICTURE
356	Picture size
357	LOAD COMPRESS PICTURE FROM FILE
358	SAVE PICTURE TO FILE
359	COMPRESS PICTURE FILE
361	ADD DATA SEGMENT
362	Read only state
363	Current default table
364	SET PRINT PREVIEW
365	PLATFORM PROPERTIES
366	MAP FILE TYPES
367	SET PLATFORM INTERFACE
368	SELECTION RANGE TO ARRAY
369	Database event
370	SET PROCESS VARIABLE
371	GET PROCESS VARIABLE
372	Process number
373	Execute on server

375	New list
376	APPEND TO LIST
377	CLEAR LIST
378	GET LIST ITEM
379	Selected list item
380	Count list items
381	SELECT LIST ITEM
382	REDRAW LIST
383	Load list
384	SAVE LIST
385	SET LIST ITEM
386	SET LIST ITEM PROPERTIES
387	SET LIST PROPERTIES
388	Form event
389	FILTER KEYSTROKE
390	Keystroke
391	SORT LIST
392	SET DEFAULT CENTURY
393	Add to date
394	RESOLVE POINTER
395	SET QUERY LIMIT
396	SET QUERY DESTINATION
397	In transaction
398	Trigger level
399	TRIGGER PROPERTIES
400	Test clipboard
401	GET CLIPBOARD
402	CLEAR CLIPBOARD
403	APPEND TO CLIPBOARD
404	Count menus
405	Count menu items
411	APPEND MENU ITEM
412	INSERT MENU ITEM
413	DELETE MENU ITEM
422	Get menu item
423	SET MENU ITEM KEY
424	Get menu item key
425	SET MENU ITEM STYLE
426	Get menu item style
428	Get menu item mark
430	Get menu title
431	SHOW MENU BAR
432	HIDE MENU BAR

433	SHOW TOOLBAR
434	HIDE TOOLBAR
437	Count screens
438	SCREEN COORDINATES
439	SCREEN DEPTH
440	Menu bar height
441	Menu bar screen
442	WINDOW LIST
443	GET WINDOW RECT
444	SET WINDOW RECT
445	Window kind
446	Window process
447	Frontmost window
448	Next window
449	Find window
450	Get window title
452	DRAG WINDOW
456	REDRAW WINDOW
457	PICTURE PROPERTIES
458	Tickcount
459	Milliseconds
460	FONT LIST
461	Font number
462	Font name
463	Mac to Win
464	Win to Mac
465	POST KEY
466	POST CLICK
467	POST EVENT
468	GET MOUSE
469	SET CURSOR
470	Get platform interface
471	VOLUME LIST
472	VOLUME ATTRIBUTES
473	FOLDER LIST
474	DOCUMENT LIST
475	CREATE DIRECTORY
476	Test path name
477	GET DOCUMENT PROPERTIES
478	SET DOCUMENT PROPERTIES
479	Get document size
480	SET DOCUMENT SIZE
481	Get document position

482	SET DOCUMENT POSITION
483	Current machine
484	Current machine owner
485	ACI folder
486	Temporary folder
487	System folder
488	Gestalt
489	Structure file
490	Data file
491	Application file
492	Compiled application
493	Application version
494	Application type
495	Version type
496	Create resource file
497	Open resource file
498	CLOSE RESOURCE FILE
499	RESOURCE TYPE LIST
500	RESOURCE LIST
501	DELETE RESOURCE
502	GET PICTURE RESOURCE
503	SET PICTURE RESOURCE
504	Get text resource
505	SET TEXT RESOURCE
506	Get string resource
507	SET STRING RESOURCE
508	GET RESOURCE
509	SET RESOURCE
510	Get indexed string
511	STRING LIST TO ARRAY
512	ARRAY TO STRING LIST
513	Get resource name
514	SET RESOURCE NAME
515	Get resource properties
516	SET RESOURCE PROPERTIES
517	GET ICON RESOURCE
519	Mac to ISO
520	ISO to Mac
521	SET PICTURE TO CLIPBOARD
522	GET PICTURE FROM CLIPBOARD
523	SET TEXT TO CLIPBOARD
524	Get text from clipboard
525	DOCUMENT TO BLOB

526	BLOB TO DOCUMENT
527	DATA SEGMENT LIST
528	Document type
529	Document creator
530	SET DOCUMENT TYPE
531	SET DOCUMENT CREATOR
532	VARIABLE TO BLOB
533	BLOB TO VARIABLE
534	COMPRESS BLOB
535	EXPAND BLOB
536	BLOB PROPERTIES
537	SET SCREEN DEPTH
538	Command name
539	Square root
540	MOVE DOCUMENT
541	COPY DOCUMENT
542	Pop up menu
543	Shift down
544	Macintosh control down
545	Macintosh option down
546	Macintosh command down
547	Caps lock down
548	INTEGER TO BLOB
549	BLOB to integer
550	LONGINT TO BLOB
551	BLOB to longint
552	REAL TO BLOB
553	BLOB to real
554	TEXT TO BLOB
555	BLOB to text
556	LIST TO BLOB
557	BLOB to list
558	COPY BLOB
559	INSERT IN BLOB
560	DELETE FROM BLOB
561	REMOVE FROM SET
562	Windows Ctrl down
563	Windows Alt down
600	COPY SET
601	SET TABLE TITLES
602	SET FIELD TITLES
603	SET VISIBLE
604	C_BLOB

605	BLOB size
606	SET BLOB SIZE
607	DRAW AND DROP PROPERTIES
608	Drop position
609	GET USER LIST
610	GET GROUP LIST
611	GET USER PROPERTIES
612	Set user properties
613	GET GROUP PROPERTIES
614	Set group properties
615	DELETE USER
616	Is user deleted
617	START WEB SERVER
618	STOP WEB SERVER
619	SEND HTML FILE
620	SET WEB DISPLAY LIMIT
621	Is a list
622	SET WEB TIMEOUT
623	SET REAL COMPARISON LEVEL
624	DELETE LIST ITEM
625	INSERT LIST ITEM
626	Copy list
627	Current form table
628	SET RGB COLOR
629	List item position
630	SELECT LIST ITEM BY REFERENCE
631	GET LIST ITEM PROPERTIES
632	GET LIST PROPERTIES
633	List item parent
634	SET HTML ROOT
635	VARIABLE TO VARIABLE

参照

Command name、コマンド名によるコマンド、EXECUTE

この章では、「ルーチン」エディタの「Math」テーマ内にある算術関数について説明します。この章では、標準的な算術演算を行なう関数について説明します。これらの関数は、数値を返します。

Abs	Int	SET REAL COMPARISON LEVEL
Arctan	Log	Square root
Cos	Mod	Sin
Dec	Random	Tan
Euro converter	Round	Trunc
Exp		

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。

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

参照
なし

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。

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	数値の少数部

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。

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

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。

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。

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。

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

参照

なし

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。

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。

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。

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)

$\$0 := \text{Square root } ((\$1^2) + (\$2^2))$

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	余り

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。

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

三角関数

この節では、三角関数について説明します。これらの関数は、単位にラジアンを使用して計算します。一度を0.0174532925199432958として扱います。

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	数値のサイン

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...) で表わされた一度 (ディグリー) 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 数値	タイプ 数値	説明 コサイン (cosine) で返す値 (単位...ラジアン)
戻り値	数値	数値のサイン

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...) で表わされた一度 (デイグリー) 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 数値	タイプ 数値	説明 タンジェント (Tangent) で返す値 (単位...ラジアン)
戻り値	数値	数値のタンジェント

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...) で表わされた一度 (デイグリー) 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	数値のサイン

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...）で表わされた一度（ディグリー）を、Radianは度数（57.29577...）で表わされた1ラジアンを返します。

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

なし

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

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
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

必要とあらば、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（結果は整数の番号です）。

修正されたユーロと11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
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)

参照

なし

実数の表示について

注意

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ひとつにすぎません。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、 $3 \times 1/3$ 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に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0.3333333333333333と表示され、WindowsやPower Macintoshでは0.33333333333333148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気になることでしょう。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。

この章のコマンドは、メニューバーの切り替え、メニューコマンドでのテキストの変更、メニューコマンドのチェック、選択可と選択不可の切り替えを行なうためのものです。

APPEND MENU ITEM	Get menu item key	Menu selected
Count menus	Get menu item mark	SET ABOUT
Count menu items	Get menu item style	SET MENU ITEM
DELETE MENU ITEM	Get menu title	SET MENU ITEM KEY
DISABLE MENU ITEM	HIDE MENU BAR	SET MENU ITEM MARK
ENABLE MENU ITEM	INSERT MENU ITEM	SET MENU ITEM STYLE
Get menu item	MENU BAR	SHOW MENU BAR

メニューの管理

用語について

メニューコマンドに関する説明では、メニューの行について述べる部分で「メニューコマンド」と「メニューアイテム」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える必要があります。

各メニューコマンドには、それに付属するひとつのプロジェクトメソッドがあります。メニューコマンドにメソッドを割り当てていないと、そのメニューコマンドを選択したときに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」メニューコマンドしかありません。「終了」アイテム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、メニューコマンド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。メニューコマンド「終了」は、「ファイル」メニューの一番最後のメニューコマンド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 (R) について...」メニューコマンド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 (R) について...」メニューコマンドは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 (R) について...」メニューコマンドと、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 (R) について...」メニューコマンドは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コマンド番号

メニューバーと同様に、メニュー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番で始まりません。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コマンドは、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番上のメニューコマンドがアイテム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現れる「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コマンドは、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コマンド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アイテムを修正する

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、メニューアイテムの追加、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アイテムはすべて「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アイテム番号が必要です。

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。

メニューアイテムは、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アイテムの番号を返します。

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
プロセス	数値	プロセス参照番号
*		メニューバーの状態保存

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。

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、複数のメニューコマンド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、メニューコマンドはすべて、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コマンド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、メニューコマンドの状態を元に戻します。

MENU BAR (3)

2. 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、メニューコマンドの状態を保存します。前に使用不可にされたメニューコマンドが、使用不可で現れます。

MENU BAR (3 ; *)

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。

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(\$vIX ; \$vIY ; \$vIButton)

Until (\$vIButton#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れている場合は、このコマンドは何も行いません。

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	アバウトメニューアイテムの新しいテキスト
メソッド	文字列	メニューアイテム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 (R) 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コマンドを選択すると、<メソッド>が実行されます。<メソッド>はダイアログボックスを開き、データベースに関するバージョンやその他の情報を表示することができます。

4th Dimensionアイコン、4th Dimensionバージョン番号、4D Compilerバージョン番号、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。

注：4D Compilerのプロジェクトで「バージョン番号の自動生成」を選択した場合、アプリケーションのコンパイルバージョン番号も表示されます。

例題

1. 以下の例は、「4th Dimension (R) について」メニューコマンドを「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 (R) について」メニューコマンドを元のアバウトボックスに戻します。

SET ABOUT ("4th Dimension (R) について..." ; "")

参照

なし

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コマンド 上位バイトにメニュー番号 下位バイトにメニューアイテム番号

説明

Menu selected関数は、フォームが表示し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コマンドを検出します。

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）のメニューコマンドに割り当てられたメソッドを使用してください。いずれのメニューコマンド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アイテム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65,536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コマンド番号を知るには、係数65,536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
```

```
メニューコマンド:=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
```

```
メニューコマンド:= Menu selected & 0xFFFF
```

メニューコマンドが選択されていない場合には、**Menu selected**関数は0を返します。

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アイテムの数

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コマンドの数を返します。

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 ; メニューアイテム {; 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
メニューアイテム	数値	メニューアイテム番号
プロセス	数値	プロセス参照番号
戻り値	文字列	メニューアイテムのテキスト

説明

Get menu item関数は、引数<メニュー>と<メニューアイテム>に渡されたメニュー番号およびメニューアイテム番号を持つメニューアイテム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 ; メニューアイテム番号 ; 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
メニューアイテム番号	数値	メニューアイテム番号
アイテムテキスト	文字列	メニューアイテムの新テキスト
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 <メニューアイテム番号> で渡された番号を持つメニューアイテム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

なし

Get menu item style

Get menu item style (メニュー ; メニューアイテム {; 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
メニューアイテム	数値	メニューアイテム番号
プロセス	数値	プロセス参照番号
戻り値	数値	メニューアイテム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<メニューアイテム>に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンド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。

以下の例は、任意のメニューコマンドが太字（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 ($vIMenu ; $vIItem) & Bold)#0)
    ...
End if
```


参照

SET MENU ITEM STYLE

SET MENU ITEM STYLE

SET MENU ITEM STYLE (メニュー ; メニューアイテム ; 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
メニューアイテム	数値	メニューアイテム番号
スタイル	数値	新しいメニューアイテム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 <メニューアイテム> に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンド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 ; メニューアイテム {; 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
メニューアイテム	数値	メニューアイテム番号
プロセス	数値	プロセス参照番号
戻り値	文字列	カレントメニューアイテムのマーク

説明

Get menu item mark関数は、引数<メニュー>と<メニューアイテム>に渡されたメニュー番号およびメニューアイテム番号を持つメニューアイテム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。

メニューコマンド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。

以下の例は、メニューアイテムのチェックマークを切り替えます。

```
SET ITEM MARK ($vMenuItem ; $vItem ; Char (18)*Num (Get menu item mark($vMenuItem ; $vItem)=""))
```

参照

SET MENU ITEM MARK

SET MENU ITEM MARK

SET MENU ITEM MARK (メニュー ; メニューアイテム ; 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
メニューアイテム	数値	メニューアイテム番号
マーク	数値	新しいメニューアイテムのマーク
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 <メニューアイテム> に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンド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、メニューコマンド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アイテム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、メニューアイテム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 ; メニューアイテム {; 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
メニューアイテム	数値	メニューアイテム番号
プロセス	数値	プロセス参照番号
戻り値	数値	メニューアイテム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<メニューアイテム>に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのキーボード (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。

メニューコマンド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 ; メニューアイテム ; 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
メニューアイテム	数値	メニューアイテム番号
キー	数値	新しいメニューアイテムキーの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 <メニューアイテム> に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのキーボード (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、メニューアイテム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 ; メニューアイテム { ; 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
メニューアイテム	数値	メニューアイテムの番号
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 <メニューアイテム> で指定したメニューアイテムを選択不可の状態（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、<メニューアイテム>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 ; メニューアイテム { ; 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
メニューアイテム	数値	メニューアイテムの番号
プロセス	数値	プロセス参照番号

説明

ENABLE MENU ITEM (メニュー番号 ; メニューコマンド番号)

ENABLE MENU ITEM コマンドは、<メニュー番号> と <メニューアイテム> で指定したメニューアイテム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、＜メニューアイテム＞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	新しいメニューアイテムのテキスト
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アイテムを追加します。

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回の呼び出しでひとつまたは複数のメニューアイテムを追加することができます。

追加されるメニューアイテム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（；）で各メニューコマンドを区切る。例えば、

“ 項目テキスト1；項目テキスト2；項目テキスト3 ”

メニューアイテム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。

メニューアイテム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ひとつを配置する。

<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)

任意のメニューアイテムにチェックマークを追加するには、引数<テキスト>の中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チェックマークが表示される。

任意のメニューアイテムにアイコンを追加するには、引数<テキスト>の中にカレット(^)を指定し、ASCIIコードから48を引いた値がMacOSベースのリソースIDである文字をその後に配置する。

任意のメニューアイテム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：メニューアイテムの数が程よいメニューを使用してください。もし、50以上のメニューアイテム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：新しいメニューアイテムには、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。

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、メニューコマンドが作成される
FONT LIST(<>asAvailableFont)
<>atFontMenuItems:=""
For ($vFont ; 1 ; Size of array(<>asAvailableFont))
    <>atFontMenuItems:=<>atFontMenuItems+";"<>asAvailableFont{$vFont}
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 ; メニューアイテム ; 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
メニューアイテム	数値	メニューアイテム番号
テキスト	文字列	挿入されるメニューアイテムのテキスト
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<メニューアイテム>に渡された番号を持つメニューアイテムの後ろに新しいメニューアイテムを挿入します。

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回の呼び出しでひとつまたは複数のメニューアイテム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もメニューアイテム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アイテム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>に渡すメニューアイテム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：新しいメニューアイテムには、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 ; メニューアイテム {; プロセス})

引数	タイプ	説明
メニュー	数値	メニュー番号
メニューアイテム	数値	メニューアイテム番号
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 <メニューアイテム> に渡されたメニュー番号およびメニューアイテム番号を持つメニューアイテムを削除します。

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、メニューコマンド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

この章では、「ルーチン」エディタの「Messages」テーマ内にあるメッセージコマンドについて説明します。この章のコマンドは、ユーザに対してメッセージを表示するものです。メッセージには、標準ダイアログボックスとメッセージウインドウ、およびカスタムのメッセージウインドウとインジケータ等も含まれます。

ダイアログボックスは形式的なもので、ユーザがボタンをクリックするか、または“enter”キーを押すことによって処理を続行します。

ALERTコマンドは、ユーザに情報を伝える場合にのみ使用します。

CONFIRMコマンドは、ユーザに情報を伝え、動作を行うべきかどうかの指示を得る場合に使用します。

Request関数は、ユーザからのテキスト情報を得る場合に使用します。

ALERT
CONFIRM
GOTO XY

MESSAGE
MESSAGES OFF
MESSAGES ON

Request

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」モードのメニューコマンド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		

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

参照
なし

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。

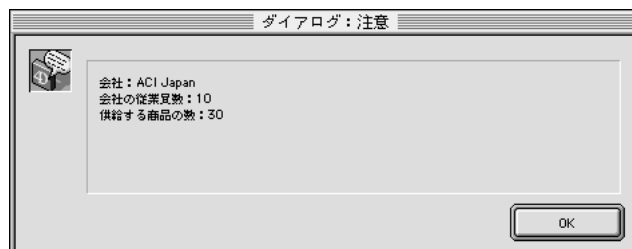
CR:=Char (13)

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]))

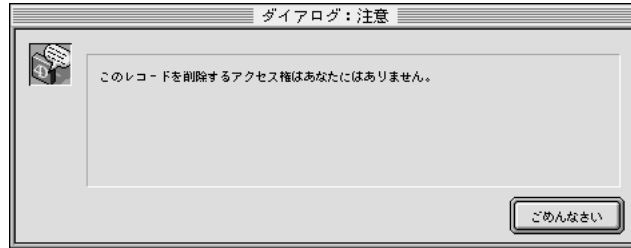
以下の図は上記の**ALERT**コマンドで表示される警告ダイアログボックスを示しています。



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、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**コマンドをコールしないでください。永くループになります。

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])

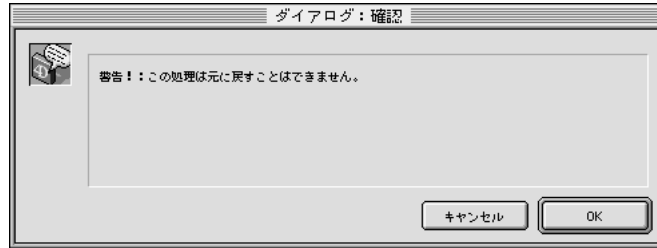
DELETE SELECTION ([元従業員])

Else

ALERT ("処理が取り消されました。")

End if

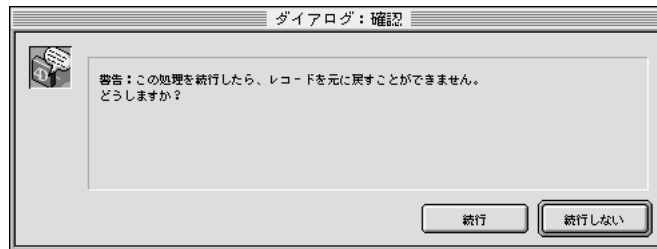
以下の図は、この**CONFIRM**コマンドで表示される「確認」ダイアログボックスを示しています。



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。

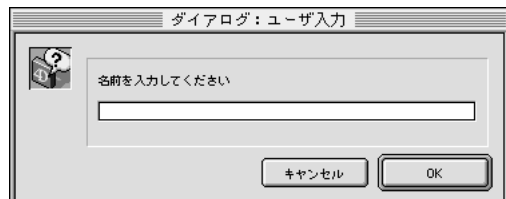
Tip : On ActivateまたはOn Deactivateフォームイベントを処理するフォームメソッド、またはオブジェクトメソッドのセクションから**Request**関数をコールしないでください。永久ループになります。

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。

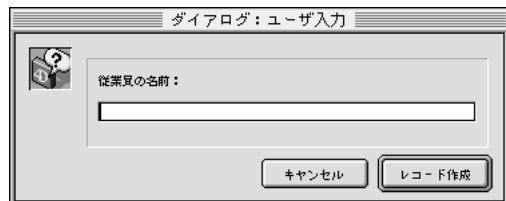
vsPrompt:=**Request** ("従業員の名前: "; ""; "レコード作成"; "キャンセル")

If (OK=1)

ADD RECORD ({従業員})

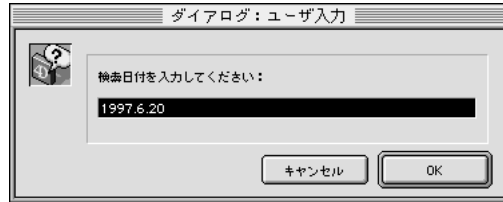
 `変数「vsPrompt」は、フォームメソッド内のOn Loadイベント中に
 [従業員]名文字フィールドにコピーされます。

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開か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開かれて、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。

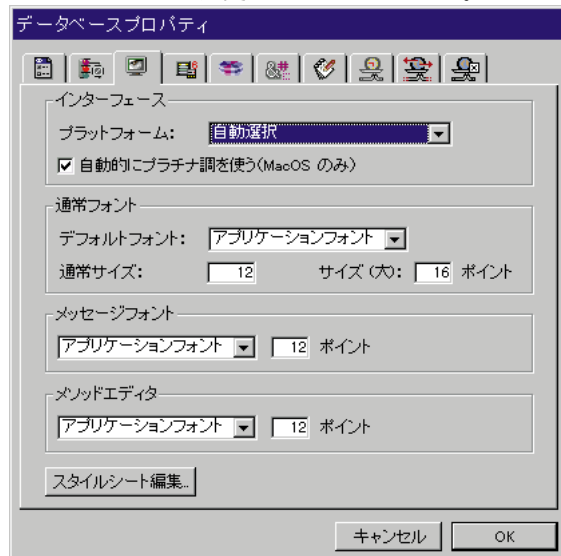
一連のメッセージは、このウインドウで表示されると、前のメッセージを消去しません。その代わりに、新しいメッセージは既存のメッセージに続けて表示されます。

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も長い場合、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。

4th Dimensionはメッセージを表示するために、メッセージフォントおよびメッセージフォントサイズプロパティを使用します。これらの設定は「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ができます。



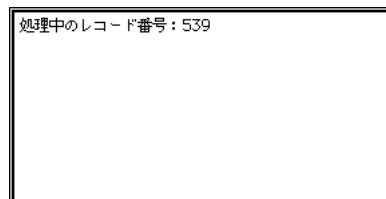
必要に応じてフォントとフォントサイズを選択することができますが、**MESSAGE**と**GOTO XY**コマンドを組み合わせて使用する場合には、等幅フォントを選択することをお勧めします。

例題

1. 以下の例は、レコードセクションを処理し、**MESSAGE**コマンドをコールしてユーザー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
```

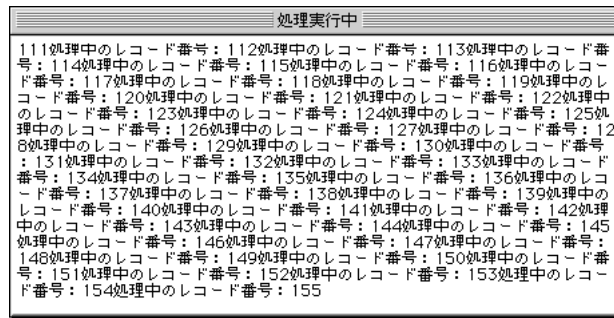
MESSAGEコマンドをコールするたびに、以下のウインドウが現れては消えます。



2. ウィンドウの“プリンキング(ちらつき)”を避けるため、以下の例題のように**Open window**関数で開いたウィ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($vIRecord;1;Records in selection ([テーブル1]))
  MESSAGE ("処理中のレコード番号 : "+String ($vIRecord))
  `レコードを使った処理を実行
  NEXT RECORD ([テーブル1])
End for
CLOSE WINDOW
```

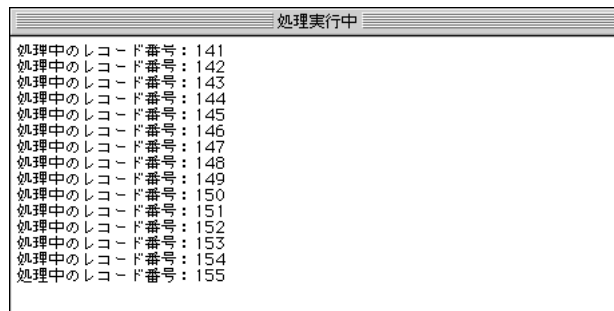
以下のウィンドウが表示されます。



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 ($vIRecord;1;Records in selection ([テーブル1]))
  MESSAGE ("処理中のレコード番号 : "+String ($vIRecord)+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
```

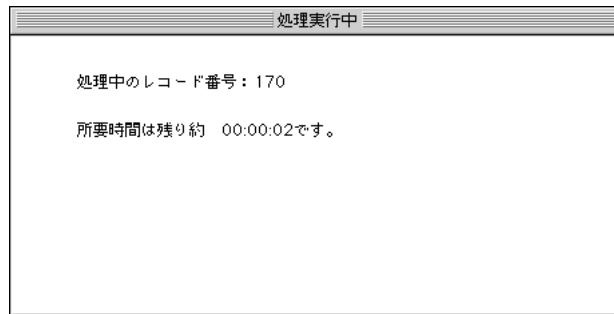
以下のような結果になります。



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")
$viNbRecords:=Records in selection ([テーブル1])
$vhStartTime:=Current time
For ($viRecord;1;$viNbRecords)
GOTO XY (5;2)
    MESSAGE ("処理中のレコード番号 : "+String ($viRecord)+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)
    $viRemaining:=(($viNbRecords/$viRecord)-1)*(Current time-$vhStartTime)
    MESSAGE ("所要時間は残り約  "+Time string ($viRemaining) + "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 ($vRow;0;9)
    GOTO XY ($vRow;0)
    MESSAGE (String ($vRow))
End for
For ($vLine;0;9)
    GOTO XY (0;$vLine)
    MESSAGE (String($vLine))
End for
$vhStartTime:=Current time
Repeat
Until ((Current time-$vhStartTime)>!!00:00:30!!))
```



参照

MESSAGE

この章では、「ルーチン」エディタの「Named selections」テーマ内にある命名セレクションコマンドについて説明します。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のみ利用することはできません。

4D Clientと4D Serverを使用している場合、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のみ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、第48章の「セットコマンド」を参照してください。

命名セレクションの例

以下の例では、データベース内で“名前”フィールドに重複した値があるかどうか調べています。これは“名前”フィールドのスクリプトです。まず、フィールド内の値を変数に割り当てています。そして、そのレコードとセレクションを保存します。同じ値を持つレコードが見つかったら、そのセレクションとレコードを取り出します。値を無効にし、ユーザに新しい値を入れるように要求します。

```
` 値を一時変数に保存
$名前:=[従業員]名前
` レコードをスタックにプッシュする
PUSH RECORD ([従業員])
` ソートしたセレクションを保存
CUT NAMED SELECTION ([従業員]; "ユーザソート")
` フィールドの値を検索する
QUERY ([従業員];[従業員]名前=$名前)
` 同じ値を持つレコードが見つかった場合
$発見:=(Records in selection ([従業員]) > 0)
` 前に保存したセレクションを使用する
USE NAMED SELECTION ("ユーザソート")
` レコードを戻す
POP RECORD ([従業員])
` レコードが見つかったら、ユーザに知らせてフィールド内の値を無効にする
If ($発見)
    ALERT ("名前フィールドに重複データが入力されています")
    REJECT ([従業員]名前)
End if
```

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>のカレントセクションを命名セクション<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>のカレントセクションとカレントレコードは変更されません。

命名セ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クションをコピーすると、セ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。命名セクションはメモリに置かれるため、命名セ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：命名セクション<名前>とカレントセクションは、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。

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、そのセクションを再利用してメモリ内の命名セクションを消去します。ソートしたセクションを後で使いたい場合には、命名セ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]) ` セクションをソートする
` ソートしたセクションを命名セ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") ` ソートした命名セクションを利用

CLEAR NAMED SELECTION ("ユーザソート") ` セ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。

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	使用する命名セクションの名前

説明

USE NAMED SELECTION コマンドは、命名セクション < 名前 > をそれが属するテーブルのカレントセクションとして使用します。

命名セクションを作成すると、その命名セ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クション < 名前 > を作成した場合は、カレントセクションが < 名前 > に設定され、< 名前 > はメモリから消去されます。

命名セクションは、それが作成された時点でのレコードのセクションを表わす点に注意してください。命名セクションが表わしているレコードが変更されると、命名セクションは正確ではなくなります。したがって、命名セクションは通常、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、命名セクションのレコードの変更、削除、および命名セ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 省略した場合は現在のデフォルト 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クションセ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クションに含める (True) か含めない (False) かを示します。

セ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

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。十分な大きさが無い場合には、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 オブジェクト名（*を指定した場合） 変数（*を省略した場合）
オブジェクト	フォームオブ ジェクト	
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。

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

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。

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 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、入力可/入力不可変数、数値のスクロールエリアに適用できます。

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を使用します。

ブールフィールドをフォーマットするには、2つの値をセミコロン（;）で区切って渡します。

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 Forcldr	倍長整数	7

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、「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。

例題

1. 以下の例は、変数“vDate”に対して、定数「Short」の日付フォーマットを設定します。

SET FORMAT (vDate ; Char (Short))

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")

参照

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。

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。船積みを行なう場合に、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

参照

なし

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 オブジェクトの名前（文字列） 省略した場合は、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） フィールドまたは変数（*を省略した場合）
入力可	ブール	True=入力可、Fals =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。

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。

このコマンドを使用することは、「フォーム」エディタの「オブジェクトプロパティ」ウインドウ内でフィールドや変数に入力可／入力不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。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。

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。船積み量が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

```

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。

☑ 現在の従業員

従業員情報

従業員氏名

住所

〒500-0000 岐阜県岐阜市 郵便番号

詳細...

キャンセル OK

または、



参照
DISABLE BUTTON、ENABLE BUTTON、SET ENTERABLE

SET COLOR

SET COLOR ({*;} オブジェクト ; カラー)

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）
カラー	数値	オブジェクトに使用する新しいカラー

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<カラー>で、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「オブジェクトプロパティ」ウィンドウでカラーパレットを見ることができます。

よく使用されるカラー番号は、下記のように前もって定義された定数で提供されます。

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。

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; 前景色; 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、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	描画色におけるRGBカラーの値
背景色	数値	背景色における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。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<描画色>と<背景色>で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が使用する4つの自動色のいずれかを指定して、色を自動設定してオブジェクトを描くことができます。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
Default foreground color	倍長整数	-1
Default background color	倍長整数	-2
Default dark shadow color	倍長整数	-3
Default light shadow color	倍長整数	-4

(標準システムにおける)これらの色を次に示します。

Default Foreground Color



Default Background Color



Default Dark Shadow Color



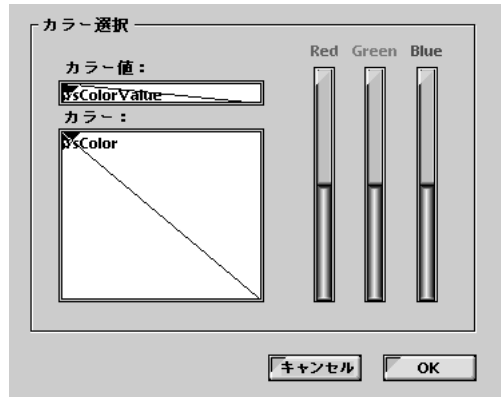
Default Light Shadow Color



注：Windows上では、これらの自動色はシステムに依存します。「カラー」コントロールパネルで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。

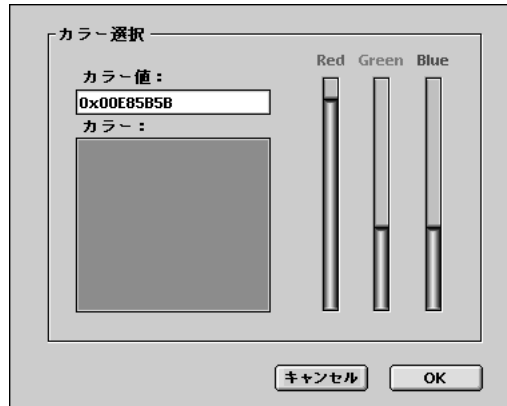
・「CLICK IN COLOR THERMOMETER」プロジェクトメソッド

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）である。 省略した場合=オブジェクトは変数で ある。
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。

"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;bottom)

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）である。 省略された場合=オブジェクトは変数である。
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。

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

この章では、「ルーチン」エディタの「On a Series」テーマ内にあるリスト上の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、一連の値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、ブレイク処理がオンの場合でフブレイクレベルが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ブレイ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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の合計

説明

Sum関数は、＜一連の値＞の合計値を返します。＜一連の値＞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、ブレイク処理をアクティブにしてセ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イク数と同じ数でなければなりません。ブレイ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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の平均値

説明

Average関数は、＜一連の値＞の平均値を返します。＜一連の値＞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。

以下の例は、出力フォームのB0ブレイ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のブレイ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、ブレイク処理をアクティブにしてセ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イク数と同じ数でなければなりません。ブレイ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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の最小値

説明

Min関数は、＜一連の値＞の最小値を返します。＜一連の値＞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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の最大値
-----	----	----------

説明

Max関数は、＜一連の値＞の最大値を返します。＜一連の値＞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。

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、フォームのブレイク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イ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、ブレイク処理をアクティブにしてセ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イク数と同じ数でなければなりません。ブレイ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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の標準偏差

説明

Std deviation関数は、＜一連の値＞の標準偏差を返します。＜一連の値＞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。

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、ブレイ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イク数と同じ数でなければなりません。ブレイ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 (一連の値) 数値

引数	タイプ	説明
一連の値	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	一連の値の分散

説明

Variance関数は、＜一連の値＞の分散を返します。＜一連の値＞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。

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 (一連の値) 数値

引数 一連の値	タイプ フィールドまたは サブフィールド	説明 平方和を得るためのデータ
戻り値	数値	一連の値の平方和

説明

Sum squares関数は、＜一連の値＞の平方和を返します。＜一連の値＞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。

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

この章では、「ルーチン」エディタの「Picture」テーマ内にあるピクチャコマンドについて説明します。この節のコマンドは、QuickTimeを使ってピクチャを圧縮するために使用し、ドキュメントとしてそれを保存します。

PICTURES

COMPRESS PICTURE

LOAD COMPRESS PICTURE FROM FILE

COMPRESS PICTURE FILE

SAVE PICTURE TO FILE

PICT TO GIF

Picture size

PICTURE PROPERTIES

PICTURE LIBRARY LIST

GET PICTURE FROM LIBRARY

SET PICTURE TO LIBRARY

REMOVE PICTURE FROM LIBRARY

また、この章では、QuickTimeの圧縮タイプとピクチャ圧縮エラーにおけるエラーコードを示します。

重要： QuickTime3より以前のQuickTimeでは、Windows上でQuickTimeのピクチャ圧縮が利用できません。しかし、上記のコマンドは4th Dimensionの中で使用することができます。Windows上では利用できませんが、Macintosh上で利用できます。QuickTime3をお使いの場合は、Windows上でもすべてのピクチャコマンドが使用できます。QuickTime 3はApple Japanのホームページからダウンロードできます (<http://www.apple.co.jp/>)。

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 95 + QT 32ビット		

ACI_Pack ReadPictureFile サポートされる形式（注を参照）

	PICT	BMP	WMF	EMF	JPEG
Macintosh			×	×	
		Mac PICT に変換			Mac PICT に拡張
Windows		Mac PICT に変換	PicComment に埋め込み	PicComment に埋め込み	Mac PICT に拡張

注：ACI Packは、ACIの4Dプラグインで4th Dimensionパッケージに付属しています。

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い場合は、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。

QuickTimeの圧縮タイプ

次に示すのは、QuickTimeで扱うことのできる圧縮のタイプの一覧です。

写真圧縮-JPEG ('jpeg') : Photo compressorピクチャの圧縮にJoint Photographic Experts Group (JPEG) アルゴリズムを採用します。JPEGは静止画を圧縮するための国際標準です。

ビデオ ('rpza') : Video compressor良品質のピクチャを維持したまま高速に解凍を行うことができます。

アニメーション ('rle') : Animation compressor動画やコンピュータ処理のビデオデータを処理する場合に用います。

なし ('raw') : Raw compressor ピクチャのピクセルデプス (深度) を変換して、ピクチャの格納容量を減らします。

グラフィックス ('smc') : Graphics compressor「アニメーション」の代替形式で、結果の品質は劣ります。

コンパクト・ビデオ ('cdvs') : Compact video compressor「ビデオ」圧縮に似ていますが、より高度な圧縮率と、良品質のピクチャ、高速な再生を行うことができます。

注：「アニメーション」と「グラフィックス」タイプの機能は、「圧縮設定」ダイアログボックスの「品質」スライダーによる設定が「最高」になっている場合にのみ有効です。

圧縮のタイプを指定するとき、タイプ名に含まれているスペースも忘れずに、引数 <方法> に指定してください。圧縮方法が空の文字列の場合、ピクチャはロードされますが圧縮されません。

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。

Windows上でApple QuickTimeを使用する

Alturaおよび4th Dimensionはいずれも純正32ビットアプリケーションのため、32ビットのQuickTime for Windows (version 2.1.1 b50以上) をインストールする必要があります。

QuickTime for Windowsは、Windows 3.1をサポートしないため、QuickTime画像をWindows 3.1で表示する方法はありません。

この画像を表示するには、QuickTime for Windowsがディスク上の作業ファイルを使用する必要があります。QuickTime画像が画面に表示されるたびに、それが最初にディスクに書き込まれ、次に削除されます。通常、テンポラリファイルはキャッシュに残り、実際にディスクに書き込まれることはないため、この操作は非常に高速に行われます。ただし、低速なPCの場合、またはメモリが不足している場合、この操作も低速になる可能性があります。

QuickTime for Windowsは、QuickTime画像を表示だけです。画像を圧縮することはできません。QuickTime圧縮を行う4th DimensionコマンドがWindows上で機能しないのはこのためです (将来も、QuickTime for Windowsが圧縮をサポートするまで機能しません)。Windowsで機能しないコマンドは以下の通りです。

COMPRESS PICTURE、COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE

SAVE PICTURE TO FILEコマンドはQuickTimeを使用しないため、Windows上でもMacintosh上と同様に機能します。これはMacintoshのPICTファイルを生成し、生成されたファイルはPhotoshopのような高度なグラフィックスアプリケーションによりWindows上でオープンできます。

注：QuickTime3をお使いの場合は、画像圧縮がサポートされていますため、Windows上でもすべてのピクチャコマンドが使用できます。

参照

COMPRESS PICTURE、COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE、PICTURE PROPERTIES、Picture size、SAVE PICTURE TO 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.

PICT TO GIF

PICT 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。

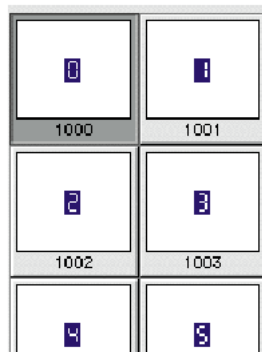
ピクチャタイプの変数またはフィールドをピクチャに、BLOBタイプの変数またはフィールドをblobGIFに渡します。コマンドの実行後、GIF用BLOBの内容はGIFフォーマットの画像になります。

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、これは87a（opaque）およびnormal（not interlaced）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
```

gifcounter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
PICT 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。



参照
なし

Picture size

Picture size (ピクチャ) 整数

引数 ピクチャ	タイプ ピクチャ	説明 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

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つの配列のひと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、ピクチャライブラ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)
$viNbPictures:=Size of array ($alPicRef)
If ($viNbPictures>0)
    SET CHANNEL (12 ; "")
    If (OK=1)
        $vsTag:="4DV6PICTURELIBRARYEXPORT"
        SEND VARIABLE ($vsTag)
        SEND VARIABLE ($viNbPictures)
        Error:=0
        For ($viPicture ; 1 ; $viNbPictures)
            $viPicRef:=$alPicRef{$viPicture}
            $vsPicName:=$asPicName{$viPicture}
            GET PICTURE FROM LIBRARY (alPicRef{$viPicture} ; $vgPicture)
            If (OK=1)
                SEND VARIABLE ($viPicRef)
                SEND VARIABLE ($vsPicName)
                SEND VARIABLE ($vgPicture)
            Else
                $viPicture:=$viNbPictures+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)

引数	タイプ	説明
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	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。

参照番号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 “ \$vIPicRef ” 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 (\$vIPicRef ; vgMyPicture)

2. **PICTURE LIBRARY LIST** コマンドの例を参照してください

説明

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 (\$vINbPictures)

If (\$vINbPictures)

For (\$vlPicture ; 1 ; \$vINbPictures)

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:=\$vINbPictures+1

ALERT ("このファイルは、ダメージを受けているみたいです。")

End if

End for

Else

ALERT ("このファイルは、ダメージを受けているみたいです。")

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	数値	ピクチャライブラリのピクチャの参照番号

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ありません。**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

この章では、「ルーチン」エディタの「Printing」テーマ内にある印刷コマンドについて説明します。レポートの印刷は、データベースの最も重要な作業です。この章のコマンドで、4th Dimensionの柔軟性の高いレポート機能を使用することができます。

ACCUMULATE	PRINT LABEL	Printing page
BREAK LEVEL	PRINT FORM	REPORT
PAGE BREAK	PRINT RECORD	SET PRINT PREVIEW
Level	PRINT SELECTION	Subtotal
PAGE SETUP	PRINT SETTINGS	

PRINT SELECTIONコマンド、**PRINT FORM**コマンド、**PRINT LABEL**コマンドを使用して、「デザイン」モードで作成したフォームからレポートを作成することができます。また、フォームを作成しなくても、**REPORT**コマンドや**PRINT LABEL**コマンドを使用してレポートを作成することができます。

フォームを使用すれば、ほとんどのレポートをデザインすることができます。レポートにはグラフィックスを入れることもできます。また、レポートを構成する要素を調整するためのさまざまな方法が用意されています。フォームで印刷されるレポートは、フォームメソッドを実行できるため、大きな処理能力が得ることができます。

「クイックレポート」エディタや「ラベル」ウィザードで作成するレポートは、一般に単純なデザインになっています。ユーザはこれらのエディタを使用して、レポートを設計することができます。また、レポートのデザインをディスクに保存することもできます。これらのレポートはフォームを使用しないため、フォームメソッドを実行することはできません。

REPORT、**PRINT SELECTION**、**PRINT FORM**の3つの主要なレポートコマンドは、柔軟性の度合いがそれぞれ異なります。一般に、コマンドの柔軟性が高いほど、レポート作成時にそのデザイン手腕が要求されます。

REPORTコマンドは、最も簡単なレポート作成コマンドです。このコマンドは、「ユーザ」モードで使用するものと同じ「クイックレポート」エディタを使用します。レポートのスタイルは、行とカラムで構成されます。これには、ヘッダ、フッタ、さまざまなフォントや書体、フォーマット、フォーミュラ、ブレイク、集計、計算、可変長のテキストを含む複数カラム等があります。レポートはグラフやASCIIファイルにも変換可能です。

PRINT SELECTIONコマンドは、最もよく使用される印刷コマンドです。これには、出力フォームを使用します。このフォームのデザインに制約はありません。このコマンドにもヘッダ、フッタ、さまざまなフォントや書体、フォーマット、フォーミュラ、ブレイク、集計、計算が使用されます。簡単なレポートは、フォームメソッドがなくても作成することはできますが、ほとんどの場合では、レポートを処理するためのフォームメソッドやオブジェクトメソッドを作成します。

PRINT FORMコマンドは、すべての印刷コマンドの中で最も柔軟性の高いものです。また、最もデザイン手腕が要求されるものです。これは、同じページに異なるフォームを配置することができます。さらに、印刷中の任意の時点で改ページ（フォーム・フィールド）を挿入することができます。

PRINT LABELコマンドは、フォームを使用してラベルを印刷します。これは、高い柔軟性を持っています。「ユーザ」モードに用意された「ラベル」ウィザードを使用するため、必要に応じてラベルをデザインすることができます。このコマンドは、レコードを並列に印刷することができるため、特殊なレポートを作成する場合にも利用することができます。

すべての印刷コマンドは、カレントセクションを印刷します。印刷前にセクションをソートするのが一般的です。

ユーザはレポートを印刷する場合に、それを印刷するかどうかを画面上で選択することができます。4th Dimensionは、印刷中に、印刷されるカレントページと印刷状態を表示します。ユーザが画面上のレポートを“印刷”する場合に、ユーザは「プリント」ボタンをクリックして、カレントページを印刷することができます。

印刷を取り消す場合は、「プリント中止」ボタンを押します。ユーザがこのボタンをクリックするか、あるいはプリンタダイアログボックスを取り消して印刷を中止すると、システム変数OKに0が代入されます。印刷がうまくいくと、システム変数OKに1が代入されます。

フォームレポートにおけるブレーク処理の生成

出力フォームのブレーク処理は、2つの方法で生成されます。

Subtotal関数を使用する。

BREAK LEVELコマンドと**ACCUMULATE**コマンドを使用する。

両方とも同じ結果を得られますが、それぞれに異なる利点を持っています。

Subtotal関数を使用したブレーク処理

Subtotal関数でブレーク処理を実行するためには、フォームメソッドまたはフォームのブレークエリアに配置された変数のオブジェクトメソッドで使用しなければなりません。レポートを印刷する前に、4th Dimensionはフォームメソッドとオブジェクトメソッドに**Subtotal**関数が指定されているか調べます。

4th Dimensionがこの関数を見つけると、ブレーク処理がアクティブになります。ブレーク処理をオンにするために、**Subtotal**関数を実行する必要はありません。例えば、フッタ行の下に配置されたオブジェクトのオブジェクトメソッドにこの関数を指定することもできます。この場合、決して印刷または実行されることはありません。

Subtotal関数をブレーク処理を生成させるために使用した場合に、ブレークを行うものよりも1つ多いレベルでソートしなければなりません。例えば、レポート中に2つのソートレベルを必要とする場合は、ソートは3レベルで行います。

BREAK LEVELコマンドおよびACCUMULATEコマンドを使用したブレーク処理

ブレーク処理をオンにするために、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用することもできます。この場合、レポートを印刷する前にこれらのコマンドの両方を実行しなければなりません。この方法を使用する場合でも、フォーム上の値を表示するためには**Subtotal**関数が必要です。またこの方法が使用される場合、1つ多いレベルでソートする必要はありません。ただし、少なくともブレークする数と同じ数のレベルでソートする必要があります。

2つの方法の比較

ブレーク処理を開始するために**Subtotal**関数を使用する主な利点は、レポートの印刷前にメソッドを実行する必要がないことです。これは、「ユーザ」モードにおいては特に有効です。

「ユーザ」モードでレポートを印刷するための行程は、以下のようになります。

1. 印刷するレコードを選択する。
2. レコードのソートを行う。1つ多いレベルでソートを行う。
3. 「ファイル」メニューから「プリント...」を選択する。

4th Dimensionは、フォームメソッドとオブジェクトメソッドを調べ、**Subtotal**関数を検索します。関数を見つけると、ブレーク処理をオンにし、レポートを印刷します。しかし、ブレーク処理に**Subtotal**関数を使用すると、以下の2つの欠点があります。

Subtotal関数は、コンパイルされたデータベースでブレーク処理を生成することができない。

1つ多いレベルでソートしなければならないため、レコードが多いと、処理に時間がかかる。

BREAK LEVELコマンドと**ACCUMULATE**コマンドによるブレーク処理の生成方法は、レポートの作成にメソッドを使用する場合に有効です。この方法を使用したレポートの印刷プロセスは、一般的に以下のようになります。

1. 印刷するレコードを選択する。
2. **ORDER BY**コマンドでレコードをソートする。少なくともブレークと同数のレベルでソートする。
3. **BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを実行する。
4. **PRINT SELECTION**コマンドでレポートを印刷する。

コンパイルモードでブレーク処理を生成するには、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用しなければなりません。ただし、**Subtotal**関数は、フォーム上の値を表示するために必要です。

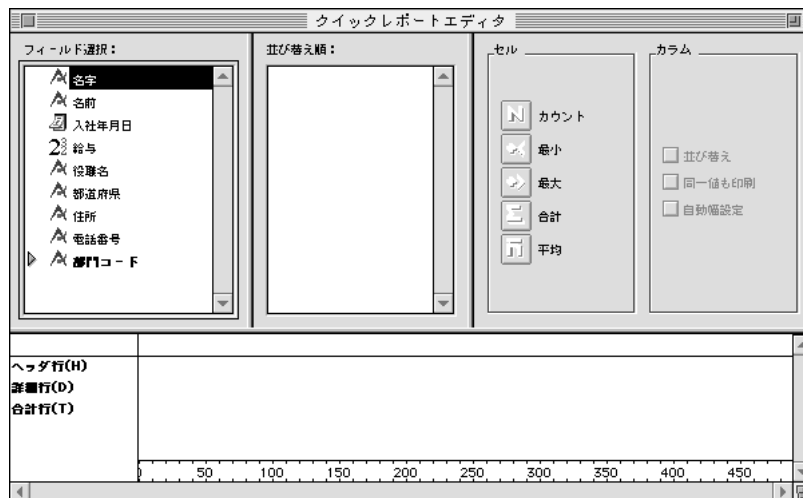
REPORT

REPORT ({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	印刷するテーブル 省略した場合、デフォルトテーブル
ドキュメント	文字列	クイックレポートのドキュメント
*	*	プリンタダイアログボックスの表示取り消し

説明

REPORTコマンドは、下図に示した「クイックレポート」エディタで作成した<テーブル>に対するレポートを印刷します。



<ドキュメント>は、「クイックレポート」エディタで作成され、ディスクに保存されたレポートドキュメントファイルです。「クイックレポート」エディタ内の「ファイル」メニューから「保存」または「新規保存」を選択してレポートドキュメントを保存します。これはレポートの形式を格納するだけで、印刷されるレコードを格納するわけではありません。

<ドキュメント>に対して空の文字列("")を指定した場合に、**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。レポートが選択されると「用紙設定」ダイアログボックスが表示されます。オプション引数にアスタリスク(*)を指定すると、プリンタダイアログボックスは表示されません。レポートは、その後で印刷されます。

<ドキュメント>に存在しないドキュメント名を指定した場合は、「クイックレポート」エディタが表示されます。

ユーザは、「クイックレポート」エディタでカスタム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ると、メニューバーは「ユーザ」モードのエディタと同じ「ファイル」、「編集」、「フォント」、「書体」の4つのメニューを表示します。「クイックレポート」エディタを使用したレポートの作成に関する詳細は、『4th Dimension ユーザリファレンス』を参照してください。

「クイックレポート」エディタを使用しない場合、レポートが印刷されるとシステム変数 OKに1がセットされます。印刷されない場合（ユーザが「用紙設定」ダイアログボックスでキャンセルをクリックした場合等）には0がセットされます。

例題

1. 以下の例は、ユーザが[従業員]テーブルを検索し、レポート“明細一覧”を自動的にプリントします。

```
QUERY ([従業員])
If (OK=1)
  REPORT ([従業員];"明細一覧";*)
End if
```

2. 以下の例は、ユーザは[従業員]テーブルを検索した後、プリントするレポートを選択します。

```
QUERY ([従業員])
If (OK=1)
  REPORT ([従業員];"")
End if
```

3. 以下の例は、[従業員]テーブルを検索した後、レポートエディタが表示されるため、ユーザは任意のレポートの設計、保存、ロード、プリントを行えます。

```
QUERY ([従業員])
If (OK=1)
  REPORT ([従業員];Char(1))
End if
```

参照
なし

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
ラベルドキュメント	文字列	省略した場合、デフォルトテーブル ディスクラベルドキュメントの名前
*	*	プリンタダイアログボックスの表示取り消し

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。

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 LABEL ([住所]; Char (1))
End if
```

参照
なし

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
*	*	指定した場合、プリントダイアログ ボックスの表示取り消し

説明

PRINT SELECTIONコマンドは、テーブルのカレントセ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...」メニューコマンドと同じ動作を実行します。セ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。アスタリスク（*）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。
- ・ブレイ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、小計やブレイク付きでソートしたセクションを印刷するには、まずそのセクションをソートしなければなりません。次に、レポートの各ブレイクエリアに、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**コマンド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）。

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
なし

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...」メニューコマンド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

なし

BREAK LEVEL

BREAK LEVEL (レベル {; ページブレイク})

引数	タイプ	説明
レベル	数値	ブレイクレベルの数
ページブレイク	数値	ページブレイクを行うブレイクの数

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イクの数指定します。

警告：コンパイルモードでは、ブレイ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イ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、ブレイ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。ブレイ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、ブレイクに対しての意味は持ちません。

生成される各ブレイクレベルは、フォーム中の対応するブレイクエリアやヘッダエリアを印刷します。フォーム中のブレイクエリアは、<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イ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<ページブレイク> は、印刷中にページブレイクを発生させるために使用します。

以下の例は、2つのブレイクレベルを持つレポートを印刷します。このセクションは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イクを指定します。[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イク処理を行う

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

なし

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、ブレイ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イ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イ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

なし

Subtotal

Subtotal (データ {; ページブレイク}) 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
ページブレイク	数値	ページブレイクを発生させるブレイクレベル
戻り値	数値	データの合計

説明

Subtotal関数は、現在または最後のブレイクレベルに対するデータの合計を返します。**Subtotal**関数は、**PRINT SELECTION**コマンドでソートされたセクションを印刷する場合と、「ユーザ」モードで「プリント...」メニューコマンド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イクエリアの変数に、**Subtotal**関数の結果を代入します。

警告：コンパイルモードでは、ブレイク処理を行ない、合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イ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イクを行います。＜ページブレイク＞が0の場合は、**Subtotal**関数はページブレイクを行いません。＜ページブレイク＞が1の場合には**Subtotal**関数は各レベル1のブレイクに対してページブレイクを行います。＜ページブレイク＞が2の場合は、**Subtotal**関数はレベル1と2の各ブレイクレベルに対してページブレイクを行います。

ソートレベルnに対してブレイクを行いたい場合、n+1のレベルでカレントセ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、予期しないブレイ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イクを行う場合は、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、フォームのブレイクエリア（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、ブレイクエリアにあります。このブレイ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
なし

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	現在のブレイクまたはヘッダのレベル

説明
Level関数は、カレントブレイ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イ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イ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。ヘッダやブレイ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

\$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->))

　　` 最初のブレイクヘッダ用のコード

　　¥ (Level = 1)

　　` レベル1のブレイクヘッダ用のコード

　　¥ (Level = 2)

　　` レベル1のブレイ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)

　　` ブレイクエリア印刷開始

Case of

　　¥ (Level = 0)

　　` ブレイクレベル0のコード

　　¥ (Level = 1)

　　` ブレイクレベル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

なし

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
		省略した場合、デフォルトテーブル
*	*	プリンタダイアログボックスの表示取り消し

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。これは、**PRINT FORM** コマンドではできない機能です。

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。

オプションの引数<*>を指定すると、プリンタダイアログボックスの表示は行われません。この場合、**PRINT RECORD**コマンドを実行する前に**PAGE SETUP**を呼び出さないかぎり、デフォルトの用紙設定を用いてレコードがプリントされます。

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")

参照
なし

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**コマンドを呼び出した場合

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

なし

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 コマンドの前に**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

なし

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、スクリーンヘプリント)」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 (スクリーンヘプリント)」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 (スクリーンヘプリント)」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

なし

PRINT FORM

PRINT FORM ({テーブル;} フォーム)

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
フォーム	文字列	印刷するフォーム

説明

PRINT FORM コマンドは、フィールドや変数の現在の値を<フォーム>に印刷します。このコマンドは、フォームのディテイルエリア（ヘッダ行とディテイル行の間のエリア）だけを印刷します。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** コマンドはレコード処理、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、すべてデザイナーが行います。**PRINT FORM** コマンド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 コマンドは、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** コマンドは、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。

PRINT FORM コマンドを使用する場合、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** コマンド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 コマンドは、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** コマンド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** コマンド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コマンドにより、フォームメソッドのOn Printing Detailイベントが発生します。

以下の例は、単純な**PRINT SELECTION**コマンドをシミュレートします。ブレーク処理はありません。このレポートは、小切手レジスタに対するものです。

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
```

参照
なし

PAGE BREAK

PAGE BREAK {(* | >)}

引数	タイプ	説明
* >		* : で開始した印刷ジョブをキャンセル > : 1つのプリントジョブを強制する

説明

PAGE BREAKコマンドは、プリンタへ送信したページを印刷し、排出します。**PAGE BREAK**コマンドは、**PRINT FORM**コマンドとともに使用し、ページブレークを強制し、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**コマンド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。

参照

前述の**PRINT FORM**コマンドの例を参照してください。

この章では、「ルーチン」エディタの「Process（Communications）」テーマ内にあるプロセス（通信）コマンドについて説明します。

CALL PROCESS

CLEAR SEMAPHORE

GET PROCESS VARIABLE

Semaphore

SET PROCESS VARIABLE

Test semaphore

VARIABLE TO VARIABLE

Semaphore

Semaphore (セマフォ) 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
tickCount	整数	最大待ち時間 (tick関数)
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。書く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**tickCount**はセマフォが既にセットされていると待ち時間 (**in ticks**) を示します。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。

```

C_TEXT ($1)                             ` コマンドに渡されるパラメータ
If (Not (Semaphore ("AccessToDoList";300)))
    ` 既にセフォマが終了していたら5秒待つ
    $vIElem:=Size of array (<>ToDoList)+1
    INSERT ELEMENT (<>ToDoList ; $vIElem)
    <>ToDoList{$vIElem}:=$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。

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。

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送信先プロセス (ストアードプロシージャ) 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。

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、ストアー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いけません。

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りません。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\$viProcess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 ($viProcess ; 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 ($viProcess ; 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 ($viProcess ; 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、\$vIProcess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 ($vIProcess ; vl_IPCom_Array ; $vISize)
For ($vIElem ; 1 ; $vISize)
  GET PROCESS VARIABLE ($vIProcess ; at_IPCom_Array{$vIElem} ; $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 ($vIProcess;at_IPCom_Array;$anArray)
For ($vIElem ; 1 ; Size of array ($anArray))
  ` $anArray{$vIElem}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 ($vIProcess ; 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 (ストアードプロシージャ) 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。

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、ストアー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せん。

ポインタ配列の要素または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 \$vIProcess 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 ($vIProcess ; vtCurStatus ; "")
```

2. 以下のコードは、番号が \$vIProcess 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 ($vIProcess ; vtCurStatus ; $vtInfo)
```

3. 以下のコードは、番号が \$vIProcess 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 ($vIProcess ; 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、\$vIProcess 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 ($vIProcess ; vI_IPCom_Array ; $vISize)
```

```
For ($vIElem ; 1 ; $vISize)
```

```
  GET PROCESS VARIABLE ($vIProcess ; at_IPCom_Array{$vIElem} ; $vtElem)
```

```
  SET PROCESS VARIABLE ($vIProcess ; at_IPCom_Array{$vIElem} ; 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。

＜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りません。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 “ \$vIProcess ” 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 ($vIProcess ; at_IPCom_Array ; $anArray)
For ($vIElem ; 1 ; Size of array ($anArray))
    $anArray{$vIElem}:=Uppercase ($anArray{$vIElem})
End for
VARIABLE TO VARIABLE ($vIProcess ; at_IPCom_Array ; $anArray)
```

参照

GET PROCESS VARIABLE、SET PROCESS VARIABLE

この章では、「ルーチン」エディタの「Process (User Interface)」テーマ内にあるプロセス (ユーザインタフェース) コマンドについて説明します。この章のコマンドは、プロセスのユーザインタフェース要素に影響します。プロセスとそのインタフェース要素は、**HIDE PROCESS**コマンドや**SHOW PROCESS**コマンドによって、表示または非表示になります。メッセージプロセスのようなプロセスが必要な場合は、**BRING TO FRONT**コマンドを使用して、プロセスを前面に配置することができます。

BRING TO FRONT
Frontmost process

HIDE PROCESS
SHOW PROCESS

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。

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。

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。

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。引数アスタリスク(*)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

この章では、「ルーチン」エディタの「Process」テーマ内にあるプロセスコマンドについて説明します。この章のコマンドは、プロセスを管理します。

New process	Process number
Execute on server	Count users
DELAY PROCESS	Count tasks
PAUSE PROCESS	Count user processes
RESUME PROCESS	EXECUTE ON CLIENT
Process aborted	REGISTER CLIENT
Current process	UNREGISTER CLIENT
Process state	GET REGISTERED CLIENTS
PROCESS PROPERTIES	

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32000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200000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、ひと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、カスタムメニューアイテム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アイテムと同じ作業を行います。

メニューアイテムを選択したり、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になります。

` “b顧客追加” ボタンのオブジェクトメソッド

START ADD CUSTOMERS

「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、メニューアイテムの「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	数値	新しく作成されるプロセスのプロセス番号 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ードプロシージャを開始します。ストアー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32000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200000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、ひと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、ストアー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を渡してストアー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ストアードプロシージャを開始します。

```
` 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ー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は必要になる
  CLEAR VARIABLE ($vxData)
  `ストアードプロシージャにより実行される処理の終了を待機
  Repeat
    DELAY PROCESS (Current process;300)
    GET PROCESS VARIABLE ($spProcessID;spErrCode;spErrCode)
    If (Undefined (spErrCode))
      `注：ストアー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ー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、ストアー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。

以下の例では、プロセスの開始を真夜中まで遅延してその後レポートを印刷します。これによって、昼間の貴重な時間を節約することができます。

```
<>レポート:=New process ("ReportProc"; 32000 ; "レポート")
```

“ReportProc” メソッドは、真夜中までのtickの数を計算してその分プロセスを遅延します。

```
` 真夜中までのtick数
$真夜中:=!!24:00:00!! - Current time * 60
DELAY PROCESS (Current process ; $真夜中)
レポート印刷 ` レポートを印刷するメソッド
```

以下の例では、[顧客]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ています。ロックされていればプロセスが1秒遅延されます。この技法はマルチユーザ環境およびマルチプロセス環境で 사용할 ことができます。

```
` 特定のレコードを検索
SEARCH ([顧客] ; [顧客]名前="山田")
If (Records in selection ([顧客]) > 0) ` 少なくとも1レコードが見つかった場合
  While (Locked ([顧客])) ` カレントレコードがロックされている場合
    MESSAGE ("そのレコードはロックされています") ` メッセージを表示
    DELAY PROCESS (Current process ; 60) ` 1秒待ってレコードを
    LOAD RECORD ([顧客]) ` 再度ロードする
  End while
  ` カレントレコードがロック解除の状態
  MODIFY RECORD ([顧客]) ` ユーザにレコードを変更させる
  UNLOAD RECORD ([顧客]) ` レコードロックを解除する
Else
  ALERT ("検索条件と一致するレコードはありません")
End if
```

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せん。**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。

以下の例は、文字フィールドとテキストフィールド内の入力項目を複製するためのチェックです。

「On startup」データベースメソッドは、“P_Duplicates”メソッドの起動中に「Duplicate Finder」プロセスを開始します。

C_POINTER (<>vフィールド)

C_STRING (80 ; <>v値)

C_LONGINT (<>v複製 ; <>vステータス)

<>v複製 := **New process** ("P_Duplicates" ; 32000 ; "Duplicate Finder")

下記に “ P_Duplicates ” メソッドを示します。

```
`複製項目を検索し、プロセスを停止する
`文字とテキストでのみ機能する
```

C_POINTER (\$theTable)

HIDE PROCESS (**Current process**) `プロセスを隠す

While (**True**) `ループを初期化する

```
` “ CheckDups ” メソッドがコールされるまでプロセスを停止する
```

PAUSE PROCESS (**Current process**)

```
$theFile := Table (Table (<>theField)) `フィールドポインタのテーブルポインタ
```

QUERY (\$theTable-> ; <>theField-> = <>v値) `複製項目の検索

```
` “ CheckDup ” にステータスを返す
```

```
<>vステータス := Records in selection ($theTable->))
```

End while

以下のフォーム内にあるオブジェクトメソッドは、“ CheckDups ” メソッドをアクティブにします。

CheckDups (**Self**)

下記は、“ CheckDups ” メソッドのコードです。

C_POINTER (\$1)

```
<>vステータス := -1 `vステータスのデフォルトのステータス
```

```
<>theField := $1
```

```
<>v値 := $1->
```

RESUME PROCESS (<>v複製)

Repeat

IDLE

```
Until (<>vステータス#0) ` “ Duplicate Finder ” プロセスが終了するまで待機する
```

```
If (<>vステータス#0)
```

```
ALERT ($1-> + "複製項目です！") `ユーザに警告する
```

```
$1-> := "" `前回の入力項目をクリアする
```

```
GOTO AREA ($1->) `そのフィールドにカーソルを戻す
```

End if

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。

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コマンドは、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 ATTRIBUTES**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。

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 ; $タスク)
  If (Process state >= Executing)          ` プロセスが中止されていないならば
    vCount:=vCount + 1                    ` 要素数をカウントする
    PROCESS ATTRIBUTES (arProcNum{$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

2つの新しいオプションの引数、ユニークIDおよび派生元がこのコマンドに追加されました。

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コマンド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 License	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 License	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

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 ;vTimeSpent)
```

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	数値	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ィ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」モードの最前面ウィ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ィ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 (最前面ウィンドウに貼り付けられるテキスト)
- C_TEXT (\$1)**
- C_LONGINT (\$vDesignPID ; \$vCount)**
- \$vDesignPID:= Design process number*
- If (\$vDesignPID # 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」イベントをボスとする
- POST KEY (Ascii ("v") ; Command key mask ; \$vDesignPID)**
- ・**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 (\$vCount ; 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のストアー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。

"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ポートが可能です。

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。

一旦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、コマンドは何も行わずOKは0になります。

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 PROCES(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。

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。

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 LLIENT

この章では、「ルーチン」エディタの「Queries」テーマ内にあるクエリ（検索）と並べ替え（ソート）コマンドについて説明します。4th Dimensionのプログラミング言語には、レコードを検索するための多くのコマンドが用意されています。これらのコマンドは基本的には同じです。つまり、テーブル上のレコードを検索し、検索条件に適合するレコードを見つけ出すという機能です。各コマンドは、実行を完了した時点で見つけ出したレコードの集合（以降では、この集合のことを“セレクション”と呼びます）を生成します。また、レコードをソートするためにいくつかのコマンドがあります。ソートコマンドは、指定されたソートキーに従ってレコードを並べ変えます。

QUERY BY EXAMPLE	SET QUERY DESTINATION
QUERY	SET QUERY LIMIT
QUERY SELECTION	Find index key
QUERY BY FORMULA	ORDER BY
QUERY SELECTION BY FORMULA	ORDER BY FORMULA
QUERY WITH ARRAYS	QUERY WITH ARRAY

検索

検索する条件には、単一条件の場合も、複合条件の場合もあります。また“従業員番号57を検索する”等のように単一のレコードを探し出す場合や、“ニューヨークにあるすべての企業を検索する”等のように複数のレコードを探し出す場合もあります。

自動リレートを持ったテーブルのレコードを検索する場合には、リレート先テーブルのフィールドを使用することもできます。テーブルの自動リレートの定義に関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

インデックスを使用して検索すると極めて高速に処理することができます。特に大量のレコードを検索する場合に有用です。

できる限り**QUERY**コマンドや**QUERY SELECTION**コマンドを使用することをお勧めします。これらのコマンドは、「ユーザ」モードにおける「クエリ」エディタと同じ検索方法を使用します。つまり最初にインデックスを使用してから、必要な検索を順次実行する、という検索処理の最適化を自動的に行います（以降では、このように自動的に最適化される検索方法を“インテリジェントな検索”と呼びます）。

QUERY BY FORMULAコマンドと**QUERY SELECTION BY FORMULA**コマンドは、非常に強力で柔軟性のある検索コマンドです。これらは、**QUERY**コマンド、**QUERY SELECTION**コマンドのような検索条件の制約がありません。フィールド内のサブストリングに対する検索や、計算結果に基づいた検索等、より高度な検索が実行することができます。ただし、これらのコマンドの実行する検索処理は、常にシーケンシャルな検索のためインデックスを使用した検索よりも遅くなります。

QUERY SUBRECORDSコマンドは、ひとつのレコードのサブレコード内で検索します。このコマンドはサブレコードに対してのみ有効で、レコードに対しては何も行いません。サブレコードに対して検索可能なのはこのコマンドだけです。他の検索コマンドは、すべてのレコードを検索します。

検索処理の実行中に検索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。これらのインジケータを表示したくない場合は、**MESSAGES OFF**コマンドを使用してください。インジケータの「中止」ボタンや検索ウインドウの「キャンセル」ボタンをクリックすると検索を中止します。中止されたかどうかは、システム変数OKの値を調べるとわかります。検索が完了していれば1が、中止されていれば0が代入されています。検索コマンドで検索した結果を表示する場合には、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドを使用します。

ソート

ソートは、データベースの処理の中でも重要な操作のひとつです。セレクションのソートに関するコマンドは、レコードの選択が表示されたり、印刷される前によく使用されます。ソートコマンドは、“あ”から“ん”への昇順ソートや、逆に“ん”から“あ”への降順ソートを行います。また、複数のレベルでソートすることもできます。**ORDER BY FORMULA**コマンドは、計算値を基にしたソートを実行することができます。また、ブレイクレベルを作成するためにレコードをソートする必要があるかもしれません。

コマンドによるソートは、一時的にレコードをソートするだけです。恒久的にソートしたい場合は、4D Toolsを使用します。

警告：テキストフィールドをソートすることはできません。

テキストフィールドをソートしたい場合、以下のようなコードを使用することにより、そのフィールドの先頭の80バイトをソートすることができます。

ORDER BY FORMULA ([テーブル]; **Substring** ([テーブル]テキストフィールド ; 1 ; 80))

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」メニューコマンド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」メニューコマンド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。

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]テーブルの名前が“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]名前 = "川@")

参照

なし

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」エディタで利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のアスタリスク (*) 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**コマンドで引数アスタリスク (*) 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が...
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が...
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）のセクションを返します。サブレコードのセ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番号と一致するレコードを検索します。

v番号:=Request ("送り状番号を入力してください。")` 送り状番号を入力する

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]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>のカレントセクションを変更し、セ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)やファーマー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> のカレントセクションを変更し、セ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

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。

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

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。

アプリケーションでは、[電話帳]テーブルは一連の固定データを持っているため、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、貴重なデータベースエンジン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現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 (\$vElem ; 1 ; 26)

asRolodex{\$vElem}:=**Char** (64+\$vElem)

abQueryDone{\$vElem}:=**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、作成した命名セクションを消去する。

For (\$vElem ; 1 ; 26)

If (abQueryDone{\$vElem})

CLEAR NAMED SELECTION ("Rolodex"+asRolodex{\$vElem})

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 ($vIField ; $vINbFields ; $vIFound ; $vICurrentRecord)
$vINbFields:=Count parameters-1
$vICurrentRecord:=Record number ($1->)
If ($vINbFields>0)
  If ($vICurrentRecord#-1)
    If ($vICurrentRecord<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 ; $vIFound)
    `指定されたフィールド番号によってクエリを作成する。
    Case of
      ¥ ($vINbFields=1)
        QUERY ($1-> ; $2->=$2->)
      ¥ ($vINbFields=2)
        QUERY ($1-> ; $2->=$2-> ; *)
        QUERY ($1-> ; & ; $3->=$3->)
    Else
      QUERY ($1-> ; $2->=$2-> ; *)
      For ($vIField;2;$vINbFields-1)
        QUERY ($1-> ; & ; $ {1+$vIField}->=$ {1+$vIField}->;*)
      End for
      QUERY ($1-> ; & ; $ {1+$vINbFields}->=$ {1+$vINbFields}->)
    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

¥ (\$vIFound=0)

\$0:=True `重複した値がない場合

¥ (\$vIFound=1)

If (\$vICurrentRecord<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

¥ (\$vIFound=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 値	フィールド	検索を実行するインデックス フィールド 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なります。

このコマンドは、カレントセ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。

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

参照
なし

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; <)

7. 以下の例は、[住所]テーブルのカレントセクションを“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、オプションの*パラメータを指定することができます。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のソート管理に役立ちます。

警告：**ORDER BY**のコールごとに1つのソートだけにレベル（フィールド）を渡すことができます。



タイトル	カテゴリ	ミュージシャン	フォーマット
Best of B.B.King	ブルース	B.B.King	LP
Gettin' Ready	ソウル	Temptations	CD
Fahrenheit	ロック	Toto	CD
Lucille and Other Classics by	クラシック	Kenny Rogers	CD
Talk	ロック	Yes	CD
Season for Love	クラシック	London Sympho	CD

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 (\$!LevelNb)

`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

 \$!LevelNb:=**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 (\$!LevelNb<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

\$!LevelNb:=**Size of array** (aPtrSortField)

If (\$!LevelNb>0) `少なくとも1つのソート条件があれば

For (\$i;1;\$!LevelNb)

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クションをソートします。ソートが終了すると、セ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

QUERY WITH ARRAY QUERY WITH ARRAY

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	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	検索された値の配列

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要素の内容と等しい、すべてのレコードを検索し、新しいカレントセレクションを作成します。

このコマンドは、速やかにかつ簡単に複数の値に対する検索を可能にします。

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のあることを覚えておいてください。

下記の例は、フランスとアメリカの顧客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)

参照

なし

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、ロードされてい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。

READ WRITE

READ ONLY

Locked

Read only state

LOAD RECORD

UNLOAD RECORD

LOCKED ATTRIBUTES

レコードのロック

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」メニューアイテムを選択する、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]; \$プ 吨&番号; \$ユーザ; \$マシン; \$プロセス名)

If (\$プ 吨&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、特定の処理を行います。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 - ザ」モ - ドでは、レコ - 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。

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が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 - 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。

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 - 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 - 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 - 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 - 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。

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	カレントレコードのロックを調べるテーブル 省略した場合、デフォルトテーブル
戻り値	ブール	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を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、**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

この章では、「ルーチン」エディタの「Records」テーマ内にあるレコードコマンドについて説明します。この章のコマンドは、レコードを管理するためのものです。新しいレコードを作成、追加、または既存のレコードに対し修正、複製、削除等を実行するためのコマンドです。

これらのコマンドは、ユーザには見えない形でデータを管理するためのものです。したがって、データやフォームは表示しません。

CREATE RECORD	DUPLICATE RECORD	Records in table
GOTO RECORD	Modified record	SAVE RECORD
Record number	DELETE RECORD	DISPLAY RECORD
Sequence number	POP RECORD	PUSH RECORD
Is new record	Is record loaded	

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サーバーマシン上で実行されるためです。

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、さらにカレントセクション（レコードが1件のみのカレントセクション）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。

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**コマンドを最初に行う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。

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。

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。

“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
```

参照
なし

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。

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ます。

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。

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 - ド番号> がデ - タベ - スの中で最も小さいレコ - ド番号よりも小さい場合や、デ - タベ - スの中で最も大きいレコ - 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 - ドが削除されたレコ - ドのレコ - 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]一連番号フィールド:= **Sequence number** ([テーブル]) + 999

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クションに依存します。セクションを変更、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」メニューコマンドを選択した後や**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、“ Sam ” のレコードが削除されたセ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、“ Liz ” という新しいレコードが追加されたセクションの状態を示しています。

新しいレコードは、カレントセクションの最後に加えられます。

“ Sam ” のレコード番号が新しいレコード (“ Liz ”) 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ック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。

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クションのレコードにした場合には、**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。

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 - ドをプッシュする
SEARCH ([顧客])	` いくつかのレコ - 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

この章では、「ルーチン」エディタの「Relations」テーマ内にあるリレートコマンドについて説明します。

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。

AUTOMATIC RELATIONS	RELATE ONE
CREATE RELATED ONE	RELATE MANY SELECTION
OLD RELATED MANY	RELATE ONE SELECTION
OLD RELATED ONE	SAVE OLD RELATED ONE
RELATE MANY	SAVE RELATED ONE

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

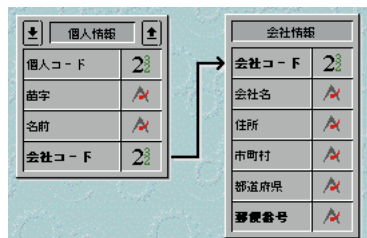
セクションに対する検索、ソート、フォーミュラでの更新

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、新しいレコードセクションがそのテーブルに作成され、セ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、会社のID番号があります。

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	コマンド	1対n
ADD RECORD		ORDER BY	×
ADD SUBRECORD	×	ORDER BY FORMULA	×
APPLY TO SELECTION	×	PRINT LABEL	×
DISPLAY SELECTION	×	PRINT SELECTION	
EXPORT DIF	×	QUERY	
EXPORT SYLK	×	QUERY BY FORMULA	
EXPORT TEXT	×	QUERY SELECTION	
MODIFY RECORD		REPORT	×
MODIFY SUBRECORD	×	SELECTION TO ARRAY	×
MODIFY SELECTION	(データ入力時)	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 - 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。

以下の表にカレントレコードをロードするコマンドを掲げます。これらは、関連するレコードを自動的に選択するものではありません。

ALL RECORDS	IMPORT DIF	QUERY BY FORMULA
CREATE RECORD	IMPORT SYLK	QUERY BY EXAMPLE
CREATE SUBRECORD	IMPORT TEXT	QUERY SELECTION
CUT NAMED SELECTION	LAST RECORD	QUERY SELECTION
DELETE RECORD	LAST SUBRECORD	BY FORMULA
DELETE SELECTION	NEXT RECORD	QUERY SUBRECORDS
DELETE SUBRECORD	NEXT SUBRECORD	ORDER BY FORMULA
FIRST RECORD	ONE RECORD SELECT	ORDER BY
FIRST SUBRECORD	POP RECORD	ORDER SUBRECORDS BY
GOTO RECORD	PREVIOUS RECORD	SCAN INDEX
GOTO SELECTED	PREVIOUS SUBRECORD	USE NAMED SELECTION
RECORD	QUERY	USE SET

AUTOMATIC RELATIONS

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

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**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。

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。

AUTOMATIC RELATIONS (True ; False)

参照

リレート、SELECTION TO ARRAY、SUBSELECTION TO ARRAY

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;	テーブル;	すべての自動リレーを設定するテーブル、 または
nフィールド	フィールド	1テーブルにマニュアルリレーする 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のリレーを行い、各リレー先テーブルのリレーレコードを選択します。これは、そのプロセスのリレー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。

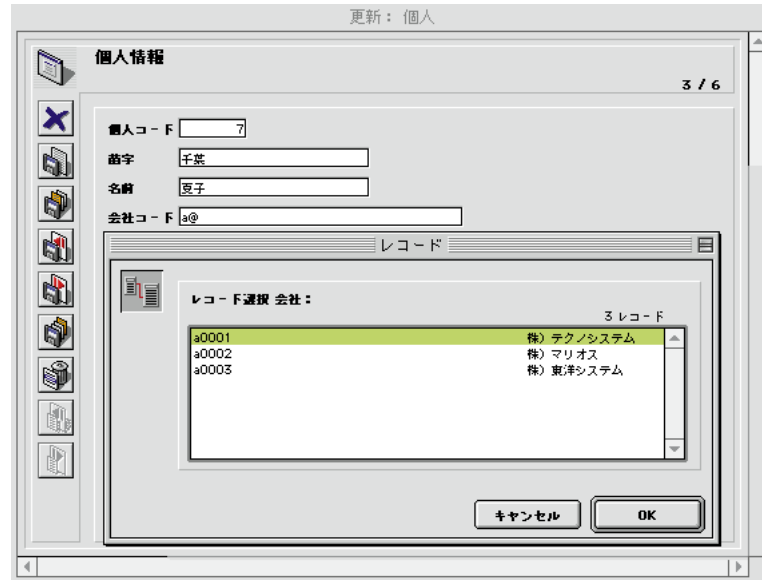
<nフィールド>の内容がワイルドカード記号(@)で終了する場合、複数のレコードが見つかる可能性があります。一致するレコードが1件しかなければ、選択リストは表示されません。<選択フィールド>を指定することは、テーブルのリレーを設定する時点でワイルドカード選択を指定するのと同じことです。ワイルドカード選択に関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

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。

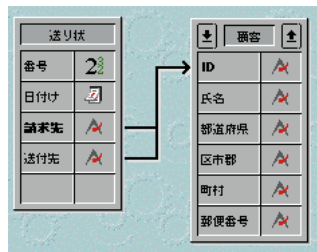


以下のコマンドを実行すると選択リストを表示します。

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。

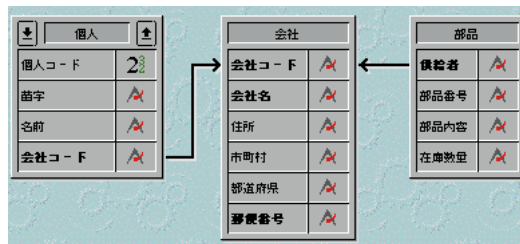
第1の形式 “**RELATE MANY (1テーブル)**” は、<1テーブル> に対してすべての1対nの自動リレートを設定します。このコマンドは、<1テーブル> に対して1対nの自動リレートを持つ各テーブルのカレントセレクションを更新します。つまり、nテーブルのカレントセレクションは、1テーブルのそれぞれのリレート先フィールドの現在値を反映します。このコマンドが実行される時はいつでも、nテーブルのカレントセレクションが再構成されます。

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

修正されたデータの管理

レコードがロードされた時点で、4th Dimensionはロードされたレコードのコピーを作成します。レコードが修正された時点で、コピーも修正します。このコピーは、保存の指示があるまでディスクには書き込まれません。コピーが保存されるまでは、この節のコマンドを使用して“更新前データ”(修正前の状態のデータ)をアクセスすることができます。

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

SAVE OLD RELATED ONE

SAVE 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

SAVE 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実行する以外は、**SAVE RELATED ONE**コマンドと同じ方法で処理を行います。**SAVE OLD RELATED ONE**コマンドを使用する前に、**OLD RELATED ONE**コマンドでレコードをロードしておく必要があります。**OLD RELATED ONE**コマンドでロードしたレコードを修正した場合にも、**SAVE OLD RELATED ONE**コマンドを使用して保存する必要があります。

SAVE OLD RELATED ONEコマンドは、ロックされたレコードを保存しません。このコマンドを使用した場合は、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

Locked、OLD RELATED ONE、トリガ

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> のレコードセクションを基にして、<1テーブル> 用のレコードセ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り使用できます。**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。

警告：このコマンドは、トランザクションの内部で使用してはいけません。

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クションを基にして、nテーブルのレコードセ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：このコマンドは、トランザクションの内部で使用してはいけません。

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。

4D Transporter

リソースフォークはWindows上には存在しないため、(4th DimensionのMacintosh版で提供される) 4D Transporterユーティリティツールを使って、4DデータベースをWindowsからMacintosh (または、MacintoshからWindows) にトランスポートすることができます。

WindowsからMacintoshに4Dデータベースをトランスポートすると、データベースストラクチャファイルの「.4DB」と「.RSR」が1つのMacintoshファイルに“マージ”されます。「.4DB」ファイルが、Macintoshストラクチャファイルのデータフォークになります。一方、「.RSR」ファイルが、Macintoshストラクチャファイルのリソースフォークになります。

逆に、MacintoshからWindowsに4Dデータベースをトランスポートすると、Macintoshストラクチャファイルは2つのファイルに“分割”されます。Macintoshのデータフォークが「.4DB」ファイル、Macintoshのリソースフォークが「.RSR」ファイルになります。

ファイルのデータフォークおよびリソースフォークを分割またはマージする機能こそが、4D Transporterツールの主要目的です。4D Transporterは、フォークおよびファイルの中に保持されている実際のデータを翻訳、あるいは修正することはできません。プラットフォーム間の4Dデータベースのトランスポートに関する詳細は、『4D Transporterリファレンスガイド』マニュアルを参照してください。

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、ACI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 Draw6.0」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くことができます。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「ASIPOINT.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。

```
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。

```
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）で利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。

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 ; $vhResFileA)
        ` 各リソースファイルに対してループ
        For ($viElem ; 1 ; Size of array ($aiResID))
            $viResID:=$aiResID{$viElem}
            ` 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 ; $asResName{$viElem}
                    ; $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。
                    $viResAttr:=Get resource properties ("PICT" ; $viResID ; $vhResFileA)
                    SET RESOURCE PROPERTIES ("PICT" ; $viResID ; $viResAttr
                    ; $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ページ（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ペ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 PROPERTIES**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)

 \$viOffset:=0

ALERT (BLOB to text (vxData ; Text without length ; \$viOffset ; **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。



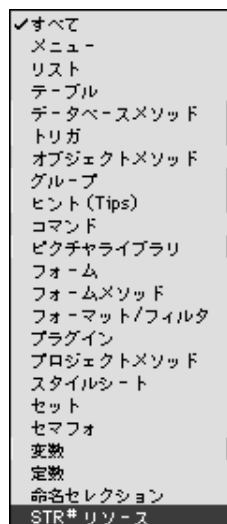
2. この時点で、このメニューバーが“STR#”リソースを参照するように変更できます。これを実行するには、4D Insiderの「ツール」メニューから「テキストからSTR#...」を選択します。

ツール	
テキストからSTR#...	
STR#からテキスト...	
STR#編集...	⌘M
検索...	
置換...	⌘R
セレクション中を置換...	⌘T
内容中を置換...	
4Dコマンドを置換...	
接頭辞を追加...	⌘V
新規グループ...	
セレクションをグループ化...	⌘G
使用オブジェクトでグループ化...	
グループ解除	
一般ドキュメント...	
ドキュメントを編集...	⌘I
ドキュメントを削除	
再解析	
スケール設定...	

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ィンドウを閉じると、このリソースが作成されます。

4. ブラウザウィ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。

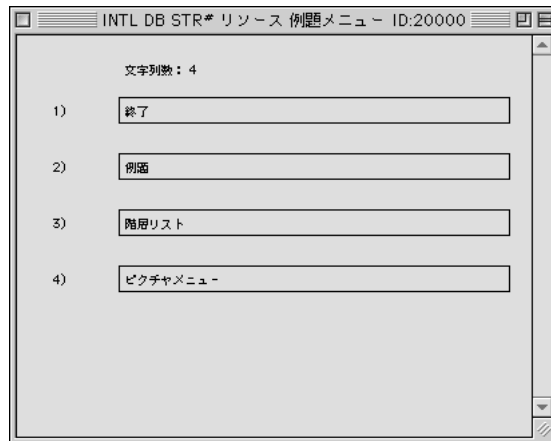


これらの文字列がリソースに保存され、データベース開発のロジックに影響を与えずに文字列の値を変更できるようになりました。

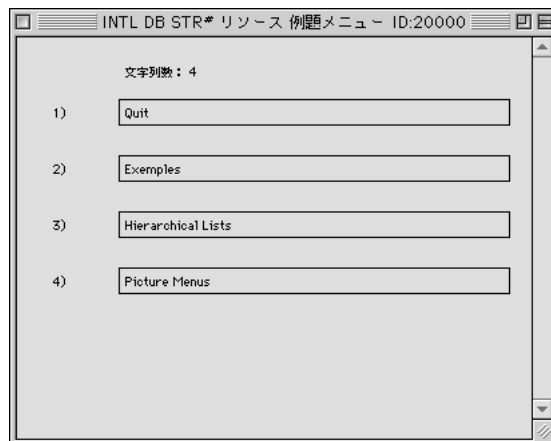
6. この値を変更するには、ブラウザウィンドウのメインリストで“例題メニュー”が選択されている状態で、4D Insiderの「ツール」メニューから「STR#編集」を選択します。



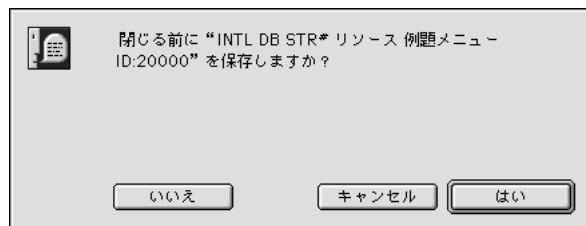
すると、STR#リソースの編集用ウインドウが表示されます。



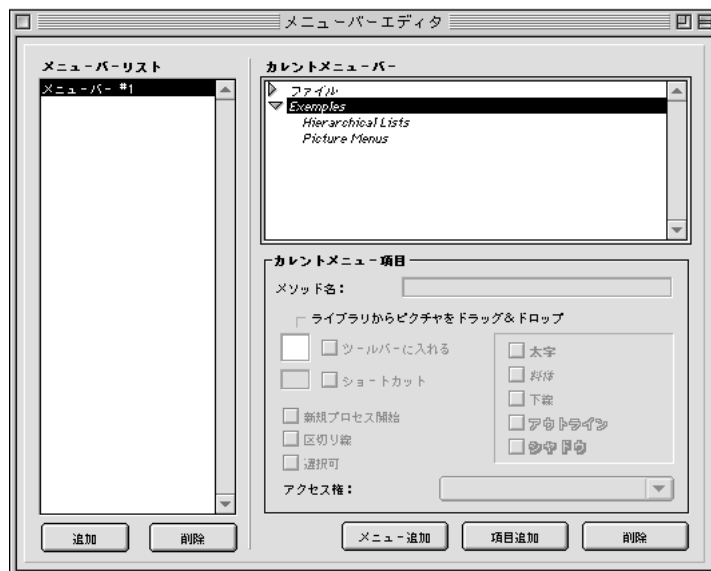
7. ここの文字列を他の言語に翻訳します。以下の図は、文字列が英語に翻訳されたところを表わしています。



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 リファレンスガイド』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	MacOS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 (13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が、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きますのを妨げません。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、そのリソースをあなた自身のファイルに格納します。

あなた自身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 использоваться す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 использоваться するためにあらかじめ予約されている番号です。あなた自身のリソースには、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	MacOS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。

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
影響を及ぼすものではありません。

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

RESOURCE 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。

RESOURCE 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が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MacOS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 (\$vElem ; 1 ; **Size of array** (\$atResType))

RESOURCE LIST (\$atResType{\$vElem} ; \$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

システム変数とシステムセット

影響を及ぼすものではありません。

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)
$Ref := Open window (50 ; 50 ; 550 ; 120 ; 5 ; "PICTリソースコピー中...")
For ($vIElem ; 1 ; Size of array ($alResID))
  GET PICTURE RESOURCE ($alResID{$vIElem} ; $vgPicture)
  If (OK=1)
    $vsName:=$atResName{$vIElem}
    If ($vsName="")
      $vsName:="PICTリソースID="+String ($alResID{$vIElem})
    End if
    ERASE WINDOW
    GOTO XY (2 ; 1)
    MESSAGE ("DBピクチャライブラリに"+$vsName+"を追加します。")
    SET PICTURE TO LIBRARY ($vgPicture ; $alResID{$vIElem} ; $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
影響を及ぼすものではありません。

STRING LIST TO ARRAY

STRING LIST TO ARRAY (リソースID ; スtring {; 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String	文字列配列	スString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String
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Stringリスト ("STR#") リソース内に格納されているスStringを引数<スString> 配列に割り当てます。

リソースが見つからない場合は、<スString> 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String> 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String>を作成します。

注：スStringリストリソースの各スStringは、最大255バイトの文字を使用することができます。

Tips：スString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Stringにします。

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 ; $vhResFile)
    If (OK=1)
        <>bFontsAreOK:=True
        For($vElem ; 1 ; Size of array ($atNewFont))
            If ($atNewFont{$vElem}# $atOldFont{$vElem}))
                $vElem:=MAXLONG
            <>bFontsAreOK:=False
        End if
    End for
    Else
        <>bFontsAreOK:=True
    End if
    CLOSE RESOURCE FILE ($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tringID {; 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
StringID	数値	String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String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Stringリスト ("STR#") リソースに格納されている1つのStringを返します。

引数<StringID>にStringの番号を渡します。StringリストリソースのStringは1からNの順に番号が振られます。StringリストリソースのすべてのString (およびそのString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String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：Stringリストリソースの各Stringは、最大255バイトの文字を使用することができます。

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。

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tring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。

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。

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。

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)

\$vINblcons:=**Size of array** (\$alResID)

ARRAY PICTURE (ag4DIcon ; \$vINblcons)

For (\$vIElem ; 1 ; \$vINblcons)

GET ICON RESOURCE (\$alResID{\$vIElem} ; ag4DIcon{\$vIElem} ; \$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：MacOS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)。

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：MacOS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、**DOCUMENT 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)
    $vIOffset:=0
    BLOB TO VARIABLE ($vxPrefData ; <>bAutoRepeat ; $vIOffset)
    BLOB TO VARIABLE ($vxPrefData ; <>lCurTable ; $vIOffset)
    BLOB TO VARIABLE ($vxPrefData ; <>sDfltOption ; $vIOffset)
    `...
    $vbDone:=False
  End if
  CLOSE RESOURCE FILE ($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

リソースを書き込んだ場合は、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。

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。

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。

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。

参照
なし

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ページ（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ペ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、ペ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ペ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、たとえページ可能であっても、ページされません。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ますようになります。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。

```
$vIResAttr:=Get resource properties ('STR#' ; 17000 ; $vhResFile)
```

```
SET RESOURCE PROPERTIES ('STR#' ; 17000 ; $vIResAttr ?+ 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 ; $1)
If (OK=1)
  For ($vIElem ; 1 ; Size of array ($aiResID))
    DELETE RESOURCE ($2 ; $aiResID{$vIElem} ; $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)
  $vIElem:=Find in array ($asResName ; $3)
  If ($vIElem>0)
    DELETE RESOURCE ($2 ; $aiResID{$vIElem} ; $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。

この章では、「ルーチン」エディタの「Selection」テーマ内にあるカレントセクションコマンドについて説明します。この章のコマンドは、画面上にレコードを表示するためのものです。リストにデータを入力、あるいは表示します。

ほとんどのコマンドは、カレントセクションの削除、修正、カレントセクション間の移動等、カレントセクションの操作を実行します。

ALL RECORDS	ONE RECORD SELECT
APPLY TO SELECTION	LAST RECORD
Before Selection	MODIFY SELECTION
DELETE SELECTION	NEXT RECORD
DISPLAY SELECTION	PREVIOUS RECORD
End selection	Records in selection
FIRST RECORD	REDUCE SELECTION
GOTO SELECTED RECORD	SCAN INDEX
HIGHLIGHT RECORDS	Selected record number

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>の全レコードを選択します。**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。

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選択
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。

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をすべて削除します。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])           `すべてのレコードを選択する
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使用

DELETE SELECTION ([従業員]) `レコードセクションを削除

End if

ALL RECORDS ([従業員]) `全レコードを選択

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ったと、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 "LockedSet" というセットに納められます。**DELETE SELECTION** コマンドの実行後、LockedSet 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])

USE SET ("LockedSet") `ロックされたレコードだけを選択

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。

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、セクションの作成方法およびセ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。

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

\$vINbRecords:=**Size of array** (a名字)

ARRAY LONGINT (a位置番号 ; \$vINbRecords)

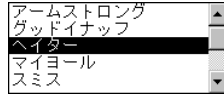
For (\$i ; 1 ; \$vINbRecords) ` 配列にレコード位置番号を取り込む

　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空の場合は、**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は確かではない場合、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。

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

NEXT RECORD

NEXT RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	以下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>のカレントセレクションにある以下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。

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クション内に戻すには、**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...」メニューコマンド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->))

` 最初のブレイ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...」メニューコマンド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} {; *} {; *})

引数	タイプ	説明
テーブル	テーブル	表示するテーブル
		省略した場合、デフォルトテーブル
*	*	レコードが1件の場合にも出力フォームを使用し、 入力フォームのスクロールバーを表示しない
*	*	入力フォームのスクロールバーを表示する (最初のオプション<*>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、そのレコードは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。

セクションを表示し、かつレコードをダブルクリックして修正を実行するには（「ユーザ」モードのウインドウでの作業と同じように） **DISPLAY SELECTION**コマンドではなく**MODIFY SELECTION**コマンドを使用します。

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。

以下の図に、**DISPLAY SELECTION**コマンドで表示された出力フォームの例を示します。

名字	名前	入社年月日	給与	役職名
柴田	豊	78/04/01	#8,680,000	次長
刀根	洋二	77/04/01	#8,235,000	次長
岩島	信次	78/04/01	#7,652,000	常務
大石	幸治	78/04/01	#9,965,100	常務
今村	直重	80/09/01	#6,978,000	
稲川	敏一	80/04/01	#6,240,000	次長代理
渡部	俊夫	80/04/02	#6,581,400	課長代理
魚住	晃	81/04/05	#6,532,000	
田中	聡助	81/09/01	#6,983,000	常務代理
内海	邦章	81/04/01	#6,238,000	
岸元	卓治	81/04/01	#6,052,000	
武蔵	剛史	82/04/01	#6,259,800	係長
白山	敬八朗	82/09/01	#6,159,000	係長代理
小島	忠男	83/04/01	#6,352,100	
浦島	能拓	83/09/01	#6,587,400	
吉野	秀太郎	83/09/01	#6,852,000	課長代理
藤上	団吉	84/04/01	#5,872,400	
西山	芳彦	84/06/01	#5,670,000	
千代田	寿重	84/02/15	#5,897,600	係長
岩澤	五朗	85/06/08	#5,235,400	
今田	喜美斗	85/04/02	#5,365,200	係長

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。レコードを1件選択するために、**FIRST RECORD**コマンドまたは**LAST RECORD**コマンドを使用してください。

MODIFY SELECTIONコマンドは入力フォーム中でレコードを修正することができますが、**DISPLAY SELECTION**コマンドは入力フォーム中でレコードを修正することができません。

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、出力フォームを使用して1レコードのセレクションが表示されます。
- ・ 1番目のオプションのアスタリスク(*)を指定し、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。

「終了」ボタンが、自動的にリストの最後に組み込まれます。フォーム上に変数、またはアクティブオブジェクトを追加すると「終了」ボタンが消えます。このボタンをクリックするとコマンドを終了します。カスタムボタンを代わりに使用することもできます。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。

ユーザは、カレントセレクションをスクロールし、該当レコードをクリックして選択することができます。ユーザが別のレコードをクリックした場合は、最初のレコードの選択が解除され、そのレコードが選択されます。また、ユーザは連続するレコードをグループとして選択することもできます。この場合は、選択する先頭のレコードをクリックし、最後のレコードを“shift”キーを押しながらマウスをクリック(shift+クリック)します。不連続のレコードを選択するには、選択したいレコードごとに、“Ctrl (Macintosh版では、command)”キーを押しながらマウスをクリックします。

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、メニューアイテムが選択された際に、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**コマンドでレコードを選択し、**PRINT SELECTION**コマンドでそのレコードを印刷します。

ALL RECORDS ([従業員])	すべてのレコードの選択
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。

- 「デザイン」モードで必要なメニューコマンド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”等のメニューコマンドです。
- 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。
- 以下のプロジェクトメソッドをメニューコマンドに関連付けます。

```
` M_SHOW_ALL (「すべて表示」メニューアイテムに付属)
$vpCurTable:=Current form table
ALL RECORDS($vpCurTable->)
```

```
` M_QUERY (「クエリ」メニューアイテムに付属)
$vpCurTable:=Current form table
QUERY($vpCurTable->)
```

```
` M_ORDDER_BY (「並び替え」メニューアイテムに付属)
$vpCurTable:=Current form table
ORDER BY($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**、**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} {;} {;} {;})

引数	タイプ	説明
テーブル	テーブル	表示して修正するテーブル 省略した場合、デフォルトテーブル
*	*	レコードが1件の場合にも出力フォームを使用し、 入力フォームのスクロールバーを表示しない
*	*	入力フォームのスクロールバーを表示する (最初のオプション<*>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**コマンドを使用すると、表示されたレコードをダブルクリックすると、既に別のプロセスやユーザに使用されていないかぎり、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
```

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	選択したままにするレコード数

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から最初の＜レコード数＞個のレコードを返します。**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。

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	返すレコードの数
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返します。 <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> 個のレコードを返します。 <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> 個のレコードをテーブルから返します。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。

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返します。

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	選択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

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。

参照

なし

HIGHLIGHT RECORDS

HIGHLIGHT RECORDS ({セット名})

引数	タイプ	説明
セット名	文字列	反転表示させるレコードのセット引数が省略されている場合はUserSet

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反転表示させる事ができます。この操作は、マウスまたはshiftキーを押しながらクリックまたはCtrlキーを押しながらクリック（Macintoshの場合はcommandキーを押しながらクリック）によって、リスト上のレコードを選択するのと同じものです。「選択された」レコードは反転表示されますが、カレントセ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。

セット名に有効な名前を渡すと、そのセットのレコードにコマンドが適用されます。

セット名引数を省略すると、コマンドは現在のUserSetに属するレコードだけに適用されます。

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Search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現れ、検索が実行された後、カレントセ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

なし

この章では、「ルーチン」エディタの「Sets」テーマ内にあるセットコマンドについて説明します。セットを使用すれば、レコードセ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（交わり、INTERSECTION）
- 集合結合（結び、UNION）
- 集合差異（違い、DIFFERENCE）

この章のコマンドは、セットの作成、管理、そして消去について記述します。

CREATE EMPTY SET	CLEAR SET	Is in set
CREATE SET	DIFFERENCE	Records in set
USE SET	INTERSECTION	SAVE SET
ADD TO SET	UNION	LOAD SET
REMOVE FROM SET	COPY SET	
CREATE SET FROM ARRAY		

セットとカレントセクション

セットは、レコードセ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。

- 順序はともかく、セ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クションにアクセスする場合（UserSet）
- セクション間で論理演算を実行する場合

カレントセクションは、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、常にカレントセクションのレコードを用いて作業を行います。また、セ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。カレントセクションは、ひとつのプロセス内の各テーブルに対してひとつしか存在しません。

カレントセクションと同様に、セットはレコードセ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ひと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、トランザクション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クションにし、カレントセ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選択

　` 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クションを作成します。

UserSetはひとつのプロセスに対してひとつしかありません。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])	・すべての従業員を選択
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。

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。

セットの説明の例を参照ください。

CREATE EMPTY SET ([従業員]; "格納セット")	新しいセットを作成
UNION ("UserSet"; "格納セット"; "格納セット")	2つのセットをマージする

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。

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 引数が省略されている場合はデフォルト 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。

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ない、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。

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。

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
$パーセント:=(Records in set ("お得意様") / Records in file ([顧客])) * 100
` お得意様の割合をパーセント表示
ALERT (String ($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。

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。

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)

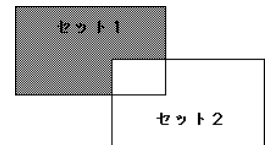
引数	タイプ	説明
セット1	文字列	オリジナルセット
セット2	文字列	排除するセット
結果セット	文字列	結果を格納するセット

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。

以下の例は、表示したセ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

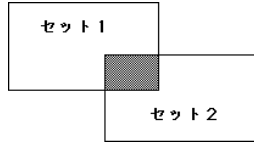
INTERSECTION (セット1; セット2; 結果セット)

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
	x	x
x		x
x	x	x

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。

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

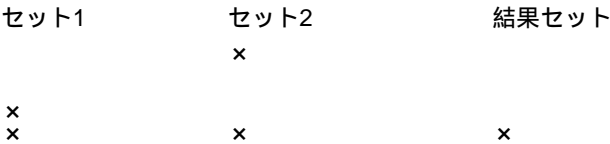
UNION

UNION (セット1; セット2; 結果セット)

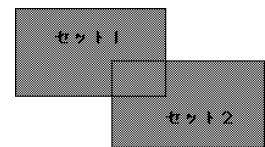
引数	タイプ	説明
セット1	文字列	第1のセット
セット2	文字列	第2のセット
結果セット	文字列	結果を格納するセット

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。



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。

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]) | すべての顧客を選択 |
| 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

なし

この章では、「ルーチン」エディタの「String」テーマ内にある文字列関数について説明します。この章では、文字列に対して機能する関数と文字参照記号について説明します。

Ascii	Length	Substring
Change string	Lowercase	Uppercase
Char	Num	Mac to Win
Convert case	Position	Win to Mac
Delete string	Replace string	Mac to ISO
Insert String	String	ISO to Mac

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。

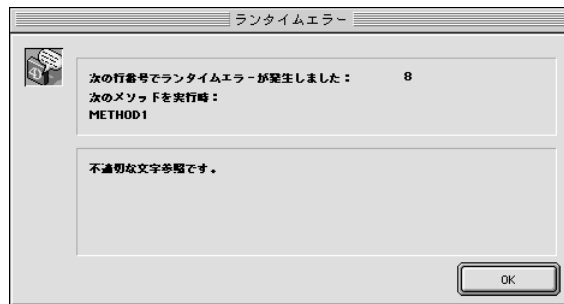
長さが足りない場合、コンパイルモードで範囲チェックオプションをオンにしてある場合に、例えば以下のようなコードを実行するとします。

```
vsAnyText:=""
```

```
vsAnyText[[1]]:="A"
```

、決してこれをしてはいけません

すると、ランタイムエラーが発生します。



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]])
```

、次に現れた文字を大文字にする

```
  End if
```

```
End for
```

参照

Ascii、Char

String

String (数値 {; 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 “|” 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 Date 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 Date 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現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が最初に現れる位置を代入
```

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)
$viElem:=Size of array ($2->)
Repeat
    $viElem:=$viElem+1
    INSERT ELEMENT ($2->,$viElem)
    $viPos:=Position (Char (Carriage return);$1)
    If ($viPos>0)
        $2->{$viElem}:=Substring ($1;1;$viPos-1)
        $1:=Substring ($1;$viPos+1)
```

```

Else
    $2->{$vElem}:= $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。

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

参照
なし

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MacOS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($vIChar;1;Length (vtText))
```

Case of

```
    ¥ (vtText[$vIChar])=Char (Carriage return))
```

```
    ` 処理を実行
```

```
    ¥ (vtText[$vIChar])=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 ($vIChar;1;Length (vtText))
  $vIAscii:=Ascii (vtText[[$vIChar]])
  Case of
    ¥ ($vIAscii=Carriage return)
    ` 処理を実行
    ¥ ($vIAscii=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MacOS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。

以下の文字は、Macintosh上でのみ利用することができます (このコード16-20は、Chicagoフォントにしか存在しません)。

コード	ASCIIコード	キー
16		+P
17		+Q
18		+R
19		+S
20		+T

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

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。

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。

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。

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。

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。

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	置き換える対象となる文字列
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 (v結果; Char (13); ""); Char (9); "")
```

参照

Change string、Delete string、Insert string

Convert case

Convert case (文字列 ; 変換タイプ ; 対象タイプ) 文字列

引数	タイプ	説明
文字列	文字列	変換対象となる文字列
変換タイプ	数値	文字列の変換タイプ
対象タイプ	数値	変換対象となる文字のタイプ

説明

「文字列」中の変換対象となる文字タイプを「対象タイプ」で指定し、「変換タイプ」で指定したタイプに変換します。「対象タイプ」の値は、加算することによって、複数の対象を同時に指定することができます。例えば、“1バイトのアルファベットと数字と記号”および“1バイトのカタカナ”を対象とする場合は、4+16の結果である 20を指定することができます。

以下に、指定できる「変換タイプ」と「対象タイプ」の値を示します。

指定できる変換タイプ

- 2 1バイトのアルファベットと数字と記号へ変換
- 3 2バイトのアルファベットと数字と記号へ変換
- 4 1バイトのカタカナへ変換
- 5 2バイトのカタカナへ変換
- 7 2バイトのひらがなへ変換

指定できる対象タイプ

- 1 テキスト全体を対象
- 4 1バイトのアルファベットと記号を対象
- 8 2バイトのアルファベットと記号を対象
- 16 1バイトのカタカナを対象
- 32 2バイトのカタカナを対象
- 128 2バイトのひらがなを対象

例

v文字:=**Convert case** ("あいうえおABCD12345+*/?ふぁーでいふぁーすと"; 2 ; -1)

上記のコードを実行すると、
変数"v文字"に「aiueoABCD12345+*/?fo-dhifa-suto」が返ります。

v文字:=**Convert case** ("あいうえおあBCD12345+*/?ふぁーでいふぁーすと"; 4 ; -1)

上記のコードを実行すると、
変数"v文字"に「アウIアBCD12345+*/?fo-d`iファ-スト」が返ります。

`v文字:=Convert case ("あいうえおABCD12345+*/?ふぉーでいふぁーすと"; 5 ; -1)`

上記のコードを実行すると、
変数"v文字"に「アイウエオアBCD12345+*/?フォーディファースト」が返ります。

`v文字:=Convert case ("ア伊IオABCD12345+*/?fo-dhifa-suto"; 7 ; 4)`

上記のコードを実行すると、
変数"v文字"に「ア伊IオあBCD12345+*/?ふぉーでいふぁーすと」が返ります。

`v文字:=Convert case ("ア伊IオABCD12345+*/?fo-dhifa-suto"; 7 ; 16)`

上記のコードを実行すると、
変数"v文字"に「あいうえおABCD12345+*/?fo-dhifa-suto」が返ります。

`v文字:=Convert case ("ア伊IオABCD12345+*/?fo-dhifa-suto"; 7 ; (4+16))`

上記のコードを実行すると、
変数"v文字"に「あいうえおあBCD12345+*/?ふぉーでいふぁーすと」が返ります。

参照

なし

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、MacOS 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、MacOS 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MacOS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

Windowsで**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MacOS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ください）していない場合には、手動でテキストをMacOS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MacOS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

Windowsで**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ください）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リファレンスガイド』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）MacOS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
<>vsFTPPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDiretory:="APS500:Spiders:" ` MacOSファイルマネージャのパス名
` ...
` ...
ARRAY STRING (31 ; $asDocuments ; 0)
DOCUMENT LIST (...; $asDocuments)
$viNbDocuments:=Size of array ($asDocuments)
jsHandler:=...
For ($viDocument ; 1 ; $viNbDocuments)
    vtHTMLCode:=vtHTMLCode+"<P><A HREF="+Char (34)+<>vsFTPPURL
    + HTML Pathname (Substring ($1+$asDocuments{$viDocument}
    ; Length (<>vsFTPDiretory)+1))+Char (34)+jsHandler
    + "> "+$asDocuments{$viDocument}+"</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MacOS 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MacOS 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MacOS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リファレンスガイド』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、MacOS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

この章では、「ルーチン」エディタの「Structure Access」テーマ内にあるストラクチャアクセスコマンドについて説明します。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。

Count fields	GET database Parameter	SET INDEX
Count tables	GET FIELD PROPERTIES	Table
Field	SET database Parameter	Table name
Field 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。

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。

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増えるためです。

```
C_LONGINT ($TableName ; $HowMany ; $i)
$TableName :=Table ($1)
` 配列を作成する
ARRAY STRING (15 ; arFields ; Count fields ($TableName))
$HowMany:=Size of array (arFields)
For ($i ; 1 ; $HowMany) ` 配列を通してループする
    arFields {$i}:=Field name ($TableName ; $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 テーブルポインタ	数値または ポインタ	テーブル番号、または 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、ウィ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")
` ウィ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 ポインタ	数値または ポインタ	テーブル番号または 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 数値	テーブル番号を渡した場合、テーブルへのポインタ 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

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 ポインタ	フィールドポインタを渡した場合、フィールド番号 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 (テーブル番号 ; フィールド番号 ; タイプ {; 長さ {; インデックス}})

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

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。

例題

1. 以下の例は、変数“vタイプ”、“v長さ”、“vインデックス”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)

2. 以下の例は、変数“vタイプ”、“v長さ”、“vインデックス”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)

参照

Field、Field name、SET INDEX

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に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。

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

この関数によって返される値については、**SET DATABASE PARAMETER**コマンドの内容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、MacOS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MacOS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シュサイズセレクタは**SET DATABASE PARAMETER**関数を使ってもセットされません。

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。

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	倍長整数	8

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からです。デフォルトでは、値は1になります (最適化モード)。

セクタ = 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、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。

セクタ = 6 (Minimum Web Process)(Webプロセスの最小数)

値 : 0 32,767

内容 : 非コンテキストモード内に保持する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)。

セクタ = 7 (Maximum Web Process)(Webプロセスの最大数)

値 : 0 32,767

内容 : 非コンテキストモード内に保持するWebプロセスの最大数 (*1)。デフォルトでは、値は0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。

セレクト = 8 (Web conversion mode)(Web変換モード)

値 : 1または2

1 = 6.0.x 変換モード

2 = 6.5 変換モード

デフォルトで値は2 (6.5モード) です。

内容 : いくつかの場合、4Dの6.0xバージョンで作成した、とくにHTMLページに {mypage.htm} のような参照を含むフォームのWeb変換は4Dの6.5では正確でないかもしれません。この場合は、フォームの両立を確実にするため、4Dの6.0xの変換モードを実行できます。このモードは**SET DATABASE PARAMETER**が呼ばれたプロセス (Webコンテキスト) でのみセットできます。6.0xのデータベースのすべてのフォームの両立を確実にするか、または、ひとつのフォームが表示される前に**On Web Connection Database**メソッドを呼ぶことが出来ます。このコマンドはコンテストモードかWebプロセスの外側からも呼び出されますが、効果はないでしょう。

注 : セレクトの追加は**Get database parameter**関数のデータベースキャッシュサイズ (9) でできます。このセレクトは**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。

参照

DISTINCT VALUES、GET database parameter、QUERY SELECTION

この章では、「ルーチン」エディタの「Subrecords」テーマ内にあるサブレコードコマンドについて説明します。この章のコマンドは、レコードを操作するのと同じようにサブレコードを処理するためのものです。これらのコマンドを使用すれば、サブレコードを追加、修正、削除、あるいはフォーミュラで更新することができます。

ALL SUBRECORDS	LAST SUBRECORD
APPLY TO SUBSELECTION	NEXT SUBRECORD
Before subselection	ORDER SUBRECORD BY
CREATE SUBRECORD	PREVIOUS SUBRECORD
DELETE SUBRECORD	QUERY SUBRECORD
End subselection	Records in subselection
FIRST SUBRECORD	

FIRST SUBRECORD、**LAST SUBRECORD**、**NEXT SUBRECORD**、**PREVIOUS SUBRECORD**の各コマンドを使用してサブセレクション内のサブレコードを移動することもできます。これらのコマンドを使用する前に正しいサブセレクションを作成することが重要です。カレントサブセレクションにサブレコードが存在しない場合には、これらのコマンドは何も行いません。

プログラミングをしないで、これらのサブレコードを移動するためのコマンドと同じ機能をフォーム上のボタンに対して割り当てることができます。

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。

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。

サブセクションを削除するには、削除しようとするサブセクションを作成し、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選択する
For ($i; 1; Records in selection ([従業員])) ` テーブルの全レコードに対して
  ` 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	すべてのサブレコードを選択するサブテーブル

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、サブセクションはす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。

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取得するサブテーブル
戻り値	数値	カレントサブセ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。

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、サブセクションから情報を収集する、またはサブセクションを修正するために使用します。

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。カレントサブセクションが空の場合は、**FIRST SUBRECORD** コマンドは何も行いません。

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。カレントサブセクションが空の場合は、**LAST SUBRECORD** コマンドは何も行いません。

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。

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。

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。

ひと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。

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。

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

この章では、「ルーチン」エディタの「System Documents」テーマ内にあるシステムドキュメントコマンドについて説明します。この章のコマンドは、ドキュメントファイルに関する作業を行います。

Append document	MAP FILE TYPES
CLOSE DOCUMENT	MOVE DOCUMENT
COPY DOCUMENT	Open document
Create document	Select folder
CREATE FOLDER	SET DOCUMENT CREATOR
DELETE DOCUMENT	SET DOCUMENT POSITION
Document Creator	SET DOCUMENT PROPERTIES
DOCUMENT LIST	SET DOCUMENT SIZE
Document type	SET DOCUMENT TYPE
FOLDER LIST	Test path name
Get document position	VOLUME ATTRIBUTES
GET DOCUMENT PROPERTIES	VOLUME LIST
Get document size	

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、システムを起動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、そしてクリエイターです。

ドキュメントファイルのタイプとクリエイター

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イターがあります。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イターがあります。この概念は、Windowsにはありません。ドキュメントのクリエイターは、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イタープロパティによって判断されます。例えば、4D V6で作成されたドキュメントのファイルクリエイターは「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、既に関わ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**関数の3つを使用すると、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使用してドキュメントにアクセスできます。

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。

ドキュメント名やドキュメントパス名の指定

この節にある、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（*）。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。パス名を渡す場合には、必ず有効なパス名でなければなりません。間違った名前や間違ったパス名を渡すと、コマンドはファイルマネージャエラーを発生しますが、これは**ON ERR CALL**メソッドを使用して検出することができます。

（*）その他の方法で指定された場合は、例外です。

警告：引数<ドキュメント>の最大長は255文字です。これより長い名前を渡すと、引数名は途中で切り取られ、ファイルマネージャエラーが発生します。

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 ($0)
C_LONGINT ($vIPlatform ; $vISystem ; $vIMachine)
PLATFORM PROPERTIES ($vIPlatform ; $vISystem ; $vIMachine)
$0:=( $vIPlatform=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 ":"(MacOS)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、ウィ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、VOLUM ATTRIBUTES、VOLUME LIST

Open document

Open document (ドキュメント {; ファイルタイプ}) 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
ファイルタイプ	文字列	MacOS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	DocRef	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。

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。

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、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ます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。

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: **docName**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現れます。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。

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)

vドキュメント:=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 (vドキュメント;"さようなら") `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 - プロソフトウェアで開いてみると、“さようなら”という文字列が含まれています。

以下の例は、ドキュメントファイル“ノート”を開き、それに“ 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。この場合、前の内容“こんにちは”が上書きされるため残りません。前の内容“さようなら”は、そのまま残ります。したがって、ドキュメントファイルの内容は、“さようなら また、会いましょう”とキャリッジリターンになります。

C_TIME (vDoc)

ドキュメント:=**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 (ドキュメント;" 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 (ドキュメント) `ドキュメントファイルを閉じる

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)

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 : docName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現れます。これが受け入れられると、Document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	MacOS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	DocRef	ドキュメントファイル参照番号

説明

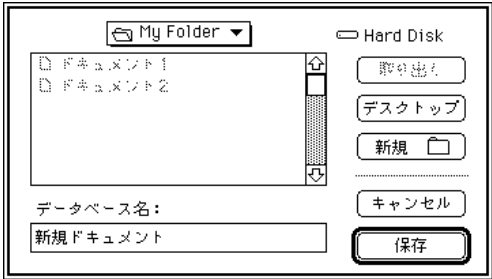
Create document関数は、<ドキュメント>で指定した名前の新しいドキュメントファイルを作成し、そのドキュメントファイルのドキュメントファイル参照番号を返します。

<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。

Windows版



Macintosh版



<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。

「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ルを作成します。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。

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、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。

このコマンドはさまざまな方法で呼び出すことが可能です。一般的なファイルタイプの例を以下に示します。

テキストファイル：

```
$DocRef:=Create document (""; ".TXT")
```

により、Windows上の.TXTファイルとMacintoshのテキストドキュメントが作られます。4th DimensionがMacintoshのファイルタイプTEXTとWindowsのファイル拡張子.TXTを自動的にマッピングしているからです。

例えば、ドキュメント名を“Information”とした場合、Windows上では“Information.TXT”が作成され、Macintosh上では“Information”が作成されます。

ピクチャファイル：

MacintoshとWindowsの両方で目的のファイルタイプのファイルを作成するためには、**MAP FILE TYPES**コマンドが利用できます。「On Startup」データベースメソッドには、以下のように書いておきます。

```
MAP FILE TYPES ("PICT"; "PCT"; "Macintoshピクチャドキュメント")
```

この命令文は、Windowsの.PCTファイルとMacintoshのPICTファイルのマッピングを設定します。そこで、以下のような命令文を使用すると、

```
$DocRef:=Create document (""; "PCT")
```

Windows上では.PCTファイルが作られ、Macintosh上ではPICTファイルが作られます。

ユーザ定義の拡張子

Macintosh上でファイル名の一部として拡張子を使用している場合、Windowsのファイル拡張子と重複しないように、プラットフォームごとに異なる**Create Document**文を書く必要があるかもしれません。例えば、Macintosh上で拡張子に「.Prefs」を付けた独自の環境設定ドキュメントを使用しているとします。例えば、以下のように利用しているとします。

```
$DocRef:=Create document ("Taro.Prefs" ; "TEXT")
```

このような場合、Windowsのファイル拡張子とMacintoshのファイル名の拡張子（これらは2種の別個なものです）の間で混同を避けようとするものでしょう。さらに、Windows上では「.TXT」ファイルではなく「.PRF」ファイルを作るかもしれません。データベースが実行されているプラットフォームを返す**PLATFORM INFO**コマンドを利用し、その後、プラットフォームに応じた**Create Document**関数をコールすればよいのです。以下のコードは、実際にこれを行っています。

```
` この例では、インタープロセス変数<>Platform は、あらかじめ  
` データベースの「On Startup」データベースメソッドで GET PLATFORM INFO  
` コマンドで初期化されているものとして使用します  
If (<>Platform=3)  
` Windows 上で .PRFファイル を作る  
  $DocRef:=Create document ("Taro" ; "PRF")  
Else  
` Macintosh 上でテキストファイルを作る  
  $DocRef:=Create document ("Taro.Prefs" ; "TEXT")  
End if
```

Windows 上

省略可能な<タイプ>引数を使用しない場合、4th Dimensionでは.TXTドキュメントを想定します。

TEXTを指定すると、4th Dimensionでは自動的にMacintoshのTEXTファイルタイプとWindowsの.TXT拡張子とを結び付け、.TXTドキュメントに対するアクセスと見なします。

あらかじめ **MAP FILE TYPES**コマンドでマッピングされたMacintoshのファイルタイプを指定すると、4th Dimensionでは自動的に Macintosh のファイルタイプとWindowsの拡張子とを結び付け、そのタイプのWindowsドキュメントを作成するものと見なします。

指定された4文字の値がマッピングされていない場合、4th Dimensionは値を切り捨てて3文字にし、以下で説明している引数として処理します。

1から3文字の値を指定すると、4th Dimensionでは<タイプ>引数をWindowsのファイル拡張子として使用し、名前の後ろに付け加えられます。

Macintosh上

省略可能な<タイプ>引数を使用しない場合、4th DimensionではTEXTドキュメントを想定します。

省略可能な<タイプ>引数を使用する場合、4文字のMacintoshファイルタイプを渡します。

警告：Windows上で、**Create Document**関数を使用し、<ドキュメント>に空の文字列を渡すと、「ファイル保存」ダイアログボックスの「ファイル名」入力エリアで、ファイル拡張子が増える可能性があります。その場合、省略可能な<タイプ>引数に指定したファイル拡張子とはまったく違うファイル拡張子のファイルが作られることとなります。そこで、ファイル拡張子を確実に使う場合、作成されたドキュメント名を確認することをお勧めします。

```
$DocRef:=Create document (""; "TXT")
```

```
If (OK=1)
```

```
  `この例では、インタープロセス変数<>Platformは、あらかじめ
```

```
  `データベースの「On Startup」データベースメソッドでGET PLATFORM INFO
```

```
  `コマンドで初期化されているものとして使用します
```

```
If (<>Platform=2)
```

```
  If (Substring (Document ; Length (Document) - 3) # ".TEXT")
```

```
    ALERT ("テキストドキュメントを作成してください。")
```

```
    CLOSE DOCUMENT ($DocRef)
```

```
    DELETE DOCUMENT (Document)
```

```
    OK:=0
```

```
  End if
```

```
End if
```

```
If (OK=1)
```

```
  `ドキュメントの内容について処理を実行する
```

```
  CLOSE DOCUMENT ($DocRef)
```

```
End if
```

```
End if
```

ユーザがドキュメントファイルを作成した場合には、システム変数OKに1がセットされます。システム変数Documentには、開いたドキュメントファイルの名前が代入されます。

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。ドキュメントファイルが**Create document**関数で作成されない場合は、ドキュメントファイルの参照番号0を返します。

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 - 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。

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	MacOS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	DocRef	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。

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 また、会いましょう ”とキャリッジリターンになります。

C_TIME (vDoc)

ドキュメント:=**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 (ドキュメント;" 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 (ドキュメント) `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
ドキュメントファイル参照番号	ドキュメントファイル参照番号	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。

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

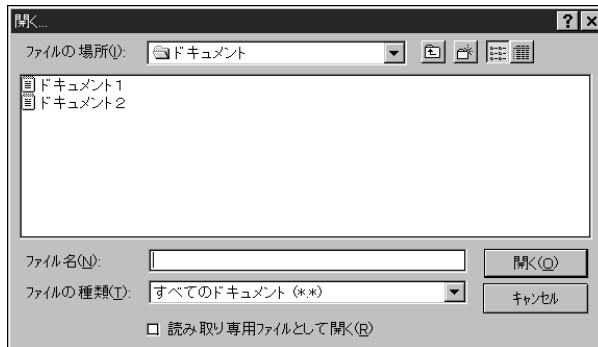
If (OK=1)

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

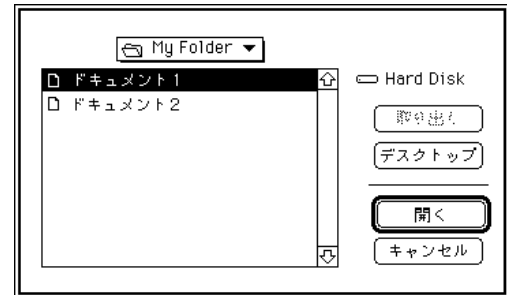
CLOSE DOCUMENT (ドキュメント) `ドキュメントファイルを閉じる

End if

Windows版



Macintosh版



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です)。

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。

参照

なし

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ければ、**Test path name**関数は1を返します。フォルダが見つければ、**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 document	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）。

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

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

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。

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 ($1 ; $vINbVolumes ; $vIVolume ; $vISize ; $vIUsed ; $vIFree)
C_REAL ($vISize ; $vIUserd ; $vIFree)
C_REAL ($vISize ; $vIUserd ; $vIFree)
C_TIME ($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)
    $vIVolume:=Find in array ($asVolumes ; "A:¥")
```

```

    If ($vIVolume>0)
        DELETE ELEMENT ($asVolumes ; $vIVolume)
    End if
    $vIVolume:=Find in array ($asVolumes ; "B:¥")
    If ($vIVolume>0)
        DELETE ELEMENT ($asVolumes ; $vIVolume)
    End if
End if
$vINbVolumes:=Size of array ($asVolumes)
` For each volume
For ($vIVolume ; 1 ; $vINbVolumes)
    `使用サイズおよび空きサイズを取得する
    gError:=0
    VOLUME ATTRIBUTES ($asVolumes{$vIVolume} ; $vISize ; $vIUsed ; $vIFree)
    If (gError=0)
        `空きサイズは十分ありますか (必要なサイズに32KBを加えたサイズ)?
        If ($vIFree>=($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{$vIVolume}+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{$vIVolume}
            $vIVolume:=$vINbVolumes+1
        End if
    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

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 (13バイト) または MacOS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
ファイルタイプ	文字列	またはドキュメントファイルへの完全なパス名 Windowsファイル拡張子 (13バイト) MacOS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イター」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

MAP FILE TYPES

MAP FILE TYPES (MacOS ; Windows ; コンテキスト)

引数	タイプ	説明
MacOS	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 < MacOS > 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。

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)

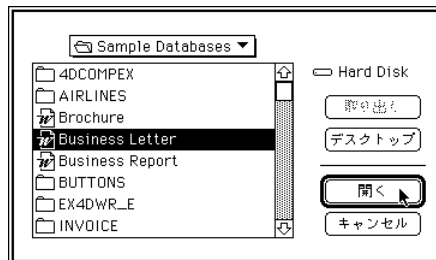
、 ...

End if

Windows版 (Windows NT3.5の画面)



Macintosh版



参照

Append document、Create document、Create resource file、Open resource file、Open resource file

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 ; ファイルクリエイター)

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
ファイルクリエイター	文字列	MacOSファイルクリエイター (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イターを設定します。

ドキュメントの新しいクリエイターは、引数 <ファイルクリエイター> 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イター」の節を参照してください。

参照

Document creator、SET DOCUMENT PROPERTIES、SET DOCUMENT TYPE

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。

文書	
番号	L
主題	A
テーマ	A
説明	T
作成日	L
更新日	L

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 at 00:00:00から2063/01/19at 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2³¹-1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な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 ; $vsDocName)
C_LONGINT ($vIDoc)
C_BOOLEAN ($vbOnWindows ; $vbDolt ; $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
$viNbRecords:=Records in selection ([ドキュメント])
$viNbDocs:=0
$vbOnWindows:= On Windows
For ($viDoc ; 1 ; $viNbRecords)
    `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
    $vbDolt:=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。
        $viElem:=Find in array ($asDocument ; $vsDocName+".HTM")
        If ($viElem>0)
            DELETE ELEMENT ($asDocument ; $viElem)
        End if
        `レコードが最後に更新された後で、ドキュメントが保存されたか？
        GET DOCUMENT PROPERTIES ($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
            $vbDolt:=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!
$vhModifiedAt:=!00:00:00!!
End if
`ドキュメントを (再) 作成する必要があるか?
If ($vbDolt)
    `その場合には、更新されたドキュメントの数を増分する
    $vINbDocs:=$vINbDocs+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 ; $vhModifiedAt)
    End if
End if
`現在実行中の処理を確認するために
SET WINDOW TITLE("ドキュメント処理中 : "+String($vIDoc)+"/"+String
($vINbRecords))
NEXT RECORD ([ドキュメント])
End for
`古いドキュメント、つまりまだ$asDocument配列内にあるドキュメントを削除する
For ($vIDoc ; 1 ; Size of array ($asDocument))
    DELETE DOCUMENT ($vsPath+$asDocument{$vIDoc})

```

```
SET WINDOW TITLE ("ドキュメント削除中 : "+Char (34)+$asDocument  
{$vIDoc}+Char (34))
```

```
End for
```

```
` We're done
```

```
ALERT ("処理されたドキュメントの数 : "+String ($vINbRecords)+Char (13)+  
"更新されたドキュメントの数 : "+String ($vINbDocs)+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 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	ドキュメント参照番号 または、文字列	ドキュメント参照番号 または、ドキュメントの名前
*	*	MacOS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	ドキュメント参照番号	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	ドキュメント参照番号	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	ドキュメント 参照番号	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		1=ファイルの最初からの相対位置 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

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。

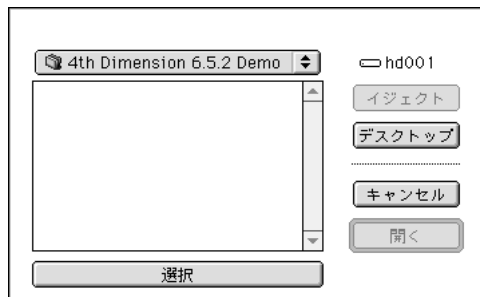
注：このコマンドは、4Dのカレントフォルダを変更しません。

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。

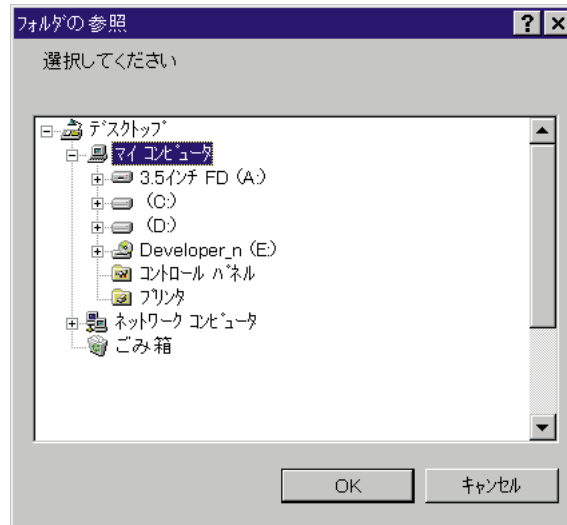
以下の例題は、"目的のフォルダを選択してください:"のメッセージです。

(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。

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

この章では、「ルーチン」エディタの「System Environment」テーマ内にあるシステム環境コマンドについて説明します。

Count screens	Gestalt	Screen Height
Current machine	LOG EVENT	Screen Width
Current machine owner	Menu bar height	SET SCREEN DEPTH
FONT LIST	Menu bar screen	System folder
Font name	SCREEN COORDINATES	Temporary folder
Font number	SCREEN DEPTH	

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=白黒またはグレースケール
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 (2⁸) 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。モニタがグレー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 (\$vIDepth ; \$vIColor)

If (\$vIDepth<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=グレースケール
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/グレー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 フォント	タイプ 配列	説明 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 フォント番号	タイプ 数値	説明 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

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、WindowsまたはMacintoshの稼働しているシステムフォルダへのパス名を返します。

参照
Aci folder、Temporary folder

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
Temporary folder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。

APPEND TO CLIPBOARD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 (セレクトア ; 値) 文字列

引数	タイプ	説明
セレクトア	文字列	4文字のGestaltセレクトア
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<セレクトア>に渡すセレクトアに基づいて、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。セレクトアがわからなければ、**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MacOSの一部です。セレクトアのいくつかは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トアについての詳細は、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://til.info.apple.com/techinfo.nsf/artnum/n9095>

例題

Machintosh上で、MacOS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。

```
$vErrCode:=Gestalt ("sysv" ; $vllInfo)
If ($vErrCode=0)
    ALERT ("システムバージョン : "+String ($vllInfo ; "&x") + "を実行しています。")
End if
```

参照
なし

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WindowsNTでのみ可能です。

このコマンドは、WindowsNT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、WindowsNT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"に現れるカスタムメッセージを追加することができますようにします。

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。

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WindowsNT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。

参照

なし

この章では、テーブルとフォームのデフォルトを設定するコマンドについて説明します。これらのコマンドは、「ユーザ」モードで“ テーブル/フォーム選択 ”を選択した場合と同じです。

Current form table	DEFAULT TABLE	OUTPUT FORM
Current default table	INPUT FORM	

4th Dimensionのプログラミング言語の多くのコマンドは、テーブルの指定が必要です。コマンドの最初の引数としてテーブルを指定することができます。また、**DEFAULT TABLE**コマンドでデフォルトテーブルを設定することもできます。各々のプロセスは独自のデフォルトテーブルを持っています。

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ひとつの目的は、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。

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

フォームを指定する

この節は、入出力のフォームを指定するコマンドについて説明します。フォームは、データ入力、印刷、データ読み込み、データ書き出し、ユーザインタフェースの作成等4th Dimension内で広範囲に使用されます。

入力フォームは、一般的にデータ入力等の1度に1つのレコードしか表示しないコマンドに付随します。出力フォームは通常、リスト形式で画面またはプリンタに複数レコードを表示するコマンドに付随します。フォームは、本質的には入力フォームでも出力フォームでもありません。フォームのいくつかは入力フォームとして、または出力フォームとして使用されます。

INPUT FORMコマンドと**OUTPUT FORM**コマンドは、テーブルに対して使用するフォームを指定します。各テーブルは、カレント入力フォームとカレント出力フォームを持っています。これらのコマンドは、その後のフォームを必要とするコマンドで使用されます。カレントフォームは、「エクスプローラ」の「フォーム」ページ上でカレント入力フォームを“ I ”およびカレント出力フォームを“ O ”で指定します。**INPUT FORM**コマンドまたは**OUTPUT FORM**コマンドで、入出力のフォームを指定しない場合には、すべての操作に対して、「デザイン」モードで指定されたフォームを使用します。

INPUT FORMコマンドと**OUTPUT FORM**コマンドは、単に使用するフォームを指定するだけで、フォームを表示するわけではありません。

INPUT FORM

INPUT FORM ({テーブル;} フォーム {;} *)

引数	タイプ	説明
テーブル	テーブル	入力フォームを指定するテーブル 省略した場合は、デフォルトテーブル
フォーム	文字列	入力フォームに設定するフォーム名
*	*	自動ウインドウサイズ

説明

INPUT FORMコマンドは、<テーブル>のカレント入力フォームに<フォーム>を指定します。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、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。

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。

入力フォームは、ユーザがサブフォームをダブルクリックした場合にも表示されます。この場合、「デザイン」モードでサブフォームを作成するときに、入力フォーム（フルページフォーム）を割り当てなければなりません。

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**関数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。

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。

以下の例は、**INPUT FORM**コマンドの一般的な使用方法です。この例では、**INPUT FORM**コマンドを入力フォームが使用される直前に記述していますが、実際には、これは必要はありません。他のメソッドで**INPUT FORM**コマンドを実行しても構いません。

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

参照

ADD RECORD、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)

引数	タイプ	説明
テーブル	テーブル	出力フォームを指定するテーブル 省略した場合、デフォルトテーブル
フォーム	文字列	フォーム名

説明

OUTPUT FORMコマンドは、<テーブル> のカレント出力フォームに<フォーム>を指定します。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。

出力フォームは3つのコマンドグループ（画面の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）で使います。**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います。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います。

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

DISPLAY SELECTION、EXPORT DIF、EXPORT SYLK、EXPORT TEXT、INPUT FORM、MODIFY SELECTION、PRINT LABEL、PRINT SELECTION

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る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

この章では、「ルーチン」エディタの「Tool Bar」テーマ内にあるツールバーコマンドについて説明します。

HIDE TOOL BAR

SHOW TOOL BAR

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れている場合は、**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

この章では、「ルーチン」エディタの「Transactions」テーマ内にあるトランザクションコマンドについて説明します。

START TRANSACTION
VALIDATE TRANSACTION

CANCEL TRANSACTION
In Transaction

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。

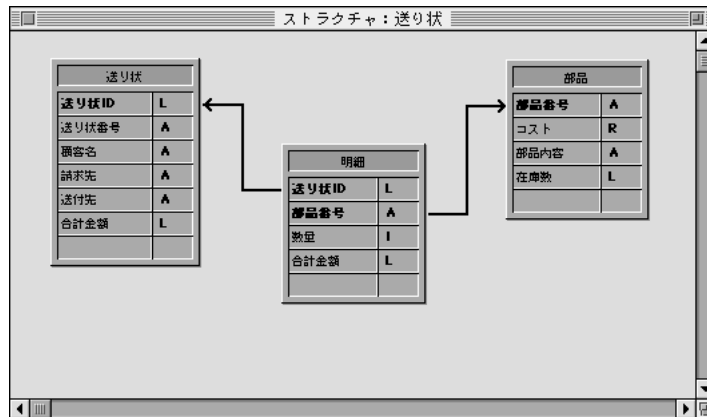
下記のコマンドは、レコード番号を使用します。トランザクションの中で使用することはできません。

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ひとつです。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、データ入力がある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い場合は、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

この章では、「ルーチン」エディタの「Triggers」テーマ内にあるトリガコマンドについて説明します。

Database event	Trigger level	TRIGGER PROPERTIES
----------------	---------------	--------------------

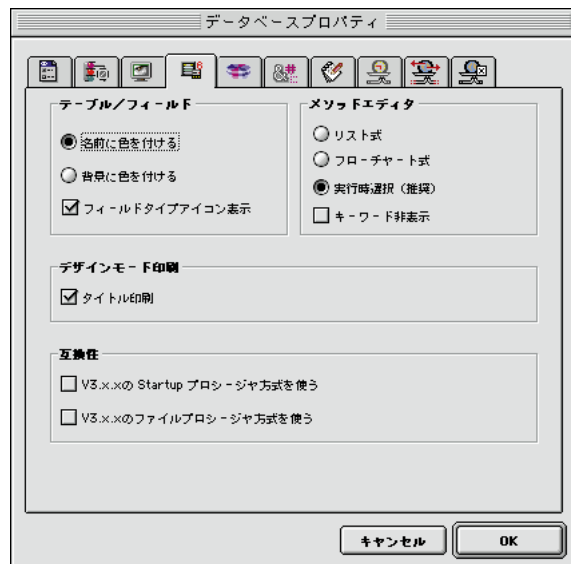
トリガについて

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。ユーザがテーブルのレコードを操作する（追加、削除、修正、ロードする）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「データベースプロパティ」ダイアログボックスにおいて、「旧バージョン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。

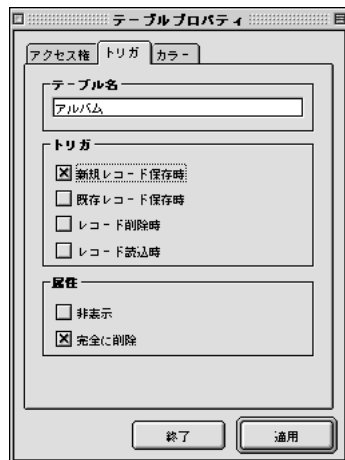
トリガをアクティブにし、4Dに対してトリガをいつ起動すべきかを知らせる。

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、「テーブルプロパティ」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）。

リレーの削除制御オプションによって、リレー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**=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(アクション)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

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、データ読み込み、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、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ます。

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ます。

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。

エンドユーザ	データベース設計者
ユーザインタフェース	プログラム環境
トリガ	
データベースエンジン	
プラットフォームハードウェア／ソフトウェア	

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。

セットと命名セクション：トリガ内からセットや命名セ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グ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ってください。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。

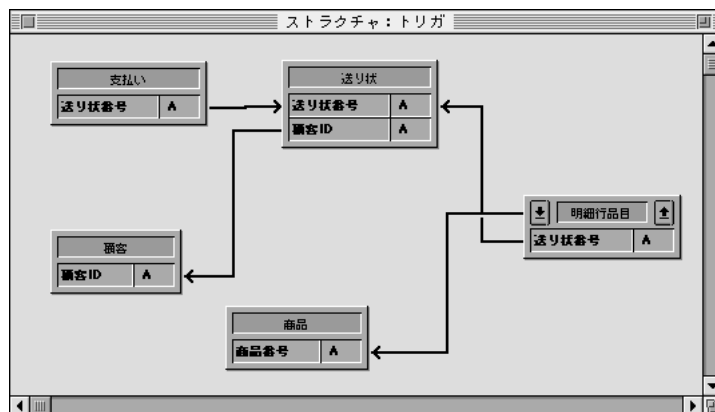
トリガとトランザクション

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となりますが、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

この章では、「ルーチン」エディタの「User Interface」テーマ内にあるユーザインタフェース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。

BEEP	Pop up menu
Caps lock down	POST CLICK
GET HIGHLIGHT	POST EVENT
GET MOUSE	POST KET
HIGHLIGHT TEXT	REDRAW
INVERT BACKGROUND	SET CURSOR
Get platform interface	SET FIELD TITLES
Macintosh command down	SET PLATFORM INTERFACE
Macintosh control down	SET TABLE TITLES
Macintosh option down	Shift down
Last object	Windows Ctrl down
PLAY	Windows Alt down

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。

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 : MacOS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 (いくつ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。“ウエルカム”の音でユーザを歓迎します。

```
PLAY ("ウエルカム")           、"ウエルカム"の音を発生する
```

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 interface	倍長整数	-1
Macintosh interface	倍長整数	0
Windows 3.1 interface	倍長整数	1
Windows 95 interface	倍長整数	2
Copland interface	倍長整数	3

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	Windows3.1
	2	Windows 95
	3	Copland

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 interface	倍長整数	-1
Macintosh interface	倍長整数	0
Windows 3.1 interface	倍長整数	1
Windows 95 interface	倍長整数	2
Copland interface	倍長整数	3

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：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現れるテーブル名
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4th Dimension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。

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なります。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。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 TABLE TITLES コマンドは、以下の場合に使用します。

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 ($vITable ; $vIField)
```

```
For ($vITable ; 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 ($vITable) ; *)
```

```
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($vITable)
```

```
[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 ($viField ; 1 ; Count fields($viTable))
`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 ($viTable ; $viField) ; *)
QUERY ([翻訳] & [翻訳]言語=$1)
If (Records in selection ([翻訳])=0)
`なければ、レコードを作成する
CREATE RECORD ([翻訳])
[翻訳]実際の名前:=Field name ($viTable ; $viField)
[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 ; $1)
C_LONGINT ($viTable ; $viNbTable ; $viField ; $viNbField)
$viNbTable:=Count tables `データベースに存在するテーブル数を数える
` SET TABLE TITLESコマンドに渡される配列を初期化する
ARRAY STRING (31 ; $asTableName ; $viNbTable)
ARRAY INTEGER ($aiTableNumber ; $viNbTable)
For ($viTable ; 1 ; $viNbTable) ` テーブルをループする
    $asTableName{$viTable}:=Table name ($viTable) ` テーブル名を取得する
    $aiTableNumber{$viTable}:= $viTable ` 実際のテーブル番号を格納する
    QUERY ([翻訳];[翻訳]実際の名前=$asTableName{$viTable}; *) 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
  $asTableName{$vITable}:=[翻訳]翻訳された名前
End if
  $vINbField:=Count fields ($vITable) `そのテーブルのフィールド数を取得する
  ` SET TABLE TITLESコマンドに渡される配列を初期化する
  ARRAY STRING (31 ; $asFieldName ; $vINbTable)
  ARRAY INTEGER ($aiFieldNumber ; $vINbTable)
  For ($vIField ; 1) `フィールドをループする
    $asFieldName{$vIField}:=Field name ($vITable; $vIField) `フィールド名を取得する
    $aiFieldNumber{$vIField}:=$vIField `実際のフィールド番号を格納する
    QUERY ([翻訳];[翻訳]実際の名前=$asFieldName{$vIField} ; *)
    `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
      $asFieldName{$vIField}:=[翻訳]翻訳された名前 `
    End if
  End for
  SORT ARRAY ($asFieldName ; $aiFieldNumber ; >)
  SET FIELD TITLES (Table ($vITable)-> ; $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

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現れるフィールド名
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 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

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	マウスボタンのステータス 0=何もしていない 1=ボタンの押下 2=右マウスボタンの押下 (Windowsのみ) 3=両方のボタンの押下 (Windowsのみ)
*		指定した場合、グローバルの座標システムが使用される。 省略した場合、ローカルの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 < * > を省略すると、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 < * > を渡すと、座標はカレントプロセスの最前面のウィ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。

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 }) 数値

引数	タイプ	説明
項目テキスト	テキスト	定義された項目テキスト
デフォルト	数値	デフォルトで選択された項目番号
戻り値	数値	選択された項目番号

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MacOS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粹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 SPEED MENU」は、ナビゲーションスピードメニューをブルダウンします。

```

`「MY SPEED MENU」プロジェクトメソッド
GET MOUSE ($vIMouseX ; $vIMouseY ; $vIButton)
If (($vIButton=2) | (Macintosh control down))
    $vItems:="このデータベースについて...<|<|<-その他のオプション;-<
    For ($vITable ; 1 ; Count tables)
        $vItems:=$vItems+";" + Table name ($vITable)
    End for
    $vUserChoice:=Pop up menu ($vItems)
    Case of
        ¥ ($vUserChoice=1)
            `情報を表示する
        ¥ ($vUserChoice=2)
            `オプションを表示する
    Else
        If ($vUserChoice>0)
            `番号が$vUserChoice-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)

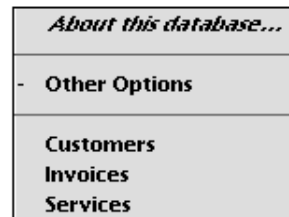
イベントを「スパイ」し、他のプロセスと通信するプロセス

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**関数の利点のひとつです。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。

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
*		指定された場合はローカルな座標システムが使用される 省略された場合はグローバ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<*>を渡した場合、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ィンドウに対して相対的に表されます。引数<*>を省略すると、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。

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。オプション引数 <*> を省略すると、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 <*> を渡した場合、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ィンドウに対して相対的に表されます。

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

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。

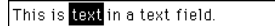
変数<先頭>に、反転表示された文字の先頭位置を代入します。変数<最終>に、反転表示された最終の文字位置に1を加えた値を代入します。

<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 以下の例は、フィールド“コメント”の反転表示された部分を検出します。**GET HIGHLIGHT**コマンドは、“v先頭”と“v最終”の2つの変数に値を代入します。以下の図の場合、“v先頭”に9が代入され、“v最終”に13が代入されます。

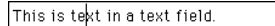
GET HIGHLIGHT (コメント ;v先頭 ;v最終) `コメントの反転表示部分を求める



フィールド上で反転表示されたテキスト

2. 以下の例は、以下の図のように、フィールド“コメント”に反転表示されたテキストがない場合です。この場合、変数“v先頭”と“v最終”の両方に11が代入されます。

GET HIGHLIGHT (コメント ;v先頭 ;v最終) `コメントの挿入ポイントを求める



フィールド上のテキスト挿入ポイント(カーソル)

3. 以下の例は、**Substring**関数を使用して反転表示されたテキストを取り出します。

GET HIGHLIGHT ([製品]コメント;v先頭 ;v最終)

If (v先頭 < v最終)

ALERT ("選択したテキストは : "+**Substring** ([製品]コメント;v先頭 t;v最終-v先頭))

End if

4. **FILTER KEYSTROKE**コマンドの例題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。

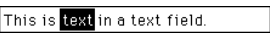
注：**HIGHLIGHT TEXT** コマンドをサブフォームのリストフォーム内にあるフィールドに対して使用することはできません。

<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

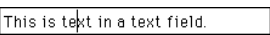
1. 以下の例は、フィールド“コメント”のテキストを反転表示します。結果を以下の図に示します。



フィールド上のテキストの反転表示

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。



フィールド上の挿入ポインタ(カーソル)の位置

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)

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	MacOS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

Last object

Last 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	最後、または現在の入力可エリアへのポインタ

説明

Last object関数は、最後または現在の入力可能エリアのポインタを返します。つまり、カーソルが現在位置するオブジェクト、または直前にカーソルが置かれていたオブジェクトへのポインタです。**Last area**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。**Last 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：この関数は、データ入力時にのみ使用してください。データ入力以外の箇所で**Last object**関数を使用すると、エラーが発生します。

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）。

```
$p:=Last object           ` ポインタを保存する
` データタイプが文字列またはテキストエリアの場合
If ((Type ($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
```

参照
なし

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。

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
  INVERT 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

この章では、「ルーチン」エディタの「User and Groups」テーマ内にあるユーザ&グループコマンドについて説明します。この章のコマンドは、パスワードやアクセス権を変更するために使用します。パスワードアクセスシステムに関する詳細は、『4th Dimension デザインリファレンス』を参照してください。

CHANGE ACCESS

CHANGE PASSWORD

Current user

DELETE USER

EDIT ACCESS

GET GROUP LIST

GET GROUP PROPERTIES

GET USER LIST

GET USER PROPERTIES

Is user deleted

Set group properties

SET USER PROPERTIES

User in group

Validate password

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、デザイナー、管理者、グループオーナーによって編集されます。デザイナーと管理者は、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。

デザイナーと管理者は、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。

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 ACCESS

CHANGE ACCESS

説明

CHANGE ACCESSコマンドは、データベースを終了しないでアクセスレベルを変更します。そして、ユーザがデータベース起動時に表示した同じ「パスワード」ダイアログボックスを表示します。ここで、ユーザは別のユーザ名として入力することができます。

「パスワード」ダイアログボックスの表示方法は、「データベースプロパティ」ダイアログボックス内の設定状況に依存します。

以下の例は、ユーザに対して「パスワード」ダイアログボックスを表示します。

CHANGE ACCESS

参照

CHANGE PASSWORD

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。

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

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。

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")>0)
        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

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。

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

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。

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」ウィンドウにはもはや現れません。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」ウィンドウにおいて緑（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」ウインドウ内に現れるユーザの名前
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」ウインドウ内に現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	デザイナー
2	管理者
3 ~ 15000	デザイナーによって作成されたユーザ （つまり、ユーザID番号3がデザイナー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 ; 最終日付 { ; グループID})

引数	タイプ	説明
ユーザID	数値	一意のユーザ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	スタートアップメソッドの名前
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
最終日付	日付	データベースに最後にログインした日付
グループID	数値配列	ユーザが属するグループの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	デザイナー
2	管理者
3 ~ 15000	デザイナーによって作成されたユーザ (つまり、ユーザID番号3がデザイナー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。

グループID番号	グループの種類
15000 ~ 32767	デザイナー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がデザイナー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。

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 ; 最終日付 { ; グループID}) 数値

引数	タイプ	説明
ユーザID	数値	ユーザアカウントの一意のID番号、または -1=デザイナーに関連したユーザの追加 -2=管理者に関連したユーザの追加 新しいユーザのユニークな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	スタートアップメソッドの名前
パスワード	文字列	新しい(暗号化されていない)パスワード、 または*を指定すると、パスワードは以前のまま
ログイン回数	数値	データベースにログインした回数
最終日付	日付	データベースに最後にログインした日付
グループID	数値配列	ユーザが属すグループの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、デザイナー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**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	デザイナー
2	管理者
3~15000	デザイナーによって作成されたユーザ (つまり、ユーザID番号3がデザイナー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)

デザイナー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。このコマンドを呼び出すと、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**コマンドは何も行いません。

このコマンドを呼び出す前に、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会ったとき) コマンドはなにもせず、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。このコマンドを呼び出す前に、引数<グループID>配列にそのユーザが属するグループの一意の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。

グループID番号	グループの種類
15000 ~ 32767	デザイナー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がデザイナー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」ウインドウ内に現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」ウインドウ内に現れるグループの名前と一意のID番号を割り当てます。

引数<グループ名>配列には、「パスワード」ウインドウ内に表示されるすべてのグループ名が代入されます。このグループ名には、使用できないグループ名（「パスワードエディタ」ウインドウに赤（レッド）で表示されるグループ名）も含まれます。

注：使用できないグループを見つけるには、**GET GROUP PROPERTIES**コマンドを使用します。

<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	デザイナー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がデザイナー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	デザイナー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がデザイナー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	デザイナー
2	管理者
3 ~ 15000	デザイナーによって作成されたユーザ (つまり、ユーザID番号3がデザイナー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	デザイナー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がデザイナー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=デザイナー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 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、デザイナー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	デザイナー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がデザイナー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)

デザイナー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。この両方のケースとも、新しいユーザを追加する際、4th Dimensionは使用不可になっている最初のグループを再利用しようとします。使用不可になっているグループがない場合にのみ新しいグループを作成します。このコマンドを呼び出すと、そのグループが正常に追加された場合は、その追加されたグループのID番号が<グループID>に返されます。

<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** コマンドは何も行いません。

このコマンドを呼び出す前に、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コマンドに渡します。

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。このコマンドを呼び出す前に、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	デザイナー
2	管理者
3 ~ 15000	デザイナーによって作成されたユーザ （つまり、ユーザID番号3がデザイナー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	デザイナー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がデザイナー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 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。

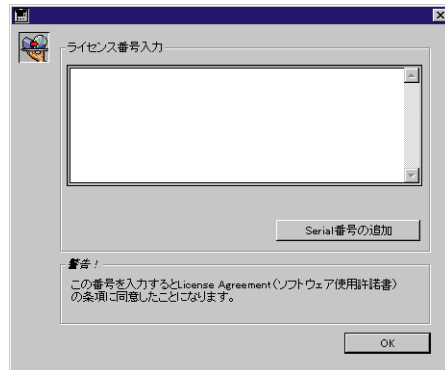
CHANGE LICENSE

CHANGE LICENSE

説明

CHANGE LICENSEコマンドは、「ACIライセンス」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って、ユーザはACIの拡張ライセンス（4Dサーバのみ）がシリアルライセンスを追加できます。

ACIライセンスのダイアログボックスです。



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、「ACI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「ACI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 WEB LICENSE コマンドは、顧客に配付されるコンパイルしてエンジンを組み込んだ4Dアプリケーションにライセンス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 LICENSE

参照

なし

この章では、「ルーチン」エディタの「Variables」テーマ内にある変数コマンドについて説明します。この章では、変数を操作するためのコマンドについて説明します。

SAVE VARIABLES	LOAD VARIABLES
CLEAR VARIABLE	Undefined

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。

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 - タベ - 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。

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 - タベ - 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ひとつの方法として、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アイテム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にセットする
```

参照

CLEAR VARIABLE

Webサービス：概要

4th Dimensionと4D Serverには、両方ともWeb上に4Dデータベースを公開できるWebサーバエンジンがあります。4D Webサーバエンジンには、以下のような非常に独創的な機能があります。

ダイレクトWebサービス

データベースは、直接Web上に公開されます。ユーザは、データベースシステム、Webサイト、またはその間のCGIインタフェースを開発する必要はありません。データベースがWebサイトになります。

接続セキュリティ

接続セキュリティが、新しいオプションと新しいデータベースメソッド「On Web Authentication」により強化されました。“一般Webユーザ”オプションは、データベース内のアクセス管理を単純化します。さらに、デフォルトHTMLの設定とルートフォルダを指定することで、ディスク上のファイルへのアクセスを制御できます。

カスタマイズセッション

Webサーバ用に多くのパラメータが付加されました。

オンラインの透過的なHTML変換

4Dはオンラインで透過的、且つダイナミックにフォームと設計コンポーネントをHTMLページに変換します。変換された新しいHTMLページは、データベースに既に接続されている場合でも、その場でWebブラウザから利用可能になります。現在では、ほとんどのWebデータベースシステムは、CGIベースのシステムまたはスタティックなHTMLベースのシステムです。

CGIシステムでは、データベース、Webサイト、CGIを開発する必要があります。スタティックHTMLベースのシステムでは、データベースの設計コンポーネントを変更するたびに、その変更をHTMLに再変換するユーティリティプログラムを実行する必要があります。どちらの場合でも、Webコンポーネントはオフラインで作成され、データベースやWeb開発者からの手動での介入が必要です。これに対して、4th Dimensionでは、必要な時に設計コンポーネントを変更することができます。そして、「デザイン」モードで変更を保存すると、変更は透過的にWebブラウザですぐに利用可能になります。このため、

アプリケーションの開発とテストを行っているときに、既に接続されているWebブラウザでその結果をすぐにテストすることができます。

レコードおよびデータへのダイナミックアクセス

4Dは、Webブラウザを4Dデータベースエンジンの標準クライアントとして扱います。例えば、ローカルな4th Dimensionデータベース上、または4D Serverのクライアントワークステーションからレコードを変更する場合でも、それらのレコードはWebブラウザで即座に利用可能になります。他のシステムのように、HTML発行のためにレコードを再処理する必要はありません。

セッション保守とデータベースコンテキスト

Webブラウザは、その名前が示す通り、ランダムにWebページをブラウズすることができます。ユーザは、あるページから別のページへ、ひとつのWebサイトから別のサイトへとジャンプできます。クライアント/サーバデータベース関連でWebブラウザ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にブラウザを合わせる必要があります。例えば、レコードを追加しているのであれば、レコードデータ入力における妥当性検査および取り消し処理の規則に従う必要があります。ユーザはブラウザナビゲーション制御経由では、レコードを終了させて、レコードを不確かな状態のままにすることはできません。4D Webサーバエンジンには、内蔵のセッション保守とデータベースコンテキスト保守が含まれています。WebページのURL全体には、独自のコンテキストID番号とサブコンテキストID番号が保守されます。これらの番号により、ブラウザで現在表示されているWebページと4D側のデータベース接続のコンテキストとの完全な同期が保証されます。

非コンテキストモード

非コンテキストモードは、4D 6.5 Webサーバの主要な改革です。コンテキストの管理から開放し、4D 6.5 Webサーバを標準のHTTPサーバにします。このモードでは、「セミダイナミック」ページを使用することで4D能力の利点を発揮できます。

透過的なマルチユーザ保守

Webブラウザが4Dのクライアントとなる場合、完全なデータベースクライアントとして扱われます。例えば、Webブラウザでレコードの変更を開始すると、4Dはレコードを自動的にロックして他のクライアントがレコードを同時に変更しないようにします。ユーザがデータ入力の妥当性検査やキャンセルを行った後に、4Dはレコードのロックを自動的に解除します。さらに、4th Dimension、または4D Clientを使用している時と同様に、トランザクション下でのデータ入力を行うこともできます。

Webプロセスの保守

4Dには、4DのWebクライアント/サーバアーキテクチャを処理する複数のプロセスがあります。内蔵のメインWebサーバプロセスは、Webへの接続処理を扱います。Webブラウザがデータベースへのアクセス権を付与されると、Web接続は接続目的のために自動作成された独立プロセスで実行を開始します。完全に統合された4Dのマルチタスクアーキテクチャの結果として、通常のクライアントやWeb 4Dクライアントは、データベースエンジンで平行して処理されるデータベースリクエスト（照会等）を同時に実行することが

できます。また、クライアントも、特定のWebクライアントから発信されたリクエストは、他のプロセスのコンテキストに介入しないことが保証されています。

最適化されたWeb Serverアーキテクチャ

4D Webサーバエンジンには、4Dデータベースエンジンと同じ機能があります。例えば、配列に一連のレコード値をロードする場合に、そのロード処理はWebサーバマシンでローカルに実行されます。その後、リクエストの結果が全体としてリクエストを出したWebクライアントに送信されます。

さらに、Web接続は、Webサーバ側の単純、且つ完全な機能を備えた4Dプロセスなため、お好みの4Dアルゴリズムを実行できます。そのアルゴリズムはWebサーバ側でローカルに実行され、結果がある場合には、その結果だけがWebブラウザに送られます。例えば、リレート、セット、合計結果の計算を含むクエリ（照会）を実行すると、結果だけがWebブラウザに返されます。4Dアプリケーションはコンパイルできるため、Webのエキスパートにならなくても複雑なWebデータベースシステムを構築することができます。

HTMLとJavaScriptのカプセル化

4DはWeb上にデータベースを公開するために必要なほとんどすべての作業を実行するのですが、HTMLとJavaScriptのコードのカプセル化によって4D開発をカスタマイズすることもできます。例えば、自分のデータベース / Webサイトを目立つHTMLページにしてホームページをインパクトのあるものにすることもできます。

ユーザはカスタムHTMLページ全体を構築してそれを**SEND HTML FILE**と**SEND HTML BLOB**コマンドでWebに公開することができます。また、HTMLを4Dフォームにカプセル化して、Webブラウザ上で4DとHTMLオブジェクトが混在するような4Dフォームを作成することもできます。カプセル化されたHTMLの内部では、JavaScriptコードを実装して、リクエストをサーバに返すことなくクライアントのWebブラウザ側でアクションやデータ制御を実行できます。

HTMLと4Dオブジェクトのバインド

4D開発において、カプセル化したHTMLコードを使用する場合には、HTMLオブジェクトに入力された値とデータを4D側で取得できる必要があります。複雑なHTML解析ルーチンをユーザが作成するのではなく、4Dには、HTMLオブジェクトを4D変数にバインドするための簡単な組み込みシステムが用意されています。つまり、HTMLオブジェクトと4D変数に同じ名前を付けるだけでバインド設定が行えます。結果として、HTMLリクエストの分析や応答は非常に実現しやすくなります。Webブラウザから戻ってきたデータが自動的に入る4D変数を処理する4Dコードを作成する必要があるだけです。

次は何をするのか？

Web公開用にマシンとデータベースを設定するには、「Webサービス：システム設定」の節を参照してください。

Web上にデータベースを公開する方法を習得するには、「Webサービス：入門編（パートI）」と「Webサービス：入門編（パートII）」の節を参照してください。

アクセスマネジメントシステムの詳細については、「Webサービス：接続セキュリティ」の節を参照してください。

HTMLカプセル化の詳細については、「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。

Webとプロセスのやり取りの詳細については、「Webサービス：「Web接続」プロセス」の節を参照してください。

非コンテキストモードの詳細については、「Webサービス：非コンテキストモード」の節を参照してください。

参照

SEND HTML FILE、SET HTML ROOT、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER、Web Services、Connection Security、Non Contextual Mode、Web Server Settings.

Webサービス：システム設定

4th Dimension 6.5と4D Server 6.5には、Web上でデータベースに透過的、且つ動的にサービスを実行するWebサービス機能があります。

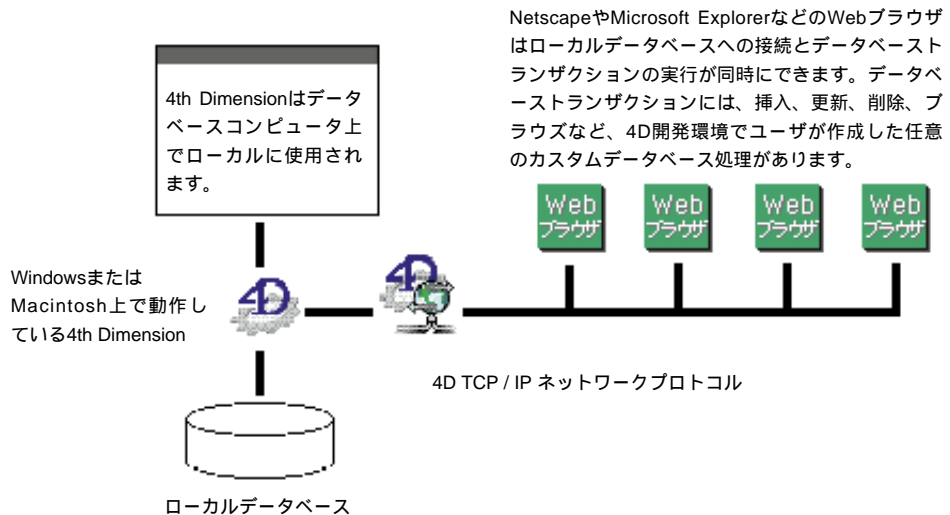
4th DimensionとWeb

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

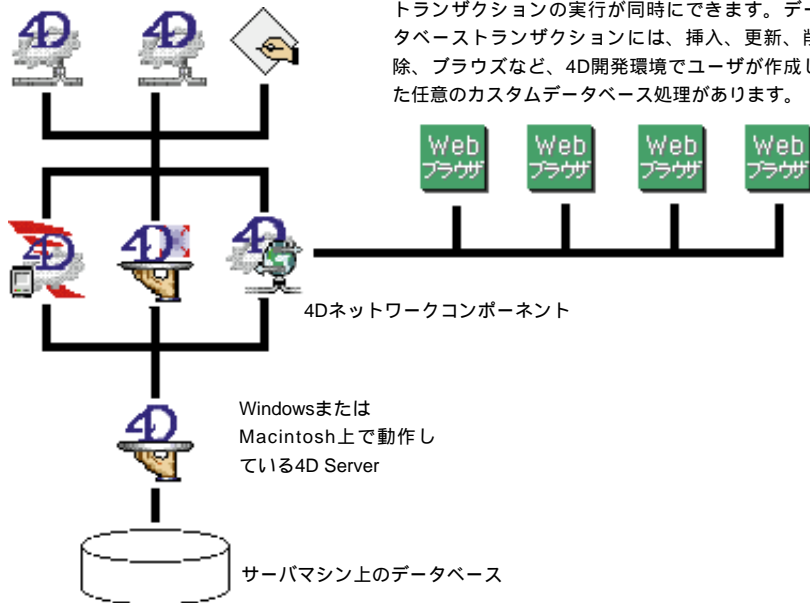
4D Openをベースにしたアプリケーション

Webブラウザ

これを以下の図にまとめます。

4D Clientと4D Openベースのワークステーションは、TCP/IP、Novell IPX、Apple ADSPプロトコルのいずれかを使用して同時に接続できます。

NetscapeやMicrosoft ExplorerなどのWebブラウザはローカルデータベースへの接続とデータベーストランザクションの実行が同時にできます。データベーストランザクションには、挿入、更新、削除、ブラウズなど、4D開発環境でユーザが作成した任意のカスタムデータベース処理があります。



Web上での4Dデータベースのサービス

4th Dimensionや4D Serverを使用してWeb上に4Dデータベースを提供するには、以下に示す適切な接続ライセンスとネットワークコンポーネントが必要です。

必要な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る必要があります。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。

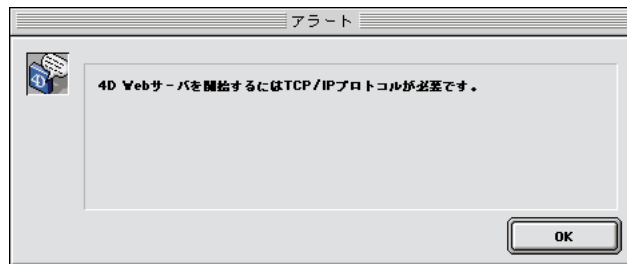
Web接続は、TCP / IPプロトコルを使用してネットワーク上に構築されます。
したがって、

- ・ 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。
- ・ 4D TCP / IP Networkコンポーネントをインストールしておく必要があります。Macintoshでは、このネットワークコンポーネントを4th Dimension、4D Server、または4D マージ済カスタムアプリケーションに直接インストールする必要があります。Windowsでは、このネットワークコンポーネントをアクティブなWINDOWSディレクトリのACI\NETWORKディレクトリにインストールする必要があります。

注：どちらの場合も、詳細については『4D Serverネットワークコンポーネント』マニュアルを参照してください。

TCP / IPのインストールまたはチェックを行った後は、4D内部からWebサービスを起動する必要があります。これについては、以下の節で説明します。

TCP / IPがまだ稼働していない場合、または4D TCP / IPネットワークコンポーネントがまだ存在していない場合に4D Webサービスを起動しようとする、4th Dimensionは以下の警告を表示します。

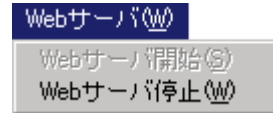
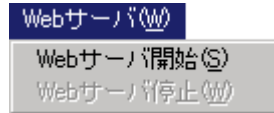


このメッセージが表示された場合には、説明されている通りにインストレーションを実行するか、TCP / IP設定のトラブルシューティングを行ってください。

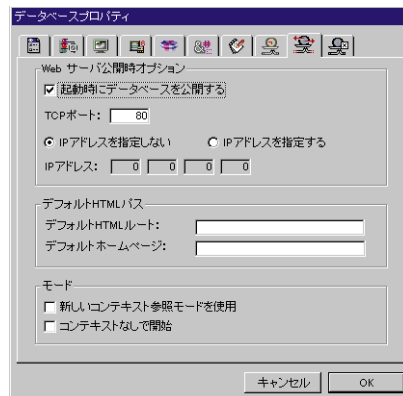
4D Webサービスの開始

4D Webサービスは、以下の3つの異なる方法で起動することができます。

4D Serverまたは4th Dimension「ユーザ」モードのメインメニューバーから「Webサーバ」メニューを使用する方法。図のように、「Webサーバ」メニューでは都合のよいときにWebサービスの開始と終了ができます。



データベースがオープンされるたびに自動的にデータベースを公開する方法。Web上にデータベースを自動的に公開するには、4D Serverまたは4th Dimensionの「デザイン」モードの「ファイル」メニューから「データベースプロパティ...」メニューコマンドを選択します。以下のような「データベースプロパティ...」ダイアログボックスが表示されます。**Web サーバ I/O**のタブ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でオープンするたびにデータベースはWeb上に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サービスの中断と再起動は「Webサーバ」メニューを使用する、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で何回でも必要なだけ行えます。

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.aci.co.jp"）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：123.4.567.89）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。

この時、何も問題がなければ接続できます。接続できない場合には、以下の状況のいずれかが考えられます。

1. "...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または4D Serverが稼働しており、そのWebサービスを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3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. 接続はできたが、Webページに"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）というメッセージが表示される場合。

このメッセージは、4D Webサービスが稼働しており、データベースに正しく接続できたことを示しています。ただし、データベースがWeb経由で操作可能な状態になるために、いくつかの最低限のコンポーネントを含む必要があります。詳細については、「Webサービス：入門編（パートI）」の節を参照してください。

3. 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。

既に独自のWebサービスを稼働しているWindows NT 4.0システム上で、1つだけの4D Webデータベースを稼働している場合。

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、以下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「Webサービス：接続セキュリティ」を参照してください）。

TCPポート番号を特定の値に設定する

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。変更するには、4D Serverまたは4th Dimensionの「デザイン」モードの「ファイル」メニューから「データベースプロパティ...」メニューコマンドを選択します（前述の図を参照）。「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、「123.4.567.89:8080」と指定します。

警告：デフォルトの80以外のTCPポート番号を使用する場合には、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（自分がしていること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。デフォルトのTCPポート番号とプロトコルの詳細については、TCP/IPプロトコルの解説書でRFC 1700標準の割り当て番号表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。何千番目かのポート番号を使用すれば他に影響しないでしょう。

参照

SEND HTML FILE、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER

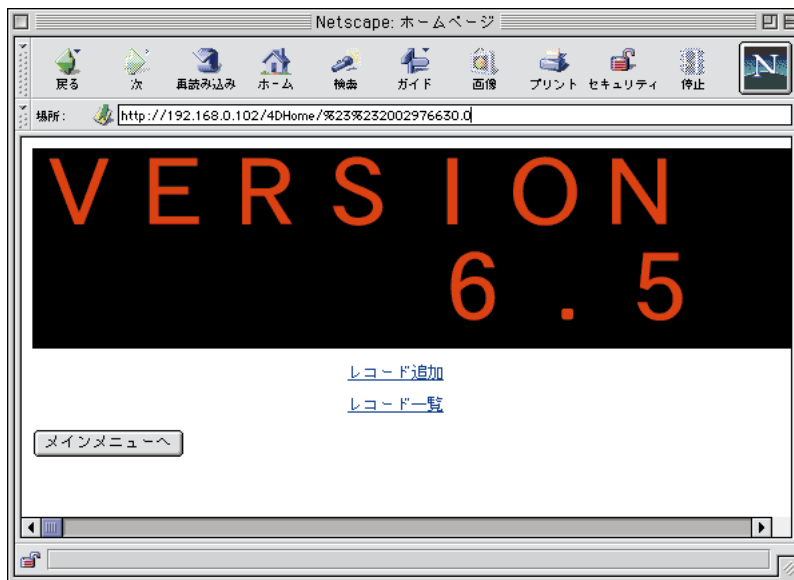
Webサービス：入門編（パートI）

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。

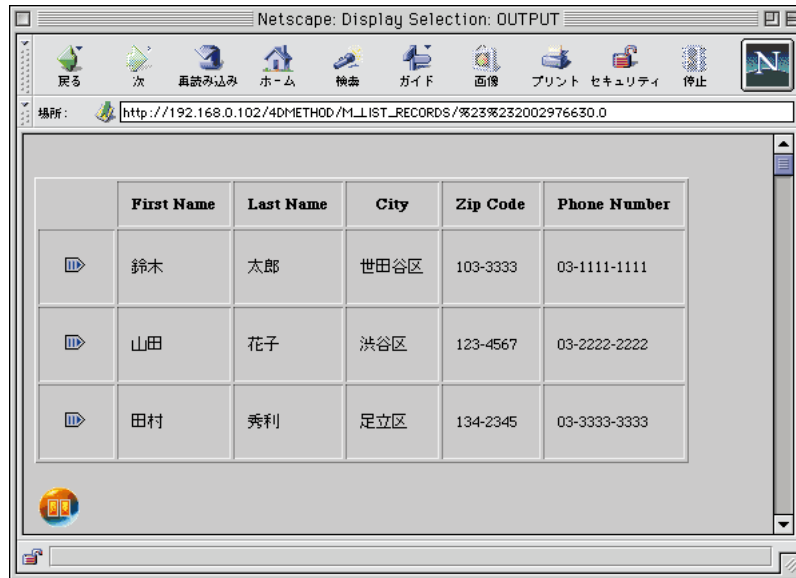


注：ブラウザ上で日本語を表示するには、必ず「データベースプロパティ」ウインドウの「接続設定」タグにある「文字セット」を「Shift-JIS (Japanese)」に設定してください。この設定を行わないと、ブラウザ側で、日本語が文字化けしてしまいます。

2. レコードの表示とブラウズ

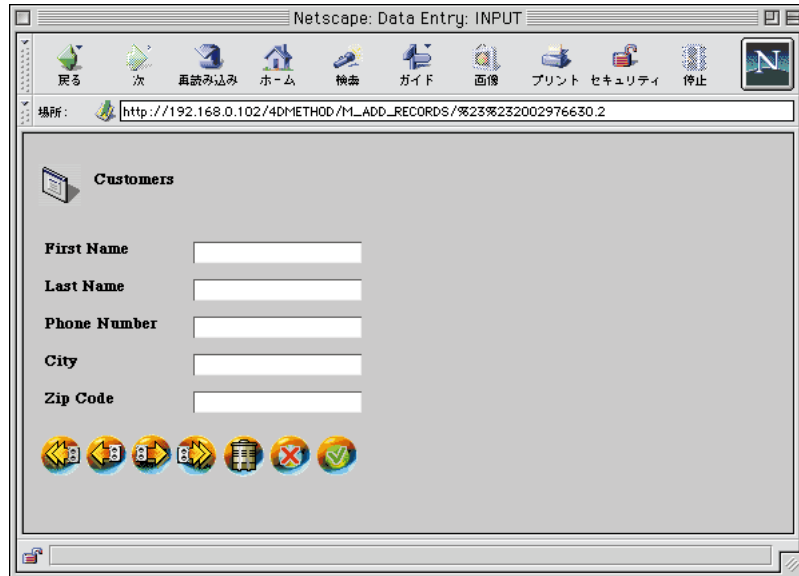
リンクされている「レコード一覧」テキストをクリックします。これによって、4Dの **DISPLAY SELECTION** コマンドで表示した画面と同等のWebの画面が、以下のように表示されます。

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

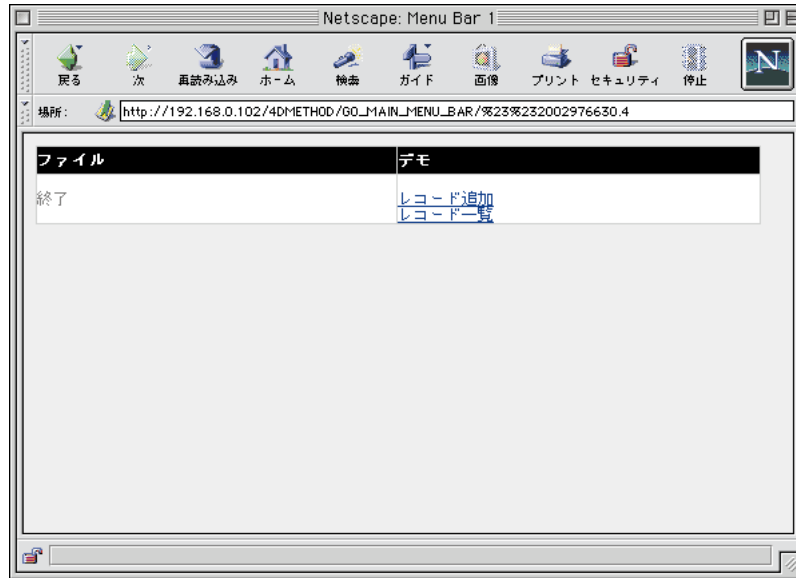
Webサイトのホームページで、リンクされた「レコード追加」テキストをクリックすると4Dの**ADD RECORD**コマンドで表示した画面と同等のWebの画面が、以下のように表示されます。



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、4Dは「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4Dは以下のアクションを実行します。

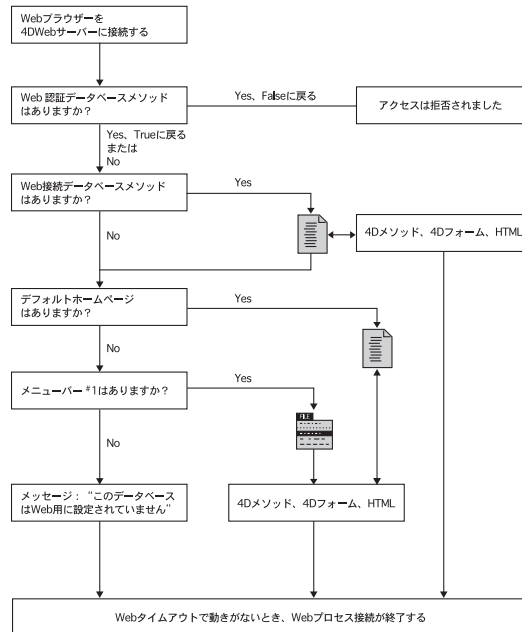
「On Web Authentication」データベースメソッドが存在する場合は、それを実行します。このデータベースメソッドがTrueを返すか存在しない場合、「On Web Connection」データベースメソッドが存在すれば、それを実行します。

「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。

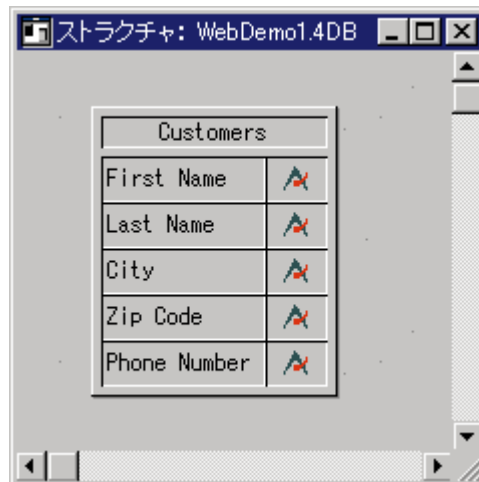
ただし、このモードで4Dにはクライアント/サーバの接続解除アクションと同等の、セッションを終了させるものを提供する方法がありません。このため、クライアント/サーバセッションの終了はタイムアウト方式によって処理されます。Web接続を処理する4Dプロセスは、データベースのWebタイムアウト設定と同じだけ遅延時間が経過しても何の動作も検出されないと終了します。

1つになったデータベースとWebサーバ

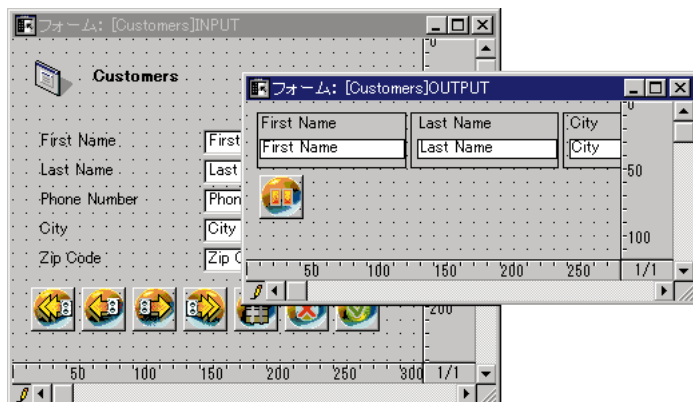
4Dメニューバー、フォーム、メソッドを使用すると、4D Webサーバセッションを完全に管理することができます。前述の例では、レコードの一覧表示と追加処理が簡単な4Dメソッドとフォームで実行されました。HTMLホームページがなかった場合には、Webブラウザには接続上でメニューバー#1が表示されていました。

HTMLホームページを省略した場合には、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。

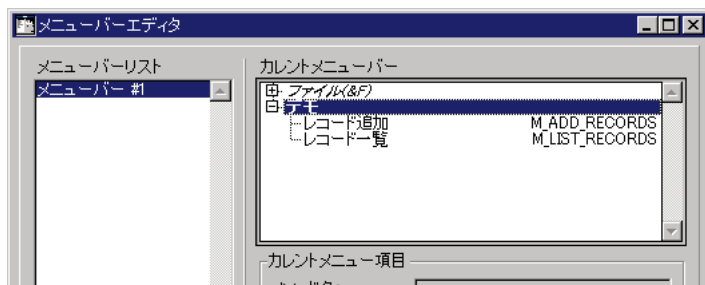
1. 以下の図はサンプルデータベースのストラクチャを示します。



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。



これで終了です。

5分もあれば、ローカルで操作できるデータベースであり、しかもイントラネットネットワークやインターネットに公開できるWebサーバでもある4Dデータベースの作成が完了します。

以下の節の「Webサービス：入門編（パートII）」を参照してください。

参照

SEND HTML FILE、SET WEB DISPLAY LIMIT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、START WEB SERVER、STOP WEB SERVER

Webサービス：入門編（パートII）

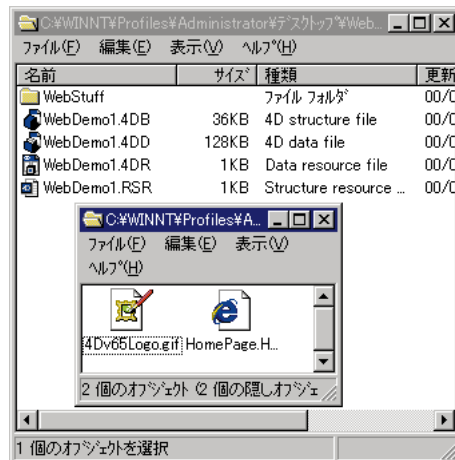
データベースにHTMLを追加する

Webユーザに対してデータベースのメニューバー#1以外にもやり取りをさせたいと考えているのであれば、「On Web Connection」データベースメソッドに機能を追加し、4DフォームもHTMLページも表示することができます。または、デフォルトの修正されたHTMLホームページを表示するようにWebサーバを設定できます。任意のHTMLツールで作成した既存のサイトにあるHTMLページを再利用できます。

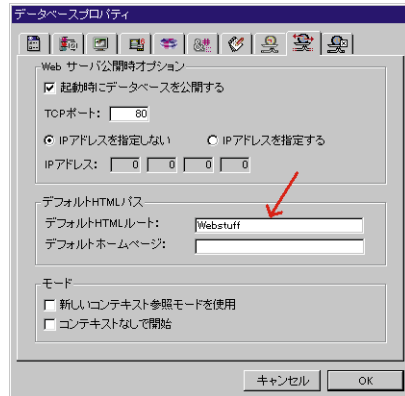
4DベースのWebサイトは、完全な4Dベースのシステムか、または4DフォームとHTMLページの組み合わせにすることができます。4DデータベースでHTMLページを使用する際に注目すべき点は、4DとHTMLの両方の開発環境から長所を得るということです。もちろん、必要がなければ、HTMLページを使用する必要はありません。

例題

この例は、既存のHTMLページをデータベースに追加します。以下の図は、データベースのディレクトリを示しています。



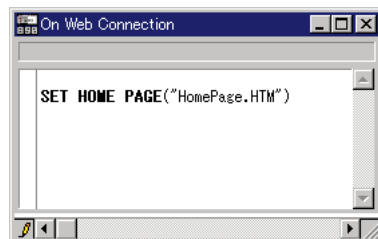
データベース用のホームページとしてHTMLドキュメント「HomePage.HTM」を使用します。ドキュメント「4DV6Logo.GIF」はこのHTMLドキュメントで使用されているピクチャです。この場合のWebサーバ定義は以下の通りです。
データベースプロパティの“ Web サーバ ” ページで、新しいデフォルトのHTMLルートを入力します。



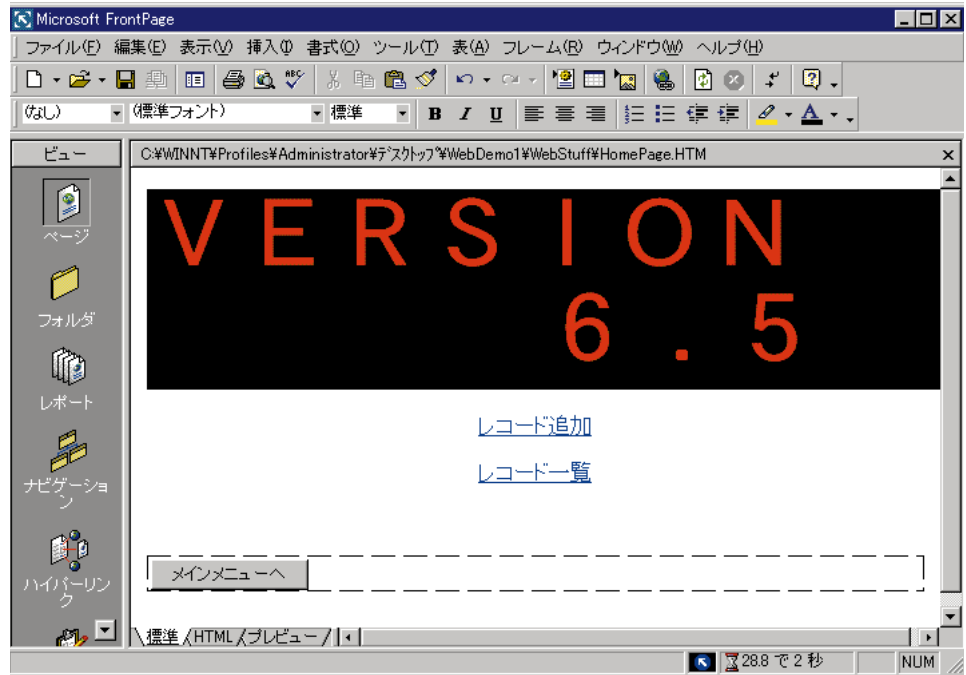
デフォルトのHTMLルートエリアは（デフォルトで）4th DimensionにHTMLドキュメントがある場所を教えます。

注：ダイアログボックスについての詳しい説明は「Webサービス：Webサーバセッティング」を参照してください。

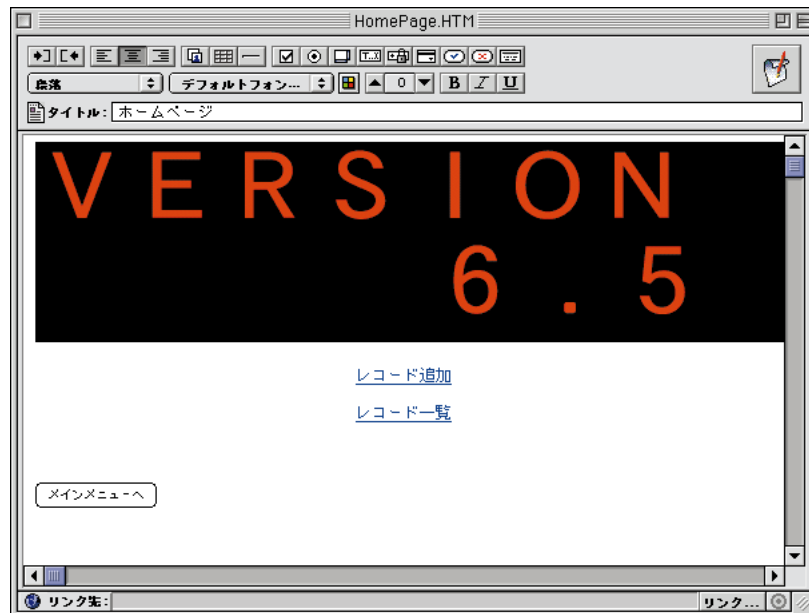
HTMLドキュメントは以下に示す**On Web Connection**データベースメソッドで、カレントWebプロセスのホームページとして宣言されます。



以下の図は、Microsoft Front PageでHomePage HTMLを表示させたものです。



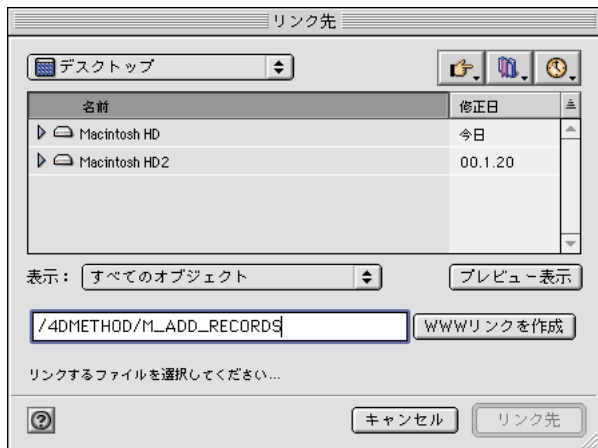
以下の図は、同じHTMLドキュメントをAdobe PageMillで表示したものです。



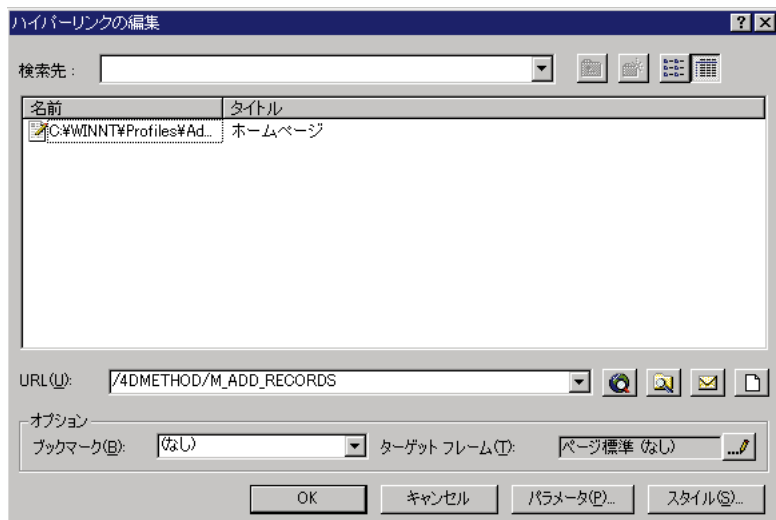
URLのリンク

「レコードの追加」と「レコード一覧」という2つのリンクされたテキスト項目は、4Dプロジェクトメソッドの「M_ADD_RECORDS」と「M_LIST_RECORDS」の実行をそれぞれのURL経由でトリガします。その表記方法は非常に簡単で、HTMLオブジェクトはURL "/4DMETHOD/メソッド名"で自分のデータベースのプロジェクトメソッドにリンクできます。

以下の図は、Adobe PageMillでのテキスト"レコード追加"へのURLです。



以下の図は、前述と同じURLをMicrosoft Front Pageで指定する場合です。



これらのリンクが定義された後、WebブラウザがURLを送り返すときに、4Dは「/4DMETHOD/」キーワードの後に指定されているプロジェクトメソッドを実行します。

プロジェクトメソッドが完了すると、実行をトリガしたHTMLページに戻ります。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ことに注目してください。

ボタン

この例のHTMLドキュメントには、レコードのサブミットに使用するボタンが含まれています。HTMLのボタンには、ノーマル、サブミット、リセットという3種類があります。

「ノーマル」ボタン：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」ボタン：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でフォームをWebサーバにサブミットします。このボタンは、普通の4DフォームよりもHTMLページ経由でデータ入力を実行する方がよい場合に便利です。

「リセット」ボタン：リセットボタンは、4D開発環境の中ではそれほど便利ではありません。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、サーバに対してはリクエストは送信しません。

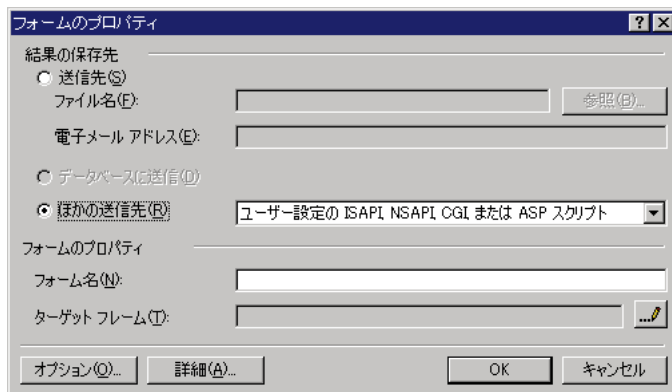
HTMLページを4Dに統合する場合、最も多く使用するのは「ノーマル」ボタンと「サブミット」ボタンです。

実行する4Dメソッドの指定

4D側からHTMLフォームをサブミットするには、フォームがサブミットされた後で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。

Microsoft Front Pageを使用して、POSTアクション4Dメソッドを指定するには以下のように行います。

1. ボタンを含むフォームの「フォームプロパティ」を表示させる。



2. オプションボタンをクリックする。以下のダイアログボックスが表示されます。



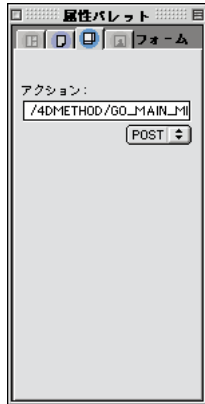
3. 「処理」入力エリアに、アクションとして「/4DMETHOD/メソッド名」を指定する。

ここで、「メインメニューへ」サブミットボタンがクリックされた時に実行される4Dメソッドとして「GO_MAIN_MENU_BAR」を入力します。

4. 「Method」ドロップダウンリストから「POST」を選択する。

Adobe PageMillを使用してPOSTアクション4Dメソッドを指定するには、以下のように行います。

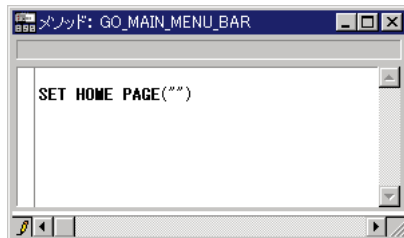
1. 「ウインドウ」メニューから「属性パレットを表示」を選択する。
以下のダイアログボックスが表示されます。



2. ポップアップメニューから「POST」を選択する。
3. アクションとして「/4DMETHOD/メソッド名」を指定する。

プロジェクトメソッド

以下は、4Dプロジェクトメソッド「GO_MAIN_MENU_BAR」です。



この例で、このメソッドの目的はたった1つだけです。その目的とは、Webブラウザで表示されているカレントのデフォルトのホームページから抜けて、カレントのメニューバーに送られるということです。4Dはデータベースのメニューバー#1に切り替わります。

これで終了です。5分間で、Webページを設計して、「GO_MAIN_MENU_BAR」4Dプロジェクトメソッドを追加することで、クライアント/サーバ機能をHTML開発環境に組み合わせたデータベースとWebサーバを持つことができます。

次は何をするのか？

Webサーバの設定についての詳しい説明は、「Webサービス：Webサーバセッティング」を参照してください。

4DにHTMLフォームとコードを統合する方法に関する詳細は、「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。

最初の4D Webサーバを設定している時に問題が発生した場合には、「Webサービス：システム設定」の節を参照してください。

参照

SEND HTML FILE、SET HTML ROOT、SET WEB TIMEOUT、START WEB SERVER、STOP WEB SERVER

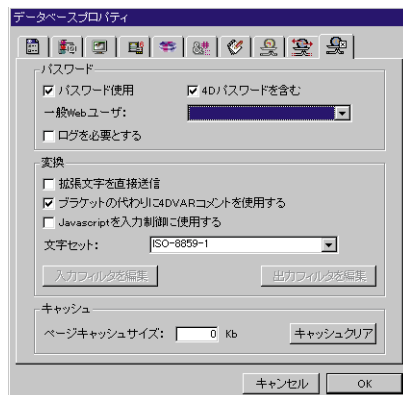
Webサービス：接続セキュリティ

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。

- ・「データベースプロパティ」ダイアログボックスのオプションのWebパスワードとOn Web Authenticationデータベースメソッド組み合わせ。
- ・ "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

Webアクセス用のパスワードマネージメントシステム

「データベースプロパティ」ダイアログ内で、Webサーバに適用したいアクセス管理システムを指定することができます。「データベースプロパティ」ダイアログ内で、“WebサーバII”タブをクリックすると以下のウィンドウが現れます。



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。

4D Web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。

「データベースプロパティ」ダイアログ内の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

以下場合があります。

オプションが何も指定されていない場合

On Web Authentication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

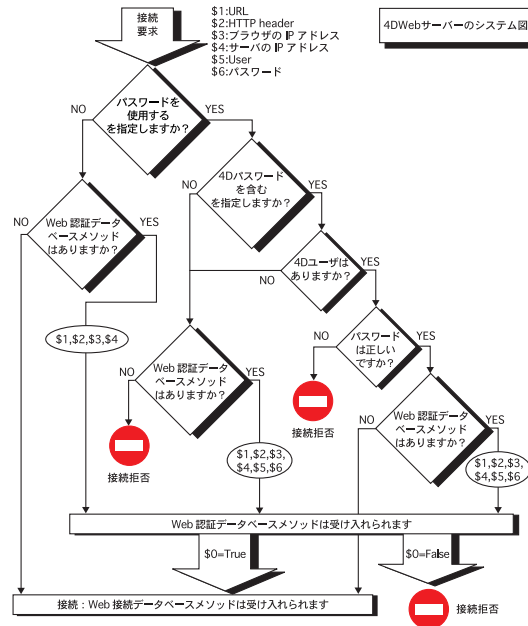
ブラウザによって送られたユーザ名が4Dのユーザとして登録されており、パスワードが正しければ、接続は受け入れられます。パスワードが誤っていれば接続は拒否されます。

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いない場合、2つの可能性があります。

On Web Authentication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、彼らにどの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。

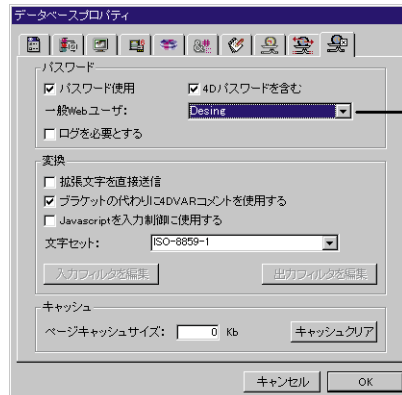
一般Webユーザ

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. デザインモードで、パスワードエディタ内にユーザを作成する。
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. 「データベースプロパティ」ダイアログで、“Web サーバII”タブをクリックする。
以下のようなページが現れます。



4Dのパスワードテーブルで
定義されたユーザのリスト

デフォルトでは、一般ユーザはデザイナーであり、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. 「一般Webユーザ」リスト内のユーザを選び、OKボタンを押す。

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

データベースプロパティのこの新しいオプションは、4Dがブラウザに送るスタティックHTMLファイルとピクチャが存在するフォルダの定義を可能にします。

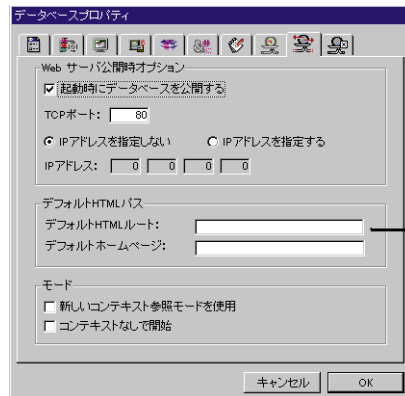
更に、HTMLルートフォルダより上位の階層へのアクセスができないように定義したことになります。

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。

デフォルトでは、HTMLルートフォルダは、データベースのストラクチャファイルがあるフォルダです。この場合には、アクセス制限が何もありませんので注意が必要です（ユーザはすべてのボリュームにアクセス可能です）。

デフォルトHTMLルートフォルダを定義するには、

1. 「データベースプロパティ」ダイアログで、「Webサーバ」タブをクリックする。
下記のページが現れます。



デフォルトのHTMLルート
フォルダ入力エリア

2. "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アクセスパスを入力する。

このダイアログボックスに入力するアクセスパスは、データベースのストラクチャファイルがあ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のフォルダにしたい場合を除きます。下記を参照してください）。

例えば、HTMLルートフォルダを"4Ddatabase"フォルダ内の"Web"サブフォルダにしたい場合は、"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。

3. OKボタンを押す。

注：HTMLルートフォルダが、「データベースプロパティ」ダイアログで変更された場合、キャッシュはクリアされ、アクセスが制限されているファイルを保存しないようになります。

データベースのプロパティとSET HTML ROOT

一旦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**コマンドでの設定は、データベースプロパティに定義されているデフォルトHTMLルートフォルダを考慮します。「データベースプロパティ」ダイアログで定義されたフォルダが“WebPages”の時、**SET HTML ROOT** (“Folder”)を実行すると、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（より詳しい情報は、「Webサービス：非コンテキストモード」参照してください）。

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

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サーバへのアクセス管理の役目があります。このデータベースメソッドは、Webブラウザがデータベースに接続を試みるたびにコールされます。

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。
これらの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
\$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)

` 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、「データベースプロパティ」ダイアログ内で設定したオプションに基づきます（「Webサービス：接続セキュリティ」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「123.4.567.89」と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
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

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。

・HTTPリクエストのヘッダ

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IPアドレスが渡されます。バージョン6.5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、「データベースプロパティ」ダイアログ内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。

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：**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。

4D Webサーバへの接続を判断するシステムは、「データベースプロパティ」ダイアログボックスのWebパスワードのオプションと**On Web Authentication**データベースメソッドの組合せによります。この点は、「Webサービス：接続セキュリティ」で詳細に説明しています。

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。

`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
```

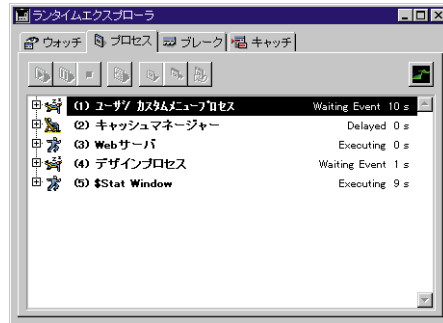
参照

On Web Connectionデータベースメソッド、URLでコールした4Dメソッドに渡されたテキスト引数、Webサービス：接続セキュリティ。

Webサービス：「Web接続」プロセス

「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」プロセスは稼働中で実行中の3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。

「ユーザ」モードの「Webサーバ」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。

「データベースプロパティ」の「起動時にデータベースを公開する」チェックボックス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。

「ユーザ」モードの「Webサーバ」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ことです。

「Webサーバ」プロセスを停止する時に、オープンされている「Web接続」プロセスが存在する場合には、WebユーザがWebサーバ接続タイムアウト（「データベースプロパティ」ウィンドウの「接続」ページでの設定、または**SET WEB TIMEOUT**コマンドをプログラムで使用して設定）と同じかそれを超える時間が経過してもデータベースの照会を行わなくなるまで、これらのプロセスは実行を継続します。

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。

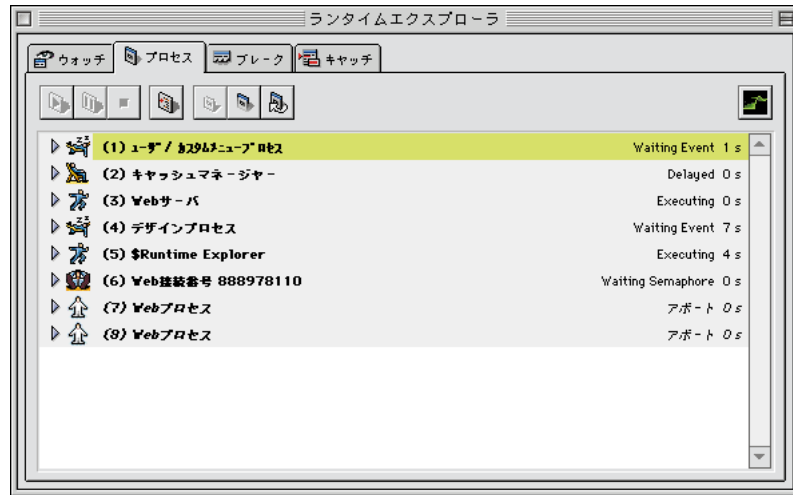
最初に、いくつかの一時的なローカル4Dプロセスを作成して、Webブラウザとの接続を評価し、管理します。

注：これらの一時的なプロセスはすべてのHTTPクエリを管理します。非常に高速に実行した後アポ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Webプロセスを5秒遅延させ、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。

コンテキストが作成されていることをコマンドクエリ必要とする場合、新規接続で利用できるリソースが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」ウィンドウは、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888978110 ” を表示しています



また、上の図の中で、7番目と8番目のアボートしたプロセスにも注目してください。このプロセスは「Webサーバ」プロセスによって起動され、停止されたものです。このプロセスがWeb接続の初期化を処理したのです。

注：コンテキストの管理についての詳しい説明は、次節の「Web 接続コンテキストの管理：コンテキストモード」を参照してください。

- ・ コンテキストを作成することをクエリが必要としない場合（例えば、データベースが非コンテキストモードで開始されている場合）、コンテキストは作成されません。必要であればクエリが処理され、結果がブラウザに返されます。
- ・ 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。

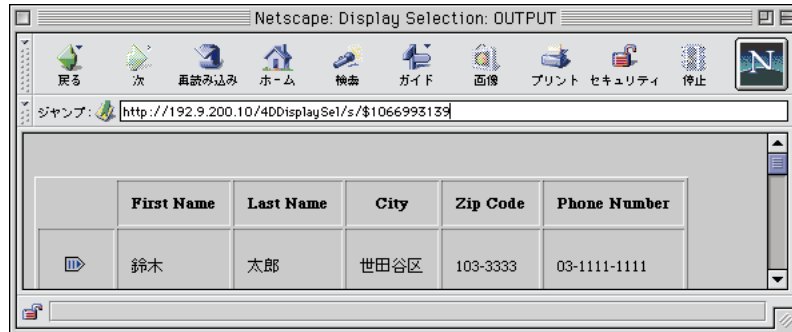
逆に言えば、セッション中に、接続を非コンテキストモードからコンテキストモードへスイッチすると、番号付きのWeb接続プロセスが作成されます。

注：非コンテキストモードについての詳しい説明は、「Webサービス：非コンテキストモード」を参照してください。

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ブラウザウィンドウでは、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ネットワークコンポーネント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接続経路で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ます。

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

上記のウィンドウでは、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、履歴ウィ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。これらはデータバ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“ Using browser navigation controls, you left a form requiring data validation. 4th Dimension will now return to that form so you can accept or cancel it ”(ブラウザのナビゲーションコントロール等(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ID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。

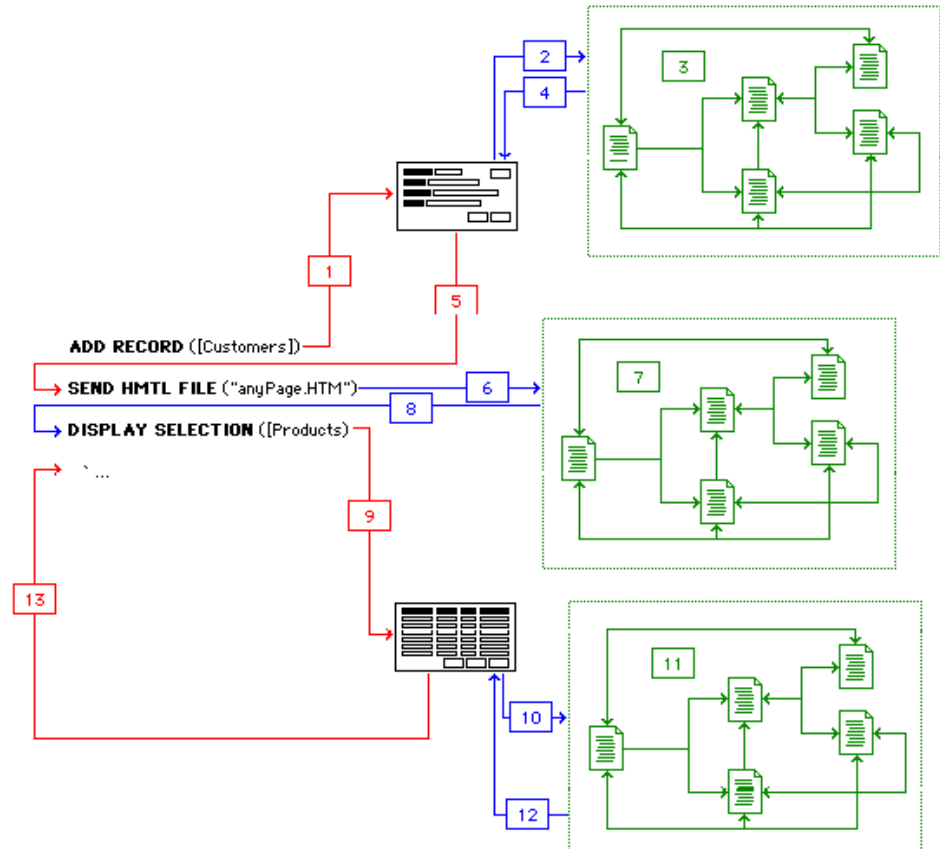
ADD RECORD ([顧客])
SEND HTML FILE ("anyPage.HTM")
DISPLAY SELECTION ([商品])

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の実行されます。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、データ入力を受け付けられるか、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。

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ひと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訳ではありません。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。

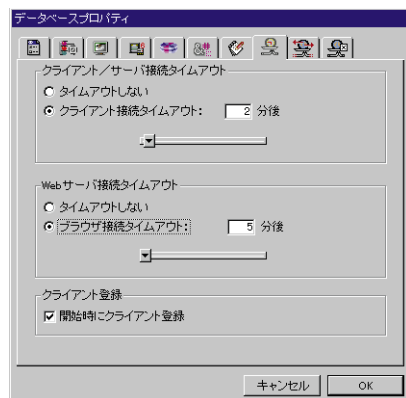
どんな種類のデータ入力でも、その間、メインメニューバーはブラウザのものであり、4Dのメニューバーではありません。フォーム内では、4Dメニューバーを頼りにしないでください。4DメニューバーはWebサーバマシン上にあり、Webブラウザマシン上にはありません。

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（例：オブジェクトタイプやフォームイベント）をすべて使用できるとは思わないでください。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。後者の場合には、「Web接続」プロセスは、「データベースプロパティ」ウインドウ（以下の図参照）で設定するか、**SET WEB TIMEOUT**コマンドでプログラムによって設定されるWebサーバ接続タイムアウトと同じかそれを超える迄、待機します。



「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「ランタイムエクスペローラ」で利用できます)。

「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、Webブラウザがコンテキストモードのデータベースへの接続を開始するたびに、またはWebサーバがコンテキストの作成を必要とした要求を受けた時に4th Dimensionまたは4D Serverから自動的に呼び出されます (下記の「On Web Connection データベースメソッドの呼び出し」を参照してください)。
もちろん、接続は (もしあれば) **On Web Authentication** データベースメソッドによって前もって許可されていなければなりませんし、データベースが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
\$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「123.4.567.89」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
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/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

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。

Windows NT上で稼働するNetscape 3.0では、以下のようなヘッダを受け取ります。

```
GET HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (WinNT; I)
Host: 192.9.200.11
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

Windows NT上で稼働するMicrosoft Internet Explorerでは、以下のようなヘッダを受け取ります。

GET / HTTP/1.0

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0d; Windows NT)

Connection: Keep-Alive

If-Modified-Since: Sunday, 10-Dec-96 01:51:37 GMT

アプリケーションがこの情報を扱う場合には、ヘッダ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。バージョン6.5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、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、「データベースプロパティ」ダイアログ内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（「Webサービス：接続セキュリティ」を参照してください）。

On Web Connection データベースメソッドの呼び出し

On Web Connectionデータベースメソッドはブラウザがデータベースに接続する際に呼び出されます。また、Webサーバの新しい"非コンテキスト"モードから4Dへの入口として役割を果たします。

実際、非コンテキストモードからコンテキストモードへの切り替えは「On Web Connection」データベースメソッドによって行なわれます（詳しい説明は「Webサーバ：非コンテキストモード」を参照してください）。

警告：非コンテキストモードで、インタフェースの要素（ALERT, DIALOG...）を表示する4Dコマンドを呼び出すと、処理を終了します。

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。

- ・ 4DWebサーバにブラウザを接続する場合、4D6.0.Xと同様にデータベースメソッドは「/<action>... URL」で呼び出されます。

- ・ 非コンテキストモードからコンテキストモードへ切り替えた場合、データベースメソッドは「/4DMETHOD/MethodName」URLで呼び出されます。
- ・ 4Dがセミダイナミックページから「/4DCGI/<action>」URLを通して呼ばれた時、データベースメソッドはURLから呼び出されます（「Webサービス：非コンテキストモード」の節を参照してください）。
- ・ Webページが非コンテキストモードで呼ばれ、<path>/<file>タイプのURLで指定されたファイルが見つからない時、データベースメソッドはURLで呼び出されます。
- ・ 非コンテキストモードにおいてタイプ<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。

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のクエリの
        'プロセスを実行する（普通は短い）
End if
```

例：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。

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す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 (\$vTable ; 1 ; **Size of array** (<>asTables))

 <>asTables{\$vTable}:=**Table name** (\$vTable)

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釈して、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/cc"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

 \$viElem:=0

ARRAY TEXT (\$atTokens ; \$viElem)

While (\$vURL # "")

 \$viElem:=\$viElem+1

INSERT ELEMENT (\$atTokens ; \$viElem)

 \$viPos:=**Position** ("/" ; \$vtURL)

If (\$viPos>0)

 \$atTokens{\$viElem}:=**Substring** (\$vtURL ; 1;\$viPos-1)

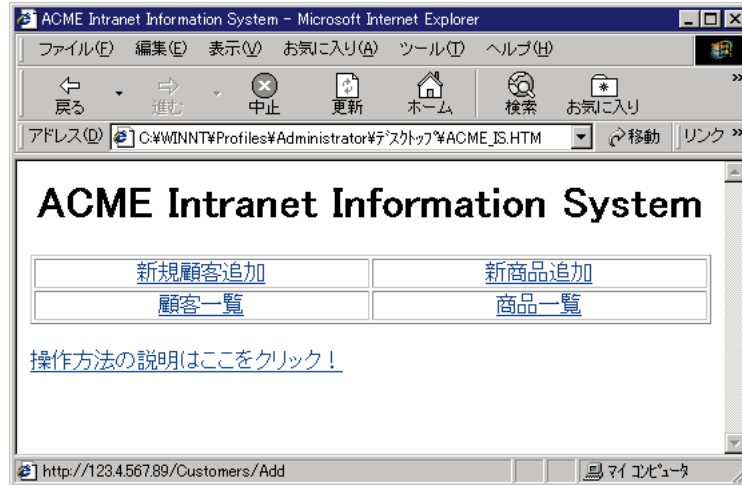
 \$vtURL:=**Substring** (\$vtURL ; \$viPos+1)

```

Else
    $atTokens{$vIElem}:=$vtURL
    $vtURL:=""
End if
End while
`URLのHOST部分の後にエクストラデータが渡された場合
If ($vIElem>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
    $vITableName:=Find in array (<>asTables ; $atTokens{1})
    If ($vITableName>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
        $vpTable:=Table ($vITableName)
        `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
        $vIAction:=Find in array (<>asActions ; $atTokens{2})
        Case of
            `レコードの追加
            ￥ ($vIAction=1)
                Repeat
                    ADD RECORD ($vpTable-> ; *)
                Until (OK=0)
                `レコードの一覧表示
                ￥ ($vIAction=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ろうとも、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://123.4.567.89/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「123.4.567.89」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。



Add New Customers

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、テキスト引数を通した4DメソッドをURL経由で呼ぶ方法、Webサービス：非コンテキストモード

Webサービス：Webサーバセッティング

4Dバージョン6.5は、ニーズに従って、Webセッションのカスタマイズすることができます。設定には、以下のオプションが有効です：

- デフォルトのホームページを定義する
- データ入力制御のためにJavascriptを使用する
- スタティックなページでの、4DVARのコメントを使用する
- HTMLテキストの変換と、Webフィルタの定義
- ページキャッシュの利用
- サーバがHTTPクエリを受けるIPアドレスを定義する

デフォルトのホームページを定義する

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。このページは、スタティック、または、セミダイナミックであることができます。

ユーザがデフォルトのホームページを指定する場合、どのモード（コンテキスト、または、非コンテキスト）が定義されても、このページはデータベースに接続する各ブラウザに送られます。4Dの前バージョンと違って、現在のメニューバーは、コンテキストモードではブラウザに送られません。

デフォルトで、ホームページは定義されません。カスタムメイドのホームページを指定しない場合、Webサーバの動作はスタートアップモードによって異なります：

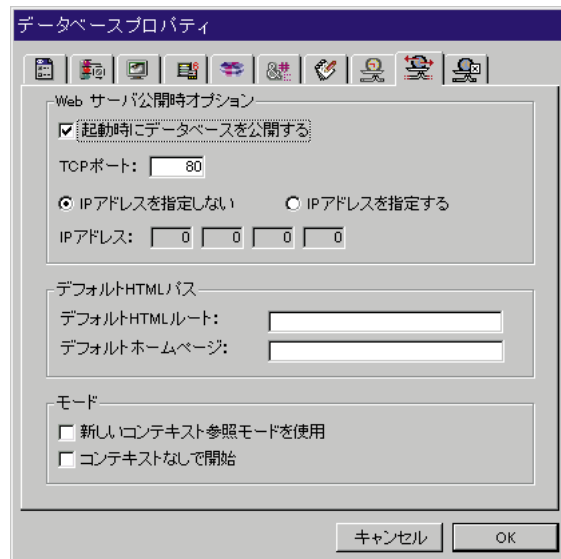
Webサーバがコンテキストモード（デフォルトで）で開始する場合、4Dの前バージョンの場合のように、カレントのメニューバー（デフォルトで、メニューバー#1）が送られます。

Webサーバが非コンテキストモード上で開始する場合、On Web Connectionデータベースメソッドだけが、コールされます。それは、手順の検索を処理するユーザ次第です。

デフォルトのホームページを定義するには、次のように行います：

1. データベースダイアログボックスで、"WebサーバII"タブをクリックする。

すると、次のようなページが表示されます。



2. 「デフォルトのホームページ」の入力エリアに、定義したいフォルダのアクセスパスを入力する。

それがデータベースのストラクチャファイルを含んでいるフォルダから設置されるので、このダイアログボックスで入力されるアクセスパスで関連づけられます。

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、4DWeb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"フォルダの中"MyHome.htm"にしたい場合、"Web/MyHome.htm"と入力します。

3. ダイアログボックスで、OKをクリックする。

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。

データ入力制御のためにJavascriptを使用する

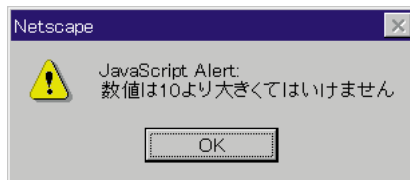
4Dバージョン6.5では、ブラウザのデータ入力制御の部分で、Javascriptを使用することができます。

ブラウザで、それらが適用されることができるデータ入力制御とデータ型（フィールド、または、変数）は、次の通りです：

最小値（数値）

最大値（数値）

必須入力



生成されたJavascript（それは小さいサイズです）は、ユーザをデータ入力（それは、まだ4Dの責任です）を受けるのを妨げることなく警告ダイアログボックスを表示します。実際に、データ入力エリアが適当でない値が入力された場合、ユーザがボタン（OK、Cancel、その他）をクリックすると、警告メッセージをブラウザに表示します：

データ入力を制御するのにJavascriptを使用するには次のように行います：

1. データベースプロパティダイアログボックスで"WebサーバII"タブをクリックする。
2. Javascriptを入力制御に使用するオプションを選択する。
デフォルトでは、このオプションは選択されていません。
3. OKボタンをクリックする。

サーバがHTTPクエリを受けるIPアドレスを定義する

データベースプロパティの"Web サーバII"ページで、WebサーバがHTTPクエリを受ける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。ユーザが「IPアドレスを指定する」オプションを選択する場合、「194.166.100.101」の様に、ユーザが固有のアドレスを入力することができるように、「IPアドレス」エリアは入力可能です。この場合、サーバはこのアドレスに送られるクエリに応答するだけです。

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。

注：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。詳細は、「4D Server ネットワークコンポーネント」マニュアルの「Macintosh上のADSP」を参照してください。

参照

「SET HOME PAGE」「Webサービス：接続のセキュリティ」「Webサービス：HTMLサポート」「Webサービス：非コンテキストモード」「Webサービス：「Web 接続」プロセス」

スタティックなページでの、4DVARのコメントを使用する

データベースプロパティの"WebサーバII"タブでの「ブラケットの代りに4DVARコメントを使用する」オプションは、スタティックページで4Dの変数に入力する時、使用する表記法を定義することを許可します。

オプションがチェックされる場合、使用しなくてはならないシンタックスは標準のHTML表記法です。

<!--4DVAR MAVARAE-->

(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)

オプションがチェックされない(デフォルト値)場合、ユーザが使用しなくてはならないシンタックスは角カッコによる表記法です。

[MAVAR]

新規のコンテキスト参照モード

デフォルトで4D Webサーバはそれぞれのユーザアクションの為にブラウザに現在のコンテキストの数を送ります。例えば、2つの段落とピクチャを含む場合、4Dはコンテキスト番号を3回送ります。

モードを参照している新しいコンテキストで、コンテキスト番号は、ドキュメントの基本のURLに設定されます。このように、それは各オブジェクトのために繰り返されません。

「新しいコンテキスト参照モード」を使用するには、次のように行います：

1. 「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"タブで、「新しいコンテキスト参照モード」オプションを選択する。
デフォルトでは、このオプションは選択されていません。

2. OKボタンをクリックして、修正が有効となるように、データベースを再起動する。

拡張ASCII文字を直接送る

デフォルトで、4D Webサーバは、送信する前に、HTML標準に従ってダイナミックでスタティックなWebページで拡張ASCII文字を変換します。それらは、その時ブラウザによって翻訳されます。HTML本体を変換することなく、拡張ASCII文字であるように送られるように、Webサーバをセットすることができます。このオプションは、特に日本のOS上で、速度増大に有効です。

4DWebサーバで直接、拡張ASCII文字を送信するには次のように行います：

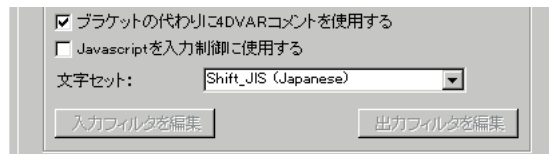
1. 「データベースプロパティ」ダイアログボックスで、"WebサーバII"タブをクリックする。
2. 「拡張文字を直接送信」オプションを選択する。
3. OKをクリックする。

4Dで文字セットの変換を修正する

ユーザは、4Dで直接入力して、データを出力するために使用されるASCII文字の文字セット（Webフィルタ）を変更することができます。この設定は、以前は、Customizer Plusで有効でした。「データベースプロパティ」ダイアログボックスで「x-user-defined」文字セットを選択するだけでWebフィルタを編集することができます。

ASCII文字の変換テーブルを編集するには次のように行います：

1. データベースプロパティダイアログボックスで、"WebサーバII"タブをクリックする。
2. 「文字セット」のリストから"x-user-defined"オプションを選択する。
「入力フィルタ編集」「出力フィルタ編集」ボタンが新たに有効となります。



3. 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する。
「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ます、そして、「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：



注：このダイアログボックスは、4Dのインポートとエクスポートのフィルタダイアログボックスに似ているように見えます。しかし、2つは独立しています：Webフィルタは、どんな形であれ、インポートとエクスポートのフィルタに関連がありません。

4. 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、クリックする。
または、「読み込み」ボタンをクリックすることによって、以前に保存されたWebフィルタを読み込む。

5. 「ASCII Code」入力エリアでは、文字の新しいASCIIコードを入力する。

注：フィルタのデフォルト値は、4Dバージョン6.0.6において使用されるそれらに同一で、データベース互換性を保証します（「MapC」から - Customizer Plusで定義される場合）。

6. フィルタしたいすべての文字のために、この操作を繰り返す。

7. よければ、フィルタを保存するために「セーブ」ボタンをクリックする。

8. 「テーブル更新」ボタンをクリックする。

9. 「データベースプロパティ」ダイアログボックスでOKのボタンをクリックする。
変更された入力や出力Webフィルタは、現在適用されています。

スタティックページのキャッシュ

4D Webサーバは、キャッシュがあります。それは、それらが要求される度に、スタティックなページ、GIFイメージ、JPEG画像（<100 kb）とスタイルシート（.cssファイル）をメモリにロードするのを許可します。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プロセスで分配されます。ユーザは、データベースプロパティでキャッシュのサイズ

をセットすることができます。デフォルトで、スタティクなページのキャッシュは、ありません（そのサイズは、0です）。

スタティクなページでキャッシュを可能にするには、次のように行います：

1. データベースダイアログボックスの"WebサーバII"で、スタティクなページのキャッシュの値（単位はKb）をセットする。



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 サービス：Web サイトに関する情報」を参照してください。

2. OKをクリックする。

一旦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、データベースプロパティの"WebサーバII"ページで「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サービス：非コンテキストモード

Webサーバは、非コンテキストモードでも使用できるようになりました（バージョン6.5以降）。このモードでは、昔ながらのHTTPサーバとなります。

コンテキストモードと非コンテキストモード

以前のバージョンは、Webプログラムの特長としてコンテキストの概念を用いていました。つまり、Webブラウザがデータベースに接続すると、現在のセクションや変数等を置いたコンテキストを作成します。ある意味では、各ブラウザは4Dクライアントとして扱われます。

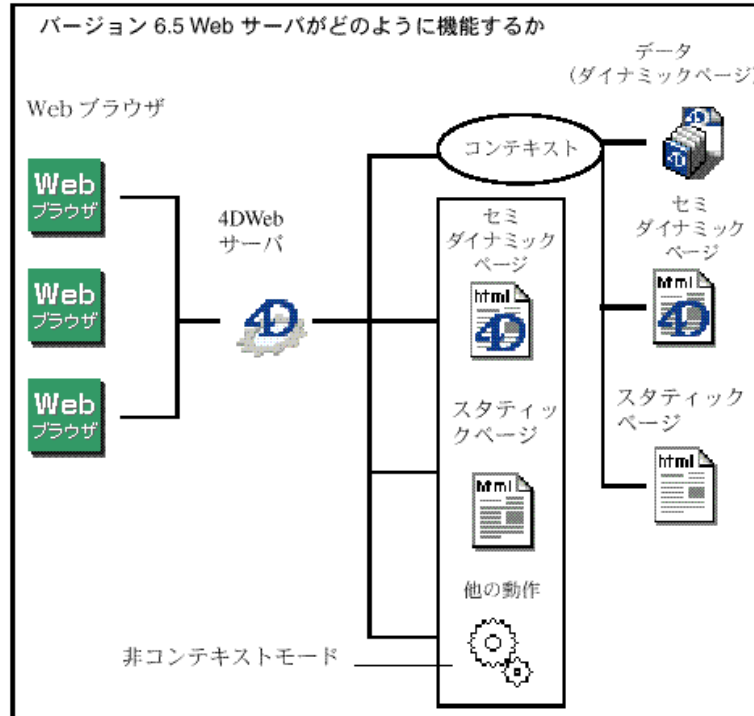
この方法によって、Webサーバはブラウザの動作を完全に管理し、データの信頼性を保証します。4DがスタティックHTMLページを送る時には、4Dは昔ながらのHTTPサーバとして使用されてるため、あまり適切な方法ではありませんでした。この場合、ブラウザからのリクエストで作成されたコンテキストは、設定されているタイムアウトになるまで、接続しているものとして残ります。例えば、ブラウザが途中でサイトを抜けるとコンテキストは無駄になり、また、標準的なブラウザの機能（「再読み込み」や「戻る」等）が使用できませんでした（「Webサービス：「Web接続」プロセス」を参照してください）。

スタティックホームページを送る際に、4D Webサーバを昔ながらのHTTPサーバとして使用するには非コンテキストモードを採用します。このモードでの4D Webサーバは、標準HTTPサーバになります。すなわち、スタティックWebページはコンテキストを保持することなく送られます。

スタティックホームページはキャッシュの恩恵を受けます（「Webサービス：Webサーバセッティング」を参照してください）。この場合もまた、ホームページ用キャッシュに記載されているWEB CACHE STATISTICSを使用して、これらのページの参照についての統計を得ることができます。

もちろん、レコードにおけるブラウザのナビゲーションをコントロールするために、コンテキストは常に使用されます。実際、4D Webサーバはコンテキストモード、非コンテキストモードまたは必要に応じてこれら2つのモード間での切り替え等、希望するどのモードでも使用することができます。

6.5 Webサーバがどのように機能するかを下図に示しました。



あるモードから別のモードへの切り替え

コンテキスト管理に関する限り、デフォルトでは、バージョン6.5 Webサーバは以前のバージョンと同じように機能し、コンテキストがそれぞれのWeb接続で作成されます。非コンテキストモードへの切り替えを、明確に指示する必要があります。

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

データベースを使用している間、いつでもコンテキストモードから非コンテキストモードへ、またはその逆の切り替えをすることができます。

注：デフォルトでは、4D Webサーバはコンテキストモードで開始しますが、非コンテキストモードでの開始を定義することも可能です。より詳しい情報は「スタートアップ時に非コンテキストモードを定義する」を参照してください。

4D Webサーバを非コンテキストモードへ切り替えるには、3つの解決法があります。

新しいコマンド **SEND HTML BLOB** を実行し、オプションパラメータに **True** (noContext) を渡します。例えば、**SEND HTML BLOB (MyBlob; ".HTM"; True)**。

4Dは、BLOBを非コンテキストモードで送り、コンテキストを管理するプロセスは直ぐに停止されます。

注：最後のパラメータを渡さないことは、Falseを渡すことと同じです。既に非コンテキストモードの場合、パラメータは無視されます。現在のモードは、Web Contextを使用して調べることができます。

新しいコマンド**SEND HTTP REDIRECT**を実行します。このコマンドは、パラメータに渡されたURLへリクエストをリダイレクトします。コンテキストモードから呼び出されると、Webプロセスを閉じます。

/4DACTIONで始まるURLを呼び出します（「4D ACTIONタグ」を参照してください）。

非コンテキストモードは、コンテキストモードがURLで呼び出されるまで有効です。

非コンテキストモードからコンテキストモードへの切り替え

一般的に、4Dの「クラシックな」コンテキストモードは、データベースがダイナミックを生成、送信しなければならなくなるとすぐに使用されます。

非コンテキストモードからの4Dデータベースの呼び出しは、特定のURL：
"/4DMETHOD/MyMethod"を通して行われます。このURLをスタティックホームページ（例えばボタン内）に置くだけです。このURLについての詳しい説明は「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。

このURLが有効になると、4D Webサーバは新しいコンテキストを作成して下記のことを実行します。

On Web Authenticationデータベースメソッドが存在している場合、実行されます。

On Web Connectionデータベースメソッドが存在している場合、実行されます。

メソッドが、新しく作成されたコンテキスト内で実行されます。

Webサーバは、これまでの4Dのバージョンと同様に機能します。コンテキストモードは、非コンテキストモードを有効にするコマンドが呼び出されない限り、そのモードで動作します。

使用されているコンテキストの数

実行されている動作に応じて、特定のWebプロセスはWebコンテキストを使用します。どのWebプロセスでもPROCESS PROPERTIESコマンドを使うことによって、使用されているコンテキストの数を知ることができます。このコマンドは、新しいパラメータoriginに、Webコンテキストが使用されているか（-11:Web Process with Context）または使用されていないか（-3:Web Process with no Context）を返します。

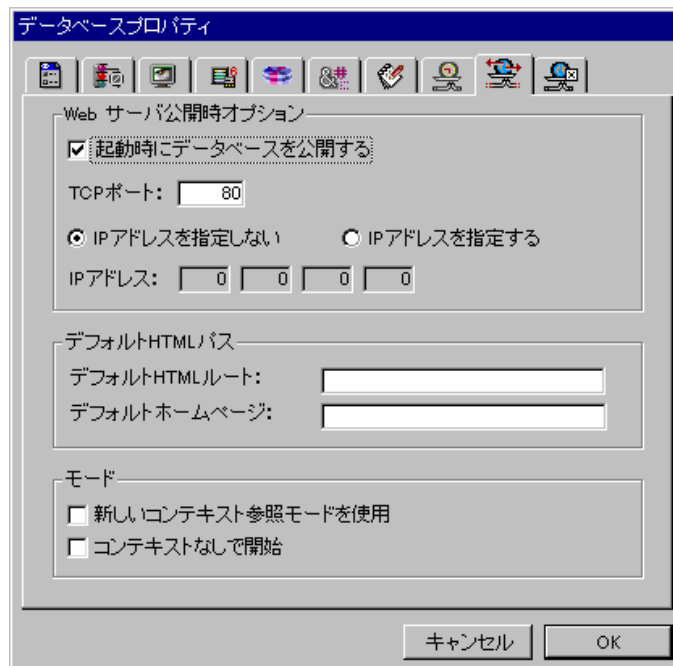
スタートアップ時に非コンテキストモードを定義する

4DWebサーバを、自動的に非コンテキストモードで開始することができます。これは、ユーザがデータベースに接続する時にコンテキストが作成されないということを意味します。

スタートアップ時に非コンテキストモードを定義するには、

1. 「データベースプロパティ」ダイアログで、“WebサーバII”タブをクリックする。
2. “コンテキストなしで開始”オプションを選択する。

デフォルトでは、このオプションは選択されておらず、4D Webサーバはコンテキストモードでスタートします。



3. ダイアログボックスを受け入れる。

4D Webサーバは、非コンテキストモードでホームページを送ります。

データベースプロパティの“デフォルトホームページ”エリアでカスタムホームページを選択すると（「Webサービス：Webサーバセッティング」を参照してください）、そのホームページが送られます。

注：URLが"/"で終る場合、4Dはホームページの名前を追加して送ろうとします。例えば、URLが"/Folder/"であり、デフォルトホームページが"MyPage.HTM"の場合、4Dは、"/Folder/MyPage.HTM"を探します。これが見つからない場合は、On Web Connectionデータベースメソッドが呼び出されます。

デフォルトホームページを定義していない、または指定されたホームページが存在しない場合は、On Web Connectionデータベースメソッドが（非コンテキストモードで）呼び出されます。また、各Webユーザ固有の接続方法を管理することもできます。例えば、データベースに接続する各ユーザ用のホームページを指定するために、新しいコマンド**SET HOME PAGE**を使用することもできます。

セミダイナミックページ

4Dではセミダイナミック方式で作成されるWebページを使用できます。セミダイナミックページは、その内容の全部または一部が4Dによって実行されたアクションから生成されるWebページです。

バージョン6.5は、セミダイナミック方式で作成される新しいタイプのWebページを採用しました。セミダイナミックページは、その内容の全部または一部が4Dによって実行されたアクションから生成されるWebページです。

セミダイナミックページは、コンテキストモードまたは非コンテキストモードのどちらでも使用できます。しかし、実際にはほとんどが非コンテキストモードで使用されます。このモードで4Dはブラウザのナビゲーションに関連した、どんな変数やセレクション等も保持しません。セミダイナミックページの目的は、4Dの利点と非コンテキストモードに関連した利点をWebページに取り入れることです。

セミダイナミックページは、下記の3つの方法で作成することができます。

"/4DCGI/<action>" URLを4D Webサーバに送る

HTMLページの拡張子に".shtm"または".shtml"を付けて送る

"/4DACTION/"タグを使用する

URL/4DCGI/<action>

4D Webサーバが/4DCGI/<action>のURLを受け取ると、このURLをそのまま\$1にセットし、On Web Connectionデータベースメソッドを呼び出します。

4DCGI/のURLは、どのファイルにも対応しません。この役目は4Dを呼び出すことです。<action>のパラメータは、どのタイプの情報も持つことができます。

URLは、どのような動作でも実行させることができます。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中で\$1の値をテストし、4Dに適切な動作をさせるだけです。例えば、レコードの作成、検索、ソートまたはGIFイメージを作成し、完全にカスタムHTMLページを構築することができます。このURLをどのように使用するかの例は、**PICT TO GIF**および**SEND HTTP REDIRECT**コマンドの説明を参照してください。

データを送るコマンド（**SEND HTML FILE**、**SEND HTML BLOB**等）を使用して、問い合わせの結果を返され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で実行させることを心掛けてください。

".shtm"ファイル

拡張子が".shtm"（または".shtml"）であるHTMLページを、4D Webサーバがブラウザに送る前に、4D変数への参照とページ内に存在するかもしれない4DACTIONのコメントを解析します。

これは、HTMLページの作成に4D内での処理実行を可能にし、スタティックホームページ内のダイナミックリファレンスの更新を確実にします。しかし、ブラウザが停止しないように、可能な限り短い動作で実行させることを心掛けてください。

注：非コンテキストモードでは、4Dピクチャ変数への参照を行なうことはできません。

WEB CACHE STATISTICSの例を参照してください。

4DACTIONタグ

4DACTIONタグは、スタティックHTMLページを送る際に4Dメソッドの実行を可能にします。このタグは3通りの使用方法があります。

HTMLコメントとして（`<!--4DACTION ...-->`）

URLとして（``）

HTMLフォームの送信（フォームアクション）

HTMLコメントとしての4DACTION

スタティックホームページ内のHTMLコメントとして/4DACTION/MyMethod/MyParamタグが存在するということは、MyParam/パラメータを文字列で\$1にセットし、MyMethodメソッドの実行を要求することです。ホームページをロードする時に、4Dは、On Web Authenticationデータベースメソッドがあれば呼び出し、戻り値としてTrueが返った場合、4Dは指定されたメソッドを実行します。

メソッドは\$0にテキストを返します。返された文字列がASCIIコードの1で始まる場合、4DにHTML文として考慮されます（同じ原理は変数にも当てはまります。「Webサービス：HTMLとJavascriptのカプセル化」を参照してください）。

警告："ブラケットの代りに4DVARコメントで使用する"オプションが、データベースプロパティで選択されていなければなりません（「Webサービス：Webサーバセッティング」を参照してください）。

ページ内容の解析は、**SEND HTML FILE**（.htm、.html、.shtm、.shtml）または**SEND HTML BLOB**（タイプtext/htmlのBLOB）が呼び出された時にも行われます。

非コンテキストモードで、URLが".shtm"または".shtml"のどちらかの拡張子を持つファイル（例えば、http://www.server.com/dir/page.shtm）を指す場合にも解析が行なわれます。

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、スタティックホームページに"Today is <!--4DACTION/MYMETHOD/MYPARAM-->"のコメントを挿入したとを考えてみましょう。

4Dはページをロードする時に、On Web Authenticationメソッドが存在していれば呼び出し後、文字列"/MYPARAM"を\$1の値としてMYMETHODメソッドを呼び出します。

メソッドは\$0にテキスト（例えば、"98/10/28"）を返すため、"Today is <!--4DACTION/MYMETHOD/MYPARAM-->"の式は、"Today is 98/10/28"になります。

MYMETHODメソッドは以下の通りです。

C_TEXT (\$0) このパラメータは常に宣言する必要あり

C_TEXT (\$1) このパラメータは常に宣言する必要あり

\$0=String (Current date)

注：4DACTIONで呼び出されたメソッドは、インタフェースエレメント（DIALOG、ALERT...）を呼び出してはいけません。

どのモードで実行していようと、現れた順に4Dが実行するメソッド内で更にドキュメント内で参照される多くの変数の値を設定するメソッドを呼び出すことができます。

注：スタティックホームページには、好きなだけ多くの<!--4DACTION ...-->コメントを挿入することができます。

URLとしての4DACTION

4Dが/4DACTION/MYMETHOD/MYPARAMリクエストを受け取ると、On Web Authenticationメソッドデータベースがあれば呼び出し、戻り値としてTrueが返った場合、MYPARAMパラメータを文字列で\$1にセットし、MYMETHODメソッドが実行されます。URLの構文は、下記の形でなければなりません。

DoSomething

注：これは、非コンテキストモードに適用されているということ以外は、/4DMETHODと同一のものです。

コンテキストモードでは、4Dが/4DACTIONタイプのリクエストを受け取るとコンテキストはアボートされ、メソッドはコンテキスト外で実行されます。これは、非コンテキストモードで/4DMETHODを呼び出すのとは逆のものです。

注：/4DACTIONで呼び出されたメソッドは、インタフェースエレメント（DIALOG、ALERT...）を呼び出してはいけません。

下記の指示をHTMLページに挿入します。

```
<IMG SRC="/4DACTION/PICTFROMLIB/1000">
```

PICTFROMLIBメソッドは下記のようになります。

C_TEXT (\$1) `このパラメータは常に宣言されなければなりません

C_PICTURE (\$PictVar)

C_BLOB (\$BlobVar)

C_LONGINT (\$Number)`

`\$1からピクチャの番号を取り出します

\$Number:=Num (Substring (\$1;2;99))

GET PICTURE FROM LIBRARY (\$Number;\$PictVar)

PICT TO GIF (\$PictVar;\$BlobVar)

SEND HTML BLOB (\$PictVar;"Pict/gif")

4DACTIONをPOSTするフォーム

非コンテキストモードで、データをWebサーバへ送るためにPOSTアクションのフォームを使用したい場合、しばしば、付加価値としての機能を提供します。POSTタイプはこれらに関連付けられなければならない、フォームの動作は/4DACTION/MethodNameで始まることが絶対条件です。

注：4Dは、GETタイプをPOSTするフォームも受け入れます。しかし、この場合、作成されたURLのHTML文を手動で処理しなければなりません。さらに、COMPILER_WEBメソッドは呼び出されません（下記参照）。

この場合、WebサーバがPOSTされたフォームを受け取る時、COMPILER_WEBプロジェクトメソッド（これが存在する場合、下記参照）を、次にはOn Web Authenticationデータベースメソッド（存在する場合）を呼び出し、Trueを返した場合に、MethodNameメソッドが実行されます。4Dは、フォーム内に存在するHTMLフィールドを解析し、これらの値を検索し、自動的に4D変数をこれらの内容で置き換えます。フォーム内のフィールドおよび4Dの変数は同じ名前ではなければなりません。

注：/4DACTIONで呼び出されたメソッドは、インタフェースエレメント（DIALOG、ALERT...）を呼び出してはいけません。

フォーム内で適用するHTML構文は、下記の形になります。

フォーム内の動作を定義するには、

```
<FORM ACTION="/4DACTION/MethodName" METHOD=POST>
```

フォーム内のフィールドを定義するには、

```
<INPUT TYPE=Field type NAME=Field name VALUE="Default value">
```

4Dはフォーム内の各フィールドの値を同じ名前の変数に設定します。

フォームのオプション（例えばチェックボックス）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に設定します。

数値データの入力では、4Dはフィールドの値を文字列から数値に変換します。

注：フォーム内のフィールドにOKと名付けられると（例えばSubmitボタン）、フィールドの値が空でなければ、システム変数OKが1に設定されます。そうでない場合は0が設定されます。

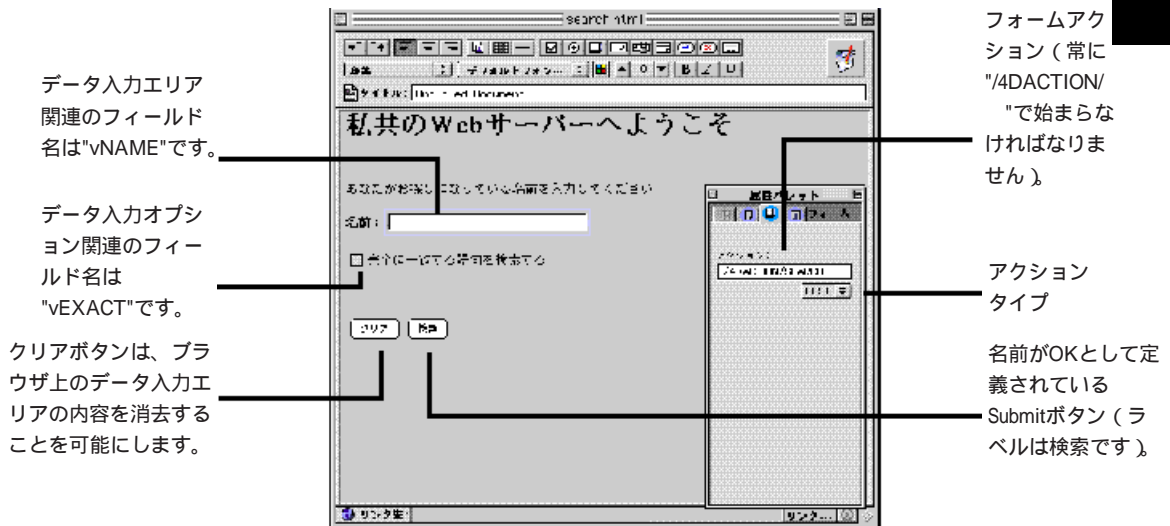
COMPILER_WEBメソッド：4D WebサーバがPostされたフォームを受け取ると、COMPILER_WEBと呼ばれるプロジェクトメソッド（それが存在している場合）を呼び出します。このメソッドには、名前がフォーム内のフィールド名と同じ名前の全変数のタイプ設定や、必要であれば変数の初期化指定が含まれていなければなりません。これはデータベースをコンパイルする時に4D Compilerによって使用されます。

COMPILER_WEBメソッドはすべてのフォームに共通です。

非コンテキストモードで開始されて使用されている4D Webデータベースがあり、スタティックHTMLページを使用してレコードの検索が行いたいとします。検索ページを"search.htm"とし、結果を返すページを"results.htm"とします。

POSTタイプは、ACTIONとして/4DACTION/PROCESSFORMがページに記述されています。

下記は、Adobe PageMillのHTMLエディタ内で現れるページです。



下記は、このページに対応するHTMLコードです。

```
<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST>
```

```
<INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR>
```

<!-- 通常は解釈上の理由からVALUEにボタン名を入れますが、4Dで使用する場合はVALUEに番号を入れなければなりません-->

```
<INPUT TYPE=CHECKBOX NAME=uExact VALUE="1">完全に一致する語句を検索する<BR>
```

<!-- OKは、特別な意味があります-->

```
<INPUT TYPE=SUBMIT NAME=OK VALUE="検索">
```

```
</FORM>
```

データ入力中に、データ入力エリア内に"ABCD"をタイプし、オプションをチェックして、検索ボタンをクリックします。

4Dは、次に下記のようなCOMPILER_WEBプロジェクトメソッドを呼び出します。

```
C_TEXT (VNAME)
```

```
VNAME:=""
```

```
C_LOGINT.(vEXACT)
```

```
vEXACT:=0
```

```
OK:=0 `特別なケース
```

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
(ボタンの名前が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、

次に下記のようにPROCESSFORMプロジェクトメソッドが呼び出されます。

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で返します

FIRST RECORD([Jockeys])

While (Not(**End selection**([Jockeys])))

 vLIST:=vLIST+[Jockeys]Name+" "+[Jockeys]Tel+"
"

NEXT RECORD ([Jockeys])

End while

SEND HTML FILE ("results.htm") `リストをresults.htmへ送ります

：

End if

参照

SEND HTML BLOB、SEND HTTP REDIRECT、テキスト引数を通したURL経由の4Dメソッドの呼び出し、Webコンテキスト、Webサービス：HTMLとJavascriptの要約

Webサービス：HTMLサポート

HTMLとJavaScriptコードを4Dアプリケーションに実装する前に、4th Dimensionが既に提供しているものは何かを確認しましょう。

メニューバー

各メニューバーは、1つのHTMLページに変換されます。各メニュータイトルはテキストとしてだけ表示され、メニュー項目は4Dメソッドへのリンクとして表示されます。

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。

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。

フォーム

オブジェクトは上から下へ、左から右への順に変換され、それらはブラウザと4Dフォームにあるものと同じ位置にあります。

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せん。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。

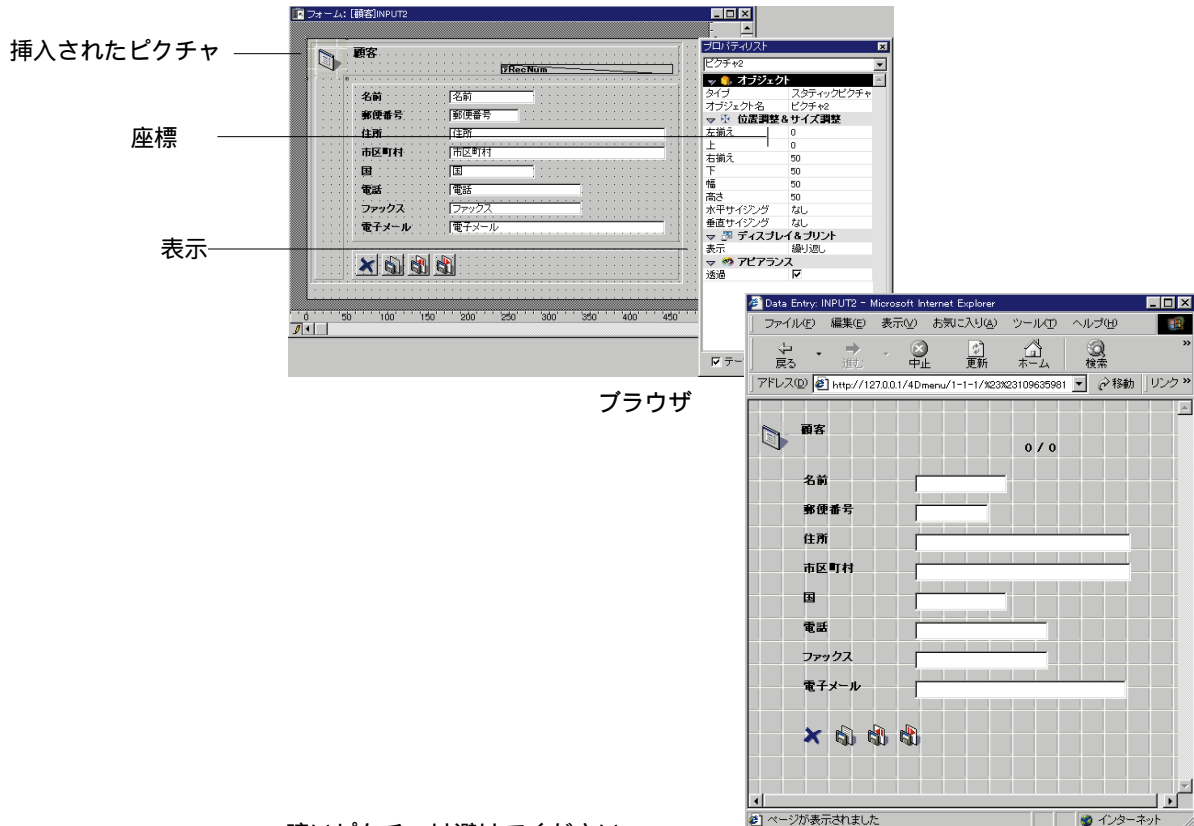
フォームイベント（On Load、On Unload、On Clicked、On Timer）がサポートされます。それ以外のイベントはサポートされません。

ヘッダ、ディテイル（詳細）、ブレイク、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現れ、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。

フォームエディタ



暗いピクチャは避けてください。

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 チェックボックス (*)	<INPUT Type="radio" ...> <INPUT Type="checkbox" ...>

ピクチャ	イメージ (常に入力不可)	
サブテーブル	HTMLではサポートしていない	なし
BLOB	HTMLではサポートしていない	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	HTMLではサポートしていない	なし
楕円	HTMLではサポートしていない	なし
角の丸い矩形	HTMLではサポートしていない	なし
静止ピクチャ	イメージ / イメージマップ (2)	
グループボックス	テキスト	<INPUT Type="image" ...> 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	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)	
プラグイン	イメージ (入力不可)	

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、ダイヤル、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、ACI_Pack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らなければならない場合があります。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 document ("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と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	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 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	デバッグウインドウは、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。
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。

Webサービス：HTMLとJavaScriptのカプセル化

はじめに

この節を参照する前に、Webサービスに関する以下の節を読んでください。

- Webサービス：概要
- Webサービス：システム設定
- Webサービス：入門編（パートI）
- Webサービス：入門編（パートII）
- Webサービス：「Web接続」プロセス
- Webサービス：HTMLサポート

4DアプリケーションにHTMLコードをカプセル化する方法には、以下の3種類があります。

1. SEND HTML FILEまたはSEND HTML BLOBコマンドを使用すると、ディスク上に格納されたWebページを送信できます。
2. 4Dフォームのスタティックテキストオブジェクト（テキストツールで作成したオブジェクト）内に、例えば“ {anyPage.HTM} ”と指定すると、そのスタティックテキストオブジェクトの位置にHTMLドキュメント"anyPage.HTM"を挿入することができます。

注：この方法は非コンテキストモードでは利用できません。詳しい説明は「Webサービス：非コンテキストモード」を参照してください。

3. フォーム内の任意の4Dテキスト変数を使い、HTMLコードを4Dフォームにカプセル化することができます。この場合、必ず最初の文字としてASCIIコードの1 “ Char (1) ” を指定してください（例：vtHTML:=Char (1) +"...HTML code..."）。

2と3では、Webブラウザ側で結果として表示されるフォームには4DとHTML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すると、ドキュメント全体が挿入されることに注意してください。テキスト変数を使用すると、いくつかのコードが挿入されます。

注：いくつかの場合、バージョン6.0.xではHTMLドキュメント（mypage.htm）の参照を含んで作成されていた4DフォームのHTMLの変換は4D 6.5では予想外な結果となりうるでしょう。その場合、**SET DATABASE PARAMETER**関数を使って6.0.xでの変換モードを使うことができます。

SEND HTML FILEコマンドまたはフォームのスタティックテキストオブジェクトを使用すると、既存のHTMLドキュメントを使用するか、または、以前にプログラムで構築して

からディスクに保存していたドキュメントを参照できます。フォームの中にあるテキスト変数を使用すると、メモリ上にHTMLコードを構築することができます。

HTMLオブジェクトと4Dメソッドとのバインド設定

HTMLを4Dアプリケーションにカプセル化する方法とは関係なく、HTMLオブジェクトに対するリンクを作成することで、4DメソッドにHTMLオブジェクトをリンクできます。オブジェクトのURLは、「/4DMETHOD/メソッド名」である必要があり、この“メソッド名”とは、HTMLオブジェクトがクリックされた時に実行される4Dプロジェクトメソッドの名前です。HTMLオブジェクトにバインドされる4Dメソッドの例については、「Webサービス：入門編（パートII）」の節を参照してください。

重要

HTMLファイルを送信する場合には、任意のリンク可能なHTMLオブジェクトに4Dメソッドをバインドできます。HTMLモードを停止する**SEND HTML FILE**（"）コールを発行するPOSTアクション4Dメソッドを持つには、HTMLページ内にサブミットボタンを作成して、HTMLページ（4D Webサーバにサブミットされる予定のページ）のPOSTアクションを実行する必要があることに留意してください。

一方、スタティックテキストまたはテキスト変数を使ってHTMLコードを4Dフォームにカプセル化する場合には、サブミットボタンは使用できません。4Dメソッドを参照するリンクを持つことはできますが、4Dがページを構築してくれるため、POSTアクションを指定する機会がありません。

HTMLオブジェクトと4D変数とのバインド設定（パート1）

HTMLオブジェクトを4D変数にバインドすることができます。

注：作業には**プロセス変数**を使用します。

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ページがサブミットして戻された後に、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、ページの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、後述の「HTMLオブジェクトと4D変数とのバインド設定：パート2」を参照してください。

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。詳しい説明は「Webサービス：非コンテキストモード」の節を参照してください。

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[VarName]を含めることにより、デフォルト値を提供できます。この“ VarName ” は、「 Web接続 」プロセスで定義された4Dプロセス変数の名前です。この変数名を角カッコで囲みます。

注：4Dのバージョン6.5から、4D変数をスタックページに挿入する際に標準のHTML表記 <!--4DVAR MAVAR-->を使用することができます。この構文を使うことができるのは、「データベースプロパティ」ダイアログボックスのオプションを対応するようにセットしなければなりません（「Webサービス：Webサーバセッティング」を参照してください）。

実際、[VarName]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せん。例えば、以下のように書いた場合に、

```
<p>Welcome to [vtSiteName]! </p>
```

4D変数「 vtSiteName 」の値が、HTMLページ内に挿入されます。

Tips：[VarName]構文を使用したバインドは、再帰的に扱われます。変数VarNameの中にあるテキスト値に他の有効な[...]参照が含まれる場合には、4Dは、参照が見つからなくなるまでその参照を変数値に置き換えます。

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 Is4DWebServer(){
      return (document.frm.vs4D.value=="4D4D")
    }

    function HandleButton(){
      if (Is4DWebServer()){
        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="[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>

```

コンテキストモードでHTMLページ（HTMLドキュメントまたは変換された4Dフォーム）を送信する前に、4Dは4D変数を参照するオブジェクトを検索するために、また、リンクのURLを再マップするためにHTMLソースコードを常に解析します（後述）。

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"[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（例：MacintoshのWeb Star）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"[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から提供されるものかどうかを検出できるのです。

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、バージョン6.0の初期リリースでは、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（つまり、最初の要素をV1へ、2番目の要素をV2へ等）。

4DからWebブラウザに対する方向でのバインドは、任意のカプセル化メソッド（**SEND HTML FILE**、**SEND HTML BLOB**コマンド、スタティックテキスト、4Dフォームのテキスト変数）を使っても動作します。

JavaScriptカプセル化

4DI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="..."）。

SEND HTML FILE又は**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。また、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。

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、下記のものと同じ例とで、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：4DI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。

HTMLオブジェクトと4D変数とのバインド設定（パート2）

SEND HTML FILE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、一度HTMLページ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。

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。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。

以下の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
    }

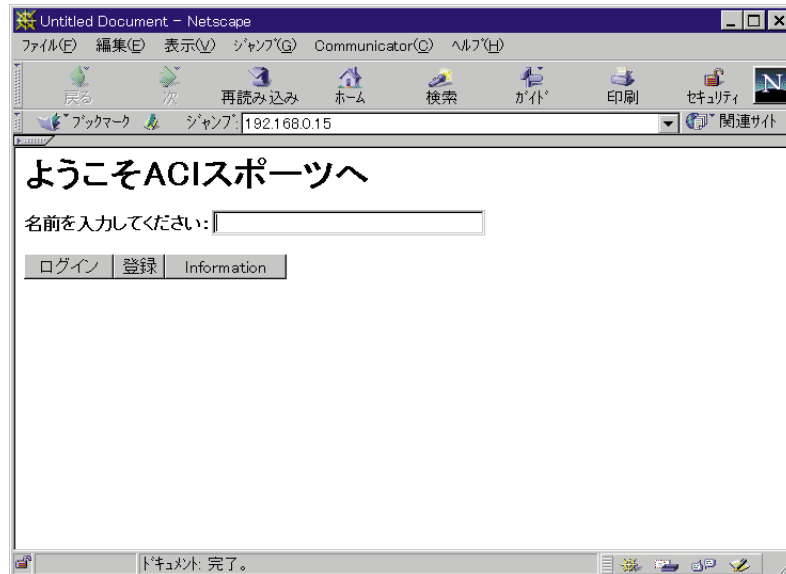
    function LogOn(formObj){
      if(formObj.vtUserName.value!=""){
        return true
      } else {
        alert("Enter your name, then try again.")
        return false
      }
    }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frmWelcome"
    onsubmit="return GetBrowserInformation(frmWelcome)">

  <H1>ようこそACISポータルへ</H1>

  <P><B>名前を入力してください:</B>
  <INPUT TYPE="text" NAME="vtUserName" VALUE="{vtUserName1}" SIZE=30></P>
  <P>
    <INPUT TYPE="submit" NAME="vsbLogOn" VALUE="ログイン" onclick="return LogOn(frmWelcome)">
    <INPUT TYPE="submit" NAME="vsbRegister" VALUE="登録">
    <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、[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   プール
`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、且つ直接的に、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。

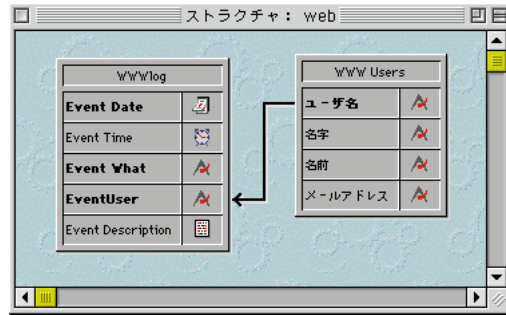
4D変数「vsbLogOn」、「vsbRegister」、「vsbInformation」は、3つのサブミットボタンにバインドされます。これらの変数は、ページがブラウザに送られるたびにリセットされることに注意してください。これらのボタンのどれかでサブミットが実行されると、ブラウザはクリックされたボタンの値を4Dに返します。4D変数はそのたびにリセットされるため、空の文字列ではなくなった変数によって、どのボタンがクリックされたのかがわかります。それ以外の2つの変数は空の文字列であり、これは、ブラウザが空の文字列を返したためではなく、ブラウザがそれらの変数について何も「言わなかった」ためです。その結果、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ります。

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に返される時に、ボタンがクリックされたかどうかのテストは、ボタンにバインドされる4D変数のテキスト値をテストすることで行われます。

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わせることで、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。

HTMLオブジェクトと4D変数とのバインド設定（パート3）

「Webサービス：HTMLサポート」の節にあるように、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又は**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、「blImageMap」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。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blImageMapと、blImageMap_X変数およびblImageMap_Y変数をバインドする必要があるだけです。ここで、この変数はクリックの縦方向の座標と横方向の座標を（テキストとして）返します。

HTMLページでは、以下のように作成できます。

```
<P> <INPUT TYPE="image" SRC="myImage.GIF" NAME="bWorldMap" BORDER= 0></P>
```

HTMLページを送信する4Dメソッドでは、以下のように作成できます。

```
blImageMap:=""  
blImageMap_X:=""  
blImageMap_Y:=""  
SEND HTML FILE ("ThisPage.HTM")
```

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 HTML FILE**（`""`）呼び出しを発行した後に、以下のようにクリックの座標を取り出します。

```
$vIX:=Num (blImageMap_X) ` 数値形式で横方向の座標を得る  
$vIY:=Num (blImageMap_Y) ` 数値形式で縦方向の座標を得る  
If (($vIX#0)&($vIY#0))  
    `取得した座標に従って処理をする  
End if
```

ファイル参照とURL（コンテキストモード）

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留意してください。

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:

参照

SEND HTML FILE、SEND HTML FILE、Webサービス：非コンテキストモード

URLでコールした4Dメソッドに渡されたテキスト引数

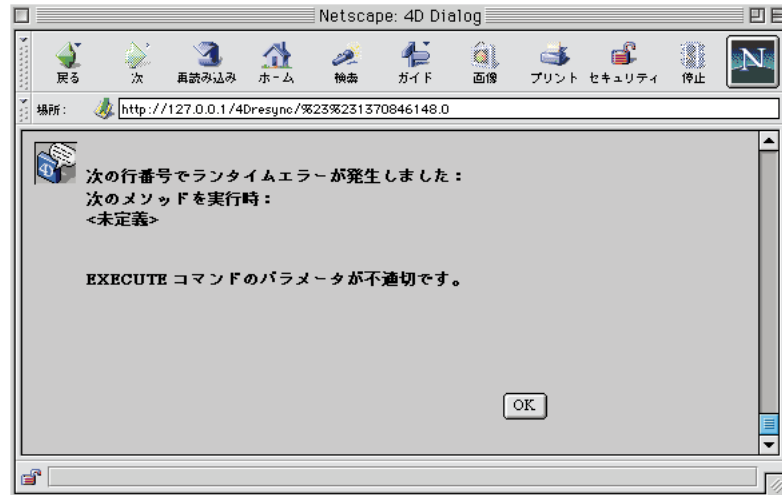
4th Dimensionは、URLでコールされた任意の4Dメソッドに対してテキスト引数を送信します（4DMETHOD/、4DACTION/...）。コンテキストでも非コンテキストモードでも、このテキスト引数に関する説明は以下の通りです。

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。このエクストラデータは、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サービス：入門編（パートI）」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（例えば（4DCGI/<action>）URLで呼ばれる）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)

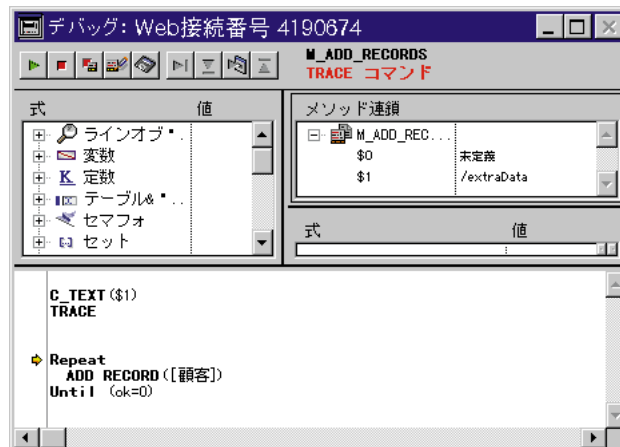
URLエクストラデータを用いた作業

4Dメソッドに渡した\$1テキスト引数には、URLに追加されたエクストラデータが返されます。

以下の『4th Dimension ランゲージリファレンス』マニュアルの例題を再度見てみましょう。メソッドM_ADD_RECORDSを参照するリンクのURLが変更されています（下図参照）。

注：この図は、Macintosh上のAdobe PageMillを使用して変更を行っているところです。

“/extraData”という文字列がURLに追加されています。この変更を行った後、4D側で「デバッグ」ウインドウを使い、実際に\$1引数へ文字列“/extraData”が返されていることを調べられます。



「On Web Connectionデータベースメソッド」の節における説明と同様の規約およびアルゴリズムを用いると、HTMLリンクにより4Dメソッドが呼び出された際に、HTMLと4D環境の間で追加データのやり取りを行えます。

URLエクストラデータのダイナミックな設定方法

独自のHTMLファイルを“その場で”作成する場合（例えば、**Create document**や**SEND PACKET**を使って）、必要に応じたHTMLを記述するだけです。

既存のHTMLファイルを使用して作業する場合、JavaScriptを利用し、オブジェクトのリンクプロパティを動的に設定することができます。

参照

Webサービス：入門編（パートII）

Webサービス：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および/4DCACHECLEAR）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作成できます。

Webサーバ管理URL

4D Webサーバは、3つの特定のURL（/4DSTATS、/4DHTMLSTATSおよび/4DCACHECLEAR）を受け入れます。

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。

/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サービス：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。

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。

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] "query" 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。

query：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

400:Incorrect query

401:Authentication required

404:Not found

500:Internal error

ログで作成される例：

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 - ACI [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"ACI"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。

デフォルトではクエリ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、

1. データベースプロパティの"WebサーバII"ページで、"ログを必要とする"オプションを選択する。
2. OKボタンをクリックする。

参照

WEB CACHE

以下の節では、「ルーチン」エディタの「Web Server」テーマ内にあるWebサーバコマンドについて説明します。

START WEB SERVER	SEND HTML FILE	WEB CACHE STATISTICS
STOP WEB SERVER	SEND HTML BLOB	SET HTTP HEADER
SET WEB TIMEOUT	Web context	SEND HTTP REDIRECT
SET WEB DISPLAY LIMITS	SET HTML ROOT	OPEN WEB URL
SET HOME PAGE		

START WEB SERVER

START WEB SERVER

説明

START WEB SERVERコマンドは、内蔵の4th Dimension Webサーバ機能を使用して、イントラネットネットワークまたはインターネット上でデータベースのサービスを開始します。

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ネットワークコンポーネントが見つからない場合には、システム変数OKは0に設定されます。

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、データベースのWebサーバとしてのサービスを停止します。データベースがWebサイトとして稼働していた場合には、すべてのWeb接続は停止され、すべてのWebプロセスが終了します。

データベースがWebサイトとして稼働していなかった場合には、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、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

「Webサービス」：「Web接続」プロセス」

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クションの4th Dimensionによる表示方法を変更します。

4th Dimensionまたは4D Clientを使用してレコードセクションを表示するとき、プログラムはセ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、各セクションページの最後に配置したい、セ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。

	Our Town	Tallapoosa	AL	35010
	Russell Mill	Tallapoosa	AL	35010
	Graystone	Blount	AL	35013

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



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。

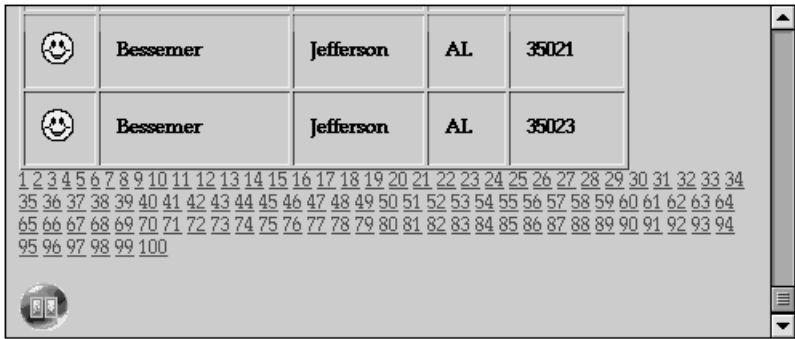
以下のピクチャがデータベースのピクチャライブラリに追加されたものとします。



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了します。



これで選択の最初の50,000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

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

この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定義することができます。

デフォルトでは、コンテキストモードでメニューバーNo.1がホームページになります。

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のバージョン6.5は、「データベースプロパティ」ダイアログ内で、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）が何であろうと、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

なし

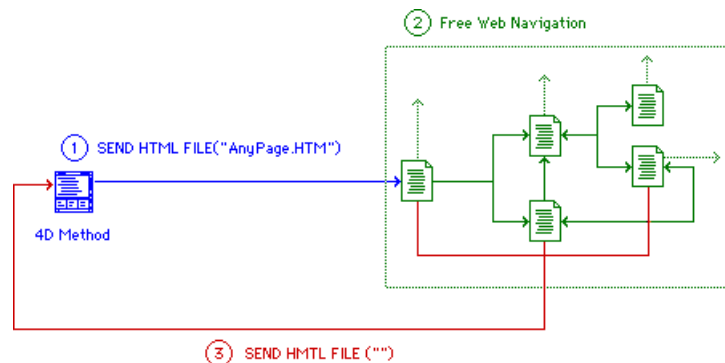
SEND HTML FILE

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、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。



1. (A) HTMLドキュメントをブラウザへ送るとき、4Dメソッド（プロジェクト、データベースのオブジェクト）はSEND HTML FILEで呼び出されて命令を出します。
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HTMLリンクするか別のWebページを送ったSEND HTML FILEで呼び出した4Dメソッドへ参照しに行きます。これらの別のページは別のページをアクセスするため4Dメソッドを参照するかリンクします。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. いくつかのWebページはSEND HTML FILE ("") を呼び出した4Dメソッドの参照を含むことができます。最初に行ったSEND HTML FILEの呼び出しを終了し、戻ってきます。最初にフリーWebナビゲーションを開始した4Dメソッドの実行を追いかけます。

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：**SEND HTML FILE**をWeb接続プロセスをしないプロセスから呼び出すと、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。

1. データベースストラクチャファイルを含むフォルダでは、HTMLドキュメントは "HomePage.HTM"を呼び出します。これはデータベースのデフォルトメニューバー#1を見る代わりにWebユーザがデータベースに接続したのを見るWebページです。このWebページは、アプリケーションのOn Web Connectionデータベースメソッドで書かれています。

'On Web Connectionデータベースメソッド

SEND HTML FILE ("HomePage.HTM")

2. データベースフォルダは以下のように組み立てます。

../Documents /CurrentWork/Databases/MyDB.4DB

../Documents/CurrentWork/Databases/MyDB.RSR

../Documents/CurrentWork/Databases/MyDB.4DD

../Documents/CurrentWork/Databases/WebStuff/HTM/HomePage

"HomePage.HTM"をWebページに送る方法

SEND HTML FILE ("WebStuff/HTM/HomePage.HTM")

または、

SET HTML ROOT ("WebStuff/HTM/")

SEND HTML FILE ("HomePage.HTM")

3. 4DWebが公開されている間、4Dフォームを使ったレコードの追加です。bHelpボタンはオブジェクトメソッドに以下のように定義します。

'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、データ入力に戻ってこられます。

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

SEND HTML BLOB、「Webサービス：HTMLとJavaScriptのカプセル化」、「Webサービス：入門編（パートII）」

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=コンテキストモードへの切り替え |

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。

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、またはその逆の指示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。引数が省略されると、デフォルトでコンテキスト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。

PICT TO GIFルーチン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。

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：Webプロセス以外のプロセスから呼び出されると、この関数は常にFalseを返します。

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。

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。

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

PROCESS PROPERTIES

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コマンドは、**SEND HTML FILE**コマンドの引数として渡すHTMLファイルを4Dが検索するデフォルトのディレクトリやフォルダを変更します。

デフォルトでは、4Dはデータベースのストラクチャファイルを含むディレクトリ内のHTMLドキュメントを検索します。

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。

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。

1. 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メソッド）は、HTMLドキュメントをブラウザに送信するSEND HTML FILEコマンドの呼び出しを発行します。
2. ブラウザに送られた最初のWebページには、他のWebページへのHTMLリンクがあったり、または他のWebページを送信するためのSEND HTML FILEコマンドを呼び出す4Dメソッドをページ自体で参照することができます。また、これらの他のページには、別のページをアクセスするための4Dメソッドへの参照やリンクが存在する場合もあります。Webページをナビゲートする間に、Backボタン等のブラウザのナビゲーション制御を使用することもできます。
3. 任意のWebページは、SEND HTML FILE（""）呼び出しを発行する4Dメソッドへの参照を含めることができます。この呼び出しは、すべてを開始したSEND HTML FILE呼び出しを終了し、自由なWebナビゲーションを本来起動した4Dメソッドの実行をさらに行って、元に戻ります。

注：「Web接続」プロセスではないプロセスから**SEND HTML FILE**コマンドを呼び出しても、コマンドは何も行わず、エラーも返りません。呼び出しが無視されるだけです。

例題

1. データベースストラクチャファイルを含むディレクトリには、"HomePage.HTM"というHTMLドキュメントがあります。これは、データベースに接続する時にデータベースのデフォルトのメニューバー番号1の代わりに表示したいWebページです。このWebページを表示するには、アプリケーションの「On Web Connection」データベースメソッドで以下のように指定します。

・「On Web Connection」データベースメソッド
SEND HTML FILE ("HomePage.HTM")

2. フォルダデータベースフォルダは、以下のように構成されています。

```
..¥Documents¥CurrentWork¥Databases¥MyDB.4DB  
..¥Documents¥CurrentWork¥Databases¥MyDB.RSR  
..¥Documents¥CurrentWork¥Databases¥MyDB.4DD  
..¥Documents¥CurrentWork¥Databases¥WebStuff¥HTM¥HomePage.HTM
```


Webページ"HomePage.HTM"は、以下の方法で送信できます。

SEND HTML FILE ("WebStuff/HTM/HomePage.HTM")

または、以下の方法でも送信できます。

SET HTML ROOT ("WebStuff/HTM/")

SEND HTML FILE ("HomePage.HTM")

3. 4D Webセッションの間、4Dフォームを使用してレコードを追加しています。このフォームには「bヘルプ」ボタンがあり、そのオブジェクトメソッドは以下の通りです。

・「bヘルプ」ボタンのオブジェクトメソッド

SEND HTML FILE ("Help.HTM")

Help.HTMドキュメントから開始して、ユーザはWebサイト用のデータベースHelpシステムを実装する多数のHTMLページを自由にナビゲートすることができます。各ページには、「Done」というサブミットボタンがあり、これによってデータ入力に戻ることができます。これを実行するには、各HTMLドキュメントにこのサブミットボタンの定義が含まれている必要があります。

<!--「bDone」サブミットボタン

<P><INPUT TYPE="submit" NAME="bDone" VALUE="Done"></P>

フォームポストアクションの設定は、以下のようになります。

<!--サブミットボタンがクリックされたら、「4D htm_Help_Done」メソッドを実行する

<FORM action="/4DMETHOD/htm_Help_Done" method="POST">

4D側では、プロジェクトメソッド「htm_Help_Done」は「bヘルプ」ボタンで起動された

SEND HTML FILEコマンドを終了させます。

・「htm_Help_Done」プロジェクトメソッド

SEND HTML FILE ("")

「bヘルプ」ボタンのオブジェクトメソッドでの**SEND HTML FILE**コマンドへの呼び出しは、メソッドの最後の行で行います。メソッドが完了した時には、データ入力に戻ります。

参照

「Webサービス：HTMLとJavaScriptのカプセル化」, 「Webサービス：入門編（パートI）」,

「Webサービス：入門編（パートII）」

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。

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。

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。

Charset、Expires、Content-Length、Content-Type、Date、Last-Modifiedのフィールドは、常に4Dによって設定されます。

カスタム "cookie" の例を示します。

SET HTTP HEADER ("SET-COOKIE:FUSER="+String (Abs (Random)) +";PATH=/")

参照
なし

SEND HTTP REDIRECT

SEND HTTP REDIRECT (url {; *})

| 引数 | タイプ | 説明 |
|-----|-----|---|
| url | 文字列 | 新しいURL |
| * | 文字列 | 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、リダイレクトした新しいURLが含まれています。この引数がファイルへのurlである場合、非コンテキストモードと同じファイル参照方法でなければいけません。例えば、**SEND HTTP REDIRECT** ("/MyPage.HTM")。HTTPヘッダ内のフィールドを設定したTextタイプのデータを引数フィールドへ渡します。

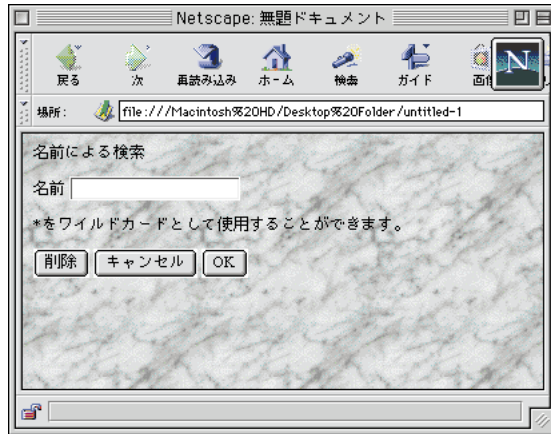
このコマンドがコンテキストモードで呼び出されると、実行された直後にWebプロセスはアポートされ、Webライセンスは開放されます。このコマンドは、同じメソッド内にあるデータを送るコマンド（**SEND HTML FILE**、**SEND HTML BLOB**等）より優先されます。

このコマンドは、他のWebサーバへ再問い合わせする事を可能にします。

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。

1. 拡張子".shtm"の付くページの内容は、常に4Dによって内容を解析され、必要な処置が施されます。

スタティックホームページを使って4D内で独自の検索を実行させるために、このコマンドを使用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。



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検索コードを
`実行するために URL を変更する
SEND HTTP REDIRECT ("/4dcgi/rech?" + name)
Else
`そうでなければ、最初のページに戻る
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
... `検索コードをここに入れる
END case

```

参照

なし

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

このコマンドは、Webブラウザを起動し <url> 引数で渡されたURL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。

4DはURLを自動的にエンコードしますが、*を渡すと4Dは翻訳しません。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。

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"

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。

下記のコードの行はブラウザを起動して、ACIのホームページに接続します。
OPEN WEB URL ("http://www.aci.co.jp/")

参照

なし

この章では、「ルーチン」エディタの「Windows」テーマ内にあるストラクチャアクセスコマンドについて説明します。この章のコマンドは、ウインドウの管理を行なうためのものです。ウインドウの管理には、カスタムウインドウのオープンとクローズ、および画面サイズの確保とウインドウタイトルの変更が含まれます。

| | | |
|----------------------------|-----------------------------|-------------------------|
| CLOSE WINDOW | HIDE WINDOW | REDRAW WINDOW |
| DRAG WINDOW | MAXIMIZE WINDOW | SET WINDOW RECT |
| ERASE WINDOW | MINIMIZE WINDOW | SET WINDOW TITLE |
| Find window | Next window | SHOW WINDOW |
| Frontmost window | Open window | WINDOW LIST |
| GET WINDOW RECT | Open external window | Window kind |
| Get window title | Open form window | Window process |
| GET FORM PROPERTIES | | |

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。

最低でもひと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がある場合は、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、再びスプラッシュ画面が表示されます。

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。

参照

Open window、ウインドウのタイプ

ウィンドウのタイプ

下記のあらかじめ定義された定数の1つを使って、**Open window**関数で開くウィ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 | フローティングウィンドウ |
|--------------------------|------|-----|--------------|
| 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 | 720 | 可 |
| Round corner window | 倍長整数 | 16 | 不可 |

フローティングウィンドウ：

Open window関数に上記の定数のひとつを渡すと、一般的なウィンドウを開きます。フローティングウィンドウを開くには、**Open window**関数に負数を渡します。

以下の図は、各ウィンドウタイプを示しています。左側がWindows、右側がMacintoshのウィンドウです。

タイプ8のウィンドウ（標準）



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

タイプ0のウィンドウ（「ズーム」ボックスなし）



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

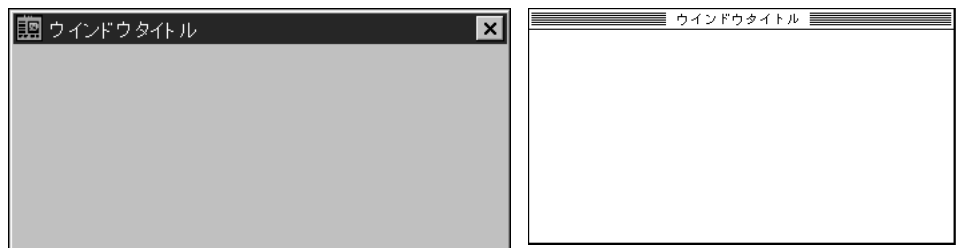
サイズ変更：可

「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

タイプ 4のウィンドウ（固定サイズ）



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

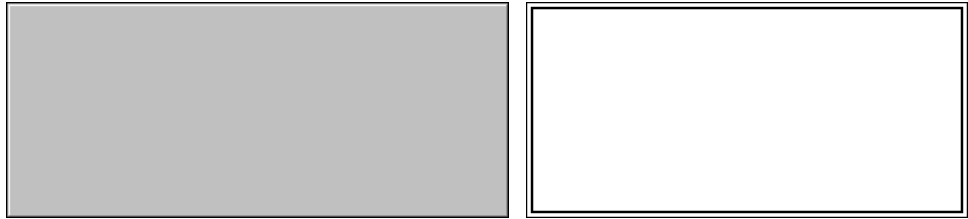
サイズ変更：Macintosh上では、不可

「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

タイプ1のウインドウ（モーダルダイアログボックス）



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

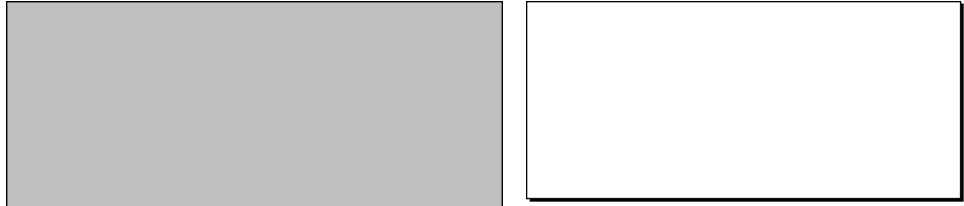
「最大化」 / 「最小化」ボタンまたは「ズーム」ボックス付加：不可

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

タイプ3のウインドウ（予備のダイアログボックス）



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

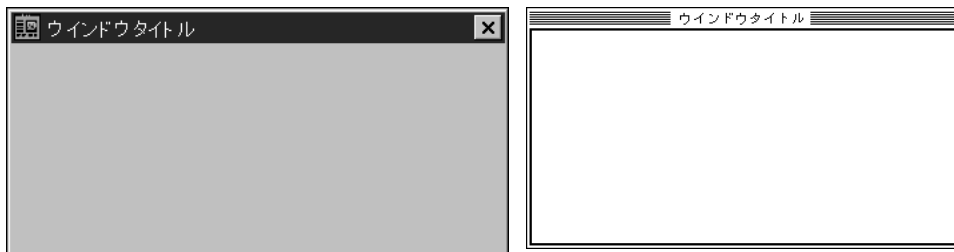
「最大化」 / 「最小化」ボタンまたは「ズーム」ボックス付加：不可

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

タイプ5のウィンドウ（移動可能なダイアログボックス）



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

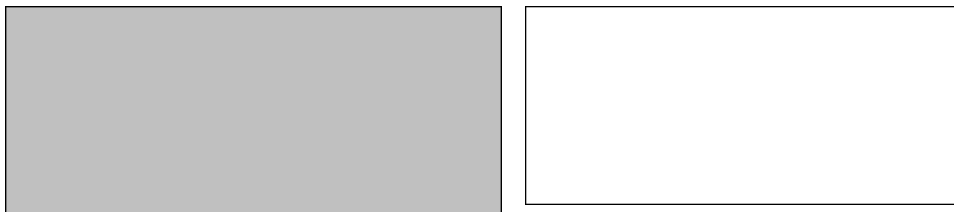
「最大化」 / 「最小化」ボタンまたは「ズーム」ボックス付加：不可

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

このタイプのウィンドウはモーダル（形式）ウィンドウですが、フローティングウィンドウとして移動、または使用することが可能

タイプ2のウィンドウ（標準のダイアログボックス）



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

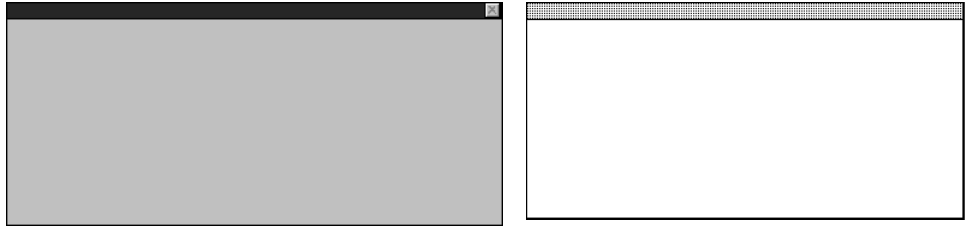
「最大化」 / 「最小化」ボタンまたは「ズーム」ボックス付加：不可

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、スブラッシュスクリーン

フローティングウィンドウとして使用されない場合、このタイプのウィンドウはモーダル（形式）ウィンドウです

タイプ (720 {+1} {+2} {+4} {+8}) のパレットウィンドウ



Open window関数を呼び出すと、パレットウィンドウに下記の定数を1つまたは複数を追加して、そのウィ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

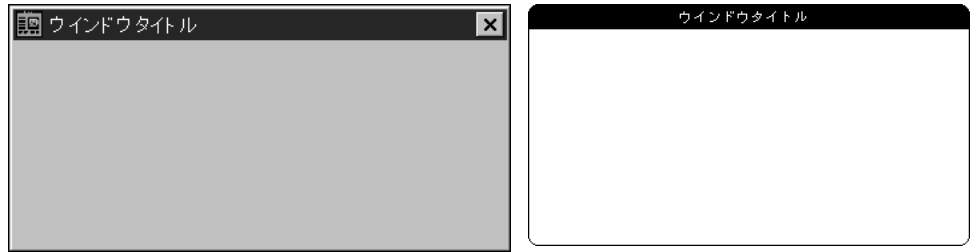
サイズ変更：可、「Has grow box」定数が指定された場合

「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（データ入力なし）を用いたフローティングウィンドウ

タイプ16のウインドウ（角の丸いウインドウ）



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：滅多に使用しない

参照

Open external 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 - 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
Macintosh版では (クロ - ズボックス) | 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を渡す場合、以下の条件でウインドウを自動的にサイズ決定するように4th Dimensionに指示します。

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、オプション引数<*>を渡した**INPUT FORM**コマンドを使ってそのフォームを選択する。

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 - 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。この技法に関する詳細は、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 - ウインドウの幅
` $2 - ウインドウの高さ
` $3 - ウインドウのタイプ (オプション)
` $4 - ウインドウのタイトル (オプション)
$S幅:=Screen width / 2
$S高さ:=(Screen height / 2) - 10
$W高さ:=$1 / 2
$W幅:=$2 / 2
` 画面の中心を求める (幅)
` 画面の中心を求める (高さ)
` ウインドウの高さの半分を求める
` 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。

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。

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。

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

SHOW WINDOW

SHOW WINDOW {{(ウインドウ)}}

| 引数 | タイプ | 説明 |
|-------|-----------|-------------------------------------|
| ウインドウ | ウインドウ参照番号 | 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

HIDE WINDOW

HIDE WINDOW {{(ウインドウ)}}

| 引数 | タイプ | 説明 |
|-------|-----------|-------------------------------------|
| ウインドウ | ウインドウ参照番号 | 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。
「デザイン」モードでプロセスリストを使用する。ウインドウを処理しているプロセスを選択し、「プロセス」メニューから「表示」を選択する。

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

If (OK=1) ` ユーザがダイアログを確定した場合

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

End if

参照

SHOW WINDOW

CLOSE WINDOW

CLOSE WINDOW {(プラグインウインドウ参照番号)}

| 引数 | タイプ | 説明 |
|--------------------|-----|--|
| プラグインウインドウ
参照番号 | 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 < プラグインウインドウ参照番号 > 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。

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

` 取り消すまでループする
` 新しいレコードを追加する
` 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の内容を消去します。

<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

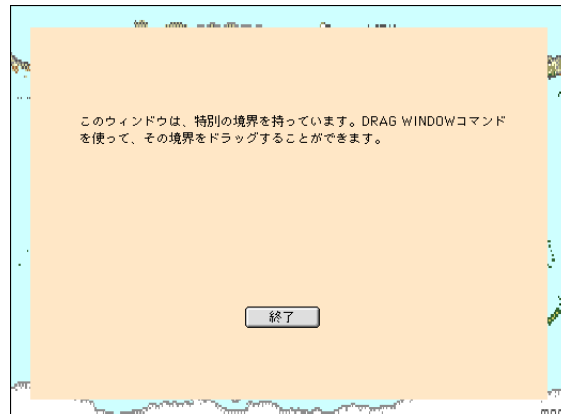
以下のフォームは、「デザイン」モードで表示されていますが、固定のピクチャで作成された枠を持っています。このピクチャの枠の各コーナーには4つの透明ボタンが設定されています。



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、ウィ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

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。

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ウ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
参照番号 | 配列 | 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 ($aWnd)
$viLeft:=10
$viTop:=80 `ツールバー用の空間を確保する
For ($viWnd ; 1 ; Size of array ($aWnd))
  If (Window kind ($aWnd{$viWnd} # Modal Dialog)
    GET WINDOW RECT ($viWL ; $viWT ; $viWR ; $viWB ; $aWnd{$viWnd})
    $viWR:=$viLeft+($viWR-$viWL)
    $viWB:=$viTop+($viWB-$viWT)
    $viWL:=$viLeft
    $viWT:=$viTop
    SET WINDOW RECT ($viWL ; $viWT ; $viWR ; $viWB ; $aWnd{$viWnd})
    $viLeft:=$viLeft+10
    $viTop:=$viTop+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。

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
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

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

SET WINDOW RECTコマンドは、引数<ウインドウ数値番号>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。

例題

WINDOW LISTコマンドの例題を参照してください。

参照

DRAG WINDOW、GET WINDOW RECT

Frontmost window

Frontmost window {(*)} ウィンドウ参照番号

| 引数 | タイプ | 説明 |
|----|-----|--|
| * | * | 指定した場合は、フローティングウィンドウを考慮
省略した場合は、フローティングウィンドウを無視 |

| | | |
|-----|----|-----------|
| 戻り値 | 数値 | ウィンドウ参照番号 |
|-----|----|-----------|

説明
Frontmost window関数は、最前面ウィンドウのウィンドウ参照番号を返します。

参照

Frontmost process、Next window

Next window

Next window {(ウィンドウ)} 数値

| 引数 | タイプ | 説明 |
|---------------|-----|---|
| ウィンドウ
参照番号 | 数値 | ウィンドウ参照番号、省略した場合は、
カレントプロセスの最前面ウィンドウ |

| | | |
|-----|----|-----------|
| 戻り値 | 数値 | ウィンドウ参照番号 |
|-----|----|-----------|

説明
Next window関数は、（ウィンドウの前後の順番を基にした）引数<ウィンドウ参照番号>に渡された参照番号を持つウィンドウの“背面”にあるウィンドウのウィンドウ参照番号を返します。もし、ウィ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ひとつを返します。

| 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

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

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。



参照

MAXIMIZE 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
参照番号 | 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ひとつを渡します。

| 定数 | タイプ | 値 |
|--------------------|------|-----|
| Plain window | 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 |

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 “ Window position ” テーマ内にある定数の内からひとつを渡します。

| 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 “ Window position ” テーマ内にある定数の内からひとつを渡します。

| 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。

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 window; On the Left; At the Top;*)

参照

Open window、GET FORMS PROPERTIES

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。

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。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。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。

参照

Open form window

この付録では、4th Dimensionのシステム変数についての概要を示します。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。以下のコマンドは、システム変数OKに値を代入します。

| | | |
|--------------------------------|-----------------------------------|-----------------------------|
| ACCEPT | ADD RECORD | ADD SUBRECORD |
| Append document | APPEND TO CLIPBOARD | APPLY TO SELECTION |
| ARRAY TO LIST | ARRAY TO SELECTION | ARRAY TO STRING LIST |
| BLOB TO DOCUMENT | CANCEL | |
| CHANGE ACCESS | COMPRESS BLOB | CONFIRM |
| Create document | Create resource file | DELETE DOCUMENT |
| DELETE RESOURCE | DIALOG | DISTINCT VALUES |
| DOCUMENT TO BLOB | EXPAND BLOB | EXPORT DIF |
| EXPORT SYLK | EXPORT TEXT | GET ICON RESOURCE |
| Get indexed string | GET PICTURE FROM LIBRARY | |
| GET PICTURE RESOURCE | GET RESOURCE | Get string resource |
| Get text from clipboard | Get text resource | |
| IMPORT DIF | IMPORT SYLK | IMPORT TEXT |
| LOAD SET | LOAD VARIABLES | MODIFY RECORD |
| MODIFY SUBRECORD | Open document | Open resource file |
| ORDER BY | ORDER BY FORMULA | PLAY |
| PRINT LABEL | PRINT SELECTION | PRINT SETTINGS |
| QUERY | QUERY BY EXAMPLE | QUERY BY FORMULA |
| QUERY SELECTION | QUERY SELECTION BY FORMULA | |
| RECEIVE PACKET | RECEIVE RECORD | RECEIVE VARIABLE |
| RELATE MANY SELECTION | | |

| | | |
|---------------------------------|--------------------------------|------------------------------|
| RELATE ONE SELECTION | REPORT | Request |
| SAVE SET | SAVE VARIABLES | SELECT LOG FILE |
| SEND PACKET | SEND RECORD | SEND VARIABLE |
| SET CHANNEL | SET PICTURE RESOURCE | |
| SET PICTURE TO CLIPBOARD | | SET RESOURCE |
| SET RESOURCE NAME | SET RESOURCE PROPERTIES | |
| SET STRING RESOURCE | SET TEXT RESOURCE | SET TEXT TO CLIPBOARD |
| SET TIMEOUT | START WEB SERVER | STRING LIST TO ARRAY |
| USE ASCII MAP | VALIDATE TRANSACTION | |

Document

システム変数Documentには、以下のコマンドで開かれたり、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 |
| REPORT | SAVE SET | SAVE VARIABLES |
| 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はマウスがクリックされた場所の水平と垂直の座標を含みます。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。および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 “ イベントの中断 ” 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。

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 | ASCII | | | MacintoshまたはWindows |
|-------|-----|-----|---------------------|-------|-----|-----|---------------------|
| 10進 | 16進 | 結果 | | 10進 | 16進 | 結果 | |
| 0 | 0 | NUL | | 9 | 9 | HT | |
| 1 | 1 | SOH | | 10 | A | LF | |
| 2 | 2 | STX | | 11 | B | VT | |
| 3 | 3 | ETX | | 12 | C | FF | |
| 4 | 4 | EOT | | 13 | D | CR | |
| 5 | 5 | ENQ | | 14 | E | SO | |
| 6 | 6 | ACK | | 15 | F | SI | |
| 7 | 7 | BEL | | 16 | 10 | DLE | |
| 8 | 8 | BS | | 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 | F | # |
| 36 | 10 | \$ |
| 37 | 11 | % |
| 38 | 12 | & |
| 39 | 13 | ' |
| 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 | F | E |
| 70 | 10 | F |
| 71 | 11 | G |
| 72 | 12 | H |
| 73 | 13 | I |

| ASCII | | MacintoshまたはWindows | ASCII | | MacintoshまたはWindows |
|-------|-----|---------------------|-------|-----|---------------------|
| 10進 | 16進 | 結果 | 10進 | 16進 | 結果 |
| 74 | 4A | J | 101 | 55 | e |
| 75 | 4B | K | 102 | 56 | f |
| 76 | 4C | L | 103 | 57 | g |
| 77 | 4D | M | 104 | 58 | h |
| 78 | 4E | N | 105 | 59 | i |
| 79 | 4F | O | 106 | 5A | j |
| 80 | 50 | P | 107 | 5B | k |
| 81 | 51 | Q | 108 | 5C | l |
| 82 | 52 | R | 109 | 5D | m |
| 83 | 53 | S | 110 | 5E | n |
| 84 | 54 | T | 111 | 5F | o |
| 85 | 55 | U | 112 | 70 | p |
| 86 | 56 | V | 113 | 71 | q |
| 87 | 57 | W | 114 | 72 | r |
| 88 | 58 | X | 115 | 73 | s |
| 89 | 59 | Y | 116 | 74 | t |
| 90 | 5A | Z | 117 | 75 | u |
| 91 | 5B | [| 118 | 76 | v |
| 92 | 5C | ¥ | 119 | 77 | w |
| 93 | 5D |] | 120 | 78 | x |
| 94 | 5E | ^ | 121 | 79 | y |
| 95 | 5F | _ | 122 | 7A | z |
| 96 | 60 | ` | 123 | 7B | { |
| 97 | 61 | a | 124 | 7C | |
| 98 | 62 | b | 125 | 7D | } |
| 99 | 63 | c | 126 | 7E | |
| 100 | 64 | d | 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 | â | option+a | â | 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 | ¶ (¶) | option+7 | ¶ | Alt+0182 |
| 167 | A7 | ß (ß) | option+s | ß | Alt+0223 |
| 168 | A8 | ® (®) | option+r | ® | Alt+0174 |
| 169 | A9 | © (©) | option+g | © | Alt+0169 |
| 170 | AA | ™ (™) | option+2 | ™ | Alt+0153 |
| 171 | AB | ´ (´) | 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 | ¨ (ʹ) | shift+option+u | ¨ | Alt+0168 |
| 173 | AD | ≠ (1) | option+= | | |
| 174 | AE | Æ (3) | shift+option+" | Æ | Alt+0198 |
| 175 | AF | Ø (ʹ) | shift+option+o | Ø | Alt+0216 |
| 176 | B0 | ∞ (-) | option+5 | | |
| 177 | B1 | ± (7) | shift+option+= | ± | Alt+0177 |
| 178 | B2 | ≤ (1) | option+, | | |
| 179 | B3 | ≥ (7) | option+.(period) | | |
| 180 | B4 | ¥ (I) | option+y | ¥ | Alt+0165 |
| 181 | B5 | μ (J) | option+m | μ | Alt+0181 |
| 182 | B6 | ∂ (k) | option+d | | |
| 183 | B7 | Σ (k) | option+w | | |
| 184 | B8 | Π (k) | shift+option+p | | |
| 185 | B9 | π (k) | option+p | | |
| 186 | BA | ∫ (1) | option+b | | |
| 187 | BB | ^a (ʹ) | option+9 | ^a | Alt+0170 |
| 188 | BC | ° (j) | option+0 (zero) | ° | Alt+0186 |
| 189 | BD | Ω (λ) | option+z | | |
| 190 | BE | æ (t) | option+: | æ | Alt+0230 |
| 191 | BF | ø (ʹ) | option+o | ø | Alt+0248 |
| 192 | C0 | ı (ʹ) | shift+option+? | ı | Alt+0191 |
| 193 | C1 | ï (ʹ) | option+l | ï | Alt+0161 |
| 194 | C2 | ¬ (ʹ) | option+l (L) | ¬ | Alt+0172 |
| 195 | C3 | √ (7) | option+v | | |
| 196 | C4 | f (t) | 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 | ≈ (ヲ) | 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 (ル) | option+`, shift+a | Ã | Alt+0192 |
| 204 | CC | Ã (リ) | option+n, shift+a | Å | Alt+0195 |
| 205 | CD | Õ (ハ) | option+n, shift+o | Ö | Alt+0213 |
| 206 | CE | Œ (ホ) | shift+option+q | | |
| 207 | CF | œ (ロ) | 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 | ‘ (ヱ) | option+[| ‘ | Alt+0145 |
| 213 | D5 | ’ (ヰ) | shift+option+[| ’ | Alt+0146 |
| 214 | D6 | ÷ (ヱ) | option+/- | ÷ | Alt+0247 |
| 215 | D7 | ◇ (ヱ) | shift+option+v | | |
| 216 | D8 | ÿ (リ) | option+u, y | ÿ | Alt+0255 |
| 217 | D9 | Ÿ (ル) | option+u, shift+y | | |
| 218 | DA | / (ル) | shift+option+l | | |
| 219 | DB | ¤ (ロ) | shift+option+2 | ¤ | Alt+0164 |
| 220 | DC | < (リ) | shift+option+3 | | |
| 221 | DD | > (リ) | 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 | ˘ | 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 |

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。 |
| 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。 |

| | |
|-----|---|
| コード | エラーの起きた理由 |
| 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。 |
| 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。 |

コード エラーの起きた理由

- 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
命令文で使うことはできません。
- 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が変数が必要です。
- 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。

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。 |
| -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セレクトが見つかりません。 |
| -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。フォームが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。

| | |
|--------|--|
| コード | エラーの起きた理由 |
| -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4D Open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。

ネットワークコンポーネントエラーコード

次の表は、ネットワークコンポーネント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きました。 |
| -10031 | 読み込みサイクル中に非同調が起きました。 |
| -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。 |
| 2 | OP Select 4D Server使用中に「他...」ボタンがクリックされました。 |

システムエラーコード

この節は、それぞれの管理者のためのエラー - コ - ドを示します。

注：次のコードは、Macintoshのオペレーションシステムによって返されるエラーを基にしています。このコードはWindows上でも同じように返されるので、いずれかのプラットフォーム上で同じエラーコードを検査することができます。

File Managerのエラ -

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てしました。 |
| -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てしました。 |
| -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てしました。 |
| -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てしました。 |
| -49 | 既に開かれているファイルを開こうとしてしました。 |
| -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てしました。 |
| -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てしました。 |
| -57 | Windows用のディスクでないもので作業しようとしてしました。 |

| コード | エラーの起きた理由 |
|------|---|
| -58 | Windows用のディスクでないもので作業しようとしてしました。 |
| -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てしました。 |
| -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てしました。 |

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。

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きました。 |
| -27 | プリンタの接続時または接続解除時にエラーが起きました。 |
| -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。 |

Resource Managerのエラ -

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。 |

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てしました。 |
| 20 | Compタイプの数を浮動小数点に変換しようとしてしました。 |
| 21 | NaNコードを0コードで作ろうとしてしました。 |
| 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。 |

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。 |

システムエラー

次の表は、MacOS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。 |

データファイルのロック状態のテスト

データファイルまたはデータファイルが配置されているボリュームのロック状態をテストするには、オペレーションシステム（OS）のファイルマネージャを呼び出す必要があります。データファイルを変更する各データベース処理に対してこれを行うと、データベースエンジンのパフォーマンスに重大な影響を与えることでしょう。

あなた自身の責任のもとでワークセッションの開始時にこのロック状態をテストします。通常、あなたのデータベースの「On Startup」データベースメソッドの中でこれを行います。4D 3.2.5と4D Server1.2.5までのバージョンでは、データファイルのロック状態をテストするには、データファイルおよびそのデータファイルが配置されているボリュームのファイルマネージャ属性を返す外部ルーチンの使用が必要でした。しかし、今バージョンの4Dと4D Serverでは、新規レコードを作成しようとする度にロックされたデータファイルに合図を送ることにより簡単にこのテストを行うことができます。データファイルまたはそのボリュームがロックされている場合、データベースエンジンはエラーコード-9999（レコードを保存するディスクの領域が足りません）を生成します。

次のコードは、データファイルのロック状態をテストする方法を示した例です。

```
`「Is data locked」プロジェクトメソッド
` Is data locked   ブール
` Is data locked -> データファイルがロックまたはいっぱいの場合はTrue
gError:=0
ON ERR CALL ("エラー処理")
CREATE RECORD ([任意テーブル])
SAVE RECORD ([任意テーブル])
ON ERR CALL ("")
If (gError=0)
    DELETE RECORD ([任意テーブル])
End if
$0:=(gError=-9999)
```

“エラー処理”と名付けられた**ON ERR CALL**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
```

すると、あなたのデータベースの「On Startup」データベースメソッドで次のように記述できます。

```
`「On Startup」データベースメソッド
```

```
` ...
```

```
If (Is data locked)
```

```
`データファイルがロックされていたり、データファイルがあるボリュームがロック
```

```
`されていたりいっぱいの場合は、その状態を示すダイアログボックスを表示します。
```

```
`どちらのケースでも、リードオンリー状態でデータファイルを使用したい場合がある
```

```
`かもしれません。つまり、そのデータファイルがCD-ROMボリューム上に配置されて
```

```
`いるかもしれません。この場合、そのダイアログボックスは「リードオンリーで使用
```

```
`中」ボタンと「終了」ボタンを持っているかもしれません。
```

```
If (bQuit=1)
```

```
QUIT 4D `データベースを抜けて、ファインダレベルに戻ったかチェックする
```

```
Else
```

```
`データファイルがロックされていることを示すインタープロセスフラグをセットする
```

```
<>gREADONLY:=True
```

```
`起動時の実行処理を続ける
```

```
End if
```

```
End if
```

ロックされたデータファイルを使ってデータベースの使用を許可する場合、そのデータベースの内容を変更するかもしれない操作へのアクセス権を制限するようにしてください。これを行うには、テスト関数を再度呼び出すか、または上記の例のようにインタープロセスフラグを管理します。例えば、顧客レコードを追加するプロセスを開始する「M_ADD_CUST」というプロジェクトメソッドがある場合、このメソッドが実行可能かどうかは前もってわかっています。もし、これがわからないと、このプロセスを開始しても無駄です。例えば、

```
`「M_ADD_CUST」プロジェクトメソッド
```

```
If (Can write data)
```

```
`先に進む
```

```
<>vIPID_CUST:=New process(...;...;...)
```

```
` ...
```

```
End if
```

「Can write data」プロジェクトメソッドを次に示します。

```
`「Can write data」プロジェクトメソッド
```

```
`Can write data -> ブール
```

```
`Can write data -> データファイルがロックされていない場合はTrue
```

```
$0:=True
```

```
If (<>gREADONLY) `または If (Is data locked)
```

```
ALERT ("この処理はリードオンリー状態のデータファイルでは実行されません。")
```

```
$0:=False
```

```
End if
```

重要：4D 3.2.5と4D Server 1.2.5（またはこれ以前）のデータベースエンジンのパフォーマンスを保護するには、データファイルの内容を修正する可能性がある各データベース操作中にデータファイルのロック状態をチェックしないでください。例えば、トランザクション内でレコードを追加している場合、そのトランザクションが有効になるまではデータファイルには手が加えられないことを覚えておいてください。トランザクション内で、変更処理中にデータのロック状態をテストすると、オーバーヘッドのテストが不要に追加されてしまいます。したがって、データファイルのロック状態は、トランザクション内ではない時に新規レコードを作成する場合にのみテストされます。これは、「ユーザ」モードにおける新規レコードの追加および読み込み処理を含みます。また、**ADD RECORD**コマンドや**SAVE RECORD**コマンド（新規レコードに使用される）および**ARRAY TO SELECTION**コマンド（新規レコードを作成する）も含みます。しかし、これはトランザクション内における新規レコード作成処理、既存レコードの修正または削除処理は含みません。

注：データファイルのロック状態のテストを行っても、データファイルへの書き込み中に発生する他のI/Oエラーのテストは続けることができます。

この付録では、バージョン6の4th Dimensionであらかじめ定義されている定数を各項目別に示します。この定数は、「デザイン」モードの「エクスプローラ」ウィンドウ上にある「定数」ページにあるものと同じです。

4D Environment

| 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

| 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 |

ASCII Codes (続き)

| 定数 | タイプ | 値 |
|----------------|------|-----|
| 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 |
| 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 |

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

| 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

| 定数 | タイプ | 値 |
|------------|------|----|
| Black | 倍長整数 | 15 |
| Blue | 倍長整数 | 6 |
| 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 |

Communications

| 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 |
| MacOS Printer Port | 倍長整数 | 0 |
| MacOS 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 |
| 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

| 定数 | タイプ | 値 |
|-------------------|------|----|
| Is new record | 倍長整数 | -3 |
| No current record | 倍長整数 | -1 |

Database Events

| 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

| 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

| 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

| 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 |

Events (Modifiers)

| 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)

| 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 |

Field and Variable Types

| 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

| 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

| 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 Events

| 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 |

Function Keys

| 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

| 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 |

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 Latin Character Entities (続き)

| 定数 | タイプ | 値 |
|-------------------------|-----|----------|
| ISO L1 Cap U circumflex | 文字列 | Û |
| ISO L1 Cap U grave | 文字列 | Ù |
| 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 |

Math

| 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 |

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 Display Formats

| 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

| 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 3.1 interface | 倍長整数 | 1 |
| Windows 95 interface | 倍長整数 | 2 |

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 |

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

| 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 |

| 定数 | タイプ | 値 |
|------------------------------|------|-----|
| 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 |

Query Destinations

| 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 |

Resources 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

| 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

| 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

| 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 |

TCP Port Numbers

| 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

| 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 |

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 |

Window Position

| 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 |
| On the Left | 倍長整数 | 131072 |
| On the Right | 倍長整数 | 196608 |
| Vertically Centered | 倍長整数 | 262144 |

Windows NT Log Events

| 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 |

A

| | |
|--|------|
| ABORT | 561 |
| Abs (数値) 数値 | 610 |
| ACCEPT | 431 |
| ACCUMULATE (データ1 {;...;データN}) | 742 |
| ACI folder 文字列 | 236 |
| Activated | 472 |
| ADD DATA SEGMENT | 231 |
| ADD RECORD ({テーブル}{;}{*}) | 389 |
| ADD SUBRECORD(サブテーブル;フォーム;{*}) | 392 |
| Add to date (日付;年;月;日) 日付 | 407 |
| ADD TO SET({テーブル;}セット) | 967 |
| After | 470 |
| ALERT(メッセージ{;OKボタンタイトル}) | 653 |
| ALL RECORDS ({テーブル}) | 936 |
| ALL SUBRECORDS(サブテーブル) | 1022 |
| Append document (ドキュメント{;形式}) ドキュメントファイル参照番号 | 1046 |
| APPEND MENU ITEM(メニュー;テキスト{;プロセス}) | 647 |
| APPEND TO CLIPBOARD (データタイプ;データ) | 334 |
| 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}}) | 514 |
| Application file 文字列 | 241 |
| Application type 倍長整数 | 244 |
| Application version({*}) 文字列 | 243 |
| APPLY TO SELECTION ({テーブル;}ステートメント) | 952 |
| APPLY TO SUBSELECTION(サブテーブル;ステートメント) | 1023 |
| Arctan (数値) 数値 | 621 |
| ARRAY BOOLEAN (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY DATE (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY INTEGER (配列名;サイズ1 ;{サイズ2}) | 271 |

| | |
|---|-----|
| ARRAY LONGINT (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY PICTURE (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY POINTER (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY REAL (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY STRING (長さ;配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY TEXT (配列名;サイズ1 ;{サイズ2}) | 271 |
| ARRAY TO LIST (配列;リスト;{項目参照}) | 281 |
| ARRAY TO SELECTION (配列1 ;フィールド1 ;{...;配列N ;フィールドN}) | 283 |
| ARRAY TO STRING LIST(ストリング;リソースID {;リソースファイル}) | 910 |
| Ascii (文字) 数値 | 987 |
| AUTOMATIC RELATIONS(1リレート{nリレート}) | 869 |
| Average (一連の値) 数値 | 702 |

B

| | |
|--|------|
| BEEP | 1130 |
| Before | 469 |
| Before selection {{テーブル}} ブール | 944 |
| Before subselection(サブテーブル) ブール | 1026 |
| BLOB PROPERTIES (blob ;圧縮{;サイズ{現在サイズ}}) | 299 |
| BLOB size(blob) 数値 | 295 |
| BLOB TO DOCUMENT (ドキュメント;blob {;*}) | 303 |
| BLOB to integer (blob ;バイト順序{;オフセット}) 数値 | 320 |
| BLOB to list (blob {;オフセット}) 数値 | 310 |
| BLOB to longint (blob ;バイト順序{;オフセット}) 数値 | 322 |
| BLOB to real (blob ;実数形式{;オフセット}) | 323 |
| 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 | 325 |
| BLOB TO VARIABLE (blob ;変数{;オフセット}) | 308 |
| BOOLEAN ARRAY FROM SET(ブール配列{;セット}) | 329 |
| BREAK LEVEL (レベル{;ページブレーク}) | 741 |
| BRING TO FRONT (プロセス) | 769 |
| BUTTON TEXT ({*;}オブジェクト;ボタンテキスト) | 680 |

C

| | |
|--------------------------------------|-----|
| C_BLOB ({メソッド;}変数{;変数2 ;...;変数N}) | 374 |
| C_BOOLEAN ({メソッド;}変数{;変数2 ;...;変数N}) | 375 |
| C_DATE ({メソッド;}変数{;変数2 ;...;変数N}) | 376 |

| | |
|--|------|
| C_GRAPH ({メソッド;}変数{;変数2;...;変数N}) | 377 |
| C_INTEGER ({メソッド;}変数{;変数2;...;変数N}) | 378 |
| C_LONGINT ({メソッド;}変数{;変数2;...;変数N}) | 379 |
| C_PICTURE({メソッド;}変数{;変数2;...;変数N}) | 380 |
| C_POINTER ({メソッド;}変数{;変数2;...;変数N}) | 381 |
| C_REAL ({メソッド;}変数{;変数2;...;変数N}) | 382 |
| C_STRING ({メソッド;}サイズ;変数{;変数2;...;変数N}) | 383 |
| C_TEXT ({メソッド;}変数{;変数2;...;変数N}) | 384 |
| C_TIME ({メソッド;}変数{;変数2;...;変数N}) | 385 |
| CALL PROCESS(プロセス) | 758 |
| CANCEL | 432 |
| CANCEL TRANSACTION | 1108 |
| Caps lock down プール | 1141 |
| CHANGE ACCESS | 1160 |
| CHANGE LICENSE | 1176 |
| CHANGE PASSWORD (パスワード) | 1161 |
| Change string (元の文字列;修正文字列;修正位置) 文字列 | 991 |
| Char (ASCIIコード) 文字列(1バイト) | 989 |
| CLEAR CLIPBOARD | 340 |
| CLEAR LIST(リスト{;*}) | 497 |
| CLEAR NAMED SELECTION (名前) | 671 |
| CLEAR SEMAPHORE (セマフォ) | 756 |
| CLEAR SET(セット) | 968 |
| CLEAR VARIABLE (変数) | 1181 |
| CLOSE DOCUMENT (ドキュメントファイル参照番号) | 1047 |
| CLOSE RESOURCE FILE(リソースファイル) | 904 |
| CLOSE WINDOW {(プラグインウインドウ参照番号)} | 1320 |
| Command name(コマンド番号) 文字列 | 578 |
| Compiled application プール | 242 |
| COMPRESS BLOB (blob {;圧縮}) | 295 |
| COMPRESS PICTURE (ピクチャ;方法;品質) | 713 |
| COMPRESS PICTURE FILE (ドキュメント参照番号;方法;品質) | 715 |
| 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}}) | 655 |
| Convert case(文字列;変換タイプ;対象タイプ) 文字列 | 995 |
| COPY ARRAY (コピー元;コピー先) | 275 |
| 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) | 328 |

| | |
|---|------|
| COPY DOCUMENT(コピー元;コピー先{*}) | 1049 |
| Copy list(リスト) 数値 | 497 |
| COPY NAMED SELECTION({テーブル;}名前) | 668 |
| COPY SET(コピー元;コピー先) | 976 |
| Cos (数値) 数値 | 620 |
| Count fields (テーブル番号;テーブルポインタ) 数値 | 1006 |
| Count list items(リスト) 倍長整数 | 499 |
| Count menu items(メニュー{*;プロセス}) 数値 | 637 |
| Count menus({プロセス}) 数値 | 637 |
| Count parameters 数値 | 567 |
| Count screens 数値 | 1081 |
| Count tables 数値 | 1006 |
| Count tasks 整数 | 790 |
| Count user processes 整数 | 791 |
| Count users 整数 | 790 |
| 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 | 1042 |
| CREATE EMPTY SET({テーブル;}セット) | 963 |
| CREATE FOLDER(パス名) | 1053 |
| CREATE RECORD({テーブル}) | 847 |
| CREATE RELATED ONE(フィールド) | 875 |
| Create resource file(リソースファイル名{*;ファイルタイプ}) DocRef | 901 |
| CREATE SELECTION FROM ARRAY({テーブル;}レコード配列
{;セクション名}) | 671 |
| CREATE SET FROM ARRAY({テーブル;}レコード配列;{セット名}) | 965 |
| CREATE SET({テーブル;}セット) | 964 |
| CREATE SUBRECORD(サブテーブル) | 1020 |
| Current date ({*}) 日付 | 400 |
| Current default table ポインタ | 1095 |
| Current form page 数値 | 481 |
| Current form table ポインタ | 1099 |
| Current machine owner 文字列 | 1088 |
| Current machine 文字列 | 1088 |
| Current process プロセス参照番号 | 784 |
| Current time ({*}) 時間 | 410 |
| Current user 文字列 | 1163 |
| CUT NAMED SELECTION ({テーブル;}名前) | 669 |

D

| | |
|--|------|
| Data file ({セグメント}) 文字列 | 239 |
| DATA SEGMENT LIST ({セグメント}) | 245 |
| Database event 数値 | 1124 |
| Date (日付文字列) 日付 | 408 |
| Day number (日付) 数値 | 405 |
| Day of (日付) 数値 | 402 |
| Deactivated | 473 |
| Dec (数値) 数値 | 611 |
| DEFAULT TABLE (テーブル) | 1093 |
| DELAY PROCESS (プロセス;遅延時間) | 778 |
| DELETE DOCUMENT (ドキュメント) | 1051 |
| DELETE ELEMENT (配列;削除位置;{要素数}) | 277 |
| DELETE FROM BLOB (blob ;オフセット;削除数) | 328 |
| DELETE LIST ITEM({リスト;項目参照番号!*;{*}) | 528 |
| DELETE MENU ITEM(メニュー;メニューアイテム{*;プロセス}) | 650 |
| DELETE RECORD({テーブル}) | 853 |
| DELETE RESOURCE(リソースタイプ;リソースID {;リソースファイル}) | 932 |
| DELETE SELECTION {(テーブル)} | 937 |
| Delete string (元の文字列;削除位置;文字数) 文字列 | 993 |
| DELETE SUBRECORD(サブテーブル) | 1021 |
| DELETE USER(ユーザID) | 1164 |
| DIALOG ({テーブル;}フォーム) | 394 |
| DIFFERENCE(セット1 ;セット2 ;結果セット) | 972 |
| DISABLE BUTTON({*;}オブジェクト) | 679 |
| DISABLE MENU ITEM(メニュー番号;メニューアイテム{*;プロセス}) | 646 |
| DISPLAY RECORD {(テーブル)} | 846 |
| DISPLAY SELECTION {(テーブル){*};{*}) | 948 |
| DISTINCT VALUES(フィールド;配列) | 284 |
| Document creator(ドキュメント) 文字列 | 1063 |
| DOCUMENT LIST(パス名;ドキュメント) | 1058 |
| DOCUMENT TO BLOB (ドキュメント;blob {;{*}) | 301 |
| Document type(ドキュメント) 文字列 | 1059 |
| DRAG AND DROP PROPERTIES(対象オブジェクト;対象要素;対象プロセス) | 424 |
| DRAG WINDOW | 1322 |
| Drop position 整数 | 423 |

| | |
|--------------------------|-----|
| DUPLICATE RECORD({テーブル}) | 848 |
| During | 469 |

E

| | |
|---|------|
| EDIT ACCESS | 1160 |
| ENABLE BUTTON({*;}オブジェクト) | 678 |
| ENABLE MENU ITEM(メニュー番号;メニューアイテム{;プロセス}) | 646 |
| End selection({テーブル}) ブール | 946 |
| End subselection(サブテーブル) ブール | 1027 |
| ERASE WINDOW {(ウインドウ)} | 1321 |
| Euro converter(値;元の通貨;変換通貨) 実数 | 623 |
| EXECUTE (ステートメント) | 577 |
| EXECUTE ON CLIENT(クライアント名;メソッド名{;引数1;...;引数N}) | 791 |
| 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 | 774 |
| Exp (数値) 数値 | 618 |
| EXPAND BLOB (blob) | 297 |
| EXPORT DATA(ファイル名{;プロジェクト{*}}) | 551 |
| EXPORT DIF({テーブル;}ドキュメント) | 547 |
| EXPORT SYLK({テーブル;}ドキュメント) | 543 |
| EXPORT TEXT({テーブル;}ドキュメント) | 539 |

F

| | |
|---|------|
| False ブール値(False) | 332 |
| 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 | 1010 |
| 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 | 1008 |
| FILTER EVENT | 557 |
| FILTER KEYSTROKE(フィルタ文字) | 438 |
| Find in array (配列;値;{開始}) 数値 | 278 |
| Find index key(インデックスフィールド;値) 倍長整数 | 821 |
| Find window (左;上{;ウインドウ部分}) ウインドウ参照番号 | 1331 |
| FIRST PAGE | 478 |
| FIRST RECORD {(テーブル)} | 942 |
| FIRST SUBRECORD(サブテーブル) | 1024 |
| FLUSH BUFFERS (ログファイル*) | 229 |
| FOLDER LIST(パス名;ディレクトリ) | 1057 |

| | |
|----------------------------------|------|
| Font name (フォント番号) 文字列 | 1085 |
| Font number (フォント名) 数値 | 1086 |
| FONT SIZE ({*};オブジェクト;フォントサイズ) | 676 |
| FONT STYLE({*};オブジェクト;フォントスタイル) | 677 |
| FONT({*};オブジェクト;フォント) | 675 |
| FONTLIST (フォント) | 1085 |
| Form event 数値 | 450 |
| Frontmost process ({*}) 整数 | 770 |
| Frontmost window ({*}) ウィンドウ参照番号 | 1330 |

G

| | |
|---|------|
| Gestalt(セクタ;値) 文字列 | 1089 |
| GET CLIPBOARD (データタイプ;データ) | 341 |
| Get database parameter({テーブル;}セクタ) 倍長整数 | 1015 |
| Get document position(ドキュメント参照番号) 数値 | 1073 |
| 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) | 1065 |
| Get document size(ドキュメント{*}) 数値 | 1072 |
| Get edited text Text | 475 |
| GET FIELD PROPERTIES(テーブル番号;フィールド番号;タイプ{長さ
{;インデックス}}) | 1011 |
| 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}}}}) | 1337 |
| GET GROUP LIST(グループ名;グループID) | 1171 |
| GET GROUP PROPERTIES(グループID ;グループ名;オーナー{メンバーID}) | 1172 |
| GET HIGHLIGHT(エリア;先頭;最終) | 1152 |
| GET ICON RESOURCE(リソースID ;リソースデータ{リソースファイル}) | 919 |
| 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 | 912 |
| GET LIST ITEM PROPERTIES(リスト;項目参照番号;入力可{スタイル
{アイコン}}) | 524 |
| GET LIST ITEM(リスト;項目位置;項目参照番号; 項目テキスト{サブリスト
{展開}}) | 529 |
| GET LIST PROPERTIES(リスト;表示形式{アイコン{線の長さ}}) | 510 |
| Get menu item key(メニュー;メニューアイテム{プロセス}) 数値 | 644 |
| Get menu item mark(メニュー;メニューアイテム{プロセス}) 文字列 | 642 |
| Get menu item style(メニュー;メニューアイテム{プロセス}) 数値 | 640 |

| | |
|---|------|
| Get menu item(メニュー;メニューアイテム{;プロセス}) 文字列 | 638 |
| Get menu title(メニュー{;プロセス}) 文字列 | 638 |
| GET MOUSE(水平;垂直:マウスボタン{;}) | 1145 |
| GET OBJECT RECT({*;}オブジェクト;左;上;右;下) | 696 |
| GET PICTURE FROM CLIPBOARD (ピクチャ) | 343 |
| GET PICTURE FROM LIBRARY (ピクチャ参照番号;ピクチャ名) | 723 |
| GET PICTURE RESOURCE(リソースID ;リソースデータ{;リソースファイル}) | 917 |
| Get platform interface 数値 | 1133 |
| Get pointer (変数名) ポインタ | 576 |
| 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)) | 759 |
| GET REGISTERED CLIENTS(クライアントリスト;メソッド) | 797 |
| 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 | 925 |
| 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 | 928 |
| 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}) | 921 |
| Get string resource(リソースID {;リソースファイル}) 文字列 | 913 |
| Get text from clipboard 文字列 | 344 |
| Get text resource(リソースID {;リソースファイル}) テキスト | 915 |
| GET USER LIST(ユーザ名;ユーザID) | 1166 |
| 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;最終日付{;グループID}) | 1167 |
| GET WINDOW RECT (左;上;右;下{;ウインドウ}) | 1328 |
| Get window title {{ウインドウ}} 文字列 | 1324 |
| GOTO AREA({*;}オブジェクト) | 445 |
| GOTO PAGE (ページ番号) | 480 |
| GOTO RECORD({テーブル;レコード番号}) | 856 |
| GOTO SELECTED RECORD({テーブル;レコード位置番号}) | 940 |
| GOTO XY(x ;y) | 663 |
| GRAPH (グラフエリア;グラフ番号;xラベル;y要素1 {;...;y要素8}) | 484 |
| 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}) | 488 |
| GRAPH TABLE {{テーブル}} | 489 |

H

| | |
|----------------------------|------|
| HIDE MENU BAR 数値 | 633 |
| HIDE PROCESS (プロセス) | 768 |
| HIDE TOOL BAR | 1101 |
| HIDE WINDOW {{ウインドウ}} | 1318 |
| HIGHLIGHT RECORDS{{セット名}} | 956 |
| HIGHLIGHT TEXT (エリア;先頭;最終) | 1154 |

I

| | |
|--|------|
| IDLE | 386 |
| IMPORT DATA(ファイル名{;プロジェクト{*}}) | 549 |
| IMPORT DIF({テーブル;}ドキュメント) | 545 |
| IMPORT SYLK({テーブル;}ドキュメント) | 541 |
| IMPORT TEXT({テーブル;}ドキュメント) | 537 |
| In break | 471 |
| In footer | 471 |
| In header | 470 |
| In transaction ブール | 1109 |
| INPUT FORM ({テーブル;}フォーム{*}) | 1097 |
| INSERT ELEMENT (配列;挿入位置;{要素数}) | 276 |
| INSERT IN BLOB (blob ;オフセット;挿入数;フィルタ) | 327 |
| 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}) | 521 |
| INSERT MENU ITEM(メニュー;メニューアイテム;テキスト{*};プロセス) | 649 |
| Insert string (元の文字列;挿入文字列;挿入位置) 文字列 | 992 |
| Int (数値) 数値 | 610 |
| INTEGER TO BLOB (整数;blob ;バイト順序{*};オフセット!*) | 312 |
| INTERSECTION (セット1 ;セット2 ;結果セット) | 973 |
| INVERT BACKGROUND ({*};テキスト変数!テキストフィールド) | 1157 |
| Is a list(リスト) ブール | 500 |
| Is a variable (ポインタ) ブール | 575 |
| Is in set(セット) ブール | 968 |
| Is new record({テーブル}) ブール | 849 |
| Is record loaded({table}) ブール | 851 |
| Is user deleted(ユーザID) ブール | 1165 |
| ISO to Mac(テキスト) 文字列 | 1002 |

K

Keystroke 文字列 433

L

Last object ポインタ 1155
 LAST PAGE 478
 LAST RECORD {{テーブル}} 943
 LAST SUBRECORD(サブテーブル) 1024
 Length (文字列) 数値 986
 Level 数値 744
 List item parent(リスト;項目参照番号) 数値 526
 List item position(リスト;項目参照番号) 数値 525
 LIST TO ARRAY (リスト;配列;{項目参照}) 280
 LIST TO BLOB (変数;blob {;}) 309
 LOAD COMPRESS PICTURE FROM FILE (ドキュメント参照番号;方法;
 品質;ピクチャ) 714
 Load list (リスト名) 数値 494
 LOAD RECORD{{テーブル}} 841
 LOAD SET({テーブル;}セット;ドキュメント) 971
 LOAD VARIABLES (ドキュメント;変数1{;...;変数N}) 1180
 Locked {{テーブル}} ブール 843
 LOCKED ATTRIBUTES ({テーブル;}プロセス;ユーザ;マシン;プロセス名) . . . 844
 Log (数値) 数値 617
 LOG EVENT(メッセージ{;重要度}) 1090
 LONGINT ARRAY FROM SELECTION({テーブル;}レコード番号配列
 {;セクション名}) 329
 LONGINT TO BLOB (倍長整数;blob ;バイト順序{;オフセット!}) 314
 Lowercase (文字列) 文字列 990

M

Mac to ISO(テキスト) 文字列 999
 Mac to Win(テキスト) 文字列 997
 Macintosh command down ブール 1143
 Macintosh control down ブール 1144
 Macintosh option down ブール 1143
 MAP FILE TYPES(MacOS ;Windows ;コンテキスト) 1061

| | |
|--|------|
| Max (一連の値) 数値 | 704 |
| MAXIMIZE WINDOW {(ウインドウ)} | 1332 |
| Menu bar height 数値 | 1084 |
| Menu bar screen 数値 | 1084 |
| MENU BAR(メニューバー番号{;プロセス}{;}) | 631 |
| Menu selected 数値 | 635 |
| MESSAGE(メッセージ) | 659 |
| MESSAGES OFF | 652 |
| MESSAGES ON | 652 |
| Milliseconds 数値 | 413 |
| Min(一連の値) 数値 | 703 |
| MINIMIZE WINDOW {(ウインドウ参照番号)} | 1333 |
| Mod (数値1 ;数値2) 数値 | 615 |
| Modified (フィールド) ブール | 396 |
| Modified record {(テーブル)} ブール | 850 |
| MODIFY RECORD ({テーブル}{;}{*}) | 391 |
| MODIFY SELECTION ({テーブル}{;}{*}{*}) | 951 |
| MODIFY SUBRECORD(サブテーブル;フォーム{*}) | 393 |
| Month of (日付) 数値 | 403 |
| MOVE DOCUMENT(移動元パス名;移動先パス名) | 1050 |
| 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}{;}) | 697 |

N

| | |
|---|------|
| New list 数値 | 496 |
| 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 | 771 |
| NEXT PAGE | 479 |
| NEXT RECORD ({テーブル}) | 942 |
| NEXT SUBRECORD(サブテーブル) | 1025 |
| Next window {(ウインドウ)} 数値 | 1330 |
| Nil(ポインタ) ブール | 575 |
| NO TRACE | 566 |
| Not (ブール値) ブール値 | 332 |
| Num (表現式) 数値 | 982 |

O

| | |
|---|------|
| OLD RELATED MANY(フィールド) | 878 |
| OLD RELATED ONE(フィールド) | 877 |
| Old(フィールド) 文字列、数値、日付、ブール、時間 | 398 |
| ON ERR CALL (エラーメソッド) | 558 |
| ON EVENT CALL (イベントメソッド{;プロセス}) | 553 |
| ONE RECORD SELECT{(テーブル)} | 955 |
| Open document (ドキュメント{;ファイルタイプ}) ドキュメントファイル参 | 1038 |
| Open external window(左;上;右;下;タイプ;タイトル;プラグインエリア) 整数 | 1317 |
| 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 | 1335 |
| Open resource file(リソースファイル名{;ファイルタイプ}) DocRef | 897 |
| OPEN WEB URL(url {; *}) | 1302 |
| 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 - ズ)ボックス))) {ウインドウ参照番号} | 1313 |
| 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)) | 827 |
| 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 | 822 |
| 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}}) | 1028 |
| OUTPUT FORM ({テーブル;}フォーム) | 1098 |
| Outside call | 473 |

P

| | |
|--|------|
| PAGE BREAK {(* ! >)} | 752 |
| PAGE SETUP ({テーブル;}フォーム) | 747 |
| PAUSE PROCESS (プロセス) | 780 |
| PICT TO GIF (ピクチャ; GIF用BLOB) | 717 |
| PICTURE LIBRARY LIST(ピクチャ参照番号;ピクチャ名) | 721 |
| 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}})) | 720 |
| Picture size (ピクチャ) 整数 | 719 |
| PLATFORM PROPERTIES(プラットフォーム;システム;マシン) | 232 |
| PLAY(サウンド名{;チャンネル}) | 1131 |
| POP RECORD ({テーブル}) | 863 |
| Pop up menu(項目テキスト{;デフォルト}) 数値 | 1146 |

| | |
|--|------|
| Position (検索文字列;文字列) 数値 | 984 |
| POST CLICK(マウスX ;マウスY {;プロセス}{;*)) | 1150 |
| 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}) | 1151 |
| POST KEY(コード{;モディファイア{;プロセス}}) | 1149 |
| PREVIOUS PAGE | 479 |
| PREVIOUS RECORD {(テーブル)} | 943 |
| PREVIOUS SUBRECORD(サブテーブル) | 1026 |
| PRINT FORM ({テーブル};フォーム) | 750 |
| PRINT LABEL ({テーブル};ラベルドキュメント){;*)) | 735 |
| PRINT RECORD({テーブル};*)) | 746 |
| PRINT SELECTION({テーブル};{;*)) | 738 |
| PRINT SETTINGS | 748 |
| Printing page 数値 | 740 |
| Process aborted ブール | 783 |
| Process number(プロセス名{;*)) 数値 | 788 |
| 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}})) | 785 |
| Process state (プロセス) 数値 | 784 |
| PUSH RECORD{(テーブル)} | 862 |

Q

| | |
|---|------|
| QUERY {(テーブル}{;検索条件){;*)) | 803 |
| QUERY BY EXAMPLE({テーブル}{;*)) | 802 |
| QUERY BY FORMULA({テーブル}{;{;検索用フォーミュラ}) | 812 |
| QUERY SELECTION BY FORMULA ({テーブル}{;{;検索用フォーミュラ}) | 814 |
| QUERY SELECTION({テーブル}{;検索条件){;*)) | 810 |
| QUERY SUBRECORDS(サブテーブル;検索条件式) | 1029 |
| QUERY WITH ARRAY QUERY WITH ARRAY(インデックスフィールド;配列) | 829 |
| QUIT 4D | 228 |

R

| | |
|------------------------------|-----|
| Random 数値 | 614 |
| READ ONLY {(テーブル!*)} | 839 |
| Read only state {(テーブル)} ブール | 840 |
| READ WRITE {(テーブル!*)} | 838 |

| | |
|---|------|
| REAL TO BLOB (実数;blob ;実数形式{;オフセット!*)) | 316 |
| RECEIVE BUFFER (受信変数 | 361 |
| 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) | 357 |
| RECEIVE RECORD {(テーブル)} | 364 |
| RECEIVE VARIABLE (変数) | 369 |
| Record number {(テーブル)} 数値 | 855 |
| Records in selection{(テーブル)} 数値 | 936 |
| Records in set(セット) 数値 | 969 |
| Records in subselection(サブテーブル) 数値 | 1022 |
| Records in table {(テーブル)} 数値 | 854 |
| REDRAW (オブジェクト) | 1156 |
| REDRAW LIST(リスト) | 501 |
| REDRAW WINDOW {(ウインドウ参照番号)} | 1322 |
| REDUCE SELECTION ({テーブル;}レコード数) | 953 |
| REGISTER CLIENT(クライアント名{;ピリオド}{;*)) | 793 |
| REJECT{(フィールド)} | 446 |
| RELATE MANY SELECTION(フィールド) | 880 |
| RELATE MANY(1テーブル!1フィールド) | 873 |
| RELATE ONE SELECTION(nテーブル;1テーブル) | 879 |
| RELATE ONE(nテーブル;nフィールド{;選択フィールド}) | 870 |
| REMOVE FROM SET({テーブル;}セット) | 967 |
| REMOVE PICTURE FROM LIBRARY(ピクチャ参照番号) | 727 |
| Replace string (元の文字列;対象文字列;置き換え文字列{;回数}) | 994 |
| REPORT ({テーブル;}ドキュメント{;*)) | 733 |
| 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}}}) | 657 |
| RESOLVE POINTER(ポインタ;変数名;テーブル番号;フィールド番号) | 573 |
| 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}) | 907 |
| RESOURCE TYPE LIST(リソースタイプ{;リソースファイル}) | 905 |
| RESUME PROCESS (プロセス) | 782 |
| Round (数値;桁位置) 数値 | 612 |

S

| | |
|--|------|
| SAVE LIST (リスト;リスト名) | 495 |
| SAVE OLD RELATED ONE(フィールド) | 878 |
| SAVE PICTURE TO FILE (ドキュメント参照番号;ピクチャ) | 716 |
| SAVE RECORD{(テーブル)} | 852 |
| SAVE RELATED ONE(フィールド) | 876 |
| SAVE SET(セット;ドキュメント) | 969 |
| SAVE VARIABLES (ドキュメント;変数1{,...;変数N}) | 1179 |
| SCAN INDEX(フィールド;レコード数{;方向}) | 954 |
| SCREEN COORDINATES (左;上 ;右;下{;画面}) | 1081 |
| SCREEN DEPTH (深さ;カラー{;画面}) | 1082 |
| Screen height{(*)} 数値 | 1080 |
| Screen width{(*)} 数値 | 1080 |
| Select folder{(メッセージ)} 文字列 | 1075 |
| SELECT LIST ITEM BY REFERENCE(リスト;項目参照番号) | 536 |
| SELECT LIST ITEM(リスト;項目位置) | 534 |
| SELECT LOG FILE(ログファイル{*)} | 230 |
| Selected list item(リスト) 倍長整数 | 532 |
| Selected record number {(テーブル)} 数値 | 939 |
| 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}) | 286 |
| SELECTION TO ARRAY (フィールド1 ;配列1{,...;フィールドN ;配列N}) | 282 |
| Self ポインタ | 572 |
| Semaphore (セマフォ) ブール | 754 |
| SEND HTML BLOB(blob;タイプ{;非コンテキスト}) | 1292 |
| SEND HTML FILE(HTMLファイル) | 1289 |
| SEND HTTP REDIRECT(url {;*) | 1300 |
| SEND PACKET({ドキュメントファイル参照番号;}パケット) | 355 |
| SEND RECORD {(テーブル)} | 363 |
| SEND VARIABLE (変数) | 368 |
| Sequence number{(テーブル)} 数値 | 857 |
| SET ABOUT(アイテム;メソッド) | 634 |
| SET BLOB SIZE (blob ;サイズ{;フィラー}) | 294 |
| SET CHANNEL(ポート;セットアップ) | 350 |
| SET CHOICE LIST ({;}オブジェクト;リスト) | 686 |
| SET COLOR({;}オブジェクト;カラー) | 690 |
| SET CURSOR{(カーソル)} | 1155 |

| | |
|---|------|
| SET DATABASE PARAMETER({テーブル;}セクタ;値) | 1016 |
| SET DEFAULT CENTURY (世紀{;}ピポット年度) | 409 |
| SET DOCUMENT CREATOR(ドキュメント;ファイルクリエータ) | 1064 |
| SET DOCUMENT POSITION(ドキュメント参照番号;オフセット{;}アンカー)) | 1074 |
| 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) | 1071 |
| SET DOCUMENT SIZE(ドキュメント;サイズ) | 1072 |
| SET DOCUMENT TYPE(ドキュメント;ファイルタイプ) | 1060 |
| SET ENTERABLE ({*;}入力エリア;入力可) | 687 |
| 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)
. | 1139 |
| SET FILTER ({*;}オブジェクト;入力フィルタ) | 684 |
| SET FORMAT ({*;}オブジェクト;表示フォーマット) | 682 |
| Set group properties (グループID ;グループ名;オーナー{;}メンバーID)) | 1174 |
| SET HOME PAGE(ホームページ) | 1288 |
| SET HTML ROOT(パス名HTML) | 1295 |
| SET HTTP HEADER(フィールド) | 1299 |
| SET INDEX (フィールド;インデックス{;}モード{;}*) | 1013 |
| SET LIST ITEM PROPERTIES(リスト;項目参照番号;入力可;スタイル;アイコン)
. | 522 |
| SET LIST ITEM(リスト;項目参照番号; 新規項目テキスト;新規項目参照番号
{;}サブリスト{;}展開)) | 531 |
| SET LIST PROPERTIES(リスト;表示形式{;}アイコン{;}線の高さ)) | 502 |
| SET MENU ITEM (メニュー番号;メニューアイテム番号;アイテムテキスト
{;}プロセス)) | 639 |
| SET MENU ITEM KEY(メニュー;メニューアイテム;キー{;}プロセス)) | 645 |
| SET MENU ITEM MARK(メニュー;メニューアイテム;マーク{;}プロセス)) | 643 |
| SET MENU ITEM STYLE(メニュー;メニューアイテム;スタイル{;}プロセス)) | 641 |
| SET PICTURE RESOURCE(リソースID ;リソースデータ{;}リソースファイル)) | 918 |
| SET PICTURE TO CLIPBOARD (ピクチャ) | 345 |
| SET PICTURE TO LIBRARY (ピクチャ;ピクチャ参照番号;ピクチャ名) | 724 |
| SET PLATFORM INTERFACE(インタフェース) | 1133 |
| SET PRINT PREVIEW(プレビュー) | 749 |
| 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)) | 762 |
| SET QUERY DESTINATION(配置先タイプ{;}配置先オブジェクト)) | 815 |
| SET QUERY LIMIT(制限) | 820 |

| | |
|--|------|
| SET REAL COMPARISON LEVEL (イプシロン値) | 622 |
| 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}) | 927 |
| 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}) | 929 |
| 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})
..... | 922 |
| SET RGB COLORS ({*};オブジェクト;前景色;背景色) | 692 |
| SET SCREEN DEPTH (深さ;カラー{;画面}) | 1083 |
| SET STRING RESOURCE(リソースID;リソースデータ{;リソースファイル}) | 914 |
| SET TABLE TITLES(テーブルタイトル;テーブル番号) | 1135 |
| SET TEXT RESOURCE(リソースID;リソースデータ{;リソースファイル}) | 916 |
| SET TEXT TO CLIPBOARD (テキスト) | 346 |
| SET TIMEOUT (秒) | 360 |
| SET TIMER | 474 |
| 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;
最終日付{;グループID}) 数値 | 1168 |
| SET VISIBLE ({*};オブジェクト;表示) | 688 |
| SET WEB DISPLAY LIMIT(レコード数{;ページ数{;ピクチャ参照番号}}) | 1285 |
| SET WEB TIMEOUT(タイムアウト) | 1284 |
| SET WINDOW RECT (左;上;右;下{;ウインドウ}) | 1329 |
| SET WINDOW TITLE (タイトル{;ウインドウ}) | 1325 |
| Shift down プール | 1140 |
| SHOW MENU BAR 数値 | 633 |
| SHOW PROCESS (プロセス) | 769 |
| SHOW TOOL BAR | 1101 |
| SHOW WINDOW {(ウインドウ)} | 1318 |
| Sin (数値) 数値 | 619 |
| Size of array (配列) 数値 | 279 |
| SORT ARRAY (配列1{;...;配列N};{ソート種別}) | 273 |
| SORT LIST(リスト{;>または<}) | 512 |
| Square root (数値) 数値 | 616 |
| START TRANSACTION | 1108 |
| START WEB SERVER | 1283 |
| Std deviation (一連の値) 数値 | 705 |
| STOP WEB SERVER | 1284 |
| String (数値{;フォーマット}) 文字列 | 980 |

| | |
|--|------|
| STRING LIST TO ARRAY(リソースID ;ストリング{;リソースファイル}) | 909 |
| Structure file 文字列 | 238 |
| Substring (文字列;先頭文字位置{;文字数}) 文字列 | 985 |
| Subtotal (データ{;ページブレイク}) 数値 | 743 |
| Sum (一連の値) 数値 | 701 |
| Sum squares (一連の値) 数値 | 707 |
| System folder 文字列 | 1087 |

T

| | |
|---|------|
| Table (テーブル番号;ポインタ) ポインタ;数値 | 1009 |
| Table name (テーブル番号;テーブルポインタ) 文字列 | 1007 |
| Tan (数値) 数値 | 620 |
| Temporary folder 文字列 | 1087 |
| Test clipboard (データタイプ) 数値 | 347 |
| Test path name(パス名) 数値 | 1052 |
| Test semaphore(セマフォ) ブール | 757 |
| TEXT TO BLOB (テキスト;blob ;テキスト形式{;オフセット;}) | 318 |
| Tickcount 数値 | 412 |
| Time (時間文字列) 時間 | 412 |
| Time string (秒) 文字列 | 411 |
| TRACE | 564 |
| Trigger level 数値 | 1126 |
| TRIGGER PROPERTIES(トリガレベル;イベント;テーブル番号;レコード番号) | 1127 |
| True ブール値(True) | 332 |
| Trunc (数値;桁位置) 数値 | 613 |
| Type (引数) 数値 | 569 |

U

| | |
|-----------------------------|------|
| Undefined (変数) ブール | 1182 |
| UNION (セット1 ;セット2 ;結果セット | 974 |
| UNLOAD RECORD{(テーブル)} | 842 |
| UNREGISTER CLIENT | 796 |
| Uppercase (文字列) 文字列 | 990 |
| USE ASCII MAP (テーブル名{;{/O}) | 370 |
| USE NAMED SELECTION (名前) | 670 |

| | |
|--|------|
| USE SET(セット) | 966 |
| User in group (ユーザ;グループ) ブール | 1163 |

V

| | |
|---|------|
| Validate password(ユーザID ;パスワード) ブール | 1162 |
| VALIDATE TRANSACTION | 1108 |
| VARIABLE TO BLOB (変数;blob {;オフセット *}) | 305 |
| 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}) | 765 |
| Variance (一連の値) 数値 | 706 |
| Version type 倍長整数 | 245 |
| VOLUME ATTRIBUTES(ボリューム;サイズ;使用サイズ;空きサイズ) | 1055 |
| VOLUME LIST(ボリューム) | 1054 |

W

| | |
|--|------|
| WEB CACHE STATISTICS(ページ;ヒット数;使用率) | 1298 |
| Web Context ブール | 1294 |
| Win to Mac(テキスト) 文字列 | 998 |
| Window kind {(ウインドウ参照番号)} 数値 | 1327 |
| WINDOW LIST (ウインドウ参照番号{;*}) | 1326 |
| Window process {(ウインドウ参照番号)} 数値 | 1328 |
| Windows Alt down ブール | 1142 |
| Windows Ctrl down ブール | 1142 |

Y

| | |
|---------------------------|-----|
| Year of (日付) 数値 | 405 |
|---------------------------|-----|

