

# 4D v14

---

*Upgrade  
Windows®/OS X®*



4D®

© 1985-2013 4D SAS. All Rights Reserved.

---

## ***4D v14 - Upgrade Windows<sup>®</sup> and OS X<sup>®</sup> Versions***

*Copyright© 1985 - 2013 4D SAS.  
All Rights Reserved.*

---

The software described in this manual is governed by the grant of license provided in this package. The software and the manual are copyrighted and may not be reproduced in whole or in part except for the personal licensee's use and solely in accordance with the contractual terms. This includes copying the electronic media, archiving, or using the software in any manner other than that provided for in the Software license Agreement.

4D, 4D Write, 4D View, 4D Server and the 4D logos are registered trademarks of 4D SAS.

Windows, Windows XP, Windows 7, Windows Vista and Microsoft are registered trademarks of Microsoft Corporation.

Apple, Macintosh, iMac, Mac OS X and QuickTime are trademarks or registered trademarks of Apple Computer Inc.

Mac2Win Software Copyright © 1990-2013 is a product of Altura Software, Inc.

ICU Copyright © 1995-2013 International Business Machines Corporation and others. All rights reserved.

ACROBAT © Copyright 1987-2013, Secret Commercial Adobe Systems Inc. All rights reserved. ACROBAT is a registered trademark of Adobe Systems Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). 4th Dimension includes cryptographic software written by Eric Young ([ey@cryptsoft.com](mailto:ey@cryptsoft.com))  
4th Dimension includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com)).

Spellchecker © Copyright SYNAPSE Développement, Toulouse, France, 1994-2013.

All other referenced trade names are trademarks, registered trademarks, or copyrights of their respective holders.

### **IMPORTANT LICENSE INFORMATION**

Use of this software is subject to its license agreement included with the software. Please read the License Agreement carefully before using the software.

# 目次

## Chapter 1 ようこそ 13

- 既存のデータベースの変換 14
  - バージョン 11 より前のデータベース 14
  - バージョン 11/12 のデータベース 15
  - バージョン 13 のデータベース 16
  - バージョン 11, 12, 13 のコンポーネント 16
- Windows 環境下における 64-bit プラグインに対するサポート 16
- IPv6 のサポート 17
- 機能的な互換性 18
  - 廃止予定の機能 18
  - 削除された機能 20
- システム要件 22

## Chapter 2 開発作業環境 23

- 新しいデータジャーナル 23
  - 「ログファイルに含める」オプション 24
  - デフォルトのプライマリーキーフィールド 26
  - プライマリーキーのエラーウィンドウ 27
  - プライマリーキー管理 29
  - MSC でのプライマリーキーを閲覧 32
- デバッグヘルプ 32
  - エラーを無視 33
- リアルタイムモニター 33
  - ログファイルを開始する 35
- 4D components 35
- アプリケーションビルダー 37
  - サーバーアプリケーションのアップデートの自動化 37
  - クライアントアプリ向けの新しい自動アップデート処理 38

- アップデートログ 38
- 新しい「自動アップデート」XML キー (Windows) 39
- OS X 環境下でのアプリケーションの認証 40
- 新しい "SignApplication" XML キー 43
- 新しいクエリエディター 44
  - エディターを呼び出す 44
  - エディターのメニュー 45
  - クエリの生成 46
  - フォームによるクエリ 50
- Mecab のサポート (日本語版) 51

## Chapter 3 フォームとオブジェクト 53

- プレースホルダーテキスト 53
- コンボボックス 55
  - 値の自動挿入 55
  - 除外リスト 56
  - OS X 環境下での、値のマウスオーバー 57
- マルチスタイルエリア 57
  - マウスオーバー 57
  - コンテキストメニュー 58
  - 新しいタグ 58
  - カラーピッカーとフォントピッカーの使用を許可 59
- 階層リスト 60
- リストボックス 61
  - 新しいワードラッププロパティ 62
  - 新しい列のプロパティ 63
  - スタイルとカラーにおける優先順位 65
  - スタイルとカラーの継承 67
  - 標準アクションのサポート 68
  - 新しいプロパティ 70
- オブジェクトに関連付けられた選択リスト 71
  - ポップアップメニューやコンボボックスとフィールド・変数の関連付け 72
  - 参照の保存 73
- テキストの回転 76
  - 回転可能なオブジェクト 76
  - 回転の設定 77
  - フィールドと変数の回転 79
  - 回転しているオブジェクトの変更 80
- ウィンドウの配置の記憶 81
  - 「配置を記憶」のオプション 81
  - 「値を記憶」のオプション 82

- 情報の保存と使用に関して 83
- 例 84
- スペルチェック 86
  - 辞書の選択 86
  - チェック処理 86
- Web エリア 91
  - 4D メソッドへのアクセス 92
  - Web インспекターへのアクセス 95

## Chapter 4SQL 99

- SQL views のサポート 99
  - CREATE VIEW 99
  - DROP VIEW 102
  - ビュー上での新しいシステムテーブル 102
- ALTER TABLE を使用した新しいオプション 102
  - ALTER TABLE 102
- システムテーブルにおける新しいカラム 103
- 日付・時間定数のサポート 104

## Chapter 54D Mobile 105

- 4D Mobile アーキテクチャー 105
- ステップバイステップ形式での解説 107
  - 1 - 4D データベースの作成と設定 107
  - 2 - Wakanda アプリケーションの作成 109
  - 3 - Wakanda ウィジェットを使用して 4D data を表示する 111
  - 4 - 4D メソッドの作成と呼び出し 114
- 4D データベースの設定 118
  - Wakanda REST サービスの有効化 118
  - REST 接続の管理 119
  - On REST Authentication 121
  - REST に公開されている 4D オブジェクトの設定 123
  - エクスプローラー 128
- Wakanda アプリケーション側の設定 128
  - mergeOutsideCatalog() メソッドの実行 128
  - 4D テーブルの呼び出し 131
  - 4D メソッドの呼び出し 133
  - openRemoteStore() と addRemoteStore() 136
- 4D Mobile のセキュリティについて 137

## Chapter 6 ランゲージ 139

### 4D 環境 139

新コマンド 139

Get last update log path 139

RESTART 4D 139

SET UPDATE FOLDER 140

修正されたコマンド 142

GET MEMORY STATISTICS 142

SET DATABASE PARAMETER, Get database parameter 142

Version type 145

### 配列 145

ARRAY BLOB 145

ARRAY OBJECT 146

ARRAY TIME 148

ARRAY TO LIST 149

LIST TO ARRAY 150

SELECTION TO ARRAY, SELECTION RANGE TO ARRAY

151

### バックアップ 152

INTEGRATE MIRROR LOG FILE 152

### BLOB 154

BLOB TO VARIABLE, VARIABLE TO BLOB 154

### コンパイラー 154

C\_OBJECT 154

### データベースメソッド 155

On Host Database Event 155

On REST Authentication 157

### 日付と時間 157

JavaScript の日付の変換 157

Time 158

null 日付の表示 158

### デザインオブジェクトアクセス 158

FORM GET NAMES 158

METHOD GET CODE 159

METHOD SET ATTRIBUTE 159

METHOD SET CODE 160

### ドラッグ&ドロップ 161

SET DRAG ICON 161

### フォームイベント 162

フォームイベント 162

### フォーム 163

Current form name 163

- FORM LOAD 164
- FORM UNLOAD 165
- グラフ 166
  - GRAPH 166
  - GRAPH SETTINGS 166
  - GRAPH TABLE 167
- 階層リスト 167
  - フォームオブジェクトとの関連付け 167
- HTTP クライアント 167
  - HTTP Get 167
  - HTTP Get certificates folder 168
  - HTTP Request 168
  - HTTP SET CERTIFICATES FOLDER 169
- JSON 170
  - JSON についての概要 171
  - JSON Parse 173
  - JSON PARSE ARRAY 174
  - JSON Stringify 176
  - JSON Stringify array 178
  - JSON TO SELECTION 180
  - Selection to JSON 181
- リストボックス 183
  - LISTBOX DUPLICATE COLUMN 183
  - LISTBOX Get array 186
  - LISTBOX GET OBJECTS 187
  - LISTBOX Get row color 188
  - LISTBOX Get row font style 190
  - LISTBOX MOVE COLUMN 191
  - LISTBOX SET ARRAY 192
  - LISTBOX SET ROW COLOR 193
  - LISTBOX SET ROW FONT STYLE 195
- メッセージ 197
  - DISPLAY NOTIFICATION 197
- オブジェクトプロパティ 197
- オブジェクト ( フォーム ) 197
  - New commands 197
  - GET STYLE SHEET INFO 197
  - LIST OF STYLE SHEETS 199
  - OBJECT Get action 199
  - OBJECT Get border style 201
  - OBJECT Get context menu 202
  - OBJECT Get data source 203
  - OBJECT GET EVENTS 203

OBJECT Get indicator type 204  
OBJECT Get list reference 205  
OBJECT GET MAXIMUM VALUE 206  
OBJECT GET MINIMUM VALUE 207  
OBJECT Get multiline 207  
OBJECT Get placeholder 208  
OBJECT GET PRINT VARIABLE FRAME 209  
OBJECT Get style sheet 210  
OBJECT Get text orientation 210  
OBJECT Get three states checkbox 212  
OBJECT Get type 212  
OBJECT Is styled text 214  
OBJECT SET ACTION 215  
OBJECT SET BORDER STYLE 216  
OBJECT SET CONTEXT MENU 217  
OBJECT SET COORDINATES 218  
OBJECT SET DATA SOURCE 219  
OBJECT SET EVENTS 221  
OBJECT SET INDICATOR TYPE 224  
OBJECT SET LIST BY REFERENCE 225  
OBJECT SET PLACEHOLDER 228  
OBJECT SET PRINT VARIABLE FRAME 229  
OBJECT SET MAXIMUM VALUE 230  
OBJECT SET MINIMUM VALUE 231  
OBJECT SET MULTILINE 232  
OBJECT SET STYLE SHEET 233  
OBJECT SET TEXT ORIENTATION 234  
OBJECT SET THREE STATES CHECKBOX 235  
変更されたコマンド 236  
OBJECT Get choice list name 236  
OBJECT Get list name 236  
OBJECT GET RGB COLORS 237  
OBJECT SET CHOICE LIST NAME 237  
OBJECT SET FONT 237  
OBJECT SET LIST BY NAME 237  
OBJECT SET RGB COLORS 239  
汎用コマンドとマルチスタイルエリアの関係性 239  
オブジェクト (ランゲージ) 240  
4D オブジェクトの構造 240  
OB Copy 241  
OB Is defined 243  
OB Is empty 244  
OB Get 245  
OB GET ARRAY 247

- OB GET PROPERTY NAMES 248
- OB Get type 250
- OB REMOVE 251
- OB SET 251
- OB SET ARRAY 254
- OB SET NULL 256
- ピクチャ演算子 256
  - "&" 演算子 (排他的論理和) 256
  - "|" 演算子 (包括的論理和) 257
- 印刷 258
  - OPEN PRINTING FORM 258
- プロセス 259
  - Execute on server 259
- クイックレポート 259
  - QR REPORT 259
  - QR SET DESTINATION 260
- スペルチェッカー 261
  - SPELL CHECK TEXT 261
  - SPELL GET DICTIONARY LIST 261
  - SPELL SET CURRENT DICTIONARY 261
- 文字列 262
  - String 262
- ストラクチャーアクセス 262
  - PAUSE INDEXES 262
  - RESUME INDEXES 263
- スタイル付テキスト 264
  - 新コマンド 264
  - ST COMPUTE EXPRESSIONS 264
  - ST FREEZE EXPRESSIONS 266
  - ST Get content type 268
  - ST Get expression 271
  - ST GET OPTIONS 273
  - ST GET URL 274
  - ST INSERT EXPRESSION 276
  - ST INSERT URL 278
  - ST SET OPTIONS 280
  - 修正されたコマンド 281
  - ST Get plain text 281
  - 行末の自動標準化 283
  - 定数を使用して選択範囲を指定 284
  - 改名されたコマンド 285
- システムドキュメント 285
  - COPY DOCUMENT 285

- Document to text 286
- TEXT TO DOCUMENT 288
- システム環境 290
  - 変更されたコマンド 290
  - FONT LIST 290
  - Font name 291
  - Font number 291
  - 新コマンド 292
  - OPEN COLOR PICKER 292
  - OPEN FONT PICKER 292
  - SET RECENT FONTS 293
- ツール 294
  - Generate digest 294
  - GET ACTIVITY SNAPSHOT 294
- ユーザーインターフェース 297
  - SHOW TOOL BAR, HIDE TOOL BAR 297
  - Tool bar height 297
- ユーザー & グループ 298
  - Validate password 298
- Web Area 298
  - WA Evaluate JavaScript 298
  - WA EXECUTE JAVASCRIPT FUNCTION 300
  - WA SET PREFERENCE 301
- Web サーバー 301
  - WEB SEND TEXT 301
- Web サービス (クライアント) 302
  - WEB SERVICE CALL 302
  - WEB SERVICE Get info 302
- ウィンドウ 303
  - Window types (OS X) 303
- 改名された定数 304
- 4D Internet Commands 304
  - Unicode の使用 304
  - SMTP\_Attachment 304
  - SMTP\_Body 305
  - SMTP\_MessageID 306
  - SMTP\_QuickSend 307
  - SMTP\_Subject 308
- 4D Pack 308
  - 4D Pack から削除されたコマンド 309
- TimePicker Widget 309
  - Clock 310
  - デジタル時計 311

TimePicker DISPLAY SECOND HAND 312  
TimePicker LCD DISPLAY AMPM 313  
TimePicker LCD DISPLAY SECONDS 313  
TimePicker LCD SET COLOR 313  
TimePicker LCD SET MODE 314



# 1

## ようこそ

4D v14 へようこそ。この新しいバージョンでは、システムアーキテクチャーが大きく進化したことに加え、あなたのアプリケーションのインターフェースをより充実させるための数々の新機能を備えながらも、既存のデータベースとの互換性も最大限確保しています。

- 4D コアの現代化 : 以前の *carbon* ライブラリーを内部的に書き直すことにより、4D のパフォーマンスはそれらのライブラリーによって課せられていた制約から解放されました。その一方で、QuickTime、4D Chart といった旧世代のテクノロジーはサポートされなくなっています。
- 4D v14 のフォームには **list object** ( リストボックス、階層リスト、コンボボックス等 )、**Web areas** (JavaScript との親和性の向上、デバッグ等)、そして **text areas** ( 静的テキストの回転、スタイル付テキストの機能拡張 ) といった分野で新機能がたくさん盛り込まれています。
- 開発環境も、アップデートの手順の簡略化、新しい query ダイアログボックス、デバッグヘルプの拡張、そして新しい **On Host Database Event** データベースメソッドなどにより強化されています。
- Wakanda アプリケーションが直接 4D データベースのデータを使用できるように、特定の機能が実装されています。
- 4D のプログラミング言語は様々な新コマンドによってさらに充実したものとなっています。特にオブジェクトプロパティをより自由に扱えるようになったコマンドの進化は顕著で、オブジェクト (**C\_OBJECT** コマンドを参照して下さい) や "**JSON**" といった新しい型のデータも扱えるようになりました。( 新テーマも設定されています。 )

## 既存のデータベースの変換

4D または 4D Server のバージョン 11.x, 12.x それと 13.x で作成されたデータベースであれば、4D v14 と互換性があります (ストラクチャファイルとデータファイル)。

- バージョン 11 より前のデータベースファイルは、ウィザードを使用して変換する必要があり、一度変換した後は元のバージョンで開くことはできません。
- バージョン 11 または 12 のデータベースファイルは v14 では直接変換され、一度変換した後は元のバージョンで開くことはできません。
- バージョン 13 のストラクチャファイルは v14 では直接変換され、一度変換した後は元のバージョンで開くことはできません。ただし、バージョン 13 のデータファイルは変換せずに v14 で開くことができ、その後も v13 で開くことができます。

---

*Note:* インタープリターストラクチャーを変換することができます。ファイルにコンパイルコードが含まれていてもかまいませんが、変換後に再度コンパイルする必要があります。

---

---

*Note:* 変換しようとしているデータベースにプライマリーキーがない場合、プライマリーキーエラーウィンドウが表示されます。これについては 23P “[新しいデータジャーナル](#)” にて説明がされています。

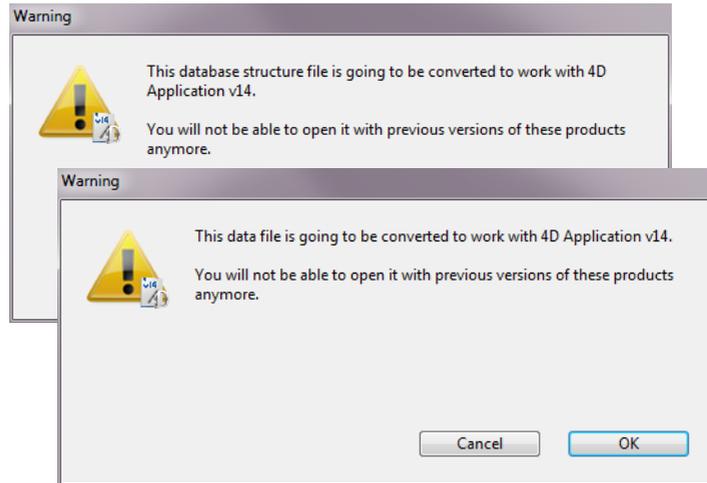
---

### バージョン 11 より前のデータベース

4D のデータベースのエンジンレベルの構造改造により、バージョン 11 より前のデータベースに関してはストラクチャファイル・データファイルともに徹底した変換が必要となりました。これには専用のウィザードが用意されています。詳細は *4D v11 SQL Upgrade マニュアル* を参照してください。すべての変更点に関する詳細 *4D v11 SQL 変換の手引き* をダウンロードして参照してください。:[http://ftp.4d-japan.com/REFERENCE/v11/Conversion\\_Guide\\_v11\\_9\\_JA.pdf](http://ftp.4d-japan.com/REFERENCE/v11/Conversion_Guide_v11_9_JA.pdf)

## バージョン 11/12 のデータベース

バージョン 11 や 12 のデータベースは、ストラクチャーファイルを 4D v13 で開く際に直接変換されます。2 つのダイアログが連続して表示され、変換されるファイルが以前のバージョンで開けなくなることを警告します。:

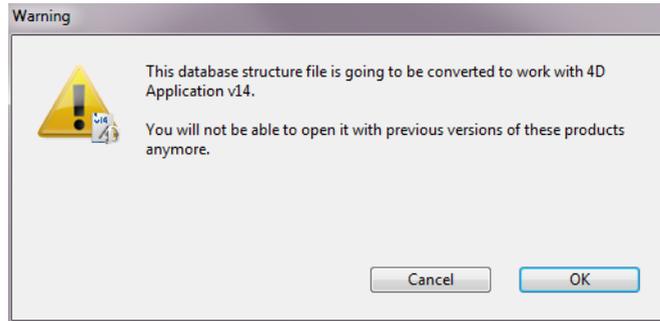


データファイルが変換される際には、インデックスも再構築されます。

Note: 古いバージョンの 4D Pack を使用している v11 データベースを変換する場合は、変換を実行する前に最新バージョンの 4D Pack v11 のプラグインをオリジナルのデータベースにインストールすることをお勧めします。

## バージョン 13 のデータベース

バージョン 13 のデータベースは、4D v14 で開く際に直接変換されます。画面にはダイアログが表示され、ストラクチャファイルが変換され、変換後は以前のバージョンで開けなくなることを警告します。:



ただしデータファイルに関しては変換なしで開くことができ、v14 で開いた後でもまた v13 で開くことが可能です。

## バージョン 11, 12, 13 のコンポーネント

4D v14 は v13、v12、v11 のコンポーネントを、変換したり警告ダイアログを表示させることなく開くことができます (コンパイルおよびインタープリター)。コンポーネントは常に読み込みのみモードで開かれる点に留意してください。

コンポーネントは再コンパイルの必要はありませんが、v14 への変換は .4DB ファイルのみ可能です。 .4DC ファイルはアップグレードできません。

## Windows 環境下における 64-bit プラグインに対するサポート

v14 では、64-ビット Windows 版のプラグインも用意されています。これを使用することによって、In v14, 4DServer の 64-ビット Windows 版の環境設定の中で 64-ビットアーキテクチャの恩恵にあずかれることができます。

64-ビット版はプラグインバンドルの中に含まれています。特別なインストール操作は必要ありません。

64-ビット Windows 版では以下のプラグインがご利用いただけます。:

v14の64-ビットプラグイン	詳細
4D Write	-

4D View	-
4D Internet Commands	v14の4D Internet Commands プラグインにも新しい機能が追加されています。詳細な情報に関しては、 <a href="#">304P “4D Internet Commands”</a> を参照して下さい。
4D Pack	v14 では 4D Pack プラグインに一部変更があります。詳細な情報に関しては <a href="#">308P “4D Pack”</a> を参照して下さい。
4D ODBC Pro	-
4D For OCI	64-bit 4D For OCI プラグインをインストールする前に、64-bit OCI をインストールする必要があります。このバージョンの OCI は Oracle, Inc. からダウンロードすることができます。

*Note:* 32-ビット版のプラグインと 64-ビット版のプラグインの機能は基本的には全く同じです。ただし、Open external window コマンドは 64-ビット版 4D Server では使用できないことに注意して下さい。

## IPv6 のサポート

4D v14 では IPv6 アドレス記法をサポートするようになりました。この新機能は 4D v14 の統合サーバー（下記）に関係します。

- Web サーバーと SOAP サーバー
- SQL サーバー

*Note* IPv6 についての詳細な情報は、以下の詳細を参照して下さい。[RFC 2460](#)

IPv6 のサポートは 4D ユーザーや 4D デベロッパが気づくことはありません。プログラムはサーバーの「IP アドレス」がすべてに設定されていれば IPv6 接続でも IPv4 接続でも無差別に受け入れます。

ただし、以下の点に気を付ける必要があります。

- **ポート番号の表記**

IPv6 記法はコロン (:) を使用するので、ポート番号を追加するときには混乱を招く恐れがあることに注意して下さい。例えば以下のような場合です。

2001:0DB8::85a3:0:ac1f:8001 // IPv6 アドレス

2001:0DB8::85a3:0:ac1f:8001:8081 // ポート 8081 指定の IPv6 アドレス

混乱を避けるため、IPv6 アドレスをポート番号と併用する際には、以下の様に [] でアドレスを囲う記法が推奨されます。

[2001:0DB8::85a3:0:ac1f:8001]:8081 // ポート 8081 指定の IPv6 アドレス

■ **TCP ポートが使用されている際の警告は出ません**

これまでのバージョンの 4D と異なり、4D v14 ではサーバーが応答する IP アドレスが「全て」に設定されていた場合には、TCP ポートが他のアプリケーションで使用されていてもサーバー起動時にそれは表示されなくなりました。IPv6 アドレスがあいているため、この場合 4D Server はどのようなエラーも検知しなくなりました。しかしながら、マシンの IPv4 アドレスを使用、またはローカルアドレス 127.0.0.1. を使用してアクセスすることは不可能になりました。

4D server が定義されたポートで反応していないようであれば、サーバーマシンで `::1` のアドレスを試してみてください (IPv6 における 127.0.0.1 と同義です。他のポート番号をテストするためには `portNum` を追加して下さい)。4D が応答するようであれば、他のアプリケーションがポートを IPv4 で使用している可能性が高いです。

## 機能的な互換性

この章では、以前のバージョンから変換されたアプリケーションの機能に変更を及ぼす可能性のある 4D v14 の新機能を説明します。

### 廃止予定の機能

以下の機能は 4D v14 でもまだサポートはされているものの、廃止予定となっている機能であり、使用することは推奨されていません。

#### PICT フォーマットのピクチャ

PICT フォーマットは次のメジャーバージョンではサポートされませんので、4D v14 ではこの機能はこれ以上使用すべきではありません。4D Pack の中に含まれる [AP Is Picture deprecated](#) 機能は、アプリケーションの移行を手助けするためにつけられたものです。

---

*Note:* Mac の "PICT" フォーマットは、Mac OS においては大分前のバージョンから Apple 社によって廃止予定とされているものです ([Wikipedia での PICT 形式の説明](#)を参照して下さい)。

---

#### QuickTime

QuickTime に関するピクチャーコーデックに対するサポートは廃止予定です。

デフォルトとして、4D v14 においては QuickTime は使用不可になっています。しかしながら、互換性の観点から、[SET DATABASE PARAMETER, Get database parameter](#) の [QuickTime support](#) オプションを通して使用可能にすることができます。

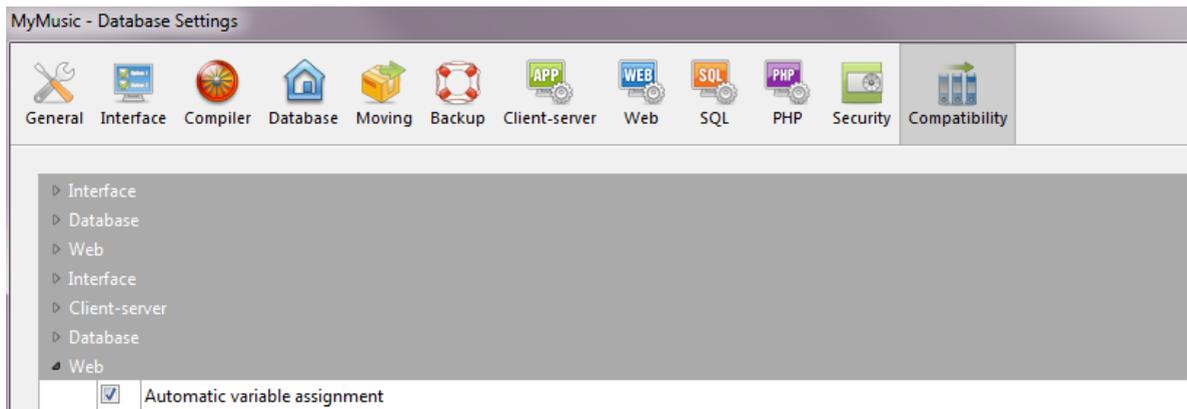
## HTTP 経由で受け取った変数の動的な割り当て

以前のバージョンの 4D では、変数同士が同じ名前であれば、Web サーバーは Web または URL 経由で送られた変数の値を自動的に 4D 変数へと代入していました。

管理と最適化の目的から、この原理は 4D v14 から廃止されました。Web 変数の値は自動的に 4D 変数へと割り当てられることはありません。POST または GET を使用して送信した変数を復元するためには、必ず WEB GET VARIABLES を使用しなければなりません。投稿したファイルを復元するためには必ず WEB GET BODY PART/WEB Get body part count コマンドを使用しなければなりません。

*Note:* 動的な割り当ては、v13.4 以降のバージョンで作成された 4D データベースでもデフォルトで廃止されています。

しかしながら、互換性のために、この機構は 13.4 より前の 4D で作成されたデータベースにはデフォルトで残されています。この場合、データベース設定の互換性ページの **Web 変数に値を自動的に代入する** のオプションを使用して不可にすることができます。



この機構は廃止予定ですので、今後のためにこのオプションのチェックを外し、必要であればコードを書き直すことが強く推奨されます。

## フォームによるクエリ

フォームによりクエリのエディターは、4D v14 では標準のクエリエディターと併合されました (44P “[新しいクエリエディター](#)” を参照して下さい)。これに関連するメニューコマンドはインターフェースからは除去されています。

## フォント番号

QuickDraw ID 番号を使用してのフォントの指定は廃止となり、今後使用されるべきではありません。Font name コマンドと Font number コマンドは 4D v14 では互換性のために残されており、将来のバージョンにおいて削除

される予定です。OBJECT SET FONT コマンドはフォント名を受け付けるようになりました。

## 削除された機能

4D v13 のリリース時にアナウンスされたように、いくつかの廃止予定の機能は 4D v14 では削除されています。

### 4D Chart の使用中止

4D Chart プラグインは v14 からサポートされてません。またその開発やメンテナンスも恒久的に中止となっています。" [グラフ](#) " テーマ内にあるコマンドにのみ SVG シンタックスを使うことができます。また、[階層リスト](#) コマンドは削除されています。

### アプリケーションモードでのツールバー

v14 では、アプリケーションモードに切り替えた時の自動ツールバーが削除されています。[SHOW TOOL BAR](#), [HIDE TOOL BAR](#) コマンドが削除され、[Tool bar height](#) コマンドはアプリケーションモードでは常に 0 を返すようになりました。データベース設定の、インターフェースページ内の " ツールバーを表示 " オプションも削除されています。

### " コーディアル " スpellチェック

コーディアル辞書は 4D v14 ではサポートされていません。ハンスpell辞書 ( と、MacOSX 環境下においてはシステムの辞書 ) のみが使用できません。詳細な情報に関しては、[86P " スpellチェック "](#) を参照して下さい。

### "&" と "|" ピクチャ演算子

モノクロのピクチャの、XOR 重ねと OR 重ねの役割をするための "&" と "|" ピクチャ演算子は、v14 ではサポートされていません。新しい演算子が用意され、それらの機能も変更されています。( 詳細な情報は [256P " ピクチャ演算子 "](#) を参照して下さい )。

### ピクチャーライブラリーエディター

4D v14 では、ピクチャーライブラリーのビットマップエディターが削除されています。ツールボックスを通して、ピクチャーの表示、並べ替え、追加、分割の機能が残されているのみです。

### ノンコンポジットモードのウィンドウ (Mac OS)

Mac OS の 4D v14 で作成された全てのウィンドウに関しては、コンポジットモードは、システム全体に使用されています。以前のバージョンの 4D では、QuickDraw に基づく内製のウィンドウ管理モードを使用してウィンドウを作成することが出来ましたが、v14 ではこのモードは削除されています。

コンポジットモードのおかげにより、4D のウィンドウは Mac OS 環境下において以下の様な新しい挙動と機能が可能になりました。:

- ウィンドウがリサイズされる際のアニメーションの表示

- ウィンドウはどの枠をドラッグしてもリサイズされるようになり、ウィンドウの右下に表示されていたリサイズボックスはなくなっています。
- HiDPI スクリーンモード (または "Retina" モード) との互換性
- メインメニューバーまで自動で隠れる "フルスクリーン" モードの採用 (Mac OS X 10.7 以降) ([303P "Full screen button"](#) を参照して下さい)。
- Mac OS 環境下において、[DISPLAY NOTIFICATION](#) コマンドに対するサポート (Mac OS X 10.8 以降)

*Note:* QuickDraw を必要とするプラグインは、コンポジットモードで使うためには何らかの修正が必要になります。未だに QuickDraw を要求してくるプラグインに対しての互換性を保証するために、4D 14 ではエミュレーションのレイヤーが用意されています。詳細な情報に関しては、4D テクニカルサポートまでお問い合わせください。

## テキストレンダリング (Mac OS)

Apple からの推奨に従い、OS X の 4D v13 ではテキストレンダリングに使用されているフレームワークが更新され統一されています (廃止予定であった QuickDraw/MLTE フレームワークを放棄し、CoreText の使用へと移行しました)。

この新しい CoreText フレームワークへの移行に伴う問題が最小限にとどまるように努力はしておりますが、OS X の 4D v13 へと変換されたアプリケーションのテキストエリアのレンダリング (特に改行に関するレンダリング) において多少の変化を引き起こす可能性があります。この場合、変換する前の v12 のデータベース内でいくつかのフォームオブジェクトをリサイズする必要があるかもしれません。

## 4D Pack

以前のバージョンにおいて 4D Pack 内にあったいくつかのコマンドは、v14 において削除されています。詳細な情報に関しては、[309P "4D Pack から削除されたコマンド"](#) を参照して下さい。

## オプション +F ショートカット (Mac OS)

アプリケーションモードとデザインモードを強制的に切り替える **オプション +F ショートカット** は、4D v14 では削除されています。デベロッパの手によって、"デザインモードへと戻る" の標準アクションをメニューコマンドを追加し、カスタムのショートカットを割り当てることはできません。

**オプション +Shift+ 右クリック** を使用してデバッグメニューへとアクセスすることは従来通りできます。

## Alt+ クリックでボックスを閉じる (Windows)

Windows の 4D のデザインモードにおいて、閉じるボックスを **Alt+ クリック** することで全てのウィンドウを閉じるというショートカットは、

v14 においては削除されています。これは標準のショートカットではありません。

全てのウィンドウを閉じる Windows 用の標準のショートカット、**Ctrl+Alt+W** はご利用いただけます。

**「互換性」内の「データエントリー時の自動処理」オプション**

以前のバージョンにはあった互換性のオプションは最近の 4D ではサポートされていないので、データベース設定の「互換性」のページ内からは削除されています。

## システム要件

4D v14 のアプリケーションを使用するためには、以下のシステム要件を満たしている必要があります。:

	Windows	Mac OS X
プロセッサ	Intel® Core Duo	Intel® Core 2 Duo
OS	Windows 7 以降	Mac OS 10.7.3(Lion) 以降
RAM メモリー	4 GB(64-bit 版 4D Server は 8GB 推奨)	

# 2

## 開発作業環境

4D v14 では、4D アプリケーションのデバッグと展開がしやすくなるように様々な面において改善されています。

- デベロッパーにとっては、データログ、メンテナンス、デバッグ、コンポーネント開発とアプリケーションの配付の機能に関して修正と改善が加えられています。
- ユーザーにとっては、4D v14 では新しいクエリとフォーミュラエディターによるクエリが提供されています。

新機能の多くはフォームオブジェクトに関するものです。これらの新機能については [53P “フォームとオブジェクト”](#) にて詳細な説明があります。

### 新しいデータジャーナル

4D v14 では、データのログファイル (*journal* ファイル) のメカニズムが新しくなり、生成方法と使い方が新しくなりました。

これらの変更によって以下の点が改善されました。

- ミラーサーバーのミラーを使用して安全なアーキテクチャーをセットアップすることができます。
- データのジャーナルを取るテーブルを選択することができます。
- ログファイル統合の処理を、レコード番号に関して厳密ではなくすることにより、抵抗力を高めています。例えば、保存されたオペレーションに関するファイル内の読み出しエラーは、ログファイルの統合全体を妨げることはありません。

4D v14 ではデータログに関して以下の様な新しい規則があります。:

- データのログを取るためには、そのテーブルにプライマリーキー (主キー) が設定されている必要があります。

- ジャーナルを取るテーブルはそれぞれ新しいオプションを使用して指定しなければなりません。

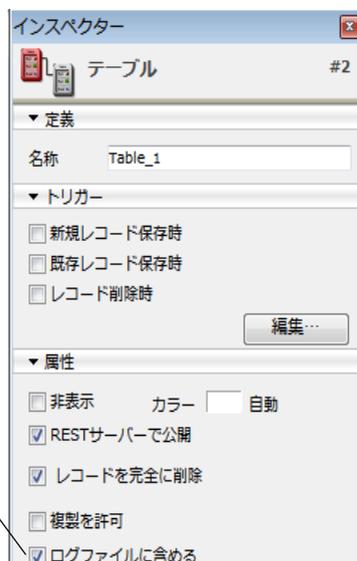
これらの新しい規則は 4D v14 で作成されたデータベースと、変換されたデータベースに対して適用されます。ストラクチャーエディターが変更されているのに加え 4D の SQL ランゲージコマンドに新しいオプションが追加されています。INTEGRATE MIRROR LOG FILE コマンドはミラーデータベースの機構を管理するための新しい引数を受け入れるようになりました。

4D v14 では、プライマリーキーを持たないテーブルのアップデートを補佐するアシスタント機能が追加されました。このアシスタント機能の詳細に関しては 29P “[プライマリーキー管理](#)” を参照して下さい。

*Note* このアシスタント機能は 4D v13 においてコンポーネントとして提供されています。これによって 4D データベースを v14 へと移行する準備をすることができます。

## 「ログファイルに含める」オプション

ストラクチャーエディターのテーブルインスペクターの中に、新しく「ログファイルに含める」というオプションが追加されました。



新オプション

このオプションは、4D v14 で作成された全ての新しいテーブルと変換されたデータベースにおいてはデフォルトでチェックがされています。

このオプションをチェックすることによって、データベースログファイルが生成されたときに、このテーブルで実行されたオペレーションのロ

グを残すように指示します。一般的に、このオプションはほとんどのテーブルにおいてチェックされるべきものです。ただし、例えば一時的なテーブルや、データをインポートするためのテーブルなどに関しては、最適化のためにチェックを外してもよいでしょう。

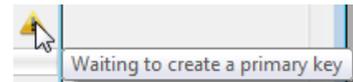
*Note* このオプションは、テーブルがプライマリーキーを持っていない場合にはグレーになっています。

ここで重要なことは、このオプションは、データベースがログファイルを使用するのであればそこにこのテーブルのデータを残すことを指示するだけであり、データベースレベルでジャーナルの処理そのものを有効化するものではないということです。

## 警告メッセージ

変換されたデータベースでは、4D はログファイルに含めるオプションの右横に警告用のアイコンを表示します。

Include in Log File  このアイコンが表示されている限り、データログは起動していません。マウスをアイコンに重ねることで警告の理由を見ることができます。



表示される警告メッセージは以下の通りです。

メッセージ	原因	必要な修正
プライマリーキーを作成する必要があります。	テーブルにプライマリーキーがない場合、オペレーションのデータのログは残せません (4D v14 からの新規則)	SQL またはテーブルのコンテキストメニューを使用してプライマリーキーを作成しましょう。
プライマリーキーの値を修正する必要があります。	プライマリーキーの値が検証され、例外が含まれています。	フィールドのレコード内の、複製された値やヌル値などを全て削除しましょう (または他のプライマリーキーを使用しましょう)。
ログファイルが有効にされる必要があります。	データログを有効にするオプションがチェックされていません。	データベース設定のバックアップページ内の「ログファイルを使用」オプションにチェックを入れましょう。
バックアップが必要です。	最初のログファイルが生成されるためにはデータベースのバックアップが必要です。	データベースのバックアップを実行して下さい。

## デフォルトのプライマリーキーフィールド

4D v14 では、新しく作成されるテーブルには全てデフォルトでプライマリーキーが含まれています。



このフィールドは、デフォルトで名前が "ID" の倍長整数型のフィールドであり、以下の属性を持ちます。

- 重複不可
- REST サーバーで公開されている
- ヌル値の入力を拒否
- 自動インクリメント
- 自動インデックス

*Note* プライマリーキーのデフォルトの名前と型は、4D の設定から変更することができます。詳細に関しては 26P “[「プライマリーキー」設定](#)” を参照して下さい。

このフィールドはそのまま使用することもできますし、必要に応じて名前やプロパティを変更する事もできます (例えば UUID フィールドを使用する等)。また、他のフィールドをプライマリーキーとして使用する場合はプライマリーキーを削除することはできません。

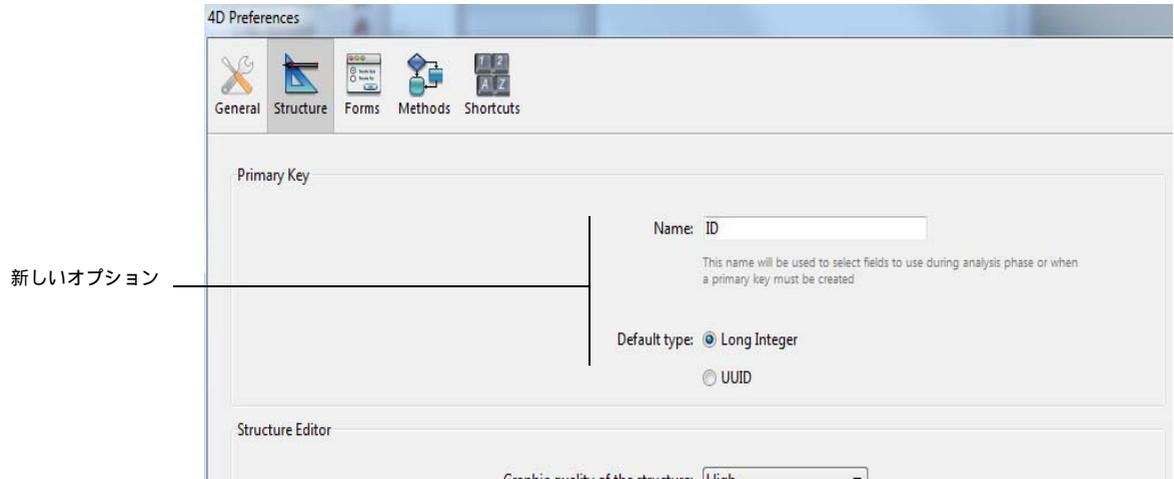
ただし、全ての 4D テーブルに最低一つずつプライマリーキーを用意しておくことが強く推奨されます。

*Note* プライマリーキーはそれぞれのテーブルのレコードを固有に識別するために使用されます。一つまたは複数のフィールドから構成されます。プライマリーキーを作成するための詳細な情報は、4D *Design Reference* マニュアルの「テーブルプロパティ」セクションを参照して下さい。

*Note* デフォルトのプライマリーキーは、SQL CREATE TABLE コマンドを使用して作成されたテーブルやデータベースにインポートされたテーブルには追加されません。

**「プライマリーキー」設定** 設定の中の二つの新しいオプションによって、新しくテーブルが追加されたときに 4D または [プライマリーキー管理](#) によって自動的に追加されるプライマリーキーのデフォルトの名前と型を変更することができます。

これらのオプションは 4D 環境設定の「ストラクチャー」ページ内にあります。



以下のオプションが設定できます。

- **Name** (デフォルトでは"ID"): プライマリーキーのフィールドのデフォルトの名前を設定します。標準的な 4D テーブルの命名規則に従う範囲内であればどんな名前も使用できます。
- **デフォルトタイプ** (デフォルトでは倍長整数): プライマリーキーフィールドのデフォルトのタイプを設定します。UUID を選択することもできます。この場合、デフォルトで作成されたプライマリーキーフィールドは文字型であり、「UUID フォーマット」と「自動 UUID」プロパティにチェックが入っています。

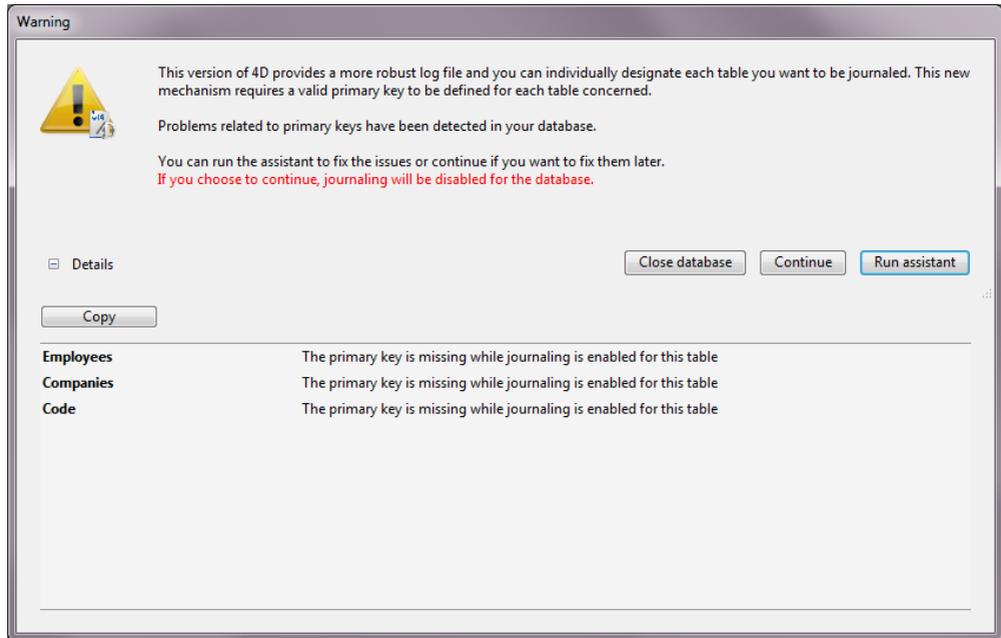
## プライマリーキーのエラーウィンドウ

4D v14 では、ログファイルを使用するためにはログを取るテーブルが全て有効なプライマリーキーを持っている必要があります。

データベース内に、有効なプライマリーキーをないテーブルが少なくとも一つある場合にはエラー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は以下のタイミングで現れます。

- データベースを開いたとき、またはログファイルを使用のオプションがチェックされている v14 のデータベースへと変換した後。
- v14のデータベース内でログファイルを使用のオプションにチェックを入れた後。

データベースストラクチャーにアクセスできる場合、ダイアログボックスはいくつかの選択肢を表示します。またボックス下部を展開すると、プライマリーキーを持っていないテーブルの一覧が表示されます。:



データベースがログファイルを使用していた場合に限り、以下のメッセージが表示されます。

You can run the assistant to fix the issues or continue if you want to fix them later.  
If you choose to continue, journaling will be disabled for the database.

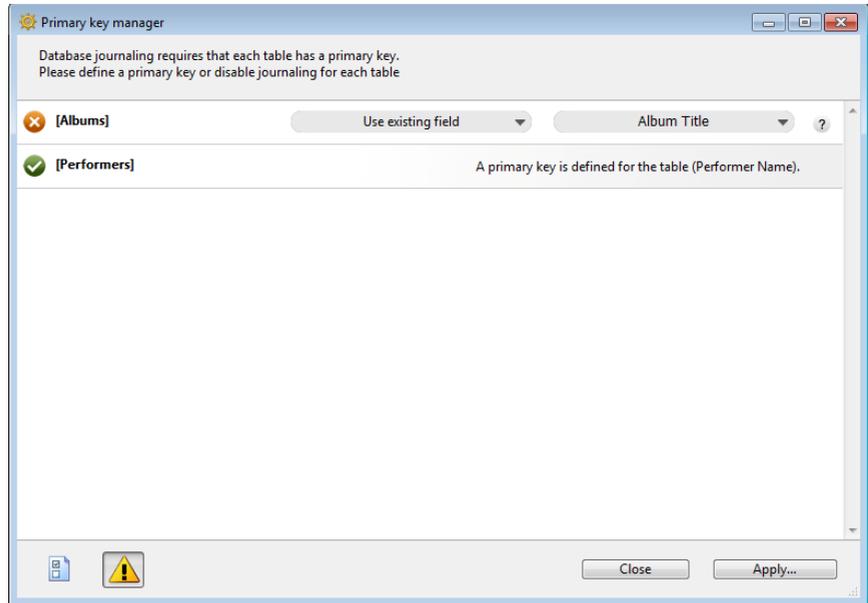
その後選択できるオプションは以下の通りです。

- **コピー** : ウィンドウからの情報を解析のためにクリップボードへコピーします。
- **データベースを閉じる** : データベースには何も触れずに閉じます。
- **続ける** : エラーを処理せずにデータベースを開きます。この場合、データベースのログは無効になり( **ログファイルを使用** のオプションのチェックが外れます)、データベースは v14 で正常に使用できます。データベースにてログファイルを使用したくない場合に有用です。
- **アシスタントを起動** : プライマリーキー管理ウィンドウを表示し、データベース内の全テーブルを更新します。データベースを更新する場合にはこのオプションの仕様が推奨されます。アシスタントについてはこの次のセクションで詳細な説明があります。

データベースストラクチャーへのアクセスがない場合、データログを有効化することはできません。データベース管理者へと連絡を取ることを勧めるメッセージが表示されます。

## プライマリーキー管理

4D v14 では、「プライマリーキー管理」という新しいアシスタントウィンドウが用意されています。これはデータログの際にプライマリーキーがないテーブルに起因するエラーの解消を手助けするためのものです。:



このアシスタントには以下の様な機能があります。

- データベースのそれぞれのテーブルと 4D v14 のデータログのメカニズムとの競合性を検証します。
- データログのメカニズム合致しないテーブルに対してそれぞれに解決方法を提示します。具体的にはプライマリーキーの作成です。

*Note* プライマリーキー管理のウィンドウは v13 でもコンポーネントとして提供されています。これを使用して v13 から v14 への変換の準備をすることができます。

## アシスタントを表示

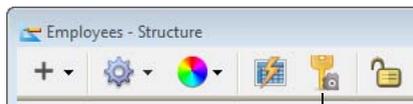
プライマリーキー管理アシスタントを表示する方法は二通りあります。

- [プライマリーキーのエラーウィンドウ](#)から表示: **アシスタントを起動**ボタンをクリックしてアシスタントウィンドウを表示します。

Run assistant

*Note* アシスタントはデザインモードで表示されます。データベースがアプリケーションモードで開かれた場合はアシスタントは直ちには表示されず、デザインモードへと切り替える必要があります。

- ストラクチャーウィンドウから: ウィンドウのツールバー内の新しいボタンをクリックするとアシスタントを表示できます。



プライマリーキー管理ウィンドウを表示

## アシスタントの使用

プライマリーキー管理ウィンドウではデータベース内の全てのテーブルに関する情報が行となって表示されます。

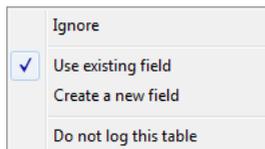


*Note* アシスタントはごみ箱に入っているテーブルは検知しません。

それぞれの行の先頭にあるアイコンはテーブルの状態をユーザーに知らせます。

- ✔ OK                    テーブルには有効なプライマリーキーがあります。
- ✘ プライマリーキーなし、使用可能なフィールド有り      テーブルにはプライマリーキーがないものの、プライマリーキーとして使用できるフィールドが少なくとも一つは含まれています。
- ✘ プライマリーキーも使用可能なフィールドもなし      テーブルにはプライマリーキーも、プライマリーキーとして使用できるフィールドもありません (プライマリーキーフィールドを作成する必要があります)。
- ! 警告                    テーブルにはプライマリーキーがありませんが、データのログを取りません (ログファイルに含めるのオプションのチェックが外されています)。この状態のテーブルは、警告アイコン  をクリックすることによって非表示にできます。

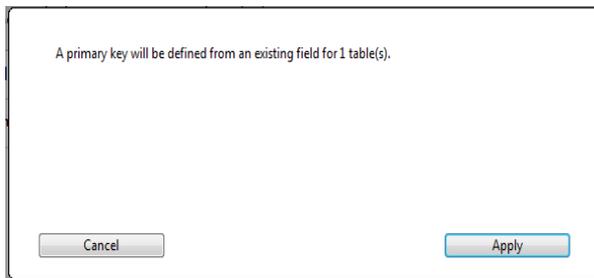
データログを残すテーブルでプライマリーキーがないものに関して、アシスタントはそれぞれに実行すべきアクションを表示します。以下の選択肢から選ぶことができます。



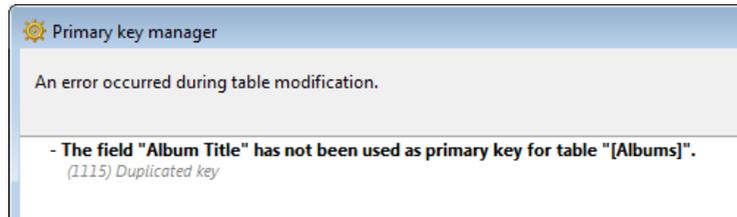
- **無視** : テーブルに何も変更を加えません。エラーは修正されず、テーブルの状態は変更されません。後でエラーを修正したい場合、またはアシスタントを使用せずにエラーを修正したい場合などにはこのオプションを選択して下さい。このオプションは特に、複数のテーブルをもとにプライマリーキーを作成したい場合に必要です。
- **既存のフィールドを使用** (プライマリーキーとして使用可能なフィールドが少なくとも一つはあるテーブルにおいてのみ表示) : テーブルのフィールドのどれかをプライマリーキーとして指定します。このオプションを選択すると、アシスタントはデフォルトでプライマリーキーとして最適なフィールドを選択して提示します。他のフィールドを使用したい場合、またはアシスタントが適切なフィールドを識別するのに失敗した場合 (この場合アシスタントは「選択可能なフィールド」を表示します) などは、行の中の二つ目のメニューをクリックして選択可能なフィールドのリストを表示して下さい。
- **新規フィールドを作成** : テーブル内に新しいプライマリーキーを作成します。このフィールドは新規テーブルが作成されたときのものと同じ属性を持ちます (26P “[デフォルトのプライマリーキーフィールド](#)” を参照して下さい)。デフォルトでは、アシスタントは "ID" という名前の倍長整数型のフィールドが提示されます。このデフォルトのプライマリーキーの名前と型は環境設定から変更することができます (26P “[「プライマリーキー」設定](#)” を参照して下さい)。プライマリーキー管理の画面の中で  をクリックすることにより、環境設定のページを開くことができます。プライマリーキー管理ウィンドウ内で直接テーブルの名前 / 型を変更することもできます。
- **このテーブルはログに含めない** : このテーブルに関しては[ログファイルに含める](#)のチェックを外します。一時的なテーブルなどの場合にこのオプションを選択することができます (24P “[「ログファイルに含める」オプション](#)” を参照して下さい)。ダイアログボックスを確定させると、「警告」の状態がテーブルへと適用されます。

設定が完了したら **適用 ...** をクリックして変更をデータベースに適用するか、データベースを変更せずに閉じる場合には **閉じる** をクリックして下さい。

適用 ... をクリックすると、実行されるオペレーションの一覧がダイアログボックスに表示され、適用かキャンセルがクリックできます。



既存のフィールドをプライマリーキーとして使用した場合、4D はそれぞれのテーブルに関して、既存のデータがそのフィールドの型の固有性やヌル値の設定とあてはまるかどうかを検証します。もし、例えばフィールドが複製値やヌル値を含んでいた場合、エラーが生成されます。



データログを有効にするためにはこれらのエラーを探して削除する必要があります。

## MSC でのプライマリーキーを閲覧

MSC のアクティビティ解析のページは、プライマリーキーの値のレコードを表示するために変更されました。

レコード /BLOB の列は、プライマリーキー /BLOB と改名され、それぞれのレコードにおけるプライマリーキーを表示するようになりました。プライマリーキーが複数のフィールドから構成される場合、値は分割された列の中に表示されます。

レコード数はページ右にある新しい列の中に表示されます。

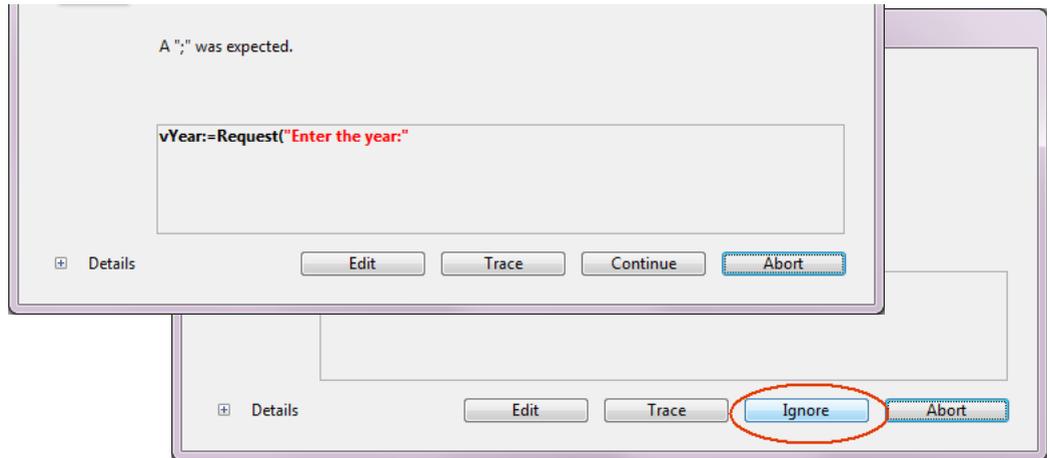
## デバッグヘルプ

4D v14 では、4D アプリケーションのデバッグとメンテナンスを簡単にするためのいくつかの新しい機能が用意されています。

*Note* デバッグログファイルでは新しいフォーマットも用意されています。詳細な情報に関しては、[142P “SET DATABASE PARAMETER, Get database parameter”](#) を参照して下さい。

## エラーを無視

以下のようなシンタックスエラーのウィンドウにおいて、Alt (Windows) または Option (OS X) を押しながら「続ける」ボタンをクリックすると、その同じシンタックスエラーはセッション中は表示されません。これに伴い、Alt (Windows) または Option (OS X) を押している間は「続ける」ボタンが「無視」ボタンに変化します。:



## リアルタイムモニター

4D v14 では、アプリケーションによって実行された、「長い」オペレーションの操作をリアルタイムでモニターできるようになりました。これらのオペレーションとは、例えばシーケンシャルクエリや 4D 式の実行などです。



- 停止ボタンをクリックする
- リストの中をクリックする
- スペースバーを押す

ページを停止させると、一時停止注のメッセージが表示され、ボタンの表示が「再起動」へと変わります。

モニタリングを停止する操作と同じ操作をすることでモニタリングを再開させることができます。

*Note* リアルタイムモニタリングは、新コマンド [GET ACTIVITY SNAPSHOT](#) を使用して行うこともできます。コマンドを使用した場合は追加のオプションもあります。

## ログファイルを開始する

v14 の 4D Server の管理ウィンドウの「メンテナンス」ページには、リクエストログを開始するためのボタンは、デバッグイベントの記録も開始するようになりました。名前もリクエストとデバッグのログを開始となっています。

Request and Debug Logs: 0 second logged.

Start Request and Debug Logs View Report

The server performance might be altered slightly during the generation of the request and debug logs.

このボタンをクリックすると、データベースの *Logs* フォルダに以下の二つの新しいファイルが生成されます。

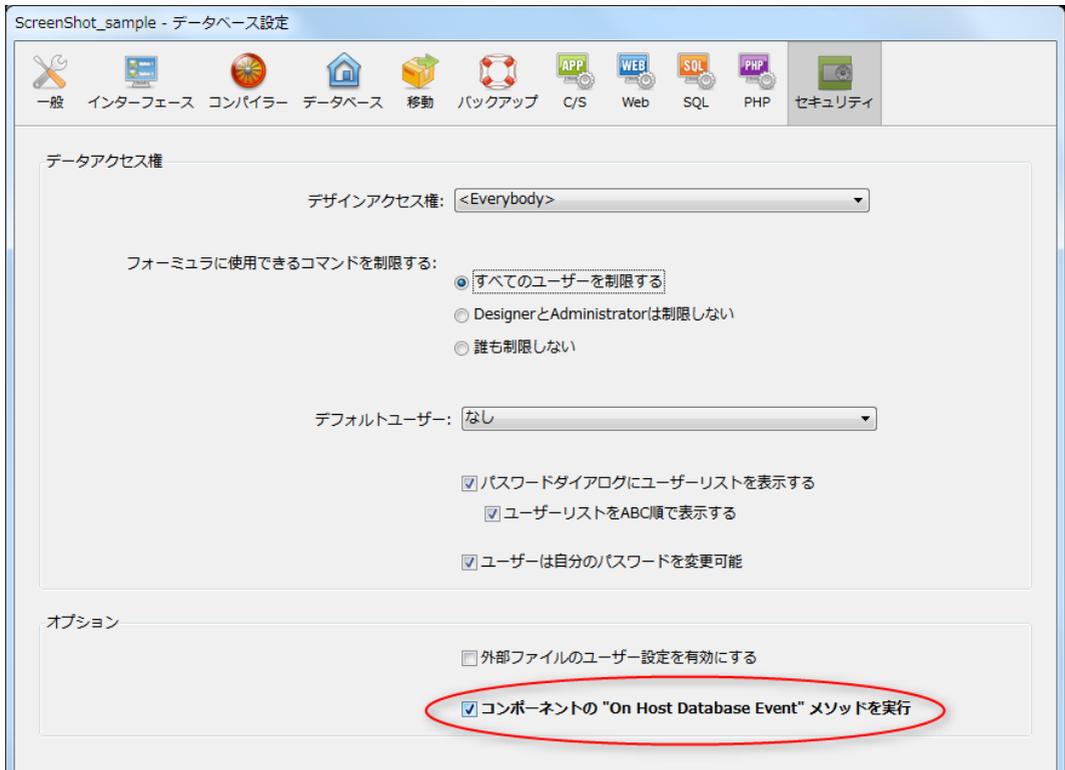
- *4DRequestsLog\_N*
- *4DDebugLog.txt*

## 4D components

4D v14 では、新しい **On Host Database Event** データベースメソッドによって 4D コンポーネントの初期化とバックアップフェースが簡単になりました ( [155P “On Host Database Event”](#) を参照して下さい )。

セキュリティ上の理由から、このメソッドの実行は、それぞれのホストデータベースにおいて、明示的に認証されなければなりません。このためには、データベース設定画面の "セキュリティ" ページ内の、「コン

ポーネントの "On Host Database Event" メソッドを実行」のオプションにチェックを入れます。



デフォルトでは、このオプションはチェックをされていません。

- このオプションがチェックされると：
  - 4D コンポーネントがロードされます。
  - コンポーネントそれぞれの **On Host Database Event** があった場合には、それらがホストデータベースによって呼び出されます。
  - メソッドのコードが実行されます。
- このオプションがチェックされていなかった場合：
  - 4D コンポーネントはロードはされるものの、スタートアップとシャットダウンは行われません (旧バージョンの 4D と同様)。
  - コンポーネントのデベロッパーは、これらのフェーズの間ホストデータベースによって呼び出される必要のあるコンポーネントについて、コンポーネントメソッドを公開しなければなりません (スタートアップとシャットダウン)。

- ホストデータベースのデベロッパは、コンポーネントの適切なメソッドを適切なタイミングで呼び出さなければなりません (適切なタイミングとコンポーネントは、デベロッパが提供します。)

## アプリケーションビルダー

4D v14 のビルダーは組み込み 4D アプリケーションのアップデートを自動化して簡単にするために、以下の様な新しい機能を備えています。

- サーバーアプリまたはシングルユーザー向けの組み込みアプリのアップデートを自動化
- クライアントアプリの新しいアップデート処理
- Windows における新しい「自動アップデート」XML キー
- OS X における認証の統合

### サーバーアプリケーションのアップデートの自動化

旧バージョンの 4D では、サーバーアプリとシングルユーザー用組み込みアプリは、ユーザーの操作 (またはカスタムのシステムルーチンのプログラミング) が必須でした。組み込みアプリの新しいバージョンができる度に製品のアプリを終了し、古いファイルを手動で新しいファイルと置き換え、アプリケーションを再起動してカレントのデータファイルを選択しなければなりませんでした。

4D v14 では、[SET UPDATE FOLDER](#)、[RESTART 4D](#) という新しいランゲージコマンドを使用してこれらの処理を自動化することができます。また、モニターオペレーションには [Get last update log path](#) を使用することができます。これらのコマンドは以下のような自動アップデートシークエンス機能をアプリケーションに実装することを目的としています。FTP サーバーに新しいファイルの有無を定期的に確認するメニューコマンドまたはバックグラウンドで実行されるプロセスとして実装することができます。

新コマンドを使用したアップデートの手順は以下の通りです。

- 1- FTP サーバーなどを通じて、サーバーアプリケーションまたはシングルユーザー向け組み込みアプリケーションの新バージョンを稼働中のマシンへと転送します。
- 2- 稼働中のアプリの中で新コマンド [SET UPDATE FOLDER](#) を呼び出します。このコマンドはカレントアプリケーションのアップデート " 予定 " のバージョンを置くフォルダの場所を指定します。また、このフォルダの中に稼働中のバージョンのカスタムエレメンツ (ユーザーファイル) を再コピーすることも可能です。

- 3-稼働中のアプリケーション内で **RESTART 4D** コマンドを呼び出します。このコマンドは "アップデーター" というユーティリティープログラムの実行をトリガーします。このプログラムはカレントアプリケーションを終了し、指定があった場合にはカレントアプリケーションをアップデート "予定" のバージョンと置き換え、カレントのデータファイルでアプリケーションを再起動します。古いバージョンの方は改名されます。

*Note* この機能は Windows のサーバーアプリケーションとも互換性があります。

また、Windows 環境下で保護されたファイルが使用できるようにインストール権限を引き上げる XML キーもあります (詳細は 39P "新しい「自動アップデート」XML キー (Windows)" を参照して下さい)。

## クライアントアプリ向けの新しい自動アップデート処理

v14 では、HTTP を使用した 4D クライアントアプリケーションの自動アップデート処理が最適化されています。v14 では "アップデーター" という名前のユーティリティーアプリケーションをクライアントマシンにインストールし、ロードをしてアップデートを管理します。

この新しい処理では、主に 4D 内部での変更が顕著です。4D アプリケーションの開発者またはユーザーの視点からみると、この新しいアップデート処理は何も変わりません。

この新しいソリューションはクライアントアプリの v12 と v13 と互換性があります。これらのバージョンであれば v14 のバージョンをダウンロードしてくることが可能です。

## アップデートログ

インストール処理は、ターゲットマシン上の組み込みアプリ (クライアントアプリまたはシングルユーザー向けアプリ) のアップデートオペレーションの詳細を記したログファイルを生成します。このファイルはインストール処理中にエラーが起きたときにそれを解析するために有用です。

アップデートログのファイル名は `YYYY-MM-DD_HH-MM-SS_log_<sequence>.txt` となっています。例えば、2013 年 8 月 25 日の 14:23 に作成されたログのファイル名は `2013-08-25_14-23-00_log_1.txt` となります。

このファイルは "Updater" アプリケーションフォルダの中に作成されます。具体的には以下の場所に作成されます。

- OS X の場合 :  
`{userName}/Library/Application Support/{ProductName}/4D/Updater/`
- Windows の場合 :  
`\{userName}\AppData\Roaming\{ProductName}\4D\Updater\`

上記の場所は、[Get last update log path](#) コマンドを使用することによっていつでも確認することができます。

## 新しい「自動アップデート」XML キー (Windows)

4D v14ではBUILD APPLICATIONコマンドで使用できる新しいXMLキーをご用意しています。このキーは、Windows 環境下においてインストール権限を引き上げることによって、" アップデーター " ユーティリティが組み込みアプリ (クライアントまたはサーバアプリ) をデスクトップのように保護されたシステムロケーションにもインストールできるようにするものです。

*Note* 一般的に、Windows では組み込みアプリを "Program Files" フォルダにインストールすることは推奨されていません。このフォルダは4D アプリケーションの動作と適合しない特定のメカニズムが存在するからです。

これらのキーが True にセットされ、" アップデーター " プログラムが保護されたロケーションにあるアプリケーションをアップデートしようとする、管理者権限が必要であるというメッセージを表示する警告ダイアログボックスが表示されます。必要であれば、マシン上でダイアログボックスが表示され、管理者アカウントへとログオンすることができます。

**RuntimeVL/StartElevated** /Preferences4D/BuildApp/AutoUpdate/RuntimeVL/StartElevated

**受け入れ可能な値** : True / False

Windows のみ : このキーが True の値を格納すると、マシンの管理者権限をもってシングルユーザーアプリケーションのアップデートが行われます。False の値 ( デフォルト ) を格納すると、権限を引き上げることなくアップデートが行われます。

OS X ではこのキーは常に False です。

**CS/Server/StartElevated** /Preferences4D/BuildApp/AutoUpdate/CS/Server/StartElevated

**受け入れ可能な値** : True / False

Windows のみ : このキーが True の値を格納すると、マシンの管理者権限をもってサーバアプリケーションのアップデートが行われます。False の値 ( デフォルト ) を格納すると、権限を引き上げることなくアップデートが行われます。

OS X ではこのキーは常に False です。

**CS/Client/StartElevated** /Preferences4D/BuildApp/AutoUpdate/CS/Client/StartElevated

**受け入れ可能な値** : True / False

Windows のみ : このキーが True の値を格納すると、マシンの管理者権限をもって組み込みアプリケーションのアップデートが行われます。  
False の値 ( デフォルト ) を格納すると、権限を引き上げることなくアップデートが行われます。

OS X ではこのキーは常に False です。

**CS/ClientUpdateWin/Start Elevated** /Preferences4D/BuildApp/AutoUpdate/CS/ClientUpdateWin/StartElevated

**受け入れ可能な値** : True / False

OS X のみ : このキーが True の値を格納すると、マシンの管理者権限をもって Windows クライアントアプリケーションのアップデートが行われます。  
False の値 ( デフォルト ) を格納すると、権限を引き上げることなくアップデートが行われます。

Windows ではこのキーは常に False です。

## OS X 環境下でのアプリケーションの認証

4D v14 アプリケーションビルダーは、OS X 環境下において組み込み 4D アプリに署名をする新機能を備えています (OS X 環境下のシングルユーザーアプリ、4D サーバーアプリおよびクライアントアプリ)。アプリケーションを署名する事により、OS X において「Mac App Store と確認済みの開発元からのアプリケーションを許可」のオプションが選択されているときに *Gatekeeper* の機能を使用してアプリケーションを実行することが可能になります。

## Gatekeeper について

*Gatekeeper* とは OS X のセキュリティ機能で、インターネットからダウンロードしてきたアプリケーションの実行を管理するものです。OS X の 10.8 Mountain Lion 以降、「Mac App Store と確認済みの開発元からのアプリケーションを許可」のオプションがデフォルトで選択されています (Apple は最低レベルの「すべてのアプリケーションを許可」のオプションを選択することを推奨していません)。もしダウンロードしてきたアプリ

ケーションが Apple Store からダウンロードしたものではない、または署名されていない場合には実行が拒否されます。



4D アプリケーションビルダーの新機能は、このオプションとデフォルトで互換性のあるアプリケーションを生成可能にします。

*Note* Apple からはアプリケーションの署名に関するドキュメントが提供されています。

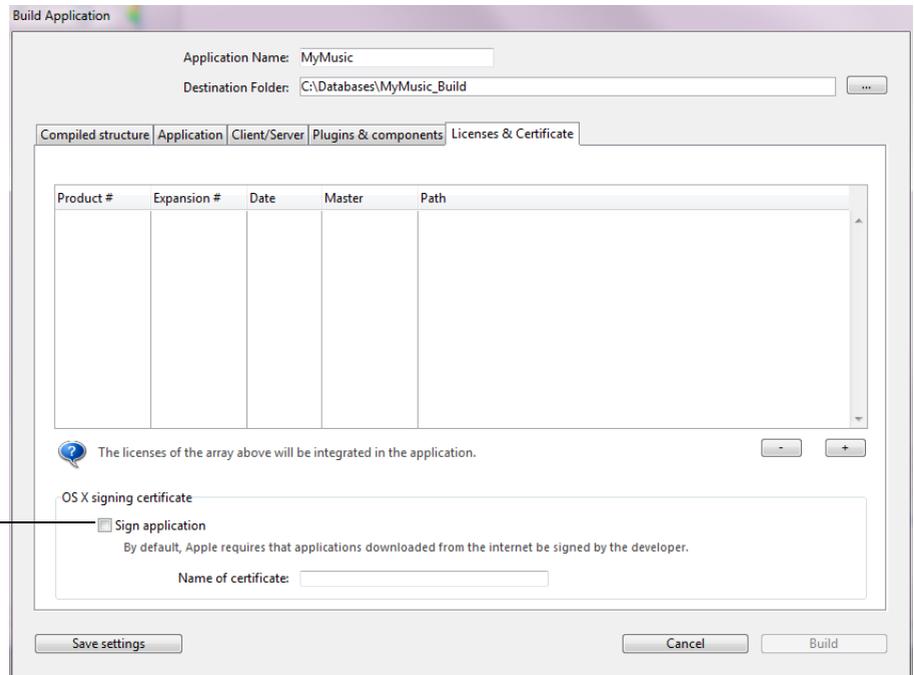
(<http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html>)4D アプリケーションビルダーはこの手順を簡略化したものです。

## アプリケーションビルダーでの新オプション

アプリケーションビルドのウィンドウの「ライセンス & 証明書」ページ内に、アプリケーションに署名という新しいオプションが追加されてい

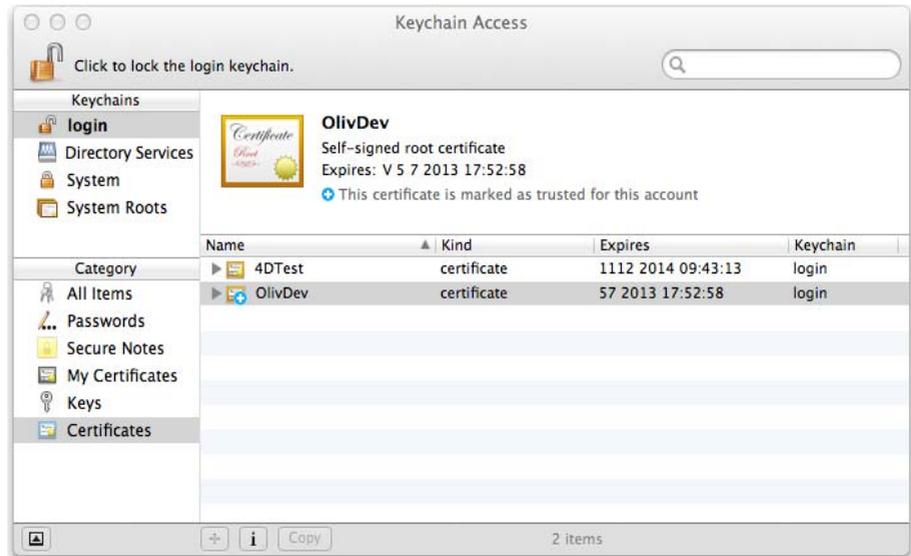
ます。これは Windows でも OS X でも表示されますが、OS X バージョンのもののみ有効となります。

新オプション



- **アプリケーションに署名**: このオプションをチェックすると、OS X のアプリケーションビルド処理に認証が含まれます。4D はビルドの際に認証に必要な要素の有無をチェックします。

- **認証名** : Apple によって有効化されたあなたのデベロッパ認証の名前を入力してください。この認証の名前は通常、キーチェーンアクセスユーティリティの中の証明書の名前と一緒にです。



Apple からデベロッパ認証を取得するためには、キーチェーンアクセスのメニューのコマンドを使用するか、以下のリンクへ移動して下さい。

<http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodesSigningGuide/Procedures/Procedures.html>

*Note* 証明書の取得には Apple の *codesign* ユーティリティが必要になります。このユーティリティはデフォルトで提供されており、通常 “/usr/bin/” フォルダの中にあります。エラーが起きた際には、このユーティリティがディスク上にあるかどうかを確認して下さい。

## 新しい "SignApplication" XML キー

OS X 環境下での認証を管理するために、新しい XML キーを BUILD APPLICATION コマンドと使用することができます。

**MacSignature/**

/Preferences4D/BuildApp/SignApplication/MacSignature/

**受け入れ可能な値** : True / False

**MacCertificate/** によって指定された認証が有効になり、ビルドされたアプリケーションが署名されるためには、このキーが True の値を含んで提示されている必要があります。

もしこのキーが False( デフォルト ) だった場合、アップデートは権限を引き上げることなく行われます。

Windows では、このキーは常に False です。

**MacCertificate/**

/Preferences4D/BuildApp/SignApplication/MacCertificate/

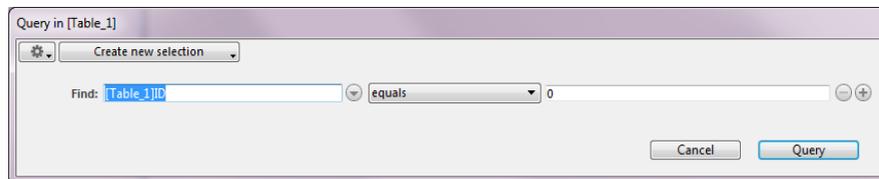
**受け入れ可能な値** : True / False

このキーは、アプリケーションを署名する際に使用する認証を、認証ツールに対して指定する際に使用します。認証の一般的な名前か、キーチェーンアクセスの証明書プリファレンスの完全な文字列を渡します。

このキーは **MacSignature/** キーが渡された際には必須となります。渡さなかった場合は組み込みアプリは署名されず、キーも考慮されません。

## 新しいクエリエディター

4D v14 では、標準のクエリエディターと、フォームによるクエリを使用することができます。新しいクエリエディターは刷新され、セッション内で直近に使用した 10 個のクエリを保存しておくことができます。



フォームによるクエリは検索モードとして使用され、キーボードショートカットを通してのみ使用できるようになります ( 46P “クエリの生成” を参照して下さい )。

### エディターを呼び出す

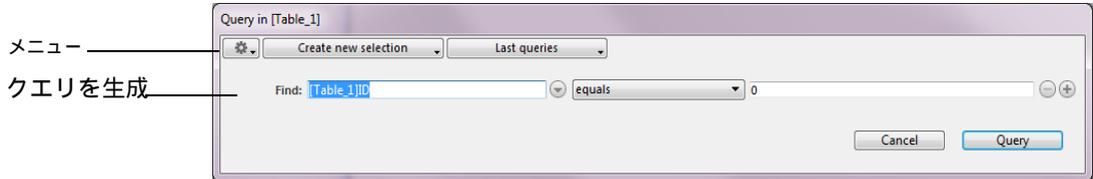
4D v14 では、以下の方法でエディターを呼び出すことができます。:

- メニューのレコード内から、クエリ > クエリ ... を選択して下さい。
- ツールバーのクエリボタンにのメニュー内にあるクエリ ... を選択して使用して下さい。
- 4D ランゲージの、QUERY コマンドまたは QUERY BY FORMULA コマンドにクエリ文字列を渡さずに実行することで使用して下さい。

*Compatibility note* これらのエディターが採用されたことにより、**レコード/クエリメニュー**内にあった**フォーミュラによるクエリ**コマンドと**クエリボタン**は4D v14では削除されています。

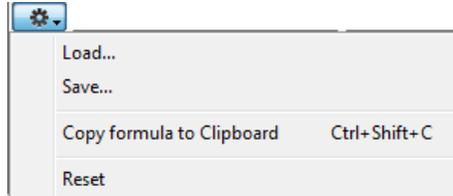
## エディターのメニュー

エディターのウィンドウにはメニューエリアとクエリを生成するエリアの二つの部分があります。セッションの間少なくとも一つのクエリが既に行われていた場合は、**直前の検索**というメニューが表示されます。:



## 編集メニュー

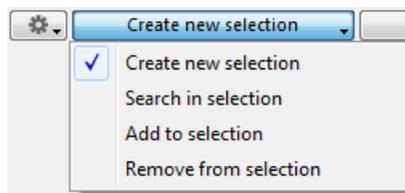
編集のメニューにはクエリコードを管理するためのコマンドがあります。



- **読み込み ...** と **保存 ...** は、以前のバージョンの4Dと同様、クエリファイルのロードと保存を管理します。  
このボタンでは標準のクエリに加え、**フォームによるクエリ**のファイルを管理することも可能です。
- **検索条件をコピー**は、編集エリア内の検索条件を、クリップボードへとコピーします。
- **リセット**は、エディター内で設定されている全てのクエリの条件を消去します。この消去の操作は**取り消しできません**。

## セレクションのアクションのメニュー

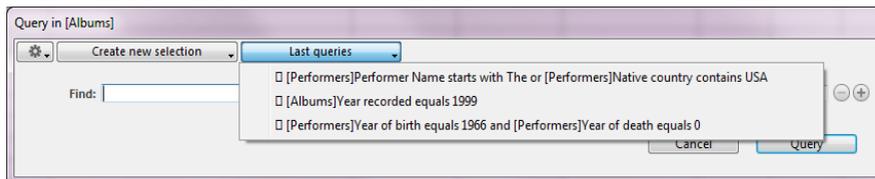
このメニューでは、既存のレコードのカレントのセレクションによって、実行するクエリアクションを選択します。



- **セレクションの作成**(デフォルトのアクション):4Dはテーブル内の全てのレコードを検索し、オリジナルのカレントセレクションの代わりにヒットしたレコードをカレントのセレクションとして表示します。
- **セレクションの絞り込み**:4Dはオリジナルのカレントのセレクションのレコード内のみを検索し、その中でヒットしたレコードをカレントのセレクションとして表示します。
- **セレクションに追加**:4Dはテーブル内の全てのレコードを検索し、ヒットしたレコードをオリジナルのカレントのセレクションに追加します。ヒットしたレコードのうち既にセレクション内に含まれていたものは、表示されますが、複製されることはありません。
- **セレクションから除外**:4Dはテーブル内の全てのレコードを検索し、ヒットしたレコードをカレントのセレクションから除外します。

## 直前の検索

このメニューは、少なくとも一つのクエリが既に行われた場合に限り表示されます。メニューにはセッション中実行されたクエリが保存されるので、よく使うクエリを簡単に繰り返し実行することができます。クエリは最大で10個まで保存されます。



ただし、クエリに関連付けられたアクション( **セレクションの絞り込み**、**セレクションに追加**、等)は保存されないことに注意して下さい。

このメニューからクエリが選択されると、その詳細がクエリを生成するエリアに表示されます。その後「クエリ」ボタンを押してそれを実行したり、必要に応じて修正したりすることができます。

## クエリの生成

新しい4D v14のクエリエディターは、標準のクエリに加えてフォームによるクエリを実行することもできます。

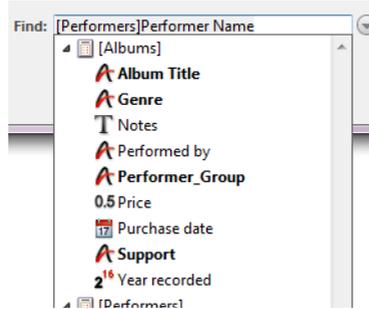
旧バージョンの4Dと同様、標準のクエリは、一つ以上のクエリラインを指定し、それらを接続詞で組み合わせることによって生成されます。

フォームによるクエリは高度なクエリモードであり、ライン追加ボタン  を **Alt+click** (Windows) または **Option+click** (Mac OS) することによって使用できるようになります(50P“[フォームによるクエリ](#)”を参照して下さい)。

## 標準のクエリ

標準のクエリを使用するためには、"フィールド 比較演算子 値" 型の条件式を書いて下さい。

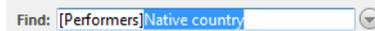
- *field* を指定するためには、編集エリアの右側にある階層リストをクリックして使用します。:



また、編集エリアをクリックしてテーブル名の最初の数文字を入力することで、自動補完機能によってテーブル名の候補が表示されます(ただし "[" は入力禁止となっています)。

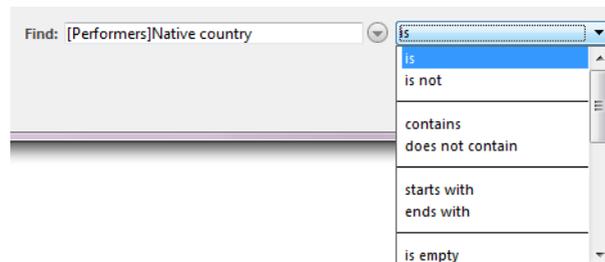


テーブルを選択し、十字キーの右矢印のボタン → を押してテーブルの候補を確定させると今度はフィールドの選択に移ります。フィールド名の最初の数文字を入力するか、十字キーの上下ボタン ↑↓ を使用してテーブル内のフィールドをスクロールさせることができます。:



リストにはデータベースの全てのテーブルとフィールドが表示されます。SET TABLE TITLES と SET FIELD TITLES コマンドを使用してヴァーチャルストラクチャが指定されていた場合、それらも一覧の中に表示されず。

- *comparison operators* のリストは定義されたフィールドの型に応じてアップデートされています。.



標準の比較演算子に加え、新しいクエリエディターでは拡張された演算

子と入力済みの値の型によって、もっとも適切なクエリへと素早くたどり着けるようになっていきます。:

演算子	文字テキスト	日付	時間	ブール	数値	ピクチャ	詳細
空の文字列 / = 空	X					X	フィールドにはデータがありません。
空の文字列 / 空	X					X	フィールドにデータはあります。
と等しい					X		数値用の標準の演算子です。
>=					X		
>					X		
<=					X		
<					X		
					X		
偽 (false) である				X			ブール用の標準の演算子です。
真 (true) である				X			
=	X	X	X				フィールドは入力された値と完全に一致します。
	X	X	X				フィールドは入力された値と異なります。
>=	X	X	X				フィールドの値は、入力された値と一致するかまたは大きいです (*)
>	X	X	X				フィールドの値は入力された値を超えます (*)
<=	X	X	X				フィールドの値は、入力された値と一致するかまたは小さいです (*)
<	X	X	X				フィールドの値は入力された値未満です (*)
範囲指定 (以上以下)		X					最初の日付は二番目以前の日付である必要があります。入力された日付も含め、間にある日付のフィールドを検索します
範囲指定		X					最初の日付は二番目以前の日付である必要があります。入力された日付は含めずに検索します。
= 今日		X					カレントの日付が表示されている
= 昨日		X					一日前の日付が表示されている
= 今週 (月・四半期・年) 中		X					とり得る値: - 週 (日 - 土) - 週 (月 - 日) - 週 (月 - 金) - 月 - 四半期 - 年 これらの値は、カレントの日付を基準として算出されます。
= 昨週 (月・四半期・年) 中		X					
= 翌週 (月・四半期・年) 中		X					

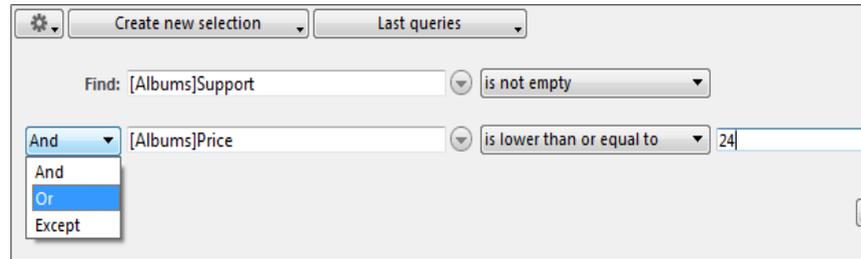
演算子	文字テキスト	日付	時間	ブール	数値	ピクチャ	詳細
= 時間 (分・秒) 以内の過去			X				とり得る値: - 時間
= 時間 (分・秒) 以内の未来			X				- 分 - 秒 これらの値は、カレントの時刻を基準に算出されます。
範囲指定 (以上以下)	X		X		X		フィールドの値は入力された値の間です (入力された値も含まれます)。 (*)
範囲指定	X		X		X		フィールドの値は入力された値の間です (入力された値は含まれません)。 (*)
=			X				とり得る値: - 時間 - 分 - 秒
>=			X				
>			X				
<=			X				
<			X				
前方一致	X						
後方一致	X						テキスト用の標準の演算子です。
含む	X						
含まない	X						
キーワード	X					X	
キーワード (含まない)	X					X	入力されたキーワードを検索します。「すべて」(を含むフィールドを検索する)と「どれか」(を含むフィールドを検索する)のどちらかを指定することができます。
<=						X	ピクチャのサイズに基づいて検索します。単位はバイト、KB、MB、GBから選択することができます。
>=						X	

(\*) 文字列の場合、クエリはアルファベットに基づいています (a < b とみなされます)。例えば、「名前が "don" よりも後である」というタイプのクエリを実行した場合、検索結果には Donna、Don、Juan、Smith、、等が含まれますが、Alves や Dominick はヒットしません。

## 複数のクエリ

複数の条件を組み合わせるクエリを実行する場合、ライン追加のボタン  を必要な数だけ押してラインを追加して下さい。

エディターにはそれだけラインが追加され、論理演算子によってそれらを組み合わせることができます。:



ライン削除ボタン  を押すことによって、条件の数を減らすこともできます。

## フォームによるクエリ

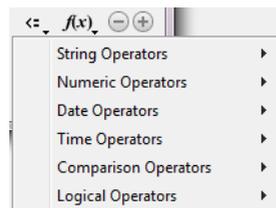
4D v14 では、フォームによるクエリは標準のクエリエディターの中に統合されました。フォームによるクエリとは高度なクエリモードです。

フォームによるクエリを定義するためには、まずラインを追加ボタン  を **Alt+ クリック** (Windows) または **Option+ クリック** (Mac OS) します。メニューが多いラインが追加されます。:

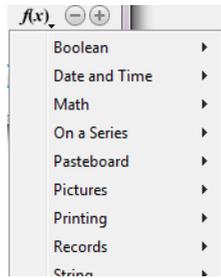


フォームによるクエリを 1 行だけ使用する場合、デフォルトで追加されている最初の行を削除して下さい。

-  : をクリックすると、フォームによるクエリで使用できる全てのデータベースとフィールドの一覧の階層リストが表示されます。
-  : をクリックすると、フォームによるクエリで使用できる全ての演算子の一覧の階層リストが表示されます。.

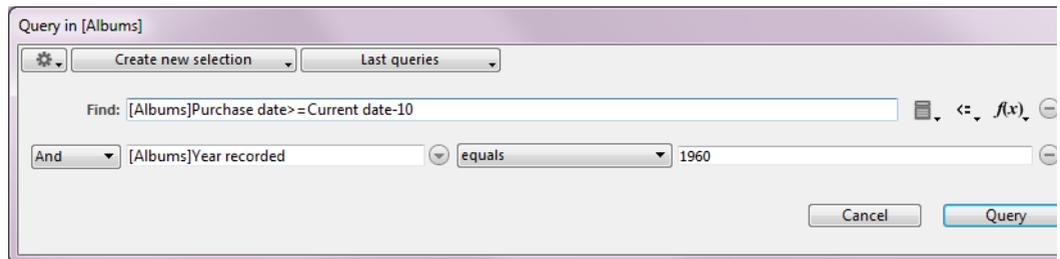


- $f(x)$  をクリックすると、フォームによるクエリで使用できる全ての 4D ファンクションの一覧の階層リストが表示されます。



*Note* 旧バージョンの 4D と同様、SET ALLOWED METHODS コマンドを使用することによってこのリストにプロジェクトメソッドを含めることもできます。

この新しい特徴をを通じて、フォームによるクエリの条件を標準のクエリの条件と組み合わせて使用することが出来るようになりました。:



また、こうしてこのダイアログボックスへと統合されたことにより、フォームによるクエリは以下の点において標準のクエリと同じ機能を持つようになりました。:

- **セレクションのアクションのメニュー** を使用してクエリアクションを選択できるようになったこと。
- **直前の検索** の一覧の中から実行済みのクエリを再度選択できるようになったこと。
- ファイルメニューを通して保存や読み込みができるようになったこと。

## Mecab のサポート (日本語版)

4D v14 の日本語版のシステムでは、日本語に特化したキーワードのインデックスアルゴリズムを兼ね備えた *Mecab* ライブラリーをサポートします。

この新しいアルゴリズムは 4D v14 の日本語版ではデフォルトで使用されています。*Mecab* ライブラリーに必要なファイルは 4D アプリケーションの **Resources** フォルダ内の **mecab** フォルダにインストールされています。

また、必要に応じて *Mecab* アルゴリズムを無効化して以前の *ICU* ライブラリーを使用することも可能です。

*Mecab* を無効化するには、データベース設定ダイアログボックス内のデータベース データストレージのページ内の**非文字・非数字のみをキーワード区切り文字とする**のオプションにチェックを入れるだけです。



---

*Note* 4D アプリケーションの **Resources/mecab** フォルダを削除または名称変更することで *Mecab* を無効化することもできます。

---

# 3

## フォームとオブジェクト

4D v14 では、フォームとオブジェクトが、4D のデベロッパの要望に応える形で刷新されています。具体的には、リストオブジェクト (ポップアップメニュー、階層リスト、等)、Web エリア、そしてテキストエリアが強化されています。

### プレースホルダーテキスト

4D v14 では、フォームのフィールド内にプレースホルダーテキストを表示する新しいオプションが追加されました。

このテキストはフィールド内で半透明のテキストとして表示され、入力されるデータに関するヘルプ、指示、具体例などを表示します。

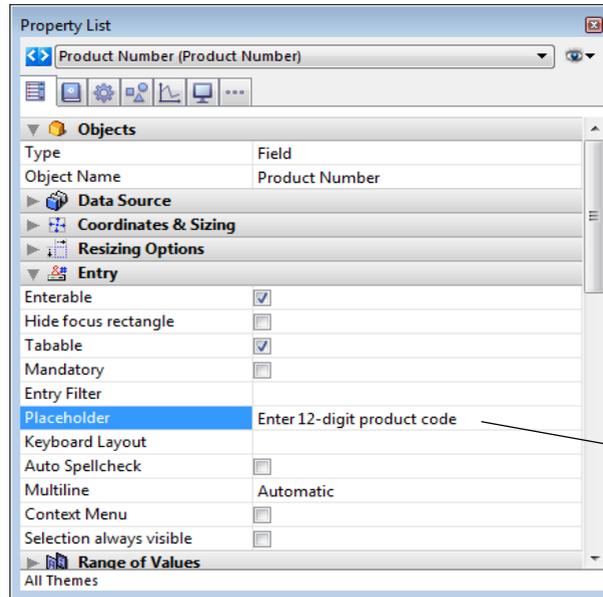
Product Number

このテキストはユーザーが文字をエリアに入力した瞬間に表示されなくなります。:

Product Number

プレースホルダーテキストは、フィールドの中身が消去されると再び表示されます。

プレースホルダーテキストは、プロパティリストの入力のテーマから設定することができます。:



プレースホルダーテキストをここで定義

プレースホルダーのオプションは以下のオブジェクトに対しても設定可能です。:

- 変数
- フィールド
- コンボボックス

プレースホルダーとして表示できるデータの型は以下の通りです。:

- 文字列 (テキストまたは文字フィールド)
- **ヌル**のとき**ブランクにする**のプロパティがチェックされていれば、日付または時刻のデータを表示することも可能です。

xliff 参照を `":xliff:resname"` の形でプレースホルダーとして使用することもできます。例えば、:

```
:xliff:PH_Lastname
```

参照を使用する場合にはプレースホルダーのフィールドには参照のみを渡して下さい。参照と静的なテキストを組み合わせることはできません。

*Note* プレースホルダーのテキストは、新しい [OBJECT SET PLACEHOLDER](#) と [OBJECT Get placeholder](#) コマンドを使用するとプログラミングによって設定したり取得したりすることができます。

## コンボボックス

4D v14 では、コンボボックスタイプのオブジェクトは選択リストに関して 2 つの新しいオプションを設定することができます。**自動挿入**と、**除外リスト**（除外された値のリスト）です。

これに加えて、OS X でのコンボボックスのインターフェースが標準のものになりました。

### 値の自動挿入

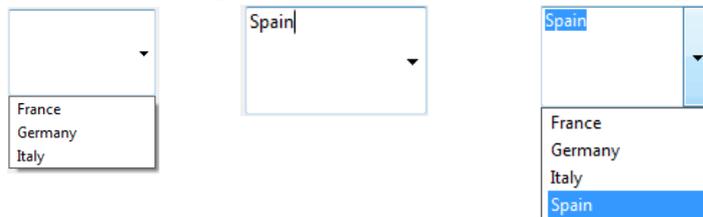
新しい **自動挿入** のオプションはコンボボックス型のオブジェクトのプロパティリスト内の、データソースのテーマ内にあります。:



*Note* このオプションはリストボックス列にも存在します。何故なら列が選択リストと関連付けられているときには、列のセルはコンボボックスとして表示されるからです。

このオプションがチェックされているときに、ユーザーが関連付けられている選択リストの中のない値を入力すると、その値が自動的にメモリー内のリストの中に追加されます。

例えば、"Countries" というコンボボックスが "France, Germany, Italy" という値を持った選択リストと関連付けられているとします。このとき、**自動挿入**のオプションがチェックされていて、ユーザーが "Spain" という値を入力すると、"Spain" という値がメモリー内のリストに追加されます。:



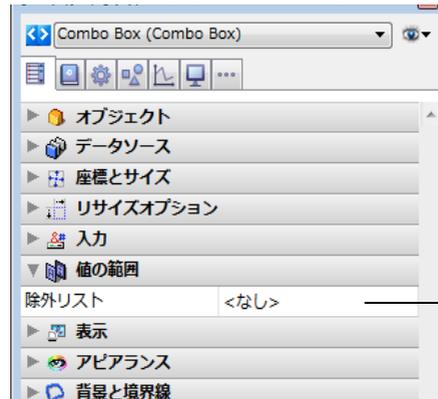
必然的に、もし除外リストが設定されていた場合にはその中に含まれる値を入力することはできません（次の段落を参照して下さい）。

*Note* デザインモードで定義されたリストが関連付けられた場合、自動挿入によって、オリジナルの選択リストが変更されることはありません。

**自動挿入** のオプションがチェックされていない場合、入力された値はオブジェクトの中には保存されますが、メモリー内のリストには入力されません（従来のバージョンの 4D と同じ挙動をします）

## 除外リスト

**除外リスト** のオプションはコンボボックス型のオブジェクトの " 値の範囲 " のテーマ内にあります。これを通して除外された値のリストをこれらのオブジェクトに関連付けることができます。ユーザーがこのリストに含まれる値を入力したとき、その入力自動的に却下されます。



除外された値のリストをコンボボックスに関連付けるためのオプション

4D v14 では、新しい **OBJECT SET LIST BY REFERENCE** コマンドと変更された **OBJECT SET LIST BY NAME** コマンドを使用することにより、プログラミングで値のリストをオブジェクトへ関連付けることができます。

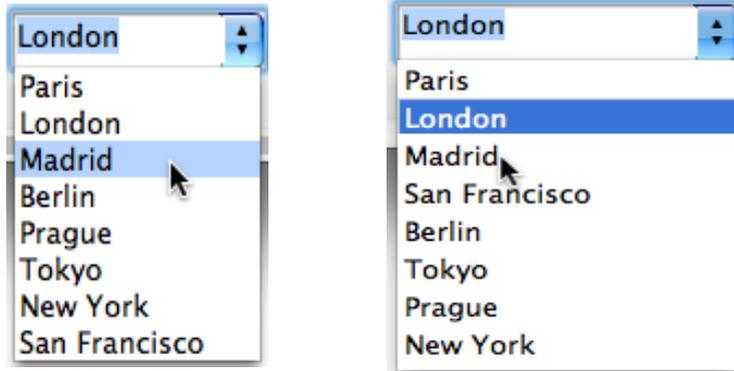
*Note* 指定リストはコンボボックスには割り当てることはできません。画面上において、オブジェクト内にいくつかの指定された値を表示したいときには、ポップアップメニュー型のオブジェクトを使用して下さい。

## OS X 環境下での、値のマウスオーバー

OS X 環境下でのコンボボックスは、仕様が変更され、以前は、関連付けられたメニューの中でマウスが重なっている値が強調されていたのが、4D v14 からは現在選択されている値が強調されるようになりました。:

4D v13 以前のコンボボックスの様子

4D v14 のコンボボックス



## マルチスタイルエリア

4D v14 のマルチスタイルエリア (またはリッチテキストエリア) では、いくつかの新機能が追加されています。

新機能はランゲージのコマンドを通して使用可能です。詳細な情報に関しては 264P “スタイル付テキスト” を参照して下さい。

### マウスオーバー

マルチスタイルの上をマウスが通過する際に、二つの自動的な機能が追加されました。:

- マウスがテキスト選択範囲の上を通過するとき、矢印に変わります。:



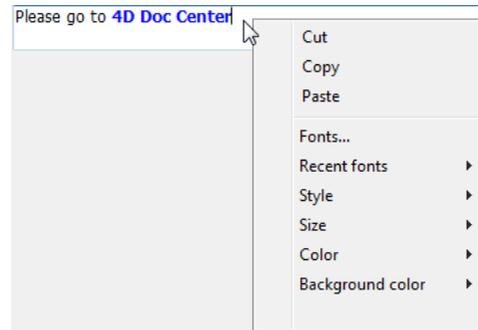
- URL の上を動く際には、アドレスが書かれたヒントが表示されます。:



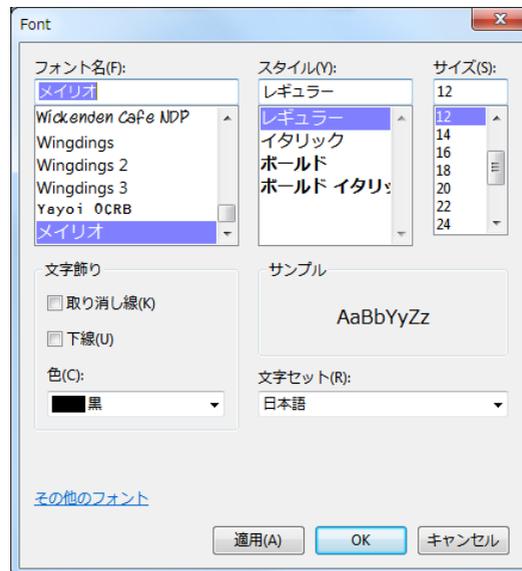
*Note* 新しい **ST INSERT URL** を使用することで URL のリンクを挿入することもできます。

## コンテキストメニュー

リッチテキストエリアに関連付けられたコンテキストメニュー内に、二つの新しいオプションが追加されました。:



- **フォント ...**: フォントマネージャースystemダイアログボックスを表示します。:



フォントマネージャ  
(Windows)

- **最近使用したフォント**: セッション中において最近使用したフォントの名前を表示します。リストには最大 10 フォントまで保存できます (それ以上は古いものから置き換えられていきます)。デフォルトではこのリストは空なのでオプションそのものが表示されません。  
[FONT LIST](#) コマンドと [SET RECENT FONTS](#) コマンドを使用して、このリストを管理することが出来ます。

## 新しいタグ

リッチテキストエリアでは新しいタグがサポートされるようになりました。

## 4D 式

```
<span style="-d4-ref:'expression'"> </span>
```

4D 式 (式、メソッド、フィールド、変数、コマンド) をテキストに挿入します。その式はトークナイズされ、以下のタイミングで評価されます。:

- 4D 式が挿入されたとき
- オブジェクトがロードされたとき
- **ST COMPUTE EXPRESSIONS** コマンドが実行されたとき
- **ST FREEZE EXPRESSIONS** コマンドが、第二 \* 演算子を渡された状態で実行されたとき

4D 式の評価された値は `<span>` タグには保存されず、参照のみが保存されます。

## URL

```
<span><a href="url">Visible label</a></span>
```

URL をテキストに挿入します。

例:

```
<span><a href="http://www.4d.com/">4D Web Site</a></span>
```

## ユーザーリンク

```
<span style="-d4-ref-user:'myUserLink'">Click here</span>
```

"ユーザーリンク" は見た目は URL と同じですが、クリックしてもソースは自動的には開きません。参照としてはどんな文字列を渡す事もできますが、クリックされたときに起こるアクションはデベロッパによってプログラムされた内容によります。

これはつまりタグがクリックされるときに、URL ではなく、ファイルや 4D メソッド等を参照し、開いたり実行したりすることができるということです。**ST Get content type** コマンドによって、ユーザーリンクがクリックされたかどうかを検知することもできます。

## カスタムのタグ

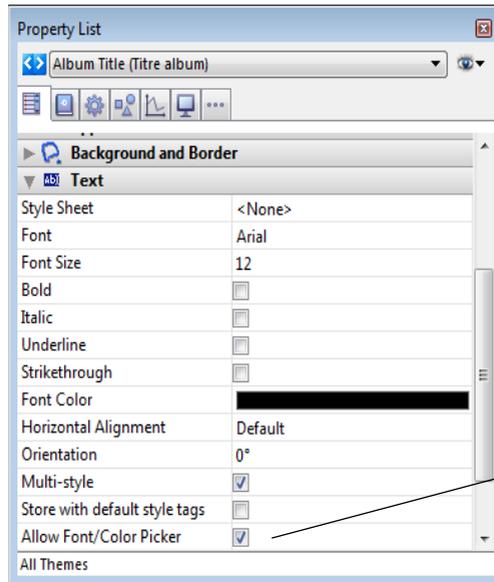
4D v14 からは、スタイル付テキストにどんなタグも挿入することが出来るようになりました。例えば `` のようなタグでもです。タグは標準テキストのコードに保存されますが、解釈されたり表示されたりすることはありません。これは例えば、HTML フォーマットの E メールの中に画像を挿入するような場合に特に有効です。

カラーピッカーと  
フォントピッカーの  
使用を許可

4D v14 では、**OPEN COLOR PICKER** コマンドと **OPEN FONT PICKER** コマンドを使用することによってカラーピッカーとフォントピッカーを表示させることが出来ます。これらのウィンドウをクリックすることにより、

ユーザーは、フォーカスされているオブジェクトのカラーやフォントを直接変更することが出来ます。

これらのユーザーの行動を管理するために、この機能を有効化するためには「テキスト」テーマ内の新しいピッカーの使用を許可のプロパティにチェックをします。



新プロパティ

このプロパティは、フィールド、変数、そしてコンボボックス型のオブジェクトに対して存在します。初期設定では全てのフォームオブジェクトにおいてこのチェックは外れています。ピッカーウィンドウを使用してカラーやフォントを変更可能にしたいオブジェクト全てについて、明示的にこのプロパティにチェックをする必要があります。

## 階層リスト

4D v14 では、Windows 環境下における階層リストのグラフィックレンダリングが向上し、これらのオブジェクトが最新の Windows のインターフェースに自然に適合するようになりました。

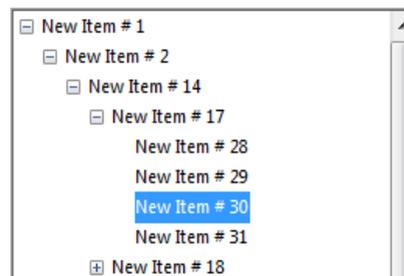
以下の点において機能が改善されています。:

- 展開 / 折り畳みのアイコンが三角形のものに変わりました。:▷ (折りたたまれた階層) と ◻ (展開された階層)

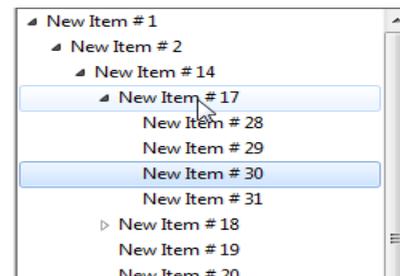
- 選択された項目の表示方法も変更になり、現在マウスが重なっている項目も強調されるようになりました。

以下の画像は 4D v13.x と 4D v14 との階層リストの外観の比較を表しています。

4D v13 での階層リスト



4D v14 での階層リスト



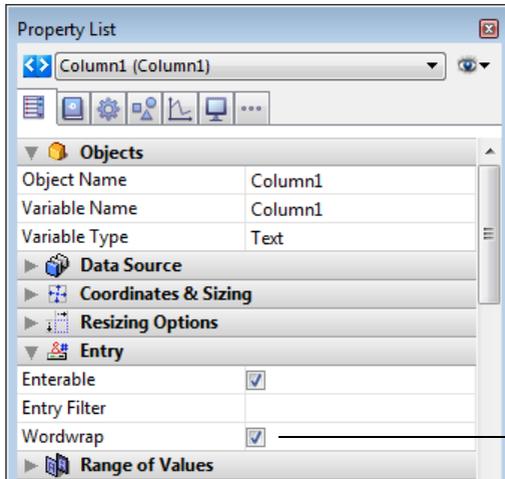
*Note* 階層リストの参照はフォームオブジェクトと関連付けることが出来るようになりました。詳細な情報に関しては、167P “階層リスト” を参照して下さい。

## リストボックス

プログラムを通してできるようになった新機能に加え（ランゲージの章の中の “リストボックス” を参照して下さい）、4D v14 では、リストボックスのスタイル、フォントカラー、そして背景色を列ごとに設定することが出来るようになりました。

## 新しいワードラップ プロパティ

新しいワードラップのプロパティが、テキスト型のリストボックスカラムの " 入力 " テーマ内に追加されました ::



新しいワードラップオプション

このオプションはカラムの中身がカラムの幅を超えた時の表示の仕方を管理します。

このオプションがチェックされていると、テキストがカラムの幅を越えたときに、カラムの高さが許容する範囲内で自動的に次の行へと改行します。

このオプションがチェックされていない場合、長すぎるテキストは全て切り落とされ、省略記号 (...) が表示されます。

以下の例では、左の列ではワードラップのオプションがチェックされていて、右の列ではされていません。

Header1	Header2
The vertical alignment will be applied as long as the full text fits in the cell. Otherwise, the text will be aligned to the top so as the beginning of the text will visible.	The vertical alignment will be ap...
You can make a field invisible in the Application environment and for the plug-ins by selecting the Invisible property for this field.	You can make a field invisible in ...

ワードラップあり

ワードラップなし

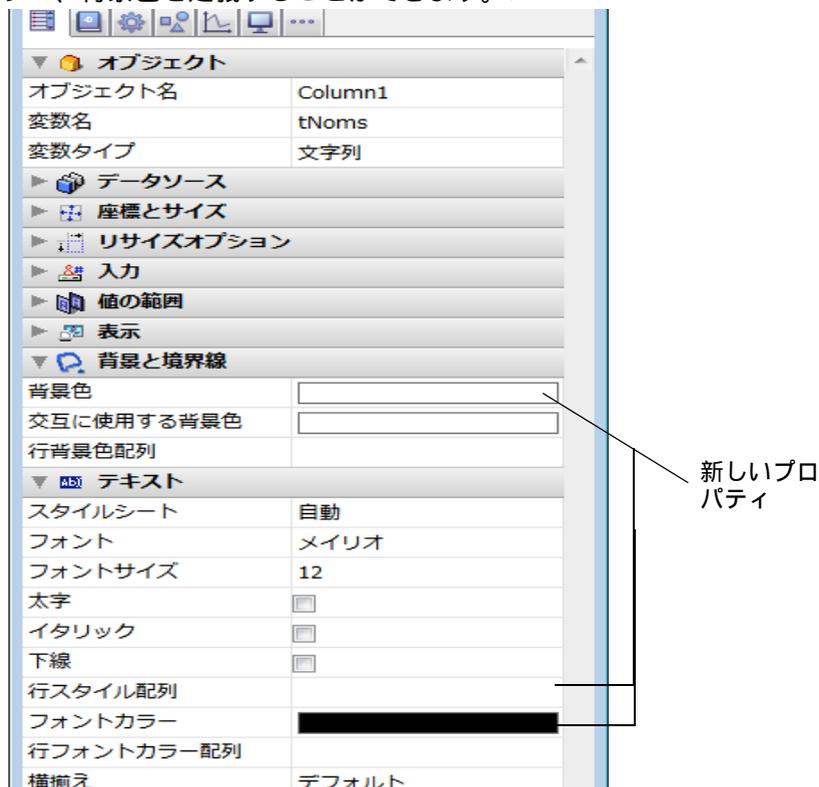
ワードラップのオプションの値に関わらず、行の高さは変化しないことに注意して下さい。改行を含むテキストがカラムの中に全て表示しきれないとき、表示しきれない部分は切り落とされ、省略記号も表示されま

せん。単一の行を表示するリストボックスの場合、テキストの最初の行のみ表示されます。

	Header1	Header2	
ワードラッ プあり	The vertical alignment will be	The vertical alignment will be ap...	ワードラッ プなし
	You can make a field invisible in	You can make a field invisible in ...	

## 新しい列のプロパティ

新しい列のプロパティでは、リストボックス列のスタイル、フォントカラー、背景色を定義することができます。:



- 行背景色配列 (配列型リストボックス) / 背景色 (セクション型リストボックス): 列のそれぞれのセルにカスタムの背景色を指定することができます。指定するには RGB カラー値を使用します。
- 配列型のリストボックスの場合には、倍長整数型配列の名前を入力しなければなりません。この配列のそれぞれの要素は列のそれぞれのセルと対応するので、二つの配列のサイズは同じである必要があります。この背景色を指定するときに "SET RGB COLORS" テーマ内の定数を使用することができます。もし上のレベルで定義されている背景色をそのまま継承したい場合 (67P “スタイルとカラーの継承” を参照して下さい) には、対応する配列の要素に -255 を渡して下さい。

- セレクション型リストボックスの場合には、4D 式か変数 (ただし配列型を除く) を入力しなければなりません。4D 式または変数は、それぞれの表示されているセルごとに評価されます。フォーミュラーエディターを使用して 4D 式を編集することが可能です。そのためにはエリアを選択したときに表示される [...] ボタンをクリックして下さい。  
"SET RGB COLORS" テーマ内の定数を使用することができます。
- **行スタイル配列** (配列型リストボックス) / **スタイル** (セレクション型リストボックス): 列のそれぞれのセルにカスタムのスタイルを指定することが出来ます。
  - 配列型のリストボックスには、倍長整数型配列の名前を入力しなければなりません。この配列のそれぞれの要素は、列のそれぞれのセルと対応するので、二つの配列のサイズは同じである必要があります。配列へは、"Font Styles" テーマ内の定数を使用することができます (メソッドを使用しての入力も可能)。定数同士を足し合わせてスタイルを組み合わせたことも出来ます。もし上のレベルで定義されているスタイルをそのまま継承したい場合 (67P “[スタイルとカラーの継承](#)” を参照して下さい) には、対応する配列の要素に -255 を渡して下さい。
  - セレクション型リストボックスの場合には、4D 式か変数 (ただし配列型を除く) を入力しなければなりません。4D 式または変数は、それぞれの表示されているセルごとに評価されます。フォーミュラーエディターを使用して 4D 式を編集することが可能です。そのためにはエリアを選択したときに表示される [...] ボタンをクリックして下さい。  
"Font Styles" テーマ内の定数を使用することができます。
- **行フォントカラー配列** (配列型リストボックス) / **フォントカラー** (セレクション型リストボックス): 列のそれぞれのセルにカスタムのフォントカラーを指定することが出来ます。指定するには RGB カラー値を使用しなければなりません。
  - 配列型のリストボックスの場合には、倍長整数型配列の名前を入力しなければなりません。この配列のそれぞれの要素は列のそれぞれのセルと対応するので、二つの配列のサイズは同じである必要があります。このフォントカラーを指定するときに "SET RGB COLORS" テーマ内の定数を使用することができます。もし上のレベルで定義されているフォントカラーをそのまま継承したい場合 (67P “[スタイルとカラーの継承](#)” を参照して下さい) には、対応する配列の要素に -255 を渡して下さい。
  - セレクション型リストボックスの場合には、4D 式か変数 (ただし配列型を除く) を入力しなければなりません。4D 式または変数は、それぞれの表示されているセルごとに評価されます。フォーミュラーエディターを使用して 4D 式を編集することが可能です。そのためにはエリ

アを選択したときに表示される [...] ボタンをクリックして下さい。  
"SET RGB COLORS" テーマ内の定数を使用することができます。

## スタイルとカラーに おける優先順位

4D v14 では、リストボックスの背景色、フォントスタイル、リストボックスといったものを設定するための新しい方法があります。列ごとに配列またはメソッドを設定するという方法です (63P “新しい列のプロパティ” を参照して下さい)。これに加え、以前までのバージョンの 4D からある既存の方法も使用することが可能です。それはリストボックスオブジェクトまたは列のプロパティを使用する方法、もしくは、配列またはメソッドをリストボックスに対して使用してセル単位で定義する方法です ( 後者はマルチスタイルテキストに限る )。

同じプロパティに異なる値が複数のレベルにわたって適用された場合、以下の優先順位が適用されます。:

優先度高	セル単位 (マルチスタイル使用時)
	列の配列 / メソッド (v14 新機能)
	リストボックスの配列 / メソッド
	列のプロパティ
優先度低	リストボックスのプロパティ

例として、リストボックスのプロパティにてフォントスタイルを設定し、列のスタイル配列を使用して異なるスタイルを設定した場合、後者の方が有効となります。

- ▶ 以下の様な、グレー / 淡いグレーを交互に繰り返す行の背景色がリストボックスのプロパティで定義されたリストボックスについて考えます。同時に、行の中の少なくともどれか一つの値が負の値である行に関しては背景色がオレンジ色になるような背景色配列が設定されていたとします。:

```
<>_BgndColors{$i}:=0x00FFD0B0 // orange
```

```
<>_BgndColors{$i}:-255 // default value
```

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

次に、負の値が入っているセルを濃いオレンジの背景色にしたい場合を考えます。これをするためには、それぞれの行に対して背景色を適用します。例えば、<>\_BgndColor\_1, <>\_BgndColor\_2 そして <>\_BgndColor\_3 のようにです。これらの配列の値はリストボックスのプロパティや一般的な背景色の設定より優先されます。:

```
<>_BgndColorsCol_3{2}:=0x00FF8000 // dark orange
<>_BgndColorsCol_2{5}:=0x00FF8000
<>_BgndColorsCol_1{9}:=0x00FF8000
```

<> \_BgndColorsCol\_1{16}:=0x00FF8000

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

新しいLISTBOX SET ROW COLOR コマンドと LISTBOX SET ROW FONT STYLE コマンドを使用しても同じ結果を得ることができます。こちらを使った方がコマンドが動的に配列を作成するので、列ごとのスタイル/カラー配列を事前に設定するのをスキップすることが出来るという利点があります。

## スタイルとカラーの継承

それぞれの属性(スタイル、カラー、背景色)について、デフォルトの値を使用した場合、属性の継承が行われるようになっています。:

- セル属性について: 行の値を受け継ぎます
- 行属性について: 列の値を受け継ぎます
- 列属性について: リストボックスの値を受け継ぎます

このように、オブジェクトに高次のレベルの属性の値を継承させたい場合は、定義するコマンドの中に -255(デフォルト値)を渡すか、対応するスタイル/カラー配列の要素の中に直接渡して下さい。

- ▶ 以下の様な、標準のフォントスタイルで行の背景色が交互に変わるリストボックスを考えます。:

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

以下の様な変更を加えます。:

- リストボックスオブジェクトの**行背景色配列**プロパティを使用して、2行目の背景色を赤に変更します
- リストボックスオブジェクトの**行スタイル配列**を使用して、4行目のスタイルをイタリックに変更します。
- 5列目の列オブジェクトの**行スタイル配列**を使用して、5列目の二つの要素を太字に変更します。
- 1、2列目の列オブジェクトの**行背景色配列**を使用して、1、2列目の要素一つずつ、計二つの背景色を濃い青に変更します。:

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	<b>standard</b>	standard
<i>standard</i>	<i>standard</i>	<i>standard</i>	<i>standard</i>	<b>standard</b>	<i>standard</i>
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

リストボックスを元の状態に戻すには、以下の手順で元に戻せます。:

- 1、2列目の背景色配列の要素2に定数 `Listbox_inherited` を渡します。これにより行の赤の背景色を継承します。
- 5列目のスタイル配列の要素3と4に定数 `Listbox_inherited` を渡します。これにより、要素4を除いて標準のスタイルを継承します(要素4はリストボックスのスタイル配列にて指定されたイタリックの属性を継承します)。
- リストボックスのスタイル配列の要素4に定数 `Listbox_inherited` を渡します。これにより、4行目のイタリックのスタイルが除去されます。
- リストボックスの背景色配列の要素2に定数 `Listbox_inherited` を渡します。これにより元の、背景色が交互に変わるリストボックスの状態に戻すことができます。

## 標準アクションのサポート

4D v14 では、ボタンまたはメニューに関連付けられた標準アクションによって "セレクション" 型のリストボックスを管理することができます。例えば、リストボックスの**サブレコード追加**の標準アクションを使ってテーブルに新しいレコードを追加することができます。この新機能に

よってリストボックスに基づいたモダンなリストボックスを簡単に作成できるようになります。

リストボックスでは3つの標準アクションが使えるようになりました：**サブレコード追加**、**サブレコード編集**、そして**サブレコード削除**の3つです。

## サブレコード追加

フォーム内に " セレクション " 型のリストボックスが少なくとも一つあるとき、**サブレコード追加**の標準アクションが関連付けられているボタンまたはメニュー項目が自動的に有効になります。フォームにリストボックスが複数含まれる場合、アクションはフォーカスがあるリスト (またはデフォルトで最前面にあるリスト) に対して適用されます。

---

*Note* v14 から、リストボックスに対しても**タブ有効**のプロパティが使用できるようになりました (70P “[タブ有効](#)” を参照して下さい)。

---

ユーザーがボタンまたはメニュー項目をクリックしたとき、リストボックスに対して定義された詳細フォームの中に新しい空のレコードが表示されます (71P “[詳細フォーム](#)” を参照して下さい)。ユーザーはここに値を入力し、確定するとまた新しい空のレコードが表示されます。これはユーザーがキャンセルボタンを押すまで繰り返されます。リストボックスのデータソースがカレントセレクションの場合、作成されたレコードは全てリストに表示されます。

## サブレコード編集

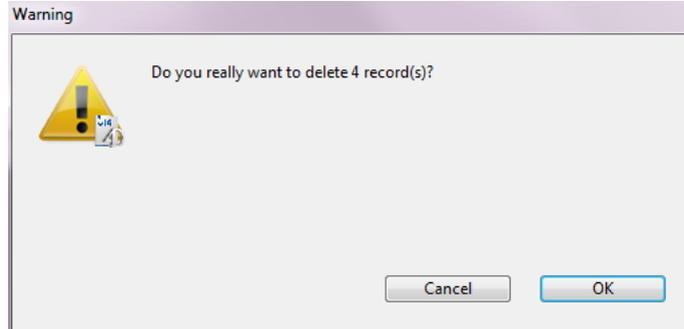
フォーム内に " セレクション " 型のリストボックス内に選択されている行が少なくとも一つあるとき、**サブレコード編集**の標準アクションが関連付けられているボタンまたはメニュー項目が自動的に有効になります。複数の行が選択されたとき、このアクションは最後にセレクションに追加された行に対してのみ適用されます。

ユーザーがボタンまたはメニュー項目をクリックしたとき、リストボックスに対して定義された詳細フォームの中にクリックした行に対応するレコードが表示されます (71P “[詳細フォーム](#)” を参照して下さい)。ユーザーは値を編集でき、確定またはキャンセルするとリストボックスへと戻ります。

## サブレコード削除

フォーム内に " セレクション " 型のリストボックス内に選択されている行が少なくとも一つあるとき、**サブレコード削除**の標準アクションが関連付けられているボタンまたはメニュー項目が自動的に有効になります。複数の行が選択されたとき、このアクションは全てのレコードに対して適用されます。

ユーザーがボタンまたはメニュー項目をクリックすると、削除をするかキャンセルするかを確認するダイアログボックスが表示されます。



### 新しいプロパティ

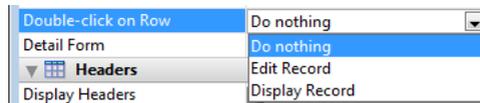
リストボックスにはいくつかの新しいプロパティが追加されています。

### タブ有効

"セレクション"型と"配列"型のリストボックスに対して、**タブ有効**のプロパティ("入力"テーマ)が設定できるようになりました。

### 行をダブルクリック

新しい行を**ダブルクリック**のプロパティは、ユーザーがリストボックスの行をダブルクリックした際に実行されるアクションを設定します(ただし"セレクション"型のリストボックスに限る)。



以下のオプションから選択することができます:

- **何もしない**(デフォルト): 行をダブルクリックしても何のアクションも起こりません。
- **レコード編集**: 行をダブルクリックするとリストボックスで定義された詳細フォームの中に対応するレコードが表示されます(71P "[詳細フォーム](#)" を参照して下さい)。レコードは読み出し - 書き込みモードで開かれるので編集が可能です。
- **レコードを表示**: レコード編集とほぼ同じですが、レコードが読み出し専用モードで開かれるので編集は不可能です。

ここで選択したアクションの内容にかかわらず、ダブルクリックした際には On Double clicked フォームイベントが生成されています。

---

*Note* 空の行へのダブルクリックは無視されます。

---

## 詳細フォーム

新しい**詳細フォーム**のプロパティはリストボックスの個々のレコードを修正または表示するために使用するフォームを指定します(ただし"セレクション"型リストボックスに限る)。

Double-click on Row	Do nothing
Detail Form	Input

定義されたフォームは以下の時に表示されます：

- リストボックスに設定された**サブレコード追加**または**サブレコード編集**の標準アクションが使用されたとき(68P“[標準アクションのサポート](#)”を参照して下さい)。
- 行をダブルクリックプロパティが"レコード編集"または"レコード表示"に設定されている行がダブルクリックされたとき(70P“[行をダブルクリック](#)”を参照して下さい)。

## フォームイベント

"セレクション"型リストボックスに対して、二つのフォームイベントが有効になりました。[On Open Detail](#) と [On Close Detail](#) です。

- [On Open Detail](#): リストボックスに関連付けられた詳細フォームにレコードが表示されようとしているとき(フォームが開かれる前)に生成されます。
- [On Close Detail](#): リストボックスに関連付けられた詳細フォームに表示されたレコードが閉じられようとしているときに生成されます(レコードが編集されたかどうかは関係しません)。

リストボックスに関連付けられた詳細フォームの表示に関する詳細な情報は、[71P“詳細フォーム”](#)を参照して下さい。

## オブジェクトに関連付けられた選択リスト

4D v14 では、選択リストとフォームオブジェクトの関連付けが拡張・改善されています。

- v14 から、ポップアップメニュー / ドロップダウンリストまたはコンボボックスオブジェクトを直接フォーム内のフィールド・変数と関連付けることが出来るようになりました。
- この新しいオプションを使用して、選択リストの項目を参照として保存できるようになりました。
- それに加えて、新コマンドを使用したプログラミングによって選択リスト、指定リスト、除外リストをフォームオブジェクトに関連付けることが出来るようになりました。詳細に関しては [167P“階層リスト”](#)を参照して下さい。

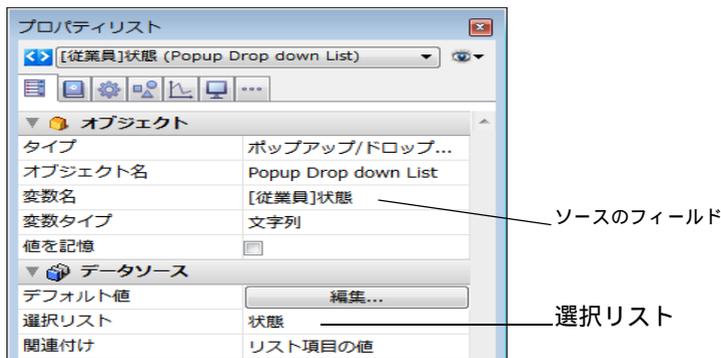
## ポップアップメニューやコンボボックスとフィールド・変数の関連付け

4D v14 では、( 選択リストと関連付けされた ) ポップアップメニュー / ドロップダウンリスト型のオブジェクトやコンボボックスのデータソースとして、フィールドや変数を直接参照できるようになりました。これにより、列挙型のフィールドの管理が簡単になりました。

以前のバージョンの 4D では、列挙されたフィールドまたは変数への入力には " リスト " ダイアログボックスを使用して行われてきました。ポップアップメニュー / ドロップダウンリストやコンボボックスを使用して列挙型のオブジェクトを管理することも可能でしたが、そのためにはオブジェクト間のやりとりを配列を使用してエンコードする必要がありました。

4D v14 では、その手順を使用せずに、ポップアップメニューやコンボボックスを直接オブジェクト関連付けることが可能になりました。例えば、"White"、"Blue"、"Green"、"Red" という値のみを含む "Color" というフィールドがあった場合、これらの値を含むリストを作成し、それを "Color" フィールドを参照するポップアップメニューに関連付けることができます。こうすることによって、あとは自動的に 4D がカレントの値の入力や表示に関して管理してくれます。

ポップアップメニュー / ドロップダウンリストやコンボボックスをフィールドや変数と関連付けるには、オブジェクトの変数名のエリアにフィールドまたは変数の名前を直接入力するだけです。:



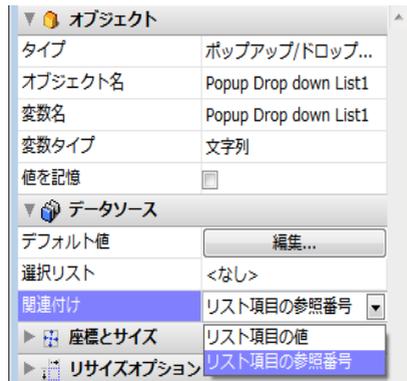
フォームを実行すると、4D が自動的に入力と表示中のポップアップメニュー / コンボボックスの状態を管理してくれます。ユーザーが値を選択

すると、その値はフィールドに保存され、このフィールドの値はフォームが表示されたときにポップアップメニューの中に表示されます。



## 参照の保存

フィールド、変数とポップアップメニュー / ドロップダウンリストの " データソース " の中に、新しい**関連付け**というオプションがあります。:



このオプションは、選択リストに関連付けされたフィールドと変数において、フィールドに保存する中身の型を指定します。:

- **リスト項目の値** (デフォルトのオプション): ユーザーによって選択された項目の値が直接保存されます。例えば、ユーザーが "Blue" という値を選択した場合、この値がフィールドに保存されます。この挙動は旧バージョンの 4D の動作と同じです。
- **リスト項目の参照番号**: 選択リスト項目の参照がオブジェクトに保存されます。この参照番号とは APPEND TO LIST または SET LIST ITEM コマンドの *itemRef* 引数、またはリストエディターを通してそれぞれの項目と関連付けされた数値です。:



このオプションにより、メモリーを節約することが出来ます。フィールドに数値を保存するのは文字列を保存するより容量が軽いからです。またこれによりアプリケーションの翻訳が簡単になります。同じ項目の参

照値を持つ、異なる言語で書かれた複数のリストを用意しておいて、アプリケーションの言語に応じたリストをロードするだけです。

リスト項目の参照番号の使用の際には、以下の点に注意する必要があります。:

- 参照を保存するためには、フィールドまたは変数は、数値型である必要があります (リスト内に表示されている値の型とは関係ありません)。
- リストの項目には有効かつ固有の参照が関連付けられている必要があります。
- ポップアップメニューにおいてこのオプションを使用した場合、ポップアップメニューは配列ではなく、フィールドまたは変数と関連付けられている必要があります。(72P “ポップアップメニューやコンボボックスとフィールド・変数の関連付け” を参照して下さい)。
- このオプションは、ストラクチャの中で定義された選択リストにも有効です。この場合は、列挙されたフィールドが使用されているそれぞれのフォームにおいてこのオプションを選択してください。

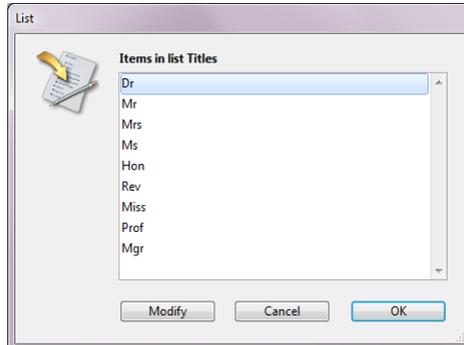
## 例

人を区別しやすくするために "Title" というフィールドを使用する場合を考えます。Mr, Ms だけでなく、Dr, Mgr, Hon, なども含めます。このためには、まず "Title" という名前の倍長整数型のフィールドを作成します。次に考えられ得る項目を全て含んだ "Title" という名の選択リストを定義し、それをフィールドと関連付けます。

入力フォームには、フォームの仕組みを説明するために "Title" のフィールドを二回表示するとします。一度目はポップアップメニューとして、もう一度は入力エリアとして。どちらのオブジェクトも同じ選択リストに関連付けられており、データは参照として保存されています。:

入力中、ポップアップメニューの値を選択すると、両方のオブジェクトに正しく表示されます。:

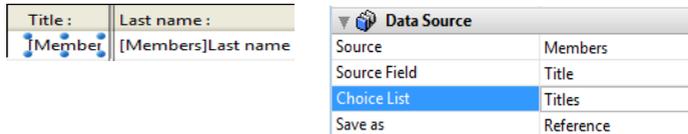
*Note* このフォーム内において、入力エリアにフォーカスが移ると、選択リストウィンドウが表示されます。:



このウィンドウを表示させないためには、入力と表示にポップアップメニューを使用して下さい。

同じやり方で、出力フォームを設定することもできます。"Title" フィールド内の関連付けのオプションでリスト項目の参照番号を選択して下さい。

:



実行すると、値は正常に表示されます。:

Title :	Last name :
Prof	Durant
Mr	Smith
Mr	Anderson
Mrs	Peterson
Ms	Harper
Rev	Trace
Ms	Johnson
Mrs	Stevenson

## テキストの回転

4D v14 では、フォーム内のテキストエリアを回転させることができます。  
:



テキストの回転機能は、フォームエディターから設定するか(恒久的なプロパティ)、新しい **OBJECT SET TEXT ORIENTATION** コマンドを使用することによって設定することができます(カレントのプロセスの間有効)。

### 回転可能なオブジェクト

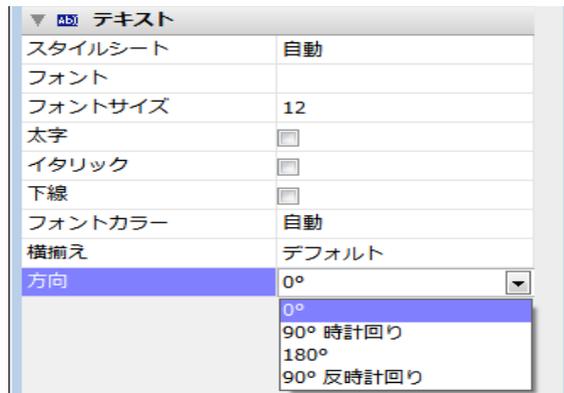
テキストの回転は、フォーム内の**入力不可**のテキストエリアにおいてのみ可能です。例えば、:

- 静的なテキスト
- 入力不可のテキスト変数またはフィールド - 厳密にいうと、テキストベースで、文字列、日付、時間、数字、複数行、またはマルチスタイル型オブジェクトを含む、オブジェクトをさします。

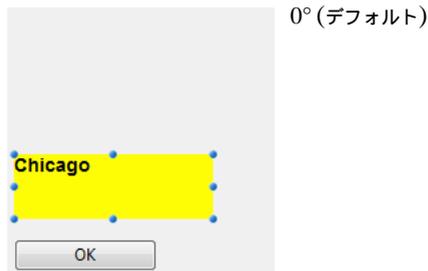
他の型のオブジェクト(ボタン、入力エリア、リスト、ラジオボタン等)は回転させることができません。

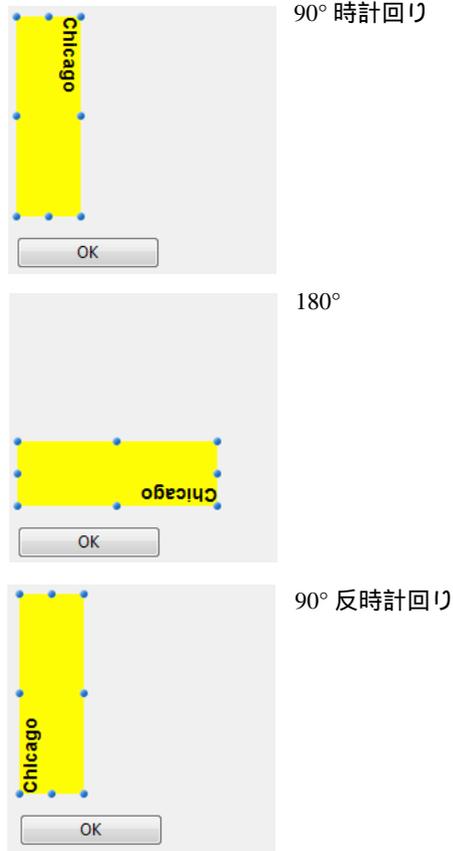
## 回転の設定

新しい " 方向 " のプロパティは、回転がサポートされているオブジェクトのプロパティリストのテキストのテーマ内にあります。テキストエリアは、90° 単位で回転させることができます。:



それぞれの回転角度を適用するとき、オブジェクトの左下の角は固定されたままで回転していきます。:





**フォームエディターまたはランゲージコマンドを使用した回転**

フォームエディター内で回転を適用したとき、テキストを含んでいるオブジェクトはそのテキストの回転に従って回転します。

ただし、**OBJECT SET TEXT ORIENTATION** コマンドを使用してテキストを回転させた場合にはこの原理はあてはまりません。このコマンドが実行されたとき、テキストは回転しますが、テキストを含むオブジェクトは回転しません。例えば、90° Left (90° 左回転) をフォームエディターの "Chicago" というテキストに対して **OBJECT SET TEXT ORIENTATION** コマンドを使用したときには、プロパティリストで設定した場合と比較して、

フォームを実行した際に異なる結果が得られます。

:

90° 反時計回り (プロパティ)



90° 反時計回り (ランゲージ)

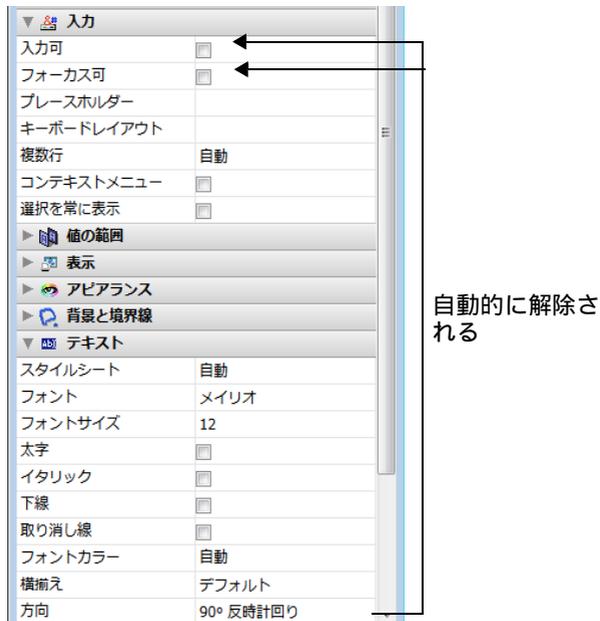


詳細な情報については、[OBJECT SET TEXT ORIENTATION](#) コマンドの詳細を参照して下さい。

## フィールドと変数の回転

4D v14 では、入力不可かつフォーカス不可のテキストオブジェクトに限り、回転させることができます。

フィールドまたは変数型のオブジェクトに対して、方向のプロパティにて 0° 以外のオプションを選んだ場合、入力可とフォーカス可のプロパティは ( 選択されていた場合 ) 自動的に解除されます。:



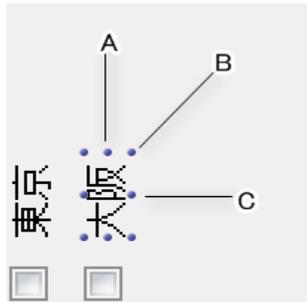
その際、このオブジェクトは自動的に入力順から除外され、背景が透明になります。

また逆に、回転しているオブジェクトに対して入力可もしくはフォーカス可のプロパティにチェックをした場合、方向のプロパティは自動的に  $0^\circ$  へとリセットされます。

## 回転しているオブジェクトの変更

テキストが変更された後でも、そのセレクションハンドルや、**OBJECT SET COORDINATES** といったランゲージコマンド、またはプロパティなどを通じて、エリアのサイズや位置などを変更することは可能です。

テキストエリアの高さと幅は、回転の方向に依らないという点に注意して下さい。:



- オブジェクトが A 方向にリサイズされる時、変更されるのは幅（縦方向）です。
- オブジェクトが C 方向にリサイズされる時、変更されるのは高さ（横方向）です。
- オブジェクトが B 方向にリサイズされる時、幅と高さ両方が同時に変更されます。

エリアの中身は、フォームエディターを使用して修正することもできます。エディットモードに切り替わる際、テキストはデフォルトの方向へと戻ります。:



## ウィンドウの配置の記憶

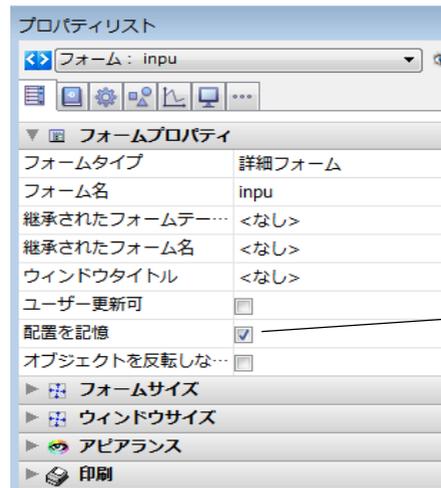
4D v14 では、ウィンドウ独自の表示方法 (" 配置 ") を、ウィンドウを閉じる際に自動的に記憶するという新機能が追加されています。これにより、ユーザーは作業を終了したときと同じ作業環境にいつでも戻って来られるようになります。

この機能にて記録されるのは、ウィンドウの座標とそのウィンドウ内に含まれるオブジェクトの位置、それからいくつかのオブジェクトのカレントの状態、すなわちタブでどれが選択されているか、などです。

*Note* この新機能は、フォームが閉じたときと同じサイズで再度開かれたときのみ有効です。結果的に、この機能は主に Open form window コマンドに \* 演算子を渡してウィンドウが開かれることを想定しています。

### 「配置を記憶」のオプション

新しい **配置を記憶** のオプションは、フォームのプロパティ内にあります。



オブジェクトの配置を保存するためのオプション

このオプションがチェックされているとき、セッション中にどのように変更したかに関わらず、以下の複数のフォームパラメーターが4Dによって自動的に保存されます。:

- カレントのページ
- それぞれのフォームの配置、大きさ、表示状態 ( リストボックス列のサイズと表示状態も含まれます )。

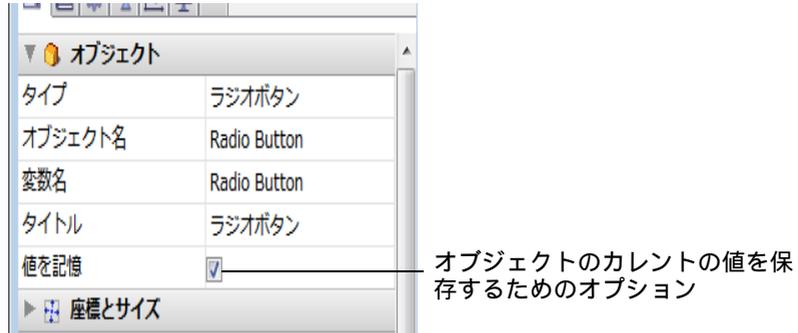
*Note* このオプションは、OBJECT DUPLICATE コマンドを使用して作成されたオブジェクトに対しては無効です。このコマンドを使用しても使用環境

を復元させるためには、デベロッパがオブジェクトの作成・定義・配置の手順をその都度行わなければなりません。

このオプションが選択されているとき、一部のオブジェクトに置いては**値を記憶**のオプションが選択可能になります（詳細は次の項目を参照して下さい）。

### 「値を記憶」のオプション

新しい**値を記憶**のオプションはプロパティリストの"オブジェクト"のテーマ内にあります。:



このオプションは以下の場合に限り有効です。:

- フォームの**配置を記憶**のオプションがチェックされていること
- 選択されたオブジェクトがフォームの全体的な配置に関するオブジェクトであること。例えばチェックボックスは、ウィンドウ内にて追加のエリアを表示したり隠したりするのにこの値を使用することができるので、チェックボックスにはこのオプションが存在します。

値を保存できるオブジェクトと保存できる値は、以下の表の通りです。:

オブジェクト	保存される値
ラジオボタン	関連付けられた変数の値 (1、0、ボタンにおいては True または False など。型による)
3D ラジオボタン	
チェックボックス	
3D チェックボックス	
リストボックスヘッダー	sort 変数の値 (0、1、-1)
タブ	選択されているタブの番号
ポップアップ/ドロップダウンリスト	選択されている項目の位置
ピクチャーポップアップメニュー	

## 情報の保存と使用に関して

4D では、Open form window コマンド (フォーム名:\*) を使用して作成されたウィンドウを閉じた際にそのウィンドウの座標を保存します。Windows 環境下では最大化した状態も保存されます。

これらの情報と、任意で保存できる情報 (配置や値など) は、ウィンドウが閉じられたときにマシンのカレントのユーザーフォルダに *json* フォーマットで保存されます。これにより、たとえ " デフォルトユーザー " アカウントを使用したときでも個別のマシンを使って接続するそれぞれのユーザーごとに、その環境を保存することが出来ます。

この情報は、フォームが閉じられたときと同じ寸法にて再度開かれたときに使用され、閉じられたときに再度保存されます。この原理には、Open form window (\*) コマンドが使用されたか、デベロッパが座標を保存するための独自のシステムを設定されていることを前提としています。

保存された情報は以下の順序で復元され、適用されます。:

- ウィンドウのサイズとポジションは Open form window コマンドが実行されたときに復元されます。
- 変数の値はフォームが On Load イベントを呼び出す前にロードされたときに復元されます。
- カレントのページは On Load イベントを呼び出す前に復元されます。
- それぞれのオブジェクトの位置、サイズ、そして表示状態は On Load イベントのすぐ後に復元されます。

サブフォームオブジェクトのプロパティは、同じ順番に従って自動的に保存・適用されます。

---

**" 配置を記憶 "、" 値を記憶 "** を使用して保存された情報は、フォームのオブジェクトがデザインモードで変更 (リサイズ、移動、追加、削除または改名) されるたび元の状態へとリセットされてしまいます。そのため、この機能を使用してユーザー設定などの恒久的な値を保存しないことを強く推奨します。

---

## 例

### スプリッターの記憶

フォームのスプリッターの相対位置を保存したい場合を考えます。この場合、ただ単に**配置を保存**のオプションをチェックするだけです。フォームが開いたとき、以下の様になるはずです。:



ユーザーがウィンドウをリサイズしたりスプリッターを動かしたりしたとします。オブジェクトはそれぞれのプロパティに応じてリサイズされます。ユーザーがウィンドウを閉じた後、再びフォームが開いたとき、オブジェクトは最新の状態を復元します。:



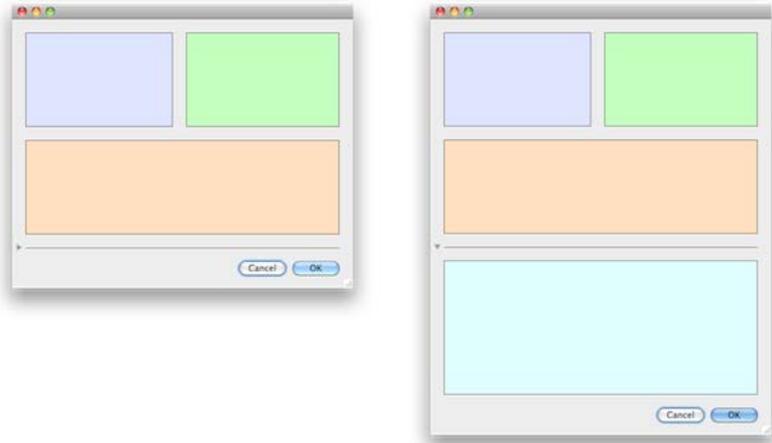
### 折り畳みエリアの記憶

フォーム内において、3D チェックボックスによって管理される、一つ以上の折り畳みエリアがある場合、そのチェックボックスはエリアが折りたたまれている場合は右向きの三角形を表示し、エリアが展開されている場合には下向きの三角形を表示します。

こういったエリアを設定するにはいくつかの方法（オブジェクトの移動または表示状態の変更、異なるフォームページの使用等）がありますが、どのケースにおいても、ウィンドウのサイズが変更になることがあります。

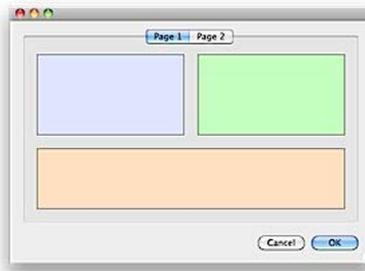
異なるセッション間で折り畳みエリアの状態を保存したい場合、以下の様にしてください。:

- フォームの**配置を保存**のオプションにチェックをし、オブジェクトの  
カレントページ、位置、表示状態が保存されるようにしてください。
- 3D チェックボックスの**値を保存**のオプションにチェックをし、関連付  
けられた変数の値（展開か折り畳みを表す 0 または 1）が保存されるよ  
うにしてください。

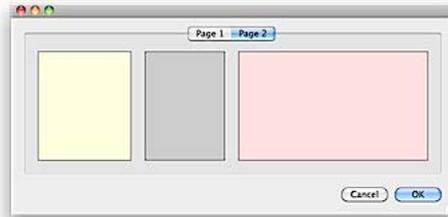


## タブの記憶

フォーム内にて、"Goto Page" の標準アクションを持ったタブを設定した  
とします。:



この場合、配置を正しく保存させるためには、フォーム内の**配置を保存**にチェックをし、タブオブジェクトの**値を保存**のオプションをチェックします。:



## スペルチェック

4D v14 では、コーディアル辞書はサポートされなくなりました (20P “[削除された機能](#)” を参照して下さい)。OS X 環境下では、システムのスペルチェッカーがデフォルトで使用されます。これに加えて、チェック処理と修正処理が改善されています。

### 辞書の選択

- 4D v14 では、スペルチェックの際に使用される辞書は以下の通りです。:
- Windows と OS X、両方の環境下では、ハンスペル辞書が使用できます (旧バージョンの 4D でも使用可能でした)。
  - OS X 環境下では、システムのスペルチェッカーも使用できます。

4D v14 は、デフォルトとして、OS X 環境下ではシステムのスペルチェッカーを使用します。SET DATABASE PARAMETER コマンドを使用して、ハンスペルのスペルチェックを使用することもできます ([SET DATABASE PARAMETER](#), [Get database parameter](#) を参照して下さい)。

なお、OS X のシステムのスペルチェッカーにも新機能が追加されています (以下の項目を参照して下さい)。

### チェック処理

4D v14 では、スペルチェックはテキストエリア内にて常時行われ、エラーはテキストの下に赤の破線が直接引かれて表示されます。:

The development workshop has also been enhanced by simplifying the procedure for updating versions, new query dialog boxes, extended debugging help, and the new On Host Database Event database method

エラー箇所

以下の修正すべき内容に応じて、異なる色の破線が表示されます。:

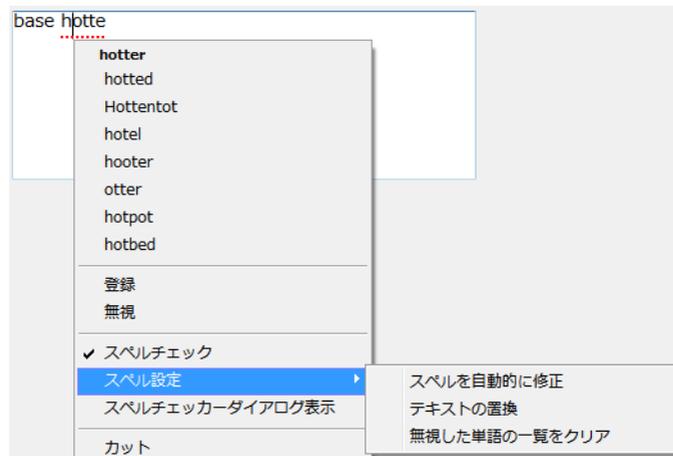
- **赤**の破線は綴りの間違いを表示します。

- 緑の破線は文法の間違いを表示します(OS Xのシステムのスペルチェッカーのみ)。
- 青の破線は置き換える単語を表示します(置き換えテキストを表示のオプションがチェックされている必要があります。以下を参照して下さい。)

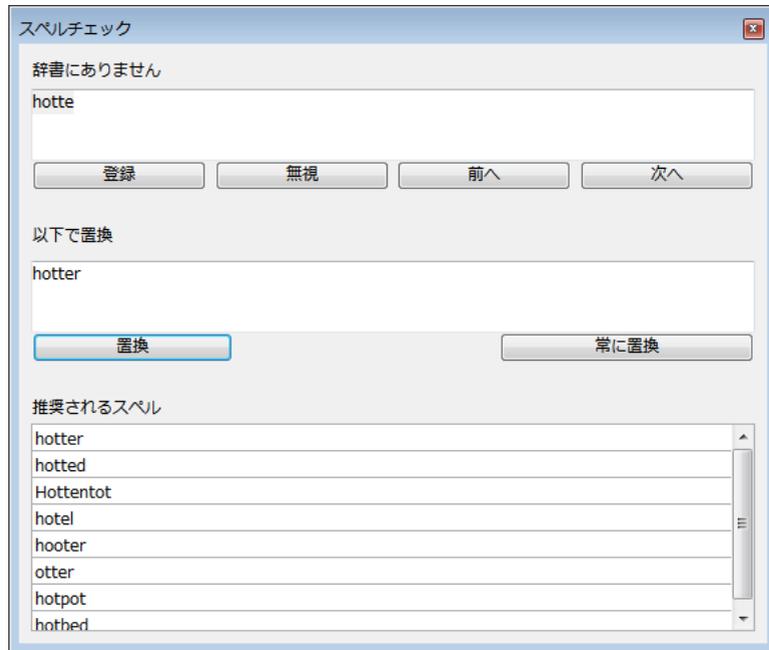
ユーザーをそれらを見て、コンテキストメニュー(下線の引かれた単語を右クリックで表示)またはスペルチェッカーダイアログを使用して修正することが出来ます。

### コンテキストメニューまたはスペルチェッカーダイアログ

スペルチェックのオプションを表示するためには、下線の引かれた単語を右クリックします。コンテキストメニューが表示され、その中にスペルチェッカーのコマンドがあります。:



スペルチェッカーダイアログを表示のオプションを選択すると、修正した値を入力するダイアログが表示されます。:



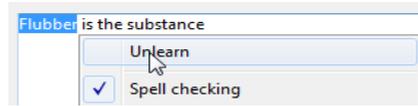
このダイアログボックスは、セッションの間、ユーザーが閉じるまでアプリケーション内の全てのウィンドウに表示されるフローティングするウィンドウです。

### スペルチェッカー機能

修正候補の提示をしてくれるだけではなく、4D v14 のスペルチェッカーでは、以下のオプションと機能が用意されています ( コンテキストメニュー内またはスペルチェッカーダイアログボックス内 )。:

- **登録**: この未知の単語を辞書に登録し、今後スペルチェッカーによって検知されることはなくなります。
  - OS X 環境下では、学習した単語は `/Users/[UserName]/Library/Spelling` の場所に恒久的に保存されています (ここで学習した単語は、システムスペルチェッカーを使用する他のアプリケーションでも表示されなくなります)。
  - ハンスペル辞書では、学習した単語はカレントのアプリケーションデータのユーザーシステムディレクトリ内のカスタムの辞書内に保存され、メインの辞書がロードされる際に同時に呼び出されます (旧バージョンと同様の動作です)。
- **学習を取り消し (コンテキストメニュー)**: このオプションは、以前に学習した単語を選択したときに表示されます。選択すると、この単語を

学習した単語のリストから除外し、その単語は再びスペルチェッカーによってエラーとして表示されるようになります。

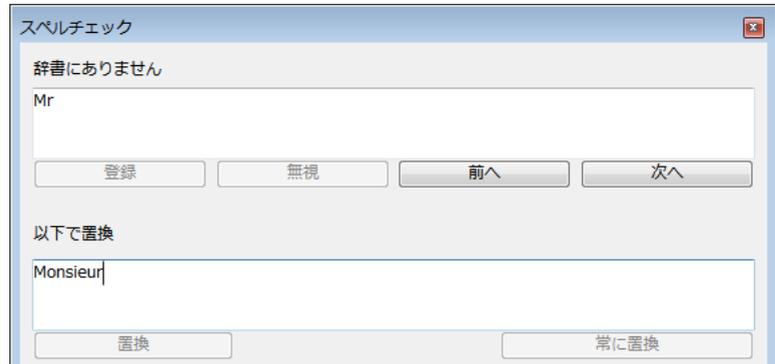


- **無視**: この未知の単語はそのままの状態の下線が表示されないようになります。ただし、その後再び検知されたときには下線が表示されます。スペルチェッカーは、ドキュメントごとに無視する単語の表を保存しています。**無視した単語の一覧を消去**のオプションを選択することで一覧表を削除することができます(以下を参照して下さい)。
- **前へ/次へ**(ダイアログボックス): この未知の単語は何も変更されず、下線を引かれたままで、スペルチェッカーはテキスト内の次の未知の単語を検索します。
- **スペルチェック**(コンテキストメニュー): カレントのプロセスにおいて、エリア内のスペルチェックを全体的に有効化または無効化します。
- **置換**(ダイアログボックス): テキスト内で選択された単語が最初の入力フィールドにある単語と一致する場合、二番目のフィールド内の単語で置換され、同時にスペルチェッカーは次のエラーを検索します。
- **常に置換**(ダイアログボックス): 置換と同じですが、その置換は記憶されます(以下の**テキストの置換**を参照して下さい)。

スペルチェックのサブメニューでは、以下の様な機能が使用可能です。:

- **スペルを自動的に修正**: カレントのプロセス内において、自動修正モードを有効化もしくは無効化します。このモードが有効化されている場合においては、未知の単語は最も近い既知の単語で自動的に置き換えられます(ただしあまりに曖昧な場合は行われません)。修正は、入力の際に行われます。  
デフォルトでは、**スペルを自動的に修正**は無効になっています。
- **テキストの置換**: テキストの置換を有効化または無効化します。この機能は、ある単語を他の単語で置き換えるというものです。例えば、"Mr." を "Mister" で置き換える、といったようなことができます。  
"置き換えられる単語/以下の単語で置き換え"の単語のペアを作成するためには、スペルチェッカーダイアログを使用する必要があります。"辞書にない単語" エリアに置き換えたい単語を入力し、それを置き換える単語を " で置き換え " エリアに入力して下さい。その後**常に置換**

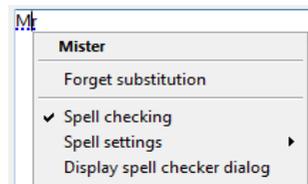
ボタンをクリックします。:



置換はアプリケーションの中全体において実行されます。ハンスペル辞書においては、スペルチェッカーがアプリケーション内の置き換えを記録する表を、アプリケーションのユーザーシステムディレクトリ内に保存します。OS X のシステムスペルチェッカーの場合、この表はシステムの置換と統合されます (システム環境設定言語とテキスト内の " 記号とテキストの置換を使用 " オプション)。

置き換えのプロセスを使用するためには、**スペル設定内のテキストの置き換えを表示**のオプションがチェックされている必要があります (以下を参照して下さい)。

- **テキストの置き換えを表示** (このオプションは、**テキストの置換**のオプションがチェックされている場合にのみ表示されます。): このオプションがチェックされているとき、スペルチェッカーは置き換える必要があり得る単語について青の下線を引きます。ユーザーはその単語を右クリックすることによって、置き換える値を選択することが出来ます。

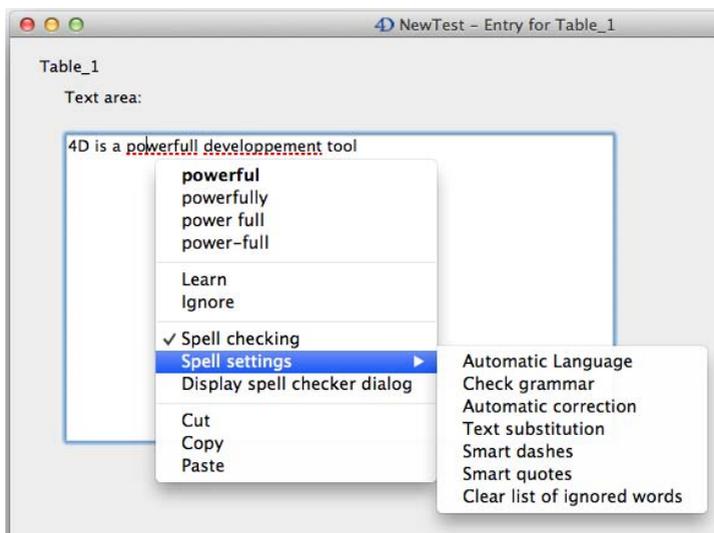


このオプションがチェックされていない場合、置換は自動的に行われます。

- **無視した単語の一覧をクリア**: ドキュメントの中で無視するよう選択された単語の一覧を消去します。

## OS X のシステムのスペルチェッカーにおけるその他のオプション

OS X のシステムのスペルチェッカーには、他にもいくつかの修正のためのオプションが存在します。:



- **言語ごとに自動**: 文章の内容をもとにして、文章の言語と使用すべき辞書を自動的に検知します。  
デフォルトでは、ハンスペルのスペルチェッカーと同じように 4D アプリケーションの言語を使用します。
- **文法をチェック**: 文章の文法のチェックを有効化します。
- **スマートダッシュ記号 と スマート引用符**:
- **スマートダッシュ記号**: 入力中、ハイフン二つ (--) をダッシュ (—) で置き換えます。
- **スマート引用符**: 引用符を、カレントの言語に合わせたスマート引用符へと置き換えます。

## Web エリア

4D と Web エリアの相互作用は、v14 にてさらに拡張されました。:

- 4D メソッドが、\$4D オブジェクト (と統合 Web キット) を使用することによって Web エリアから使用することが出来るようになりました。
- コードデバッグのためにインスペクタにアクセスすることが可能になりました。
- カスタムのドラッグ&ドロップのサポート

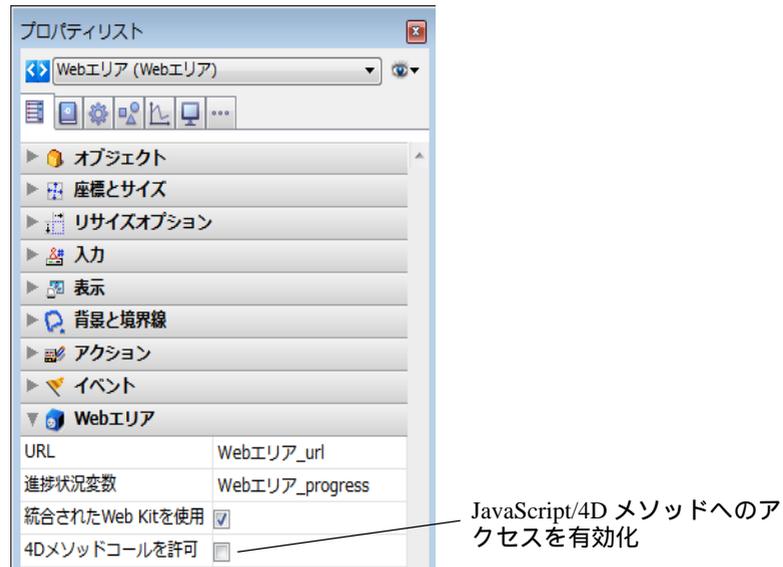
## 4D メソッドへのアクセス

4D v14 では、Web エリアの中で実行された JavaScript から 4D メソッドを呼び出し、戻り値を得ることが出来るようになりました。

**重要:** この新機能は Web エリアがレンダリングエンジンとして統合 Web Kit を使用している場合に限り、使用可能です。

### Web エリアの設定

4D メソッドを Web エリアから呼び出せるようにするためには、エリアのプロパティリスト内の、**4D メソッドコールを許可**のオプションにチェックをする必要があります。:



*Note* このオプションは、**統合された Web Kit を使用**のオプションにチェックをしている場合のみ有効です。

このプロパティがチェックされている場合、特別な JavaScript オブジェクト (\$4d) が Web エリア内に表示され、これを使用して 4D プロジェクトメソッドの呼び出しを管理することが出来るようになります。

### \$4D オブジェクトの使用

4D メソッドに**アクセス**のオプションにチェックがされている場合、4D の統合 Web Kit は、"." オブジェクト記法を使用することによって 4D プロジェクトメソッドと使用できる \$4d という JavaScript オブジェクトをエリアに提供します。

例えば、*HelloWorld* という 4D メソッドを呼び出す場合には、以下の宣言を実行するだけです。: `$4d>HelloWorld()`;

*Note* JavaScript は大文字小文字を区別するので、この場合、オブジェクトの名前は \$4d (d は小文字) であることに注意して下さい。

4D メソッドへの呼び出しのシンタックスは以下のようになります。:

```
$4d.4DMethodName(param1,paramN,function(result,error){})
```

- *param1...paramN*: 4D メソッドに対して必要なだけ引数を渡すことが出来ます。これらの引数は、JavaScript にサポートされている型であればどんな方でも渡す事が出来ます (文字列、数値、配列、オブジェクト)。
- *function(result,error)*: 最後の引数として渡される関数。この "コールバック" 関数は、4D メソッドが実行を終えると同時に呼び出されます。この関数はふたつの引数を受け取ります。:
  - *result*: 4D メソッドの実行の戻り値です。"\$0" という 4D 式の中に返されます。戻り値は JavaScript でサポートされている型 (文字列、数値、配列、オブジェクト) のどれかになります。新しい **C\_OBJECT** コマンドを使用して、オブジェクトを返すこともできます。

*Note* デフォルトとして、4D は UTF-8 文字コードで動作しています。拡張された文字 (アクセントが付いた文字など) を含むテキストを返す場合には、Web エリアで表示されるページの文字コードが UTF-8 に宣言されていることを確認して下さい。文字コードが UTF-8 でない場合、文字が正しく表示されない可能性ががあります。この場合、以下の 1 行を HTML ページに追加して文字コードを宣言して下さい。:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

- *error* (任意): エラースタックが格納された、オブジェクトの配列です。この配列は、4D 側で (ON ERR CALL を使用して) エラーを管理しなかった場合にのみ作成されます。それぞれのオブジェクトは三つのプロパティから構成されます。:
  - *message* (テキスト): エラーの詳細
  - *errorCode* (数字): エラーコード
  - *componentSignature* (テキスト): エラーの発生場所を示す内部コンポーネントの識別コード (4 文字)
 例: "dbmg" はデータベースエンジンをさします。

## 例 1

*today* という名の 4D プロジェクトメソッドがあり、そのメソッドは引数を受け付けず、カレントの日付を文字列として返す場合について考えてみます。

- *today* メソッドの 4D コードは以下のようになります。:

```
C_TEXT ($0)
```

```
$0:=String (Current date;System date long)
```

Web エリアでは、4D メソッドは以下のシンタックスで呼び出し可能です。

```
$4d.today()
```

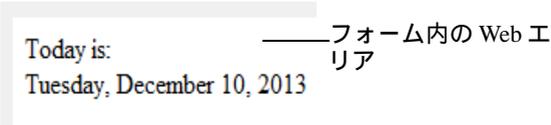
この 4D メソッドは引数を何も受け取りませんが、\$0 の値を、4D によって呼び出されたコールバック関数へとメソッドの実行後に返します。

Web エリアによってロードされた HTML ページ内に日付を表示したい場合を考えます。

- HTML ページのコードは以下の様になります。:

```
<html >
<head>
<meta http-equiv="Content-Type" content="text/html ;
charset=UTF-8" />
<script type="text/javascript">

$4d.today(function(dollarZero)
{
  var curDate = dollarZero;
  document.getElementById("madi v").innerHTML=curDate;
});
</script>
</head>
<body>Today is: <div id="madi v"></div>
</body>
</html >
```



Today is: \_\_\_\_\_ フォーム内の Web エリア  
Tuesday, December 10, 2013

## 例 2

*calcSum* という 4D プロジェクトメソッドがあり、そのメソッドが (\$1...\$n) という引数を受け取り、その合計を \$0 に返すという場合について考えます。:

- *calcSum* メソッドの 4D コードは以下の様になります。:

```
C_REAL ($1) // n 個の実数型の引数を受け取ります。
```

```
C_REAL ($0) // 実数の値を返します。
```

```
C_LONGINT ($i;$n)
```

```
$n:=Count parameters
```

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

```
  $0:=$0+${$i}
```

```
End for
```

- Web エリア内で実行される JavaScript コードは以下のようになります。:

```
$4d. calcSum(33, 45, 75, 102.5, 7, function(dollarZero)
{
    var result = dollarZero // result is 262.5
});
```

### 例 3

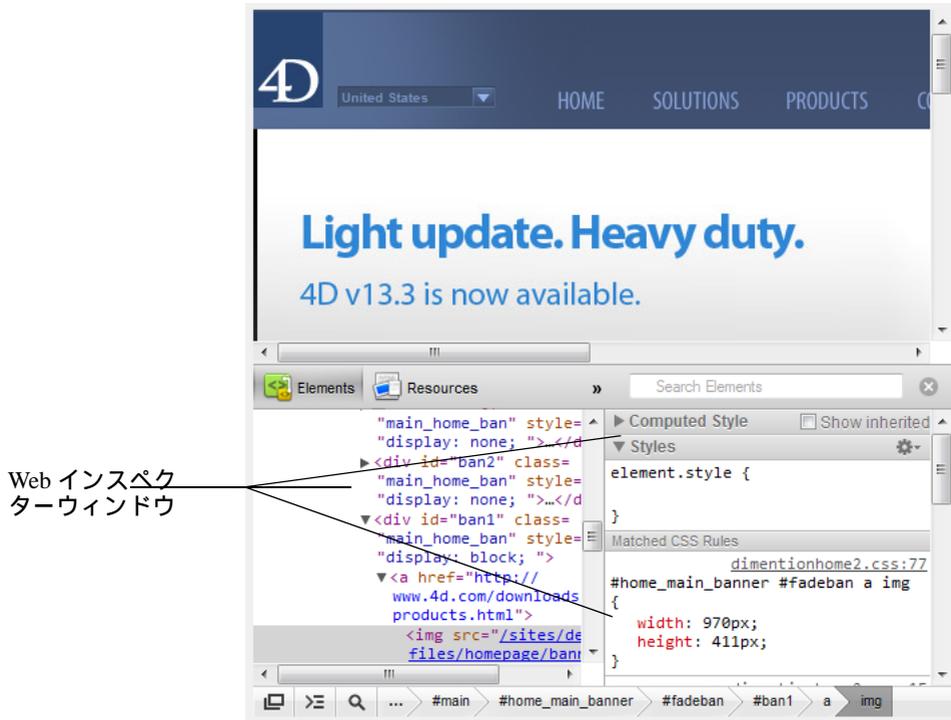
ここでは、例 2 で登場した *calcSum* メソッドに、エラーハンドラーを追加したものを使用する場合を考えます。:

```
$4d. calcSum("alpha", 45, 75, 102.5, 7, function(result, error)
{
    if (error != null)
        alert( "got error: " + error[0].message);
    else
        alert( "got result: " + result);
});
```

### Web インспекター へのアクセス

4D v14 では、フォームの Web エリア内で Web インспекターを見たり使用したりすることが出来るようになりました。Web インспекターは統合された Web Kit デバッガーであり、Web ページの情報の、コードとフ

ローを解析します。この Web インスペクターは Chrome のブラウザに含まれているものと基本的に同じものです。



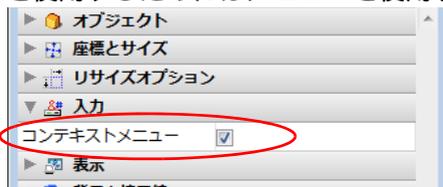
### Web インスペクターの表示

Web エリア内で Web インスペクターを表示させるためには、以下の手順に従ってください。:

- Web エリアの、統合された Web Kit のオプションを選択して下さい。(Web インスペクターを使用するためにはこのオプションを選択する必要があります)。



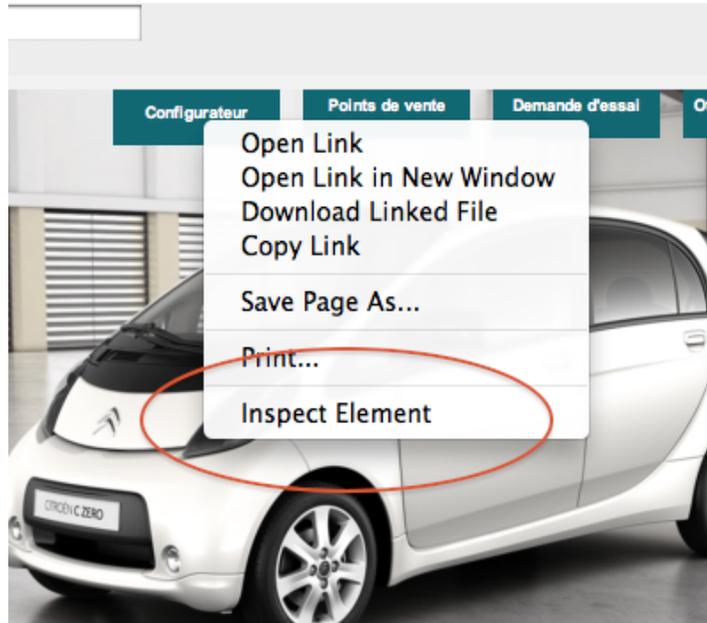
- エリア内のコンテキストメニューを有効化して下さい(インスペクターを使用するためにはメニューを使用する必要があります)。



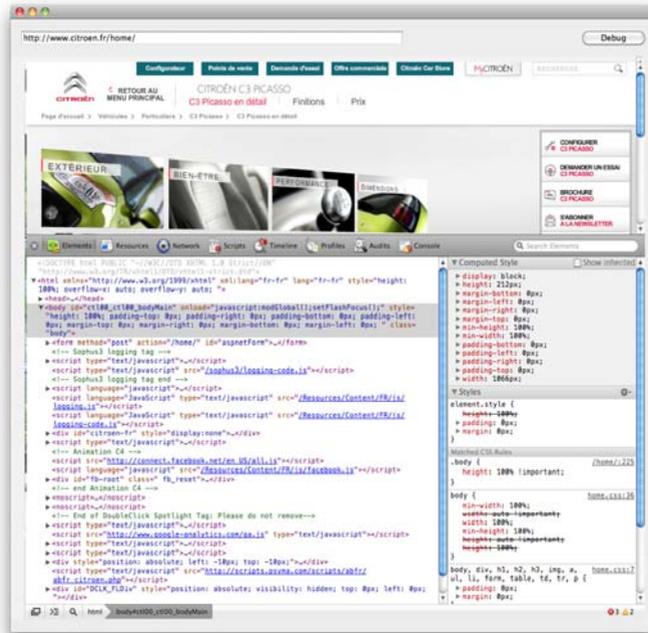
- 以下の宣言を Web エリア内で行い、インスペクターの使用を有効化して下さい。 :  
**WA SET PREFERENCE(\*;"WA";WA enable Web inspector;True)**  
詳細な情報に関しては、301P “WA SET PREFERENCE” を参照して下さい。

## Web インスペクターの使用

上記の手順を踏んで設定を完了すると、エリア内でコンテキストメニューを開くと**要素の詳細を表示**という新しいオプションが追加されているはず。 :



このオプションを選択すると、Web エリアは二つに分割され、Web インスペクターは下側のパネル内に表示されます。:



また、インスペクターは別のウィンドとして表示して使用することもできます。エリアの左下にある。"Undock in separate windows" ボタンをクリックして下さい。:



Web インスペクターは、Web Kit レンダリングエンジンに含まれています。このデバッガーの機能の詳細に関しては、以下のアドレスにて Google より提供されているドキュメントを参照して下さい。:

<https://developers.google.com/chrome-developer-tools/docs/scripts>

# 4

# SQL

この章では 4D v14 の SQL エンジンの新機能や変更点について説明していきます。

## SQL views のサポート

4D v14 の統合 SQL エンジンは、標準の SQL views をサポートするようになりました。ビューとは複数の異なるデータベースのテーブルからのデータを受け取れる仮想テーブルです。ビューは定義されると SELECT 宣言の中で実際のテーブルと同じように使用することができます。

ビューの中のデータは SELECT コマンドに基づいた定義クエリによって定義されます。定義クエリの中で使用される実際のテーブルは「ソーステーブル」と呼ばれます。SQL ビューには標準のテーブルと同じように列と行がありますが、実際に存在しているわけではありません。処理され、メモリーに格納された結果を表示しているにすぎません。データベースに実際に保存されているのはビューの定義だけです。

4D v14 ではビューを管理するための新しい SQL コマンドが二つ用意されています。CREATE VIEW と DROP VIEW です。

### CREATE VIEW

```
CREATE [OR REPLACE] VIEW [schema_name.]view_name [(column_list)]
AS select_statement[:]
```

CREATE VIEW コマンドは、column\_list 引数で定義された列を内包する、view\_name (標準の sql\_name) という名前を持った SQL ビューを作成します。この列が関数であるか、または演算操作によって作成されたものである場合は列の名前を指定する必要があります。異なる列に同じ名前が付くことを避けたい場合 (JOIN 操作の途中など) や、元になった列と異なる名前を使用したい場合なども列の名前を指定する必要があります。

*column\_list* 引数を渡す場合、引数には、ビューの *select\_statement* の定義クエリの中にある列の数と同じ数が入っていなければなりません。もし *column\_list* が省略された場合、ビューの列の名前は、ビューの *select\_statement* 内の列と同じ名前になります。

ビューとテーブルは固有の名前を持たなければなりません。

**OR REPLACE** オプションを渡した場合、ビューが既に存在する場合には自動的に作成されなおされます。このオプションは、カレントのビュー内で定義済みのオブジェクトを、消去 / 再作成 / 権利の割り当て等を行うことなく既存のビューの定義を変えたい時には有用なオプションです。ビューが既に存在していて **OR REPLACE** オプションを渡さなかった場合には、エラーが返されます。

*schema\_name* も標準の *sql\_name* で、ビューを格納するスキーマの名前を指定するのに使用することができます。*schema\_name* を渡さなかった場合、もしくは存在しないスキーマの名前を渡した場合には、ビューは自動的に "DEFAULT\_SCHEMA" という名前の標準のスキーマへと割り当てられます。

*select\_statement* はビューの定義クエリである SELECT 宣言を指定します。*select\_statement* は 4D の標準の SELECT と同じですが、以下の様な制限がかかります。

- INTO、LIMIT、OFFSET 節を使用することはできません。なぜなら 4D での限界、オフセット、変数の割り当てはビューを呼び出す SELECT 宣言によって実行されるからです。
- GROUP BY 節は使用できません。
- ビューは読み出し専用モードとなっており、更新はできません。

ビュー定義は静的であり、ソースとなるテーブルが変更されたり削除されたりしてもアップデートされることはありません。つまりあるテーブルに列が追加されても、そのテーブルを参照しているビューには列は表示されないという事です。同じように、削除済みの列にビューを通してアクセスしようとした場合、エラーが返されます。

ただし、削除済みのソースビューを参照しているビューは動き続けます。なぜなら、ビューを作成する時点で、どんなビュー参照もソーステーブルの参照へと変換されるからです。

ビューはグローバルなスコープを持ちます。**CREATE VIEW** を使用してビューが作成されると、それはアプリケーション内の全ての部分 (SQL を使用する 4D リモート、CREATE DATABASE コマンドを使用する外部データベース、SQL LOGIN コマンドを使用する他のデータベース、等々) か

らアクセス可能となり、その状態は **DROP VIEW** コマンドを使用してビューが削除されるまで続きます。

## 例

PEOPLE という名前の、以下の列を含むテーブルについて考えます。

```
ID          INT64
FIRST_NAME  VARCHAR(30)
LAST_NAME   VARCHAR(30)
DEPARTMENT  VARCHAR(30)
SALARY      INT
```

ビュー定義には以下の様な例があります。

### ▶ 制限のないビュー

```
CREATE VIEW FULLVIEW AS
  SELECT * FROM PEOPLE;
```

### ▶ 「水平的な」制限のあるビュー。例えば、マーケティング部にいる人間のみを表示したい場合、以下の様に記述します。

```
CREATE VIEW HORIZONTALVIEW (ID, FirstName, LastName, Salary) AS
  SELECT ID, FIRST_NAME, LAST_NAME, SALARY FROM PEOPLE
  WHERE DEPARTMENT = 'Marketing';
```

### ▶ 合算されたビュー

```
CREATE VIEW AGGREGATEVIEW (FirstName, LastName AnnualSalary) AS
  SELECT FirstName, LastName, SALARY*12 FROM PEOPLE;
```

### ▶ 「垂直的な」制限のあるビュー。例えば、SALARY の列を表示したくない場合は、以下の様に記述します。

```
CREATE VIEW VERTICALVIEW (ID, FirstName, LastName, Department) AS
  SELECT ID, FIRST_NAME, LAST_NAME, DEPARTEMENT FROM
  PEOPLE;
```

一度ビューが定義されれば、そのビューは標準のテーブルと同じように使用することが出来ます。例えば、給料が 5,000 ユーロを超える人間を全て取得したい場合、以下の様に記述します。

```
SELECT * FROM FULLVIEW
  WHERE SALARY < 5000
  INTO :aID, :aFirstName, :aLastName, :aDepartment, :aSalary;
```

また、例えばマーケティング部の中で "Michael" という名前を持つ人間を全て取得したい場合、以下の様に記述します。

```
SELECT ID, LastName, Salary FROM HORIZONTALVIEW
```

```
WHERE FirstName='Michael'
INTO :aID, :aLastName, :aSalary;
```

## DROP VIEW

```
DROP VIEW [IF EXISTS] [schema_name.]view_name[;]
```

**DROP VIEW** コマンドは、*view\_name* で名前を指定したビューをデータベースから削除します。

**IF EXISTS** 定数が渡されたとき、*view\_name* で指定したビューがデータベースに存在しなかったとしてもエラーは返されず、コマンドは何もしません。

*schema\_name* も標準の *sql\_name* で、ビューを格納するスキーマの名前を指定するのに使用することができます。*schema\_name* を渡さなかった場合、もしくは存在しないスキーマの名前を渡した場合には、ビューは自動的に "DEFAULT\_SCHEMA" という名前の標準のスキーマへと割り当てられます。

## ビュー上での新しいシステムテーブル

ビューの情報を管理するために、4D v14 では二つの新しいシステムテーブルが用意されています。

<b>_USER_VIEWS</b>	<b>型</b>	<b>データベースユーザーのビューの詳細</b>
VIEW_NAME	VARCHAR	ビューの名前
SCHEMA_ID	INT32	ビューが所属する <i>schema_name</i> の ID

<b>_USER_VIEW_COLUMNS</b>	<b>型</b>	<b>データベースユーザーのビューの列の詳細</b>
VIEW_NAME	VARCHAR	ビューの名前
COLUMN_NAME	VARCHAR	列の名前
DATA_TYPE	INT32	列の型
DATA_LENGTH	INT32	列のサイズ
NULLABLE	BOOLEAN	列が NULL 値を受け取る場合は True、それ以外の場合は False

## ALTER TABLE を使用した新しいオプション

### ALTER TABLE

**ALTER TABLE** SQL コマンドは、4D v14 から以下のものを有効化・無効化できるようになりました。

- **[TRAILING]** (ADD キーワードと使用): 列を作成する際に、ストラクチャーファイル内のテーブル内にある列のすぐあとに作成するように強制します。このオプションは、データを格納している列が (データを消去することなく) テーブルから削除されたときに有用です。なぜ

なら、既存のデータが新しい列に対して割り当てられるのを防ぐことができるからです。

- テーブルのジャーナル
- 倍長整数型のフィールドに対しての "Autoincrement" オプション
- UUID 型の文字フィールドに対しての "Auto UUID" オプション

```
ALTER TABLE sql_name
{ADD column_definition [PRIMARY KEY] [TRAILING]}
DROP sql_name |
ADD primary_key_definition |
DROP PRIMARY KEY |
ADD foreign_key_definition |
DROP CONSTRAINT sql_name |
[{ENABLE | DISABLE} REPLICATE] |
[{ENABLE | DISABLE} LOG] |
[{ENABLE | DISABLE} AUTO_INCREMENT] |
[{ENABLE | DISABLE} AUTO_GENERATE] |
SET SCHEMA sql_name}
```

以下の場合においてはエラーを返します。

- 任意の **ENABLE LOG** 引数が渡されているのに有効な主キーが定義されていない場合。
- **DISABLE LOG** を使用してジャーナルを無効化せずにテーブルの主キーの定義を変更または削除しようとした場合。

## システムテーブルにおける新しいカラム

`_USER_TABLES` と `_USER_COLUMNS` システムテーブルには新しいカラムが追加されています：

<code>_USER_TABLES</code>	型	データベースのユーザテーブルを記述
<code>REST_AVAILABLE</code>	ブール	テーブルが REST サービスで公開されていれば True、それ以外は False

<code>_USER_COLUMNS</code>	型	データベースのユーザテーブルのカラムを記述
<code>UNIQUENESS</code>	ブール	重複不可であれば True、それ以外は False

AUTOGENERATE	ブール	カラムの値が新しいレコードごとに自動的に生成される場合は True、それ以外は False
AUTOINCREMENT	ブール	カラムの値が自動インクリメントされるのであれば True、それ以外は False
REST_AVAILABLE	ブール	カラムが REST サービスで公開されていれば True、それ以外は False

*Note* AUTOGENERATE と AUTOINCREMENT カラムは v13.4 以降の 4D v13 でも使用可能です。

## 日付・時間定数のサポート

4D v14 の統合 SQL サーバーでは、ODBC API に沿った日付と時間の定数をサポートするようになりました。

ODBC 日付・時間定数のシークエンスのシンタックスは以下の通りです。

{constant\_type 'value'}

<i>constant_type</i>	<i>value</i>	詳細
d	yyyy-mm-dd	日付のみ
t	hh:mm:ss[.fff]	時間のみ
ts	yyyy-mm-dd hh:mm:ss[.fff]	日付と時間 ( <i>timestamp</i> )

*Note* *fff* はミリ秒を意味しています。

例えば、以下の様な定数を使用することができます。

```
{ d '2013-10-02' }
{ t '13:33:41' }
{ ts '1998-05-02 01:23:56.123' }
```

# 5

## 4D Mobile

4D SAS によって開発された Wakanda は、JavaScript や HTML5 といった標準のテクノロジーに基づいた Web アプリケーションを開発・公開するためのプラットフォームです。

"4D Mobile" アーキテクチャーを用いれば 4D-Wakanda 間にダイレクトなリンクを設定することができます。この場合、最新世代の Wakanda の Web インターフェースの豊富なグラフィックと機能を、4D データベースの実力と組み合わせて使用することができます。

---

*Note* もし 4D と Wakanda の最初のリンクをすぐに設定したいのであれば、以下にある適切な環境と設定があることを確認の上、[107P “ステップバイステップ形式での解説”](#) を参照して下さい。

---

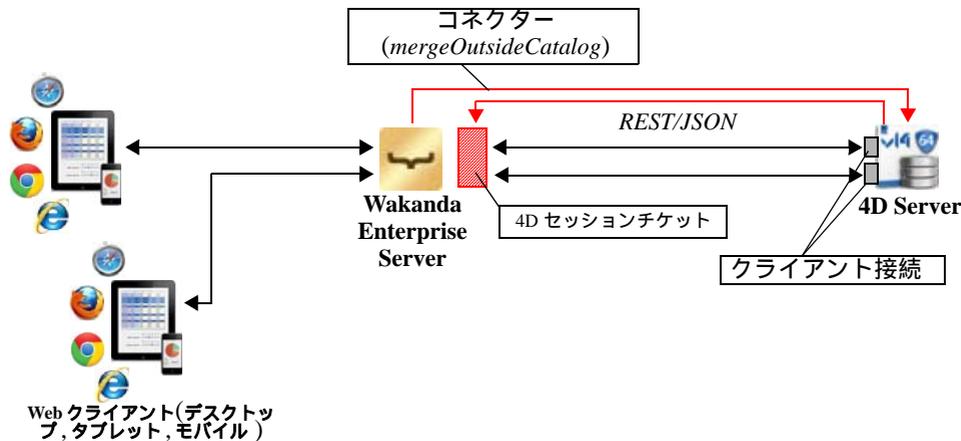
### 4D Mobile アーキテクチャー

4D/Wakanda コネクタを使用してアーキテクチャーを設定する場合、以下のシステム要件が最低限必要になります。

- **4D Server v14**  
ただし 4D Mobile コネクタを使用してソリューションを開発・テストしたい場合には、シングルユーザーの **4D v14(Developer Professional)** をお使いいただくことも可能です (この場合にはクライアント接続が同時に三つまでご使用いただけます)。
- **Wakanda Enterprise Server v7 と Wakanda Enterprise Studio v7**  
開発に必要なになります。どちらも <http://www.wakanda.org> からダウンロードすることができます。
- 相互に通信をするための 4D データベースと Wakanda アプリケーション

4D 側では、Wakanda アプリケーション側で利用したい全てのテーブル、属性、そしてメソッドを、アクセス可能な状態にしておく必要があります ( [118P “4D データベースの設定”](#) を参照して下さい)。

4D Mobile アーキテクチャーは以下の様に表すことができます。:



Wakanda ソリューションが開始したときに Wakanda サーバーによって `mergeOutsideCatalog()` JavaScript メソッドが実行されると (詳細は [mergeOutsideCatalog\(\) メソッドの実行](#) を参照して下さい)、Wakanda Enterprise Server と 4D Server の間にリンクが構築されます。接続が 4D Server によって認証されると (119P “REST 接続の管理” を参照して下さい)、REST セッション「チケット」が Wakanda サーバーへと渡されます。このチケットは Wakanda によって以降の全ての REST クライアントリクエストに対して使用されます。

このリンクによって、Wakanda サーバーは 4D データベースの二種類のリソースにアクセスできるようになります。

- テーブルとその属性
- プロジェクトメソッド

認証が完了すると、これらのリソースは Wakanda アプリケーションのローカルのカatalogに入っていたかのように、Wakanda 側にて直接使用されます (この際の接続は Wakanda アプリケーションに対しては透過的です)。

WebクライアントがWakandaサーバーへ4Dデータベースへのアクセスを必要とするリクエストを送った場合、このリクエストはカレントのチケットを使用して4Dサーバーへと送られ、4D Server マシンでは標準のクライアント接続が開かれます。この接続はデフォルトで60分のタイムアウトを上限として、ユーザがリクエストを送り続ける限り開いたままになります。このデフォルトのタイムアウト時間は `mergeOutsideCatalog()` メソッドの `timeout` 引数を使用して実行することによって変更することができます。セッション中に、4Dサーバーにて認証されたクライアント接続

数がライセンス数に達してしまった場合、エラーメッセージが Wakanda サーバーに返されます。

*Note* [addRemoteStore\(\)](#) と [openRemoteStore\(\)](#) メソッドを使用して、Wakanda と 4D アプリケーションとの間に動的で一時的なリンクを構築することもできます。これらのメソッドに関しては [openRemoteStore\(\)](#) と [addRemoteStore\(\)](#) を参照して下さい。

## ステップバイステップ形式での解説

このセクションでは、ステップバイステップ形式で一つずつ手順を追って Wakanda / 4D コネクタの機能を紹介していきます。具体的には以下のような手順を解説します。

- 4D データベースの作成と設定
- 単一のページの Wakanda アプリケーションの作成
- 4D データベースからのデータを Wakanda のページに表示する

解説を簡単にするために、ここでは 4D アプリケーションと Wakanda アプリケーションが同じマシン上にある場合を考えていきます。もちろん、リモート構造を使用することも可能です。

### 1 - 4D データベースの作成と設定

- 1 4D アプリケーションまたは 4D Server アプリケーションを起動して新規にデータベースを作成します。

ここでは "Emp4D" という名前をつけたという仮定で解説を進めます。

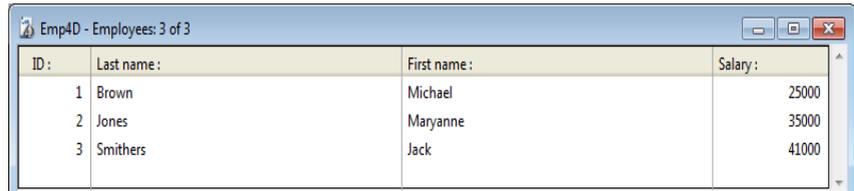
- 2 ストラクチャーエディターの中で、[Employees] というテーブルを作成して以下のフィールドを追加します。

- LastName (文字列)
- FirstName (文字列)
- Salary (倍長整数)



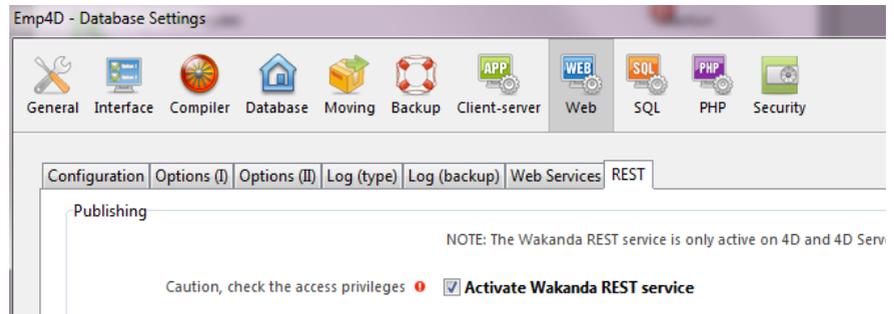
"REST サービスで公開" の属性は、全てのテーブルにおいて最初からチェックがされており、この設定は変更しないで下さい。

- 3 テーブルのボタンをクリックして 4D にデフォルトフォームを作成させたのち、実際のデータを数レコード分作成します。

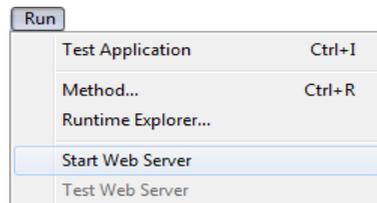


ID :	Last name :	First name :	Salary :
1	Brown	Michael	25000
2	Jones	Maryanne	35000
3	Smithers	Jack	41000

- 4 データベース設定ダイアログボックスの中から "Web" のページの中を表示させ、"REST" タブをクリックします。
- 5 "Wakanda REST サービスを有効化" のオプションをクリックしてオンにし、OK をクリックします。



- 6 実行のメニューの中から Web サーバー開始を選択します。:



これで 4D データベースは Wakanda からの REST リクエストに回答する準備が出来ました。なお、ここでは簡略化のために REST 接続の管理まではしていないという点に注意して下さい。実際の製品やオープンアーキテクチャの場合は REST 接続を安全に管理することが必要不可欠となります (詳細な情報に関しては [137P "4D Mobile のセキュリティについて"](#) を参照して下さい)。

## 2 - Wakanda アプリケーションの作成

- 1 "Wakanda Enterprise Studio" アプリケーションを起動し、Create a New Solution ボタンをクリックします。:



- 2 作成ダイアログボックスにて、名前を記入して **OK** をクリックします。ここでは "EmpWakanda" という名前をつけて説明を続けます。



アプリケーションプロジェクトが作成され、Wakanda Studio Explorer のデフォルトの項目がウィンドウの左側に表示されます。

- 3 Model をダブルクリックします。



デフォルトのデータスタクラスを内包した Wakanda Model Designer が表示されます。しかしここでは外部モデルを作成したいので、このエディターは使用しません。

- 4 Wakanda のエクスプローラーエリアにて Model.js ファイルをダブルクリックします。

このファイルは GUI エディターを開いたときに自動的に作成されます。ここにはモデルの定義が JavaScript コードで記述されています。

- 5 以下の JavaScript 文を入力します。

```
model.mergeOutsideCatalog("Emp4D", "localhost", "", "");
```

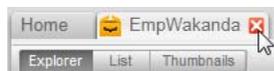


このコードはコネクタを有効化し、Wakanda プロジェクトと 4D の間のリンクを開きます。

- "Emp4D" は Wakanda Enterprise Studio 側で表示されるローカル名です。ここには任意の名前を入力できますが、ここでは簡略化のために 4D データベースの名前を使用します。
- "localhost" は 4D v14 の HTTP サーバーのアドレスです (必要であればホスト名または IP アドレスを入力して下さい)。
- 残り二つの空の文字列には名前とパスワードを入力しますが、この例では使用しないので空欄のまま話を進めます。

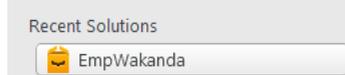
ソリューションを一度閉じた後再度開き、Wakanda Enterprise Studio にモデルをロードします。

#### 6 エクスプローラー内のソリューションの閉じるボタンをクリックします。

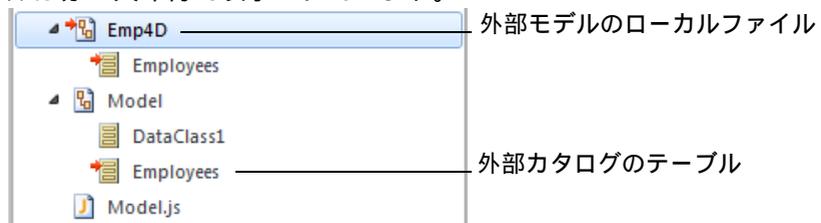


モデルファイルをまだ保存していない場合、ダイアログボックスが表示されます。変更を全て保存するためには **Save All** をクリックして下さい。

#### 7 "Recent Solutions" のリスト内からソリューションをクリックして再度 Wakanda Enterprise Studio で開きます。



すると、"Emp4D" 外部モデルが Wakanda アプリケーションのファイル内に表示され、4D アプリケーションの [Employees] テーブルがローカルモデルの *datastore classes* 内に表示されているのが確認できます。外部モデルは赤い矢印付で表示されています。



#### うまく行かない場合は ...

この段階でリストにテーブルが表示されていないのであれば、以下の点をチェックして下さい。

- サードパーティサービスやソフトウェア (例えばインスタントメッセージャーなど) が 4D HTTP サーバーの公開ポートと競合していないか (初期設定値では 80)。
- 4D 側で、4D Web サーバーが開始され、Wakanda REST サービスが有効化されていて、テーブルが REST へ公開されているか。

- mergeOutsideCatalog() へ渡されたアドレスが有効であるか。

4D Server が実際に REST リクエストに反応しているかどうかを調べるためには、以下の URL をブラウザに入力して下さい。

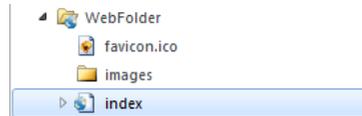
<address>/rest/\$catalog/\$all  
(REST に公開されている全てのテーブルを返します。)

<address>/rest/my\_table/my\_method  
(メソッドが結果を返すならば、その結果を全て返します。)

### 3 - Wakanda ウィジェットを使用して 4D data を表示する

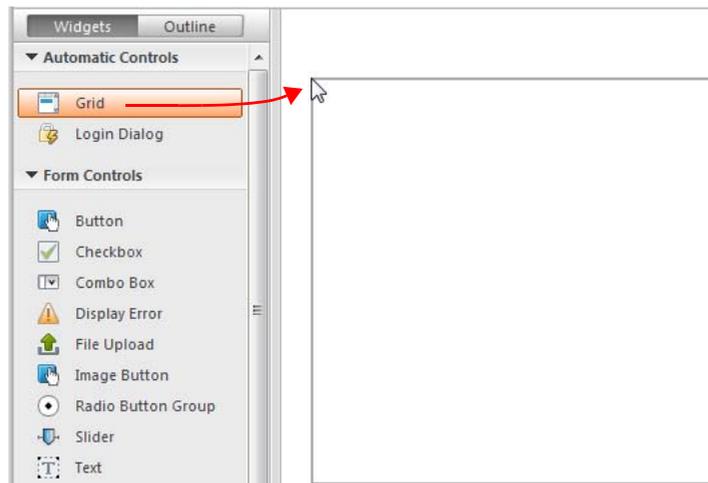
ここでは単純なドラッグ & ドロップによって 4D テーブルと Wakanda ウィジェットを関連づけ、Wakanda Enterprise Server を起動してデータを表示させます。

- 1 エクスプローラー内の "WebFolder" のフォルダーを開き、Index のページをダブルクリックし、Wakanda の GUI デザイナーを開きます。

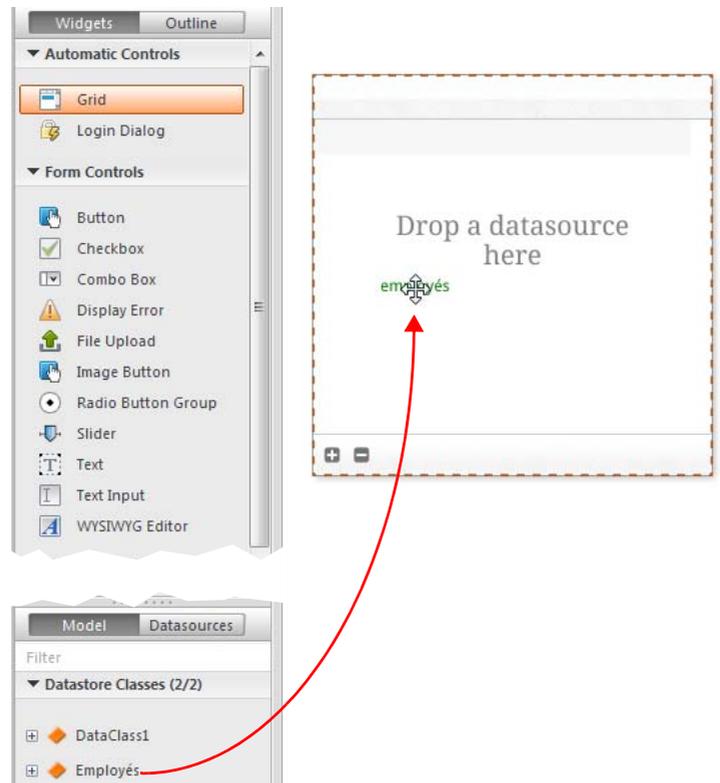


*Note* "WebFolder" にはプロジェクトの中の Web 公開に必要な要素が置かれています。"Index" はプロジェクトのデフォルトのページになります。

- 2 ウィジェットのリストの中の、"Grid" をクリックしてワークエリアにドロップします。:



- 3 モデルの *Datastore Classes* のリスト内の "Employees" をクリックし、作成したグリッドの中にドロップします。



この時点で、エディターは "Employees" クラスをもとにした *datasource* を自動的に作成します。これはウィジェットのコンテンツを管理します。この *datasource* とは Wakanda によって管理される JavaScript オブジェクトで、デフォルトでは "employees" という名前がついています (クラス名の頭文字が小文字になったものです)。

ウィジェットには中身のプレビューが表示されます。ウィンドウを広げることによってデータソースの全フィールドを表示することができます。

ID	Last name	First name	Salary
Text	Text	Text	Text

これによってデータソースとウィジェットの関連付けが完了しました。

- 4 エディターのツールバーの **Save**  ボタンをクリックします。  
今度はブラウザを使用してデータを表示させてみましょう。
- 5 Wakanda Enterprise Studioのツールバーの **Run project** をクリックします。



これをクリックすることにより Wakanda Enterprise Server が開始し、"EmpWakanda" アプリケーションをパブリッシュします。先に設定してお

いた 4D Mobile リンクのおかげで、4D データベースのデータを既定のブラウザのウィンドウ内に表示させることができます。

ID	Last name	First name	Salary
1	Brown	Michael	25000
2	Jones	Maryanne	35000
3	Smithers	Jack	41000

Web 側でデータを変更することによってリンクのダイナミックな性質をテストすることもできます。例えば、ここでは Maryanne Jones' の名字を "Jackson" に変えたのが、4D 側でも直ちに反映されています。

ID	Last name	First name	Salary
1	Brown	Michael	25000
2	Jackson	Maryanne	35000
3	Smithers	Jack	41000

ID :	Last name :	First name :
1	Brown	Michael
2	Jackson	Maryanne
3	Smithers	Jack

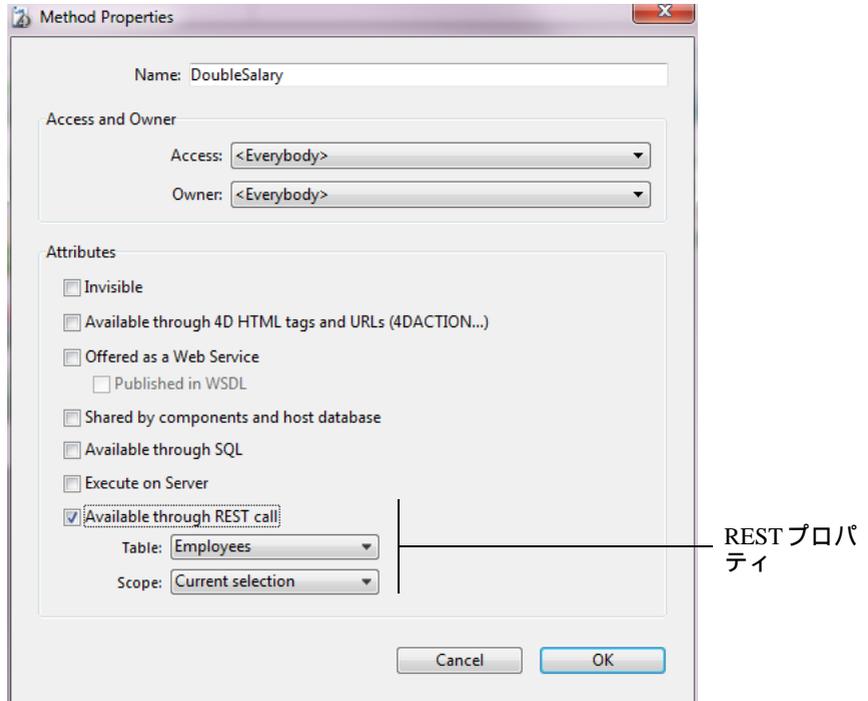
#### 4 - 4D メソッドの作成と呼び出し

ここではとても単純なプロジェクトメソッドを 4D 側で作成し、Web ページ側から実行します。このメソッドは全ての salary の値を二倍にします。

- 1 4D 側で、*DoubleSalary* という名前のプロジェクトメソッドを作成し、以下のコードを入力します。

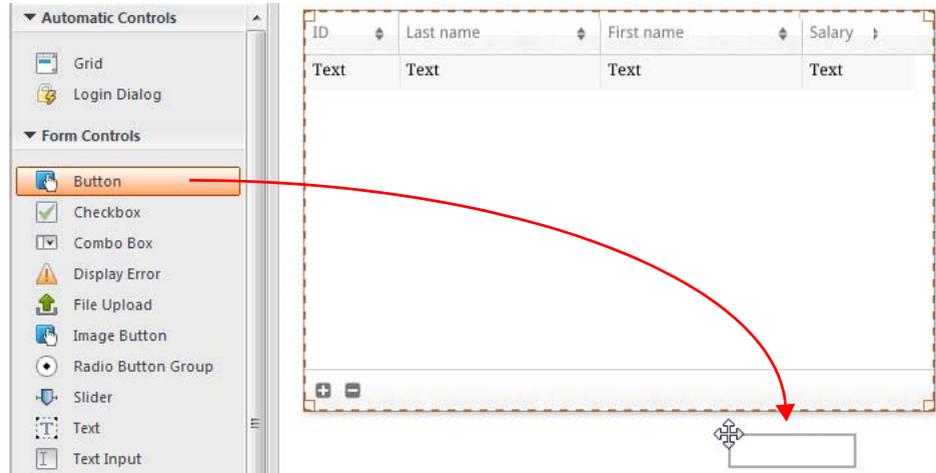
```
FIRST RECORD ([Employees])
While (Not (End selection ([Employees])))
    [Employees]salary:=[Employees]salary*2
SAVE RECORD ([Employees])
NEXT RECORD ([Employees])
End while
```

## 2 メソッドプロパティの REST の設定をし、OK をクリックします。



Wakanda では、クラスメソッドは以下のどれかに適用されます。エンティティ (レコード)、エンティティコレクション (セレクション)、データストアクラス (全レコード)。これらの内容を 4D 側で指定する必要があります。

- 3 Wakanda Enterprise Studio 側で、*GUI Designer* 中の Index に戻り、ウィジェットのリストからボタンを選択して追加します。



- 4 ボタンをダブルクリックし、"Double salaries" という名前をつけます。

Double salaries

- 5 "Double salaries" ボタンが選択されているのを確認したうえで、*GUI Designer* の右側にある Events ボタンをクリックします。
- 6 "On Click" のアイコンをクリックし、イベントを追加します。



コードエディターが表示され、ボタンがクリックされたときに実行したいコードを記述することができます。ここでは単純に 4D の *DoubleSalary* メソッドを呼び出し、コールバックファンクション (*onSuccess*) にて全レコードをリロードするようにトリガーします。

- 7 以下のコードを記述します。

```
sources.employees.DoubleSalary({
  onSuccess:function(event){
    sources.employees.allEntities();
  }});
```

コードエディター内は以下の様になるはずです。

```
button1.click = function button1_click (event)
{
    sources.employees.DoubleSalary({
        onSuccess:function(event) {
            sources.employees.allEntities();
        });
    });
};
```

"employees" という単語は頭の "e" が小文字になっていることに注意して下さい。ここではクラスがウィジェットと関連付けられた際に自動的に作成されたデータストアクラスを使用しているからです。

- 8 エディターのツールバーの **Save**  ボタンをクリックして保存します。これで 4D のメソッドを呼び出すテストの準備が出来ましたが、その前にモデルを Wakanda Enterprise Server 上でリロードする必要があります。
- 9 Wakanda Enterprise Studio のツールバーの中にある **Reload Models**  ボタンをクリックします。
- 10 ブラウザのページを再読み込みして **Double salaries** ボタンを表示させ、ボタンをクリックします。

ID	Last name	First name	Salary
1	Brown	Michael	25000
2	Jackson	Maryanne	35000
3	Smithers	Jack	41000

3 item(s)

**Double salaries**

Salary の欄の値が倍増したのが確認できます。

ID	Last name	First name	Salary
1	Brown	Michael	50000
2	Jackson	Maryanne	70000
3	Smithers	Jack	82000

ただし、ここで紹介した例はあくまで Wakanda/4D コネクタの設定を解説するためのものであり、ここで紹介した簡略化されたメソッドは製品では使用できるものではないことに注意して下さい。

## 4D データベースの設定

セキュリティ上・パフォーマンス上の理由から、REST 要求を使用している 4D データベースのテーブル、データ、そしてメソッドへの接続 (Wakanda サーバーが使用するプロトコル) は、有効化され、明示的に認証されている必要があります。そのためには 3 段階のアクセスの設定をしなければなりません。

- Wakanda REST サービスのスタートアップ
- REST アクセスの管理 (任意ですが推奨されます)
- それぞれのデータベースオブジェクト (テーブル、属性、プロジェクトメソッド) の REST サーバーへの公開は必要に応じて個別に設定する必要があります。初期設定では、
  - テーブルと属性は全て REST からアクセス可能
  - プロジェクトメソッドは REST からアクセス不可となっています。

### Wakanda REST サービスの有効化

デフォルトとして、4D Server の v14 は REST 要求には反応しません。Wakanda/4D コネクタの設定ができるようにこれらの要求が処理されるようにするためには、Wakanda REST サービスを有効化する必要があります。

---

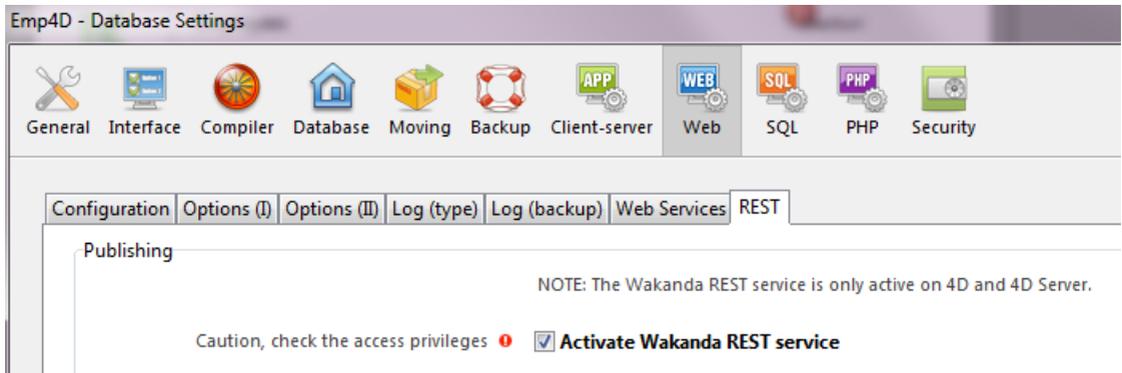
*Note* REST サービスは 4D HTTP サーバーを使用します。そのため、4D Web サーバーまた 4D Server が開始されていることを確認して下さい。

---

Wakanda REST サービスを有効化するためには以下の手順に従って下さい。

- 1 データベース設定において Web のページの REST タグをクリックします。

## 2 Activate Wakanda REST service のオプションにチェックをします



Wakanda REST サービスが有効化されると、「警告：アクセス権が正しく設定されているか確認して下さい。」という警告メッセージが表示されます。これは REST 接続が適切に管理されていない限り、デフォルトでデータベースオブジェクトへは自由にアクセスできてしまうためです（詳細は以下を参照して下さい）。

### REST 接続の管理

REST 接続の管理とは、Wakanda REST リクエストの後に、4D 側でそのセッションを開くかどうかの認証をすることです。

*Note* Wakandaを通したREST 接続の場合、[mergeOutsideCatalog\(\) メソッドの実行](#)の際に送られた名前とパスワードによって認証が行われます。

包括的に REST 接続を管理する方法は二つあります。

- 4D パスワードを用いて自動的に管理する方法
- 新しい On REST Authentication データベースメソッドを用いてプログラムによって管理する方法

これらのコントロールモードはどちらかしか選択できません。つまり [On REST Authentication](#) が定義されると 4D パスワードを使用した自動アクセス管理は無効化されます。

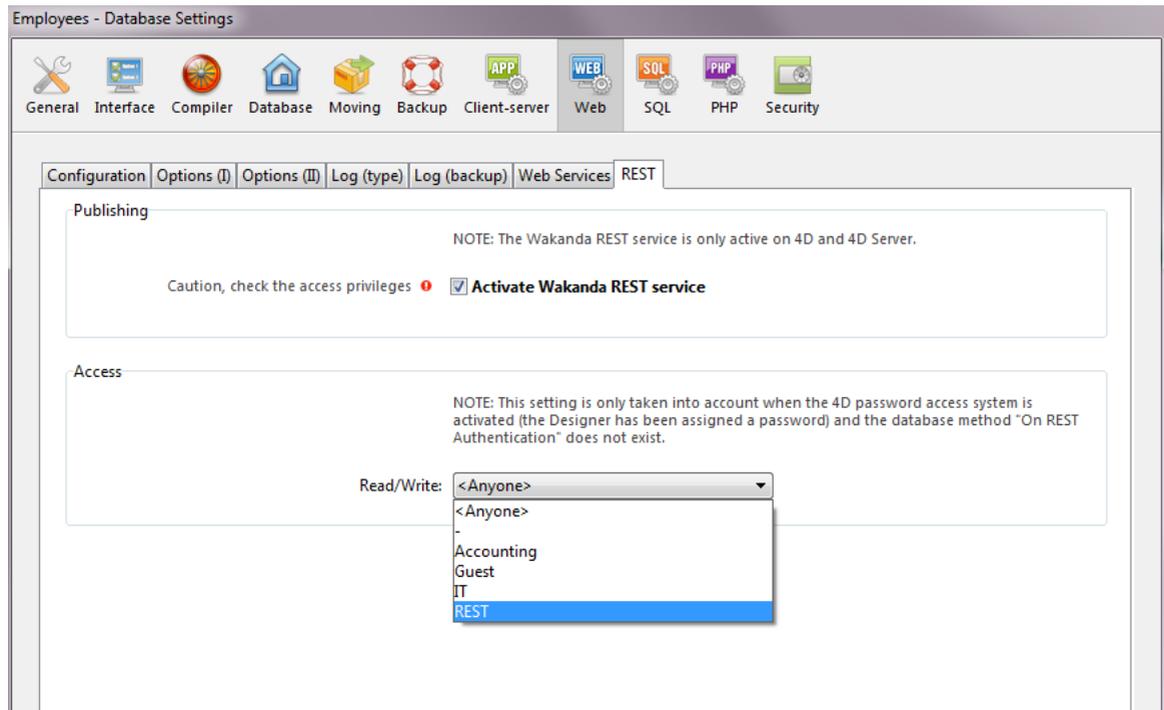
警告：これらのアクセス管理がどちらも有効化されていない場合、REST を経由してのデータベースへのアクセスは常に受理されてしまいます。この状態は推奨されません。

### 4D パスワード使用した自動コントロール

4D では、Wakanda アプリケーションから 4D サーバーへのリンクを設定できる 4D ユーザーのグループを指定することが出来ます。

以下の手順でアカウントを指定して下さい。

- 1 データベース設定の画面から Web REST のページを表示。
- 2 アクセス権のエリア内の「読み込み / 書き出し」のボックスから使用するグループを選択します。



初期設定では、メニューには <すべて> と表示されています。これは REST 接続は全てのユーザーにオープンであるという状態を示しています。

グループの指定が終わると、そのグループに所属するユーザーのみが Wakanda REST リクエストを通して 4D へとアクセスできるようになります。具体的には、4D Server 上で `mergeOustideCatalog()` メソッドを使用してセッションを開くことができますということです。このグループに所属していないアカウントの場合は、4D はリクエストの送信者に対して認証エラーを返します。

この設定を有効にするために以下の点に注意して下さい。

- 4D パスワードシステムが起動している(パスワードが Designer に割り当てられている)必要があります。
- **On REST Authentication** が定義されていないことを確認して下さい。定義されてしまうと、データベース設定のアクセス設定が全て無効になってしまうからです。

## On REST Authentication

On REST Authentication ( \$1; \$2 ; \$3 ) → \$0

引数	型	説明
\$1	Text	← ユーザー名
\$2	Text	← パスワード
\$3	Boolean	← True = ダイジェストモード False = ベーシックモード
\$0	Boolean	← True = リクエスト承認 False = リクエスト拒否

**On REST Authentication** データベースメソッドを使用することにより REST セッションの接続の設定を自在に管理できるようになります。

Wakanda Server から *mergeOutsideCatalog()* メソッドを使用した REST セッションを開くリクエストが来ると (一般的なケース)、接続の識別子がリクエストのヘッダーに供給されます。続いて **On REST Authentication** データベースメソッドが呼ばれこれらの識別子を評価します。4D データベースのユーザーのリストを使用することもできますし、独自の識別子のテーブルを使用することもできます。

**重要** : **On REST Authentication** が定義される (つまり中にコードが記述される) と、4D は REST リクエストの管理をそちらに全て一任します。このとき、データベース設定の Web/REST ページ内の「読み込み / 書き出し」メニューで設定した内容は、無視されます (119P “4D パスワード使用した自動コントロール” を参照して下さい)。

このデータベースメソッドは二つのテキスト型の引数 (\$1 と \$2) と一つのブール型の引数 (\$3) を 4D から受け取り、ブール型の引数 \$0 を返します。これらの引数は以下の様に宣言されている必要があります。

```
// On REST Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
... // Code for the method
```

\$1 には接続に使用したユーザー名が入り、\$2 にはパスワードが入ります。

リクエストに使われるモードにより、パスワード (\$2) は標準テキストまたはハッシュ値で受け取る事が可能です。このモードは \$3 引数によって指定され、適切に処理することができます。

- パスワードが標準テキスト (ベーシックモード) である場合、\$3 には **False** が渡されます。
- パスワードがハッシュ値 (ダイジェストモード) である場合、\$3 には **True** が渡されます。

REST 接続リクエストが Wakanda Server から来るとき、パスワードは必ずハッシュ値で送られてきます。

リクエストが Wakanda 以外から送られてきた場合、デベロッパが責任を持って "username-4D" フィールドと "password-4D" フィールドを HTTP ヘッダーに含めることによって認証を管理して下さい。この場合、パスワードは 4D REST サーバーに標準テキストで送られてなければなりません (サードパーティからの干渉のリスクを避けるために SSL を使用して下さい)。

REST 接続の識別子は、データベースメソッド内でチェックしなければなりません。通常、ユーザー独自のテーブルを使用して名前とパスワードをチェックします。もし識別子が有効であるなら、\$0 に **True** を渡します。すると、リクエストが受理されます。4D はこのリクエストを実行して結果を JSON 形式で返します。

それ以外の場合は \$0 に **False** を渡します。この場合、接続は拒否され、サーバーはリクエストの送信者へ認証をエラーを返します。

ユーザーがデータベースの 4D ユーザーのリストの中に載っているとき、以下の宣言によってパスワードを直接チェックすることができます。

```
$0 := Validate password($1 ; $2 ; $3)
```

---

*Note* [Validate password](#) コマンドは拡張され、第一引数にユーザー名、第二引数にパスワードを渡し、任意の第三引数でパスワードがハッシュド形式で書かれているかどうかを指定できるようになりました。(298P “[Validate password](#)” を参照して下さい)。

---

4D データベースのものとは別の独自のユーザーリストを使用したい場合、そのユーザー達のパスワードを、Wakanda Server が On REST Authentication データベースメソッドに接続リクエストを送る時のアルゴリズムと同じものを用いてハッシュドフォームにて \$2 引数に保存することができます。

この方法を使用してパスワードをハッシュする場合、以下の様に記述して下さい。

```
$HashedPasswd := Generate digest( $ClearPasswd ; 4D digest)
```

---

*Note* [Generate digest](#) コマンドにはハッシュアルゴリズムとして [4D digest](#) を受け取れるようになりました。これは 4D のパスワードの内部管理で使用されているメソッドと対応しています (詳細は 294P “[Generate digest](#)” を参照して下さい)。

---

- ▶ 以下の [On REST Authentication](#) の使用例は、4D ユーザーにマッチしない "admin" というユーザーと "123" というパスワードの組み合わせのみを認証します。

```

C_TEXT($1;$2)
C_BOOLEAN($0;$3)
  //$1: ユーザー
  //$2: パスワード
  //$3: ダイジェストモード
If($1 = "admin")
  If ($3)
    $0 := ($2 = Generate digest( "123" ; 4D digest))
  Else
    $0 := ($2 = "123")
  End if
Else
  $0 := False
End ifi

```

- ▶ 以下の [On REST Authentication](#) の使用例は、接続リクエストが 4D データベースのユーザーに保存されている二つの認証済みの Wakanda サーバーのどちらかから来ていることをチェックします。

```

C_TEXT($1;$2)
C_BOOLEAN($0)
ON ERR CALL("REST_error")
If ($1="WAK1") | ($1="WAK2")
  $0 := Validate password($1 ; $2 ; $3)
Else
  $0:=False
End case

```

## REST に公開されている 4D オブジェクトの設定

Wakanda REST サービスが 4D データベース内で有効化されると、デフォルトで REST セッションは全てのテーブルとフィールドにアクセスすることができ、またそのデータを使用することが出来ます。例えば、あるデータベースに [Employee] というテーブルがあった場合、Wakanda 側で以下の様に記述することでデータを取得することができます。

```

var emp=ds.Employee.query("name == 'Martin'");
  // 名前のフィールドが 'Martin' である従業員の全データを返します。

```

---

*Note* 「非表示」のオプションにチェックがされている 4D のテーブル / フィールドに関しても、REST サーバーへと公開されます。

---

さらに Wakanda サーバーは 4D データベースのプロジェクトメソッドにアクセスすることもできます。しかしながら、セキュリティ上の理由からこのアクセスはデフォルトでは無効化されています。

データベースのオブジェクトの REST への公開をカスタマイズしたい場合は、

- 公開したくないテーブル/フィールドは「REST サーバーに公開」のチェックを外します。
- 公開したいテーブル/フィールドは「REST サーバーに公開」にチェックをします。

REST リクエストが認証されていないリソース ( テーブルまたはプロジェクトメソッド ) にアクセスをしようとした場合、4D はエラーを返します。

## テーブルの公開

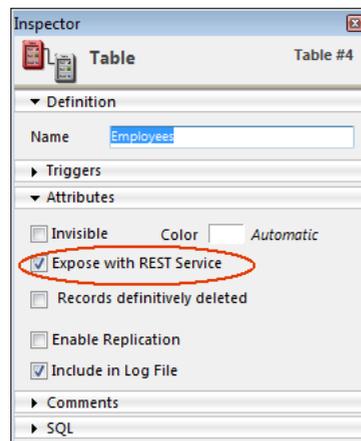
デフォルトでは、全てのテーブルは REST サーバーに公開されています。

セキュリティ上の理由からデータベースの一部のテーブルのみ REST サーバーに公開したいという場合があるかもしれません。しかし、ユーザー名とパスワードを記録した [Users] というテーブルを作成していた場合は、これは公開しない方がよいでしょう。

テーブルの REST サーバーでの公開は以下の手順で修正します。

- 1 ストラクチャーエディター内で公開したいテーブルのインスペクターを表示します。

デフォルトでは、REST サービスで公開のオプションにチェックがされています。:



- 2 REST サーバーで公開のオプションのチェックを外します。  
または

公開するテーブルに関してはオプションにチェックをしてください。  
公開・非公開を修正したいテーブルそれぞれに関して上記の操作をして下さい。

## フィールドの公開

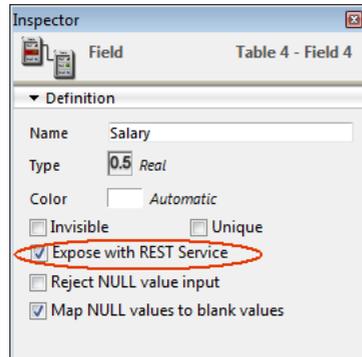
デフォルトでは、フィールドは全て REST サーバーで公開されています。

しかしテーブル内のフィールドのうち、REST サーバーで公開したくないものもあるでしょう。例えば、[Employees] というテーブルの給料のフィールドなどは公開したくないかもしれません。

フィールドごとの REST 公開については以下の様に修正します。

- 1 ストラクチャーエディター内で公開したくないフィールドのインスペクターを表示します。

デフォルトでは、REST サービスで公開のオプションにチェックがされています。



- 2 REST サーバーで公開のオプションのチェックを外します。

または

チェックされていないフィールドを公開するためにはチェックをします。  
公開・非公開を修正したいフィールドそれぞれに関して上記の操作をして下さい。

フィールドが REST で公開するためには、テーブルも同様に公開されてなければならないことに注意して下さい。親のテーブルが公開されていないとき、その中のフィールドは公開状態に関係なく非公開になります。これを利用して、テーブルの REST での公開設定を選択することにより、個々のフィールドの REST での公開設定を変えることなく公開 / 非公開を切り替えることができます。

## プロジェクトメソッドの公開

デフォルトではどのプロジェクトメソッドも REST では公開はされていません。

しかし、場合によっては一部のプロジェクトメソッドを REST に公開したいことがあるかもしれません。そのためには適切なオプションを選択し、メソッドの実行コンテキストを定義する必要があります。

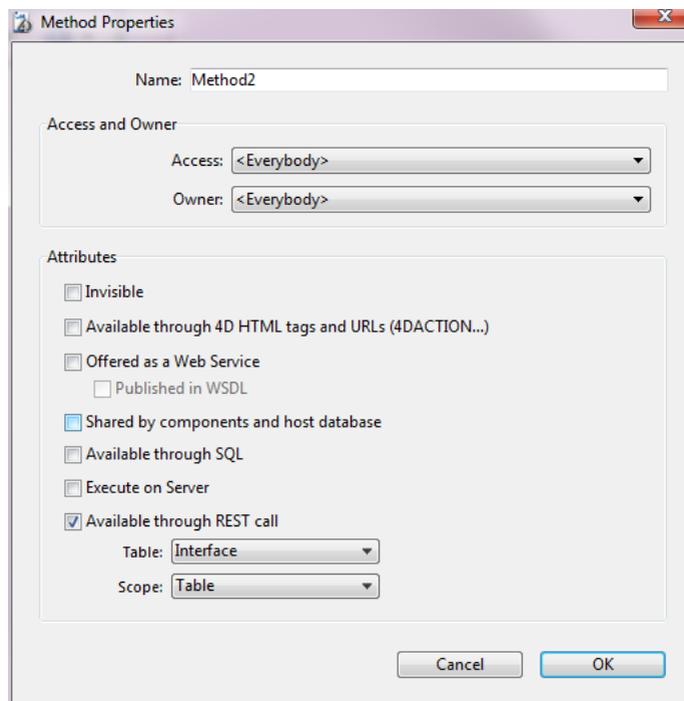
*Note* 4D メソッドにアクセスグループが関連付けられている場合、REST のグループがこのグループに含まれている必要があります。

プロジェクトメソッドの REST 公開は以下の様に設定します。

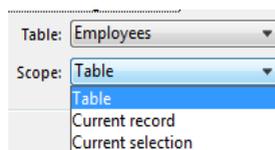
### 1 メソッドプロパティのダイアログボックスを表示します。

*Note* メソッドプロパティのダイアログボックスは、エクスプローラー内のメソッドのページのコンテキストメニューか、メソッドエディターのメソッド情報のボタンから行くことができます。

### 2 REST 呼び出しからの利用を許可のオプションにチェックを入れます。



### 3 テーブルとスコープを使って REST 実行コンテキストを定義します。



これらの設定は REST シンタックスと Wakanda のロジックでは必須です。この点についての詳細は、以下のセクションを参照して下さい。

#### 4 OK をクリックして変更を確定させます。

REST を介して使用可能なプロジェクトメソッドは、4D エクスプローラーの「REST メソッド」内に一覧で表示されます (128P “[エクスプローラー](#)” を参照して下さい)。

#### プロジェクトメソッドの ペアレントテーブルとス コープについて

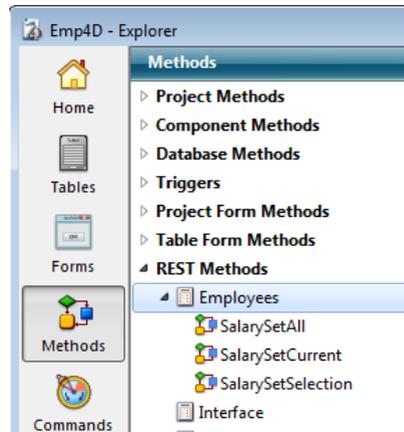
REST リクエストを介して使用可能なプロジェクトメソッドを宣言するとき、その呼び出しコンテキストを **テーブル** と **スコープ** を通じて明示的に宣言する必要があります。

- **テーブル**: プロジェクトメソッドと関連付けられているテーブルです。REST 呼び出しの最中、メソッドは `/tableName/methodName` の形の構文で呼び出すことができます。メニューには REST に公開されているデータベースのテーブルの一覧が表示されます。メソッドが扱うデータを含むテーブルを選択して下さい。また、例えば [RESTInterface] というテーブルを作成して REST に公開されている全てのプロジェクトメソッドを関連付けることもできます。
- **スコープ**: メソッドが適用される範囲を指定します。この宣言は必須です。なぜなら、Wakanda 側ではメソッドは JavaScript オブジェクトのプロパティとしてみなされ、これらのオブジェクトを使用しないと呼び出せないからです。公開されている 4D メソッドはそれぞれ明示的に呼び出されるデータベースコンテキストと関連付けられている必要があります。テーブル、カレントレコード、そしてカレントセレクションから選択できます。
  - **テーブル**: このオプションは、4D メソッドが指定されたテーブルの全てのレコードを使用して実行されるという事を意味します。Wakanda 側では、メソッドはデータスタッククラスという型を使用して呼び出されます。
  - **カレントレコード**: このオプションは、4D メソッドが指定されたテーブルのカレントレコードを使用して実行されるという事を意味します。Wakanda 側では、メソッドはエンティティという型のオブジェクト上で呼び出されます。
  - **カレントセレクション**: このオプションは 4D メソッドが指定されたテーブルのカレントセレクションを使用して実行されるという事を意味します。Wakanda 側では、メソッドはエンティティコレクションという型のオブジェクト上で呼び出されます。

4D 側でプロジェクトメソッドの公開設定やスコープの設定を変更した場合、Wakanda 側でリモートモデルをリロードしてこれらの変更を有効化する必要があります。

## エクスプローラー

Wakanda REST サービスが有効化されているとき、REST に公開されているテーブルとそれに関連付けられているプロジェクトメソッドの一覧は、4D エクスプローラー内の REST メソッドのセクションに表示されます。



## Wakanda アプリケーション側の設定

Wakanda Enterprise 側では、通常 `mergeOutsideCatalog()` JavaScript メソッドを使用して 4D v14 データベースと接続します。接続が Wakanda と 4D の間に設立されると、Wakanda は 4D で公開されている全てのテーブル、属性、そしてプロジェクトメソッドをローカルオブジェクトと同じように使用することが出来るようになります。

またこのメソッドを使用して追加で JavaScript コードを実行する事もできます。例えば、リモート属性のプロパティをローカルに修正したり、クラスをクラスを拡張したり、計算属性を追加したりできます。

**`mergeOutsideCatalog()` メソッドの実行** `mergeOutsideCatalog()` JavaScript メソッドはリモートデータのカatalogを指定し、それをカレントの Wakanda モデル内で使用します。このメソッドはカレントのモデルに関連付けられている `.js` ファイル内で呼び出され、Wakanda サーバーによって実行されなければなりません。

このとき、以下の二つのシンタックスのどちらかを使用できます。

- ダイレクトシンタックス

```
model.mergeOutsideCatalog(localName, address, user, password);
```

#### ■ オブジェクトを使用したシンタックス

```
model.mergeOutsideCatalog(localName, {
    hostname: address,
    user: userName,
    password: password,
    jsFile: jsFilePath
    timeout: minutes });
```

オブジェクトを使用したシンタックスの利点は、4D データベースに接続したあとに実行される .js ファイルを追加できることです。このファイルはリモートデータベースから参照されるカタログをローカルに修正することができます。

引数	型	説明
<i>localName</i>	文字列	リモートカタログのローカル名
<i>ipAddress</i>	文字列	リモートデータサーバーのアドレス (セキュリティのために HTTPS を使用して下さい)
<i>userName</i>	文字列	セッションを開くためのユーザー名
<i>password</i>	文字列	セッションを開くためのパスワード
<i>jsFilePath</i>	文字列	JavaScript ファイルへのパス名 (任意)
<i>timeout</i>	数字	4Dデータベースへのクライアント接続のタイムアウト (分、初期設定は 60。任意)

*Note* より詳細な情報に関しては、Wakanda Server 側の API マニュアルの、[documentation of the \*mergeOutsideCatalog\(\)\* method](#) を参照して下さい。

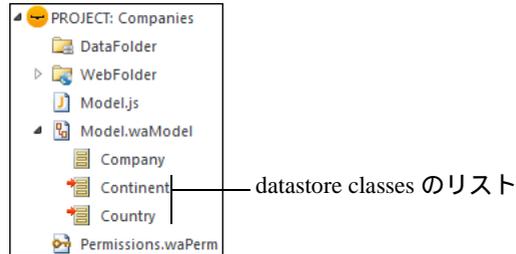
#### model

*model* オブジェクトは、Wakanda アプリケーションのカレントの「モデル」をあらわします。つまり、Wakanda の "datastore classes"(テーブル)とメソッド一式のことです。4D Mobile アーキテクチャにおいては、Wakanda モデルは空であっても構いません。Wakanda アプリケーションにオブジェクトが含まれる場合、リモート 4D アプリケーションから参照されたクラスとメソッドはローカルのモデルと組み込みされます。

接続が正常に確立されると、「公開」されている 4D テーブルが Wakanda 側のモデルのクラスのリストに表示されます。Wakanda Enterprise Studio 側では、リモートテーブルがローカルモデルのクラスのリストの中に表

示されているのが確認できます。これらにはリモートモデルであることを示す赤い矢印がついています。

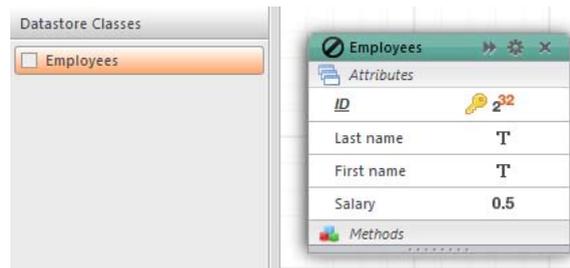
Wakanda Studio - エクスプローラー



外部カタログは Wakanda Studio 側でも *localName.waRemoteCatalog* というファイル名で表示されます。

 Emp4D.waRemoteModel

このファイルをダブルクリックすると Wakanda Studio のモデルエディターの中に外部カタログが表示されます。



## jsFile

*jsFile* プロパティにはモデルと同じフォルダ内にある JavaScript ファイルへの相対パスを渡すことができます。このファイルは外部カタログが組み込まれた後、Wakanda によって実行されます。また、このファイルはカスタマイズ、最適化、セキュリティなどの目的のためにモデルのローカルバージョンを変更することができます。さらに、このファイルを使用することによって以下の様なことが可能です。

- イベントやスコープなどのデータスコアクラス属性のプロパティを変更できます。以下の様に記述します。

```
model.className.attributeName.scope = "publicOnServer"
```

- データスコアクラスに計算属性を追加することができます。以下の様に記述します。

```
model.className.attributeName.onGet = function()
model.className.attributeName.onSet = function()
```

- 外部カタログのテーブルから派生させたローカルのデータストアクラスを作成し、クライアントへ送られるデータを完全に管理することができます。派生されたデータストアクラスは外部テーブルのカスタムビューを表示することができる一方、Wakanda Server 上の拡張された（親の）データストアクラスへもアクセスすることができます。以下の様に記述します。

```
model.DerivedClass = new DataClass("Emps", "public", "My4DTable")
```

- セキュリティのため、またはネットワークトラフィックを最適化するために、派生したデータストアクラスから属性を除去することができます。以下の様に記述します。

```
model.DerivedClass = new DataClass("Emps", "public", "My4DTable")
model.DerivedClass.removeAttribute("salary");
model.DerivedClass.removeAttribute("comments");
model.DerivedClass.removeAttribute("...");
```

上記のコードは、"My4DTable" をもとに派生した "DerivedClass" という名前のクラスを作成し、このクラスはネットワークを使用して必要な属性のみを送ります。

モデルと組み合わせて使用できる JavaScript コードに関しては、Wakanda のドキュメントの中の [Model](#) の章を参照して下さい。

- ▶ ダイレクト接続の例 :

```
model.mergeOutsideCatalog("base4D", "localhost:80", "admin", "123456");
```

- ▶ オブジェクトを使用した接続例 :

```
model.mergeOutsideCatalog("base4D", {
  hostname: "http://localhost:8050",
  user: "wak",
  password: "123456",
  jsFile: "Model2.js"
  timeout: 15 });
```

## 4D テーブルの呼び出し

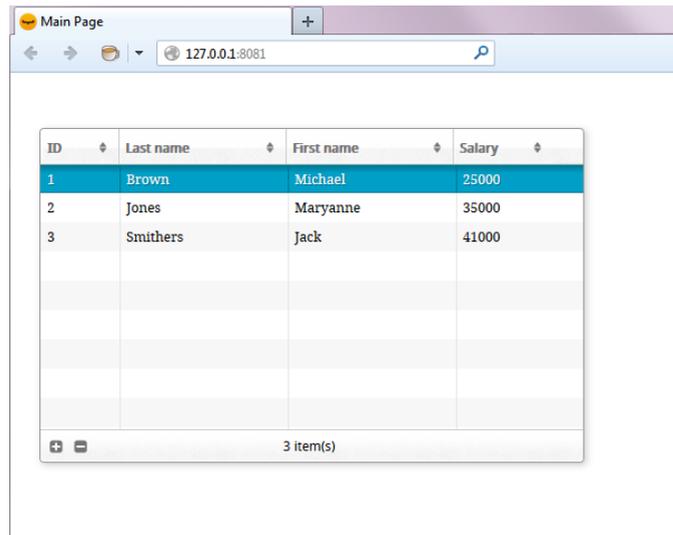
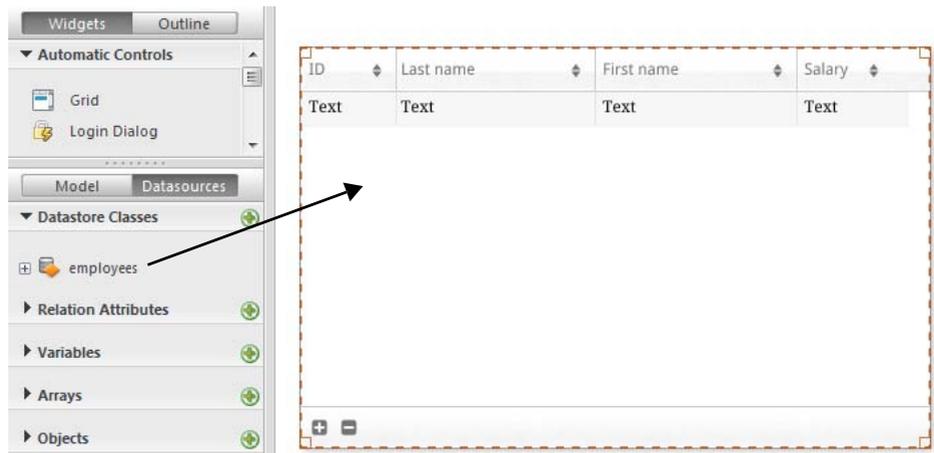
Wakanda アプリケーションから参照されている 4D テーブルは、データストアクラスと同じように、**ds** オブジェクトのプロパティとして JavaScript のコードの中で直接使用することができます。

*Note* **ds** オブジェクトは Wakanda のカレントのデータストアを内包しています。

例えば、[Employees] テーブルのレコード内でクエリを実行しようとした場合、以下の様に記述します。

```
var emp = ds.Employees.query("age > :1",30);
// Employees テーブルから、年齢が 30 歳を超える
// レコードのコレクションを emp 変数に返します。
```

また、ウィジェット付随の、データストアクラスに基づいたデータソース自動メカニズムを使用することもできます。例えば、‘employees’ データソースを ‘Grid’ 型のウィジェットと関連付けると employees のリストが自動的に表示されます。



テーブルがデータソースと関連付けられているとき、データソースを使用してテーブルのデータにアクセスすることもできます。例えば、

‘employees’ データソースのレコードのコレクションをソートしたい場合、以下の様に記述します。

```
sources.employees.orderBy("age");
//employees のコレクションを年齢順にソートします。
```

datasource クラスの仕様については、Wakanda のドキュメントを参照して下さい (<http://doc.wakanda.org>)。

## 4D メソッドの呼び出し

### スコープとオブジェクト

Wakanda 内で参照されている 4D メソッドは、**datastore class**、**entity collection** または **entity** オブジェクトのプロパティとして JavaScript のコードの中で直接使用することができます。どれのプロパティとして呼び出されるかは 4D 側で定義されたスコープによって決まります (127P “プロジェクトメソッドのペアレントテーブルとスコープについて” を参照して下さい)。Wakanda オブジェクトとプロジェクトメソッドの対応表は以下の様になります。

4D スコープ	Wakanda オブジェクト
テーブル	datastore class
カレントセレクション	entity collection
カレントレコード	entity

*Note* 4D メソッドはデータソースを使用することによってクライアント側で呼び出すことも可能です (以下を参照して下さい)。この場合は全てのメソッドが使用可能で、データソースが状況に応じてカレントコレクションかカレントエンティティに適用するかを自動的に判別します。

例えば、前章で使用したクエリメソッドを使用してクエリを実行した場合、Wakanda はエンティティコレクションを返します。このコレクションに対しては、スコープが「カレントセレクション」と宣言されている 4D プロジェクトメソッドであればどれでも使用可能です。

### サーバーとクライアント

4D メソッドが JavaScript から呼び出される方法は 3 通りあります。

- **SSJS Datastore API** を使用して (SSJS) サーバー上で JavaScript を実行して呼び出し：この場合、4D メソッドは先に説明のあったように **datastore class**、**entity collection** または **entity** オブジェクトのプロパティとして呼び出されます。以下の様に記述します。

```
var vTot = ds.Emp.raiseSalary(param))
```

- Wakanda Ajax Framework (WAF) を使用して、クライアント上 (ブラウザなど) で実行された JavaScript コードで JavaScript から呼び出し: この場合、使用する API によって二通りの方法があります。

- **WAF Datasource API** を使用する方法: このハイレベルな API はデータを管理するための様々な自動機能を提供します。この API を使用した場合、**datastore classes** に関連付けられた**データソースのプロパティ**として呼び出され、内容に応じて自動的にデータストアークラス、カレントエンティティコレクション、もしくはカレントエンティティに適用されます。メソッドの戻り値やエラーを処理するのであれば、全て非同期シンタックスを使用して管理しなければなりません (クライアントでコードを実行するためには必須です)。記述例としては以下のようになります。

```
sources.employee.raiseSalary(param,
{onSuccess: function(event)
{ ... // メソッド終了時に実行されるべきコード }
}))
```

ここではコールバック関数の使用は必須ではありません。何故ならデータソースオブジェクトはクエリ後のカレントコレクションに合わせて表示を更新するなどの動作をサポートする自動機能があるからです。

- **WAF Dataprovider API** を使用する方法: このローレベルなクライアント API を使用するとオブジェクトを直接扱うことができます。SSJS Datastore API 同様、4D メソッドは **datastore class**、**entity collection** または **entity** オブジェクトのプロパティとして呼び出されます。しかしながらメソッドの戻り値やどのエラーも、非同期シンタックスを使用して管理しなければなりません (クライアントで実行されるコードのためには必須です)。記述例としては以下のようになります。

```
ds.Employee.raiseSalary(param, // シンタックスは SSJS の呼び出しに
{onSuccess: function(event)// 似ていますが、これはクライアント側の
// コードなので非同期呼び出しのコールバック関数を管理する必要があります。
{ ...// メソッド終了時に実行されるべきコード }
}))
```

呼び出す場所 (サーバーかクライアント) と、使用すべき API はアプリケーションによって異なり、その詳細は Wakanda ドキュメントに説明があります。

## 引数

標準的なメソッド同様、呼び出し中にメソッドに引数を渡す事ができません。これらの引数は \$1、\$2、... という順番で引き受けられていきます。同じように、\$0 がメソッドからの返り値になります。

- ▶ 給料が 1500 未満の従業員に対して 5% の昇給を行いたい、という場合を考えます。
  - 4D 側では、*IncreaseSalary* メソッドを REST サーバーに公開し、スコープを「カレントセクション」に設定して、コードを以下の様に記述します。

```
C_REAL($1)
READ WRITE ([Employees])
FIRST RECORD ([Employees])
While (Not (End selection ([Employees])))
  [Employees]salary:=[Employees]salary*$1
  SAVE RECORD ([Employees])
  NEXT RECORD ([Employees])
End while
UNLOAD RECORD ([Employees])
```

- Wakanda 側では、以下のコードをサーバー上で実行します。

```
var emp = ds.Employees.query("salary < :1",1500);
// emp には salary が 1500 未満の従業員のコレクションが入ります。
emp.IncreaseSalary(1.05);
// コレクションに対して IncreaseSalary を実行します。
// 以下の様に記述することもできます :
//"ds.Employees.query("salary < :1",1500).IncreaseSalary(1.05);
```

## 4D コンテキストの更新

Wakanda link を使用して 4D メソッドを呼び出す場合、メソッドの適用される範囲によって挙動が以下の様に変わります：

- メソッドが適用される範囲がセクション (*entity collection*) であるとき、そのセクションがカレントセクションとなり、4D はこのセクションの最初のレコードに位置します。セクションのロードやリンクの有効化はしません。セクションが空であるとき、**Selected record number** コマンドは 1 ではなく 0 を返します。
- メソッドが適用される範囲がレコード (*entity*) であるとき、そのレコードがカレントのレコードとなり、読み書き可能な状態でロードされます。カレントセクションはこのレコードにまで狭められ、**Selected record number** コマンドは 1 を返します。
- メソッドが適用される範囲がテーブル (*datastore class*) であるとき、カレントセクションもカレントレコードも、どちらも影響されません。

メソッドを REST を通じて実行したあと、4D コンテキストは以下の様にリセットされることに注意して下さい：

- セレクションは 0 に減らされます。
- レコードはレコードスタックから取り除かれ、ロードも解除されます。
- プロセス用のローカルセレクションとセットは破壊されます。
- メソッド実行中に開かれたトランザクションは全てキャンセルされません。
- フィールド、クエリ格納先、サーバー上のクエリによる自動リレーションの設定がリセットされます。
- 印刷ジョブはキャンセルされます。
- ウィンドウは閉じられます。
- 全ての SQL, PHP, または HTTP 接続は閉じられます。

## スコープエラー

4D メソッドのスコープは、それを呼び出す Wakanda オブジェクトの方と対応し、合致している必要があります。そうでない場合には *"TypeError: 'undefined' is not a function"* というエラーが Wakanda によって返されます。

例えば、以下のコードによって記述された `getcursel` という 4D メソッドについて考えてみましょう。

```
$0:=Records in selection([Table_1])
```

Wakanda 側に以下の `run` というメソッドがあったと仮定します。

```
var tt = ds.Table_1.query("Field_2 = 'a*").getcursel();
```

`query()` メソッドはコレクションを返します。もし `getcursel` メソッドのスコープが「カレントレコード」に設定されていた場合、Wakanda は以下のエラーを返します。

```
TypeError: 'undefined' is not a function (evaluating 'ds.Table_1.query("Field_2 = 'a*').getcursel()')
```

## `openRemoteStore()` と `addRemoteStore()`

Wakanda と 4D 間の動的なリンクは、`openRemoteStore()` と `addRemoteStore()` を使うことによっても設定することができます。

これらのメソッドは `mergeOutsideCatalog()` のように、4D データベースのデータへのダイナミックなアクセスを可能にしますが、仕組みが異なります。

- これら二つのメソッドは Wakanda セッション中であればソリューションがロードされたときでなくてもいつでもリモートモデルを参照することができます。

- 外部モデルのテーブル、属性、メソッドは、個別のデータストアを使用してアクセス可能です。Wakanda アプリケーションのローカルモデル (ds オブジェクトによってアクセス可) と統合はされません。

`openRemoteStore()` はカレントの JavaScript のコンテキストの中でのみ有効な参照を返しますが、`openRemoteStore()` はセッションの間はずっと参照を維持し続けます。

より詳細な情報に関しては、[Wakanda documentation](#) の `openRemoteStore()` と `addRemoteStore()` についての説明を参照してください。

## 4D Mobile のセキュリティについて

4D データベースのテーブルから REST を通じて公開されたデータが Wakanda カタログと統合されたあとは、一部のデリケートなリソースに関してはアクセスを制限する必要があります。

4D とは違い、Web アプリケーションではインターフェースを使用して公開されているデータを管理することはできません。例えば、あるフィールドが表示されていないからといって、それがユーザーからアクセスできないわけではない、ということです。HTTP リクエストと JavaScript を使用することで、悪意あるユーザーがプロテクトが不完全な Web サーバーから自由にデータを取得してしまう事態も起こり得ます。

この章では 4D Mobile アプリケーションにおいてセキュリティ面で取るべき全ての対策を挙げているわけではないですが、公開しているデータを保護するために最低限必要な情報がまとめられています。

- **4D データベースへの REST アクセスの保護** : REST 接続のリクエストは保護されている必要があります。以下二つのどちらかを使用しましょう。
  - 4D パスワード([4D パスワード使用した自動コントロール](#)を参照のこと)
  - [On REST Authentication](#) データベースメソッド
- **4D 側で REST サーバーへの公開を管理** : REST サーバーへの公開・非公開はそれぞれのテーブル、属性、そしてメソッドごとに設定することができます。本当に必要なデータとメソッドのみ公開するようにしましょう。例えば、使用していないフィールド等は公開する必要はありません。
- **公開されているデータの保護** : ブラウザ経由で公開されているデータに関しては、Wakanda のセキュリティシステムを使って管理して下さい。以下の様にいくつかの手段があります (同時に複数併用することも可能です)。
  - **スコープの調整** : 属性やメソッドのスコープを 4D データベースから調整します。特に、スコープを **Public on Server** に設定するとサーバーが

らはコード実行のために自由にアクセスできますが、Web クライアントからはアクセスできなくなります。

この設定は `mergeOutsideCatalog()` メソッドによって実行される追加の `.js` ファイルによって設定することができます。詳細な情報に関しては Wakanda ドキュメントの [Model API](#) を参照して下さい。

- **計算属性を使用** : 計算属性は標準の属性と同じように使用できますが、その値には特定の関数 (`onGet`、`onSet` 等) を通してのみアクセスできません。これはつまり、4D データベースのフィールドを直接公開せずに必要な計算属性のみを公開するといったことができるということです。4D フィールドへのアクセスは Wakanda サーバーから安全な方法で実行されます。

動的な属性は、`mergeOutsideCatalog()` メソッドによって実行される追加の `.js` ファイルへと追加することができます。詳細な情報に関しては Wakanda ドキュメントの [Attributes](#) を参照して下さい。

# 6

# ランゲージ

この章では 4D の v14 で新しく追加された、または変更されたランゲージについて説明しています。

## 4D 環境

### 新コマンド

#### Get last update log path

Get last update log path → String

引数	型	説明
----	---	----

戻り値	文字列	← 直近のアップデートログへのパス名
-----	-----	--------------------

[Get last update log path](#) コマンドは、呼び出されたマシンで見つかった、直近のアップデートログファイルへの完全なパス名を返します。

アップデートログは、自動アップデートプロセスのために 4D が使用するアップデーターツールによって生成されます。これには実行されたアップデートに関する情報に加え、発生したエラーの情報も保存されています。

*See also:* [RESTART 4D](#), [SET UPDATE FOLDER](#)

#### RESTART 4D

RESTART 4D {( time{; message} )}

引数	型	説明
----	---	----

time	倍長整数	→ 4D を再起動するまでのディレイ (秒)
------	------	------------------------

message	文字列	→ クライアントマシンに表示するテキスト
---------	-----	----------------------

[RESTART 4D](#) コマンドは、カレントの 4D 組み込みアプリケーションを再起動させます。

このコマンドは、組み込みアプリケーション（クライアント / サーバーまたはシングルユーザー）においての使用を主眼においており、**SET UPDATE FOLDER** と組み合わせて使用しなければなりません。これにより自動アップデートプロセスは起動し、**RESTART 4D** によってアプリケーションが再起動した時点で、**SET UPDATE FOLDER** によって指定されたアプリケーションの新バージョンがカレントのバージョンを置き換えます。データファイルへのパス名は自動的に保存され、使用されます。

*Note* カレントセッションにおいて **SET UPDATE FOLDER** コマンドを使用する際に何のアップデート情報も指定しなかった場合、このコマンドは同じデータファイルで組み込みアプリケーションを再起動します。

**RESTART 4D** コマンドはこれ以外での使い方はできません。特に組み込まれていない 4D データベースには効力がありません。

*time* 引数を用いて、クライアントマシンが接続を切断するために再起動までの時間差をつけることができます。*time* 引数に指定する時間の値を秒数で渡します。この引数を省略した場合、サーバーアプリケーションは 10 分を上限として全てのクライアントアプリケーションが接続を切断するまで待ちます。それを過ぎると全てのクライアントアプリケーションは自動的に接続が切断されます。

*Note* *time* 引数は、サーバーアプリケーションに対してのみ有効です（シングルユーザーアプリケーションの場合はこれは無視されます）。

任意の *message* 引数は、接続したクライアントアプリケーションに表示するメッセージ指定します。

コマンドが正常に実行されれば、OK システム変数は 1 に設定されます。それ以外の場合には変数は 0 に設定され、アプリケーションは再起動されます。このコマンドによって生成されたエラーはどんなものでも ON ERR CALL コマンドを使って実装したメソッドによって割り込み可能です。

*See also:* **SET UPDATE FOLDER**

## SET UPDATE FOLDER

SET UPDATE FOLDER ( folderPath{; silentErrors})

引数	型	説明
folderPath	文字列	→ アップデートしたアプリケーションを格納するフォルダ (OS X ではパッケージ) へのパス名
silentErrors	ブール	→ False (デフォルト) = エラーメッセージを表示 True = エラーログのみ記録する (表示をしない)

新しい **SET UPDATE FOLDER** コマンドは、カレントの 4D 組み込みアプリのアップデートを格納しているフォルダを指定します。この情報は **RESTART 4D** が呼び出されるまで、4D セッション内に保存されます。アプリケーションが手動で終了された場合、情報は保存されません。

このコマンドは組み込みアプリケーションの自動アップデート処理に使われることを想定しています (サーバーあるいは単一のユーザー)。詳細な情報に関しては、37P “サーバーアプリケーションのアップデートの自動化” を参照して下さい。

---

*Note* このコマンドは、4D Volume Desktop にて組み込みされた単一ユーザー向けアプリか 4D Server においてのみ動作します。

---

*folderPath* 引数には、組み込みアプリの新しいバージョンのフォルダへの完全なパス名を渡します (Win 環境下では my4DApp.exe アプリケーション、または OS X 環境下では my4DApp.app パッケージを含んでいるフォルダ)。この新しいバージョンは 4D v14 アプリケーションビルダーで生成されている必要があります。細かく言うと、リモートアップデートを管理するためのアップデーターツールのカレントのバージョンを含んでいる必要があります (詳細な情報は 37P “アプリケーションビルダー” を参照して下さい)。

---

*Note* アプリケーションフォルダはアップデートの際に置換されることから、アプリケーションの新しいバージョンのファイルはオリジナルのものと同一名前を使用することを推奨します。これらのファイルに異なる名前を使用した場合、保存されたショートカットやパスは全て作動しなくなります。

---

引数が有効であれば、アップデートは **RESTART 4D** コマンドが呼び出されるまで、セッション中は「ホールド」された状態になります。**SET UPDATE FOLDER** を、**RESTART 4D** コマンドを呼び出す前に複数回呼び出した場合は、最後の有効な呼び出しが使われます。

予期せぬ事態が起きた場合にはエラーが生成されます。その際、エラーを表示するかどうかは *silentErrors* 引数によって決まります (以下参照)。

*folderPath* 引数に空の文字列 ("") を渡す事により、カレントのセッション中のアップデート情報をリセットすることができます。

任意の *silentErrors* 引数は、エラーがアップデーターによってどのように表示されるかを指定します。

- この引数に **False** を渡すかこれを省略した場合、エラーはアップデートジャーナルに記録され、警告ダイアログボックスが表示されます。

- **True** を渡した場合、エラーはアップデータジャーナルに記録されるにとどまり、何も表示は出ません。

例外：アップデータツールがジャーナルファイルを作成することが出来なかった場合は、*silentErrors* 引数の値に関わらず警告ダイアログボックスが表示されます。

このコマンドが正常に実行された場合、OK システム変数は 1 に設定されます。それ以外の場合には 0 に設定されます。このコマンドから生成されるエラーは全て ON ERR CALL コマンドを使用して実装したメソッドで割り込み可能です。

- ▶ "MyUpdates" フォルダをディスク上に作成し、その中に "MyApp" アプリケーションの新しいバージョンを置いたとします。エラーを表示しないようにアップデートを準備したい場合、以下の様に記述します。

```
// Windows シンタックス
SET UPDATE FOLDER("...MyUpdates"+Folder separator+ "MyApp";True)
```

```
// OS X シンタックス
SET UPDATE FOLDER("...MyUpdates"+Folder separator+
"MyApp.app";True)
```

See also: [RESTART 4D](#)

## 修正されたコマンド

### GET MEMORY STATISTICS

```
GET MEMORY STATISTICS ( infoType ; arrNames ; arrValues ;
arrCount )
```

The [GET MEMORY STATISTICS](#) コマンドは、*infoType* 引数にセレクター 1 を渡した場合に返される一般的なメモリ情報にさらに 2 つ追加で情報を返すようになりました。 *stack memory* と *free stack memory* です。

### SET DATABASE PARAMETER, Get database parameter

```
SET DATABASE PARAMETER ({table; }selector; value)
```

```
Get database parameter ({table; }selector{; stringValue}) → Real
```

[SET DATABASE PARAMETER, Get database parameter](#) は3つの新しいセレクターを受け入れるようになりました。また、セレクター 34 が修正されています。

- **Selector = 34** (*Debug log recording*)

4D v14 より、新しくてよりコンパクトでタブがつけられたテキストがイベントログファイルで使用されるようになりました。このログファイルは "4DDebugLog[\_n].txt" と名前がつけられます (\_n にはファイルのセグメント番号が入ります)。セレクター 34 は、この新しいフォーマットを有効化して設定可能にするために、機能が一部変更されています。

- *Values:* bit フィールドを含む倍長整数が入ります : value = bit1(1)+bit2(2)+bit3(4)+bit4(8)+...).
  - Bit 1 (value 1) はファイルの有効化をリクエストします (他のヌルでない値もファイルを有効化することに注意して下さい)。
  - Bit 2 (value 2) はメソッドとコマンドに引数を呼び出すことをリクエストします。
  - Bit 3 (value 4) 新しいタブ付されたフォーマットを有効化します。
  - Bit 4 (value 8) それぞれのオペレーションのディスクへの直接書き込みを禁止します (デフォルトでは許可されています)。直接書き込みは遅いですがクラッシュの原因を調査する際などは効果的です。このモードを禁止すると、ファイルの中身はよりコンパクトになりより早く生成されます。
  - Bit 5 (value 16) プラグインの呼び出しを記録することを禁止します (デフォルトでは許可されています)。

(*Compatibility note:* 実行時間は v14 からは常にファイルに含まれています。)

例 :

SET DATABASE PARAMETER (34;1) // モード v13 のファイルを、引数なし、ランタイム付で有効化します。

SET DATABASE PARAMETER (34;2) // モード v13 のファイルを、引数とランタイムとともに有効化します。

SET DATABASE PARAMETER (34;2+4) // ファイルを v14 フォーマットで引数とランタイムとともに有効化します。

SET DATABASE PARAMETER (34;0) // ファイルを無効化します。

このフォーマットと 4DDebugLog[\_n].txt の使用方法についてのより詳細な情報に関しては、4D のテクニカルサポートまでお問い合わせ下さい。

- **Selector = 81 (スペルチェッカー)**
  - *Description:* OS X 環境下においてハンスペル辞書を有効にします。
  - *Values:* 0 (デフォルト) = OS X のシステムのスペルチェッカーを使用 (ハンスペルは無効化)、1 = ハンスペルを有効化  
この引数は OS X 環境下においてハンスペルのスペルチェッカーを使用可能にします。デフォルトではシステムのスペルチェッカーが使用される設定となっています。例えば、クロスプラットフォームアプリケーションのインターフェースを統一するためなどにハンスペル辞書を使用するケースなどが考えられます。(Windows 環境下ではハンスペ

ルのスペルチェッカーのみ使用可能です。) 4D v14 のスペルチェックに関する詳細は、86P “[スペルチェック](#)” (インターフェイス) と 261P “[スペルチェッカー](#)” (ランゲージ) を参照して下さい。

- **Selector = 82** (QuickTime support)
  - *Description:* QuickTime サポートを起動します。
  - *Values:* 0 (default) = QuickTime は無効 1 = QuickTime を有効にする  
4D v14 では、デフォルトとして QuickTime コーデックはサポートされていません。(廃止予定の機能です。18P “[QuickTime](#)” 参照して下さい。) 互換性のために、このセレクターを使用してアプリケーションの中で QuickTime コーデックを再度有効にすることが出来ます。しかしながら、4D では将来 QuickTime サポートが一切なくなる予定であることに注意する必要があります。
- **Selector = 85** (JSON use local time)
  - *Description:* 4D の日付と JSON 間での変換モードを選択します。
  - *Values:* 0 = ローカルのタイムゾーンを無視する、1 (デフォルト) = タイムゾーンを考慮に入れる  
4D v14 では 4D の日付を JSON 形式に変換する際、標準時を考慮するようになりました。例えば、!23/08/2013! という日付を JSON 形式に変換すると、サマータイム中のフランス (GMT+2) においては "2013-08-22T22:00:00Z" という結果が得られます。これによって JavaScript そのまま使える形式になります。  
この変換は、JSON の日付の値を他のタイムゾーンにいる相手に送るときにエラーの原因となり得ます。例えば、フランスで [Selection to JSON](#) を使用してテーブルを変換し、それをアメリカで [JSON TO SELECTION](#) を使用して再度変換しようとしたときなどに起こります。デフォルトでは、日付はタイムゾーンごとに異なるので、データベースに保存されている値はタイムゾーンごとに異なります。こういった場合、このセレクターに 0 を渡す事により、タイムゾーンを無視する変換モードへと切り替えることができます。こうすることにより、!23/08/2013! というどんな場合でも "2013-08-23T00:00:00Z" を返すようになります。
- ▶ 4D の日付を JSON へと変換する場合を考えます。データの変換は日付がオブジェクトに保存されたときに起こるので、SET DATABASE PARAMETER コマンドは [OB SET](#) を呼び出す前に呼び出さなければならないことに注意して下さい。

**C\_OBJECT(\$o)**

```

SET DATABASE PARAMETER(JSON use local time; 0)
OB SET( $o ; "myDate" ; Current date) // JSON への変換
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(JSON use local time; 1)

```

■

## Version type

Version type → Longint

戻り値                      倍長整数                      ← デモ版か製品版か、64-bit 版か 32-bit 版か、  
4D データベースかビルドされたアプリか

このコマンドにおいて、"4D Environment" テーマに追加された新しい定数を使用することによって、カレントのバージョンがビルドされたアプリなのかそうでないのかを判別します。:

Constant ( <i>value</i> )	Comments
<u>Merged application</u> (2)	このバージョンは 4D エンジンによってビルドされたアプリです。

- ▶ これにより、そのバージョンが 4D によってビルドされたアプリなのか 4D サーバーによって開かれたデータベースなのかを判別し、それに応じて違うコードを実行する、ということが出来るようになります。:

**If** (**Version type** ?? Merged application)

// ここにビルドされたアプリ用のコードを書きます。

**Else**

// ここに 4D によって実行されたデータベース用のコードを書きます。

**End if**

■

## 配列

### ARRAY BLOB

ARRAY BLOB ( arrayName ; size { ; size2 } )

引数	型	説明
arrayName	配列	→ 配列名
size	倍長整数	→ 配列の要素の数、もしくは size2 が指定されていた場合は配列の数
size2	倍長整数	→ 2次元配列の要素の数

新しい **ARRAY BLOB** コマンドはメモリ上の BLOB 型の要素を持つ配列を作成・リサイズします。

- *arrayName* 引数には配列の名前を渡します。
- *size* 引数には、配列の要素の数を渡します。
- *size2* 引数は任意の引数です。指定時にはコマンドは 2 次元配列を作成します。この場合、*size* 引数はそれぞれの配列の行の数を指定し、*size2* 引数はそれぞれの配列の列の数を指定します。2 次元配列内のそれぞれの行は、要素としても配列としても扱うことができます。これはつまり、配列の 1 次元目を扱うときは、このテーマ内の他のコマンドを使用することによって 2 次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

**ARRAY BLOB** コマンドを既存の配列に対して使用する場合、以下のことに気を付けてください。:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素は空の BLOB で初期化されます (BLOB サイズ = 0)
  - サイズを縮小する場合は、" 底 " にある要素から削除されていきます。
- ▶ 以下の例は BLOB 型の要素を 100 個含んだプロセス配列を作成します。:
- ```
ARRAY BLOB (arrBlob;100)
```
- ▶ 以下の例は 50 個の BLOB 型の要素を含んだ行を 100 行持ったローカルな配列を作成します。:
- ```
ARRAY BLOB ($arrBlob;100;50)
```
- ▶ 以下の例は 50 個の BLOB 型の要素を含んだ行を 100 行持ったローカルな配列を作成します。*\$vByteValue* 変数には、その BLOB 配列の 5 行目・7 列目の、10 バイト目の BLOB が渡されます。:

```
C_INTEGER ($vByteValue)
ARRAY BLOB ($arrValues;100;50)
...
$vByteValue:=$arrValues{5}{7}{9}
```

## ARRAY OBJECT

ARRAY OBJECT ( *arrayName* ; *size* { ; *size2* } )

引数	型	説明
<i>arrayName</i>	配列	→ 配列名
<i>size</i>	倍長整数	→ 配列の要素の数、もしくは <i>size2</i> が指定されていた場合は配列の数
<i>size2</i>	倍長整数	→ 2 次元配列の要素の数

新しい **ARRAY OBJECT** コマンドはメモリ上のランゲージオブジェクト型の要素を持つ配列を作成・リサイズします。

---

ランゲージオブジェクトとは 4D v14 からサポートされるようになった新しいデータの型です。詳細に関しては 240P “**オブジェクト (ランゲージ)**” を参照して下さい。

---

- *arrayName* 引数には配列の名前を渡します。4D のルールに則った名前であればどんな名前でも使用することができます。
- *size* 引数には、配列の要素の数を渡します。
- *size2* 引数は任意の引数です。指定時にはコマンドは 2 次元配列を作成します。この場合、*size* 引数はそれぞれの配列の行の数を指定し、*size2* 引数はそれぞれの配列の列の数を指定します。2 次元配列内のそれぞれの行は、要素としても配列としても扱うことができます。これはつまり、配列の 1 次元目を扱うときは、このテーマ内の他のコマンドを使用することによって 2 次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

**ARRAY OBJECT** コマンドを既存の配列に対して使用する場合、以下のことに注意して下さい。:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素は未定義の要素になります。**OB Is defined** コマンドを使用することによって要素が定義済みかどうかを調べることができます。
  - サイズを縮小する場合は、“底”にある要素から削除されていきます。
- ▶ 以下の例はオブジェクト型の要素を 100 個含んだプロセス配列を作成します。:

```
ARRAY OBJECT (arrObjects;100)
```

- ▶ 以下の例は 50 個のオブジェクト型の要素を含んだ行を 100 行持ったローカルな配列を作成します。:

```
ARRAY OBJECT ($arrObjects;100;50)
```

- ▶ 以下の例はローカルなオブジェクト配列を作成してデータをそこに代入します。:

```
C_OBJECT ($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT ($arrayChildren;0)
OB SET ($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY ($arrayChildren;$ref_richard)
OB SET ($ref_susan;"name";"Susan";"age";4)
```

```

APPEND TO ARRAY ($arrayChildren;$ref_susan)
OB SET ($ref_james;"name";"James";"age";3)
APPEND TO ARRAY ($arrayChildren;$ref_james)
// $arrayChildren{1} -> {"name":"Richard","age":7}
// $arrayChildren{2} -> {"name":"Susan","age":4}
// $arrayChildren{3} -> {"name":"James","age":3}

```

*See also:* C\_OBJECT, オブジェクト (ランゲージ)

## ARRAY TIME

ARRAY TIME ( arrayName ; size {; size2} )

引数	型	説明
arrayName	配列	→ 配列名
size	倍長整数	→ 配列の要素の数、もしくは size2 が指定されていた場合は配列の数
size2	倍長整数	→ 2次元配列の要素の数

**ARRAY TIME** コマンドはメモリ上の 時間型の要素を持つ配列を作成・リサイズします。

4D では時間は数の値として処理されるということに注意して下さい。以前のバージョンの 4D では、時間の配列を管理するためには倍長整数配列と表示フォーマットを組み合わせる必要がありました。

- *arrayName* 引数には配列の名前を渡します。
- *size* 引数には、配列の要素の数を渡します。
- *size2* 引数は任意の引数です。指定時にはコマンドは 2 次元配列を作成します。この場合、*size* 引数はそれぞれの配列の行の数を指定し、*size2* 引数はそれぞれの配列の列の数を指定します。2 次元配列内のそれぞれの行は、要素としても配列としても扱うことができます。これはつまり、配列の 1 次元目を扱うときは、このテーマ内の他のコマンドを使用することによって 2 次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

**ARRAY TIME** コマンドを既存の配列に対して使用するときには以下の点に注意して下さい。:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素は null 時間の値 (00:00:00) で初期化されます。
- サイズを縮小する場合は、" 底 " にある要素から削除されていきます。

**SELECTION TO ARRAY, SELECTION RANGE TO ARRAY** のコマンドを *Time* 型のフィールドに対して適用した場合、これらのコマンドはコピー

先の配列が他の型（例えば倍長整数など）に定義されていない場合に限り、時間型の配列を作成します。

*Note* **Time** 機能は 4D v14 から秒数を現す数字の引数を受け取ることが出来るようになりました。

- ▶ 以下の例は時間型の要素を 100 個含んだプロセス配列を作成します。:

```
ARRAY TIME (arrTimes;100)
```

- ▶ 以下の例は 50 個の時間型の要素を含んだ行を 100 行持ったローカルな配列を作成します。:

```
ARRAY TIME ($arrTimes;100;50)
```

- ▶ 時間の配列が数字の値を受け取ることが出来るようになったことによつて、以下のようなコードを使用することが出来るようになります。:

```
ARRAY TIME ($arrTimeValues;10)
```

```
$CurTime:=Current time+1
```

```
APPEND TO ARRAY ($arrTimeValues;$CurTime)
```

```
$Found:=Find in array ($arrTimeValues;$CurTime)
```

*See also:* [Time](#)

## ARRAY TO LIST

```
ARRAY TO LIST ( array ; list { ; itemRefs } )
```

引数	型	説明
array	配列	→ 配列要素のコピー元配列
list	文字列 / ListRef	→ 配列要素のコピー先リストの名前または参照
itemRefs	配列	→ 項目参照番号の数値配列

**ARRAY TO LIST** コマンドは 4D v14 から *list* 引数に *ListRef* (倍長整数) を受け取ることが出来るようになりました。これにより、階層リストを直接配列へと変換することが出来るようになりました。

このコマンドが正常に動作するためには、コピー先のリストは **New list** コマンドなどを使用して既に作成されている必要があります。

このコマンドの動作により、*list* で指定したリストの第一要素レベルの要素のみ *array* 引数のデータで定義することが出来ます。

- ▶ フィールドの異なる値をリストに入れて、例えば階層ポップアップメニューを作成したい場合を考えます。その場合、以下の様を書くことが出来ます。:

```
ALL RECORDS ([Company])
DISTINCT VALUES ([Company]country;$arrCountries)
CountryList:=New list
ARRAY TO LIST ($arrCountries;CountryList)
```

*See also:* [LIST TO ARRAY](#)

## LIST TO ARRAY

LIST TO ARRAY ( *list* ; array {; itemRefs} )

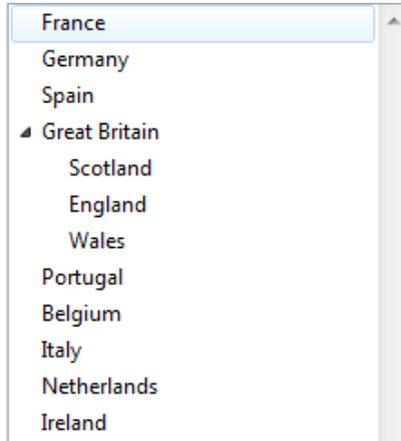
引数	型	説明
<i>list</i>	文字列 / <i>ListRef</i>	→ 一番目の項目をコピーするコピー元のリスト名または <a href="#">参照</a>
array	配列	→ コピー先の配列
itemRefs	配列	→ リスト項目の参照番号

[LIST TO ARRAY](#) コマンドは *list* 引数において *ListRef* ( 倍長整数 ) を受け取れるようになりました。これにより、階層リストを直接配列へと変換することが出来るようになりました。コマンドの動作により、リストの最初のレベルの項目のみが *array* で指定された配列へと書き出されます。指定した配列が存在しない場合、コマンドによって自動的に作成されリサイズされます。

- ▶ 以下のように定義された階層リストについて考えます。:

```
MyList2:=New list
APPEND TO LIST (myList2;"Scotland";1)
APPEND TO LIST (myList2;"England";2)
APPEND TO LIST (myList2;"Wales";3)
myList1:=New list
APPEND TO LIST (myList1;"France";1)
APPEND TO LIST (myList1;"Germany";2)
APPEND TO LIST (myList1;"Spain";3)
APPEND TO LIST (myList1;"Great Britain";4;MyList2;True)
APPEND TO LIST (myList1;"Portugal";5)
APPEND TO LIST (myList1;"Belgium";6)
APPEND TO LIST (myList1;"Italy";7)
APPEND TO LIST (myList1;"Netherlands";8)
APPEND TO LIST (myList1;"Ireland";9)
```

このリストは以下の様に表示されます：



これに対し以下の宣言を実行すると、：

```
LIST TO ARRAY (myList1;$MyArray)
```

... 結果は以下の様になります。：

```
$MyArray{1} = "France"  
$MyArray{2} = "Germany"  
$MyArray{3} = "Spain"  
$MyArray{4} = "Great Britain"  
$MyArray{5} = "Portugal"
```

...

*See also:* [ARRAY TO LIST](#)

## **SELECTION TO ARRAY,SELECTION RANGE TO ARRAY**

4D v14 では、新しい [ARRAY TIME](#) コマンドを使用して時刻型の配列を作成することが出来ます。

SELECTION TO ARRAY と SELECTION RANGE TO ARRAY コマンドを `w` と `Time` 型のフィールドに対して適用した場合、これらのコマンドはコピー先の配列が他の型に定義されていない場合に限り、時間型の配列を作成します。例えば、以下の例の中で、`myArray` 配列は実行後も倍長整数型の配列として残ります。：

```
ARRAY LONGINT (myArray; 0)  
SELECTION TO ARRAY ([myTable]myTimeFielde; myArray)
```

*See also:* [ARRAY TIME](#)

# バックアップ

## INTEGRATE MIRROR LOG FILE

INTEGRATE MIRROR LOG FILE ( *pathName* {; *operationNum* } )

引数	型	説明
<i>pathName</i>	テキスト	→ 統合されるログファイルの名前もしくはパス名
<i>operationNum</i>	倍長整数変数	→ 統合が開始されるオペレーションの番号 ← 最後に統合されたオペレーションの番号

新しい **INTEGRATE MIRROR LOG FILE** コマンドは、*pathName* で指定したログファイルの、任意の引数 *operationNum* 以降のオペレーションを、4D Server データベースへと統合します。このコマンドはどんなログファイルをもデータベースに統合することができます（たとえログファイルがデータファイルと対応していなくても受け入れます）。このコマンドは特にミラーデータベースのコンテキストで使用することを目的としています。

4D v14 以降、ログファイルを " ミラー " データベースの一部として使用することができます。" ログファイルを使用 " のオプションは、論理ミラーとして使用されている 4D Server のデータベース設定においてチェックが出来るようになりました。これにより、ミラーサーバーのミラーを実装することが出来るようになりました。

既存の INTEGRATE LOG FILE コマンドとは異なり、**INTEGRATE MIRROR LOG FILE** コマンドは実行を終了したあとにカレントログファイルを統合されたログファイルで置き換えることはしません。データベースのカレントログファイルは引き続き使用されます。これにより、統合の最中に変更されたものは全てカレントのログファイルへと記録されません。

*pathName* 引数には、データベースフォルダへの絶対パスまたは相対パスを渡します。この引数に空の文字列を渡した場合、標準のファイルを開くダイアログボックスが開くので、そこから統合したいファイルを選択することができます。このダイアログボックスがキャンセルされると、どのファイルも統合されることなく、OK システム変数は 0 に設定されません。

デフォルトで、*operationNum* 引数を省略した場合、コマンドはログファイルの全てのオペレーションを統合します。

*operationNum* 引数に値を渡した場合、統合は指定された値のオペレーションから開始されます。この場合、統合が完了すると *operationNum* 引数には最後に統合されたオペレーションの番号が渡されます。この返された値を次回 **INTEGRATE MIRROR LOG FILE** コマンドを呼び出す際に直接渡す事で、連続したログファイル統合をすることができます。

以下のダイアグラムは異なる場合における統合のプロセスの詳細を説明しています。統合されるログファイルにおいて、X は最初のオペレーションを、Y は最後のオペレーションを表す数値です：

```

Case of
: (operationNum < X-1)
    --> Error 1260 (Selected log file is too recent)
: (operationNum >= X-1) and (operationNum <= Y)
    --> Start integration
    Case of
    : (Type of operation = "Addition")
        If (Record exists)
            --> Do nothing
        Else
            --> Add record
        End if
    : (Type of operation = "Modification")
        If (Record exists)
            --> Edit record
        Else
            --> Do nothing
        End if
    : (Type of operation = "Deletion")
        If (Record exists)
            --> Delete record
        Else
            --> Do nothing
        End if
    End case
Else // operationNum > Y
    --> Error 1261 (Selected log file is too old for database )
End case

```

統合は、エラーが起これるとその時点で中止されてしまうことに注意して下さい。この場合、統合を続行したい場合には MSC を使用する必要があります。

## BLOB

### BLOB TO VARIABLE, VARIABLE TO BLOB

BLOB TO VARIABLE ( blob ; variable { ; offset } )

VARIABLE TO BLOB ( variable ; blob { ; offset | \* } )

これらのコマンドは、ランゲージオブジェクトに対応するために v14 から変更されたものです。(詳しくは [240P “オブジェクト\(ランゲージ\)”](#) を参照して下さい。)

VARIABLE TO BLOB では、C\_OBJECT オブジェクトを *variable* 引数に渡した場合、コマンドが自動的に文字コード UTF-8 の JSON 文字列として BLOB に渡します。

もし C\_OBJECT オブジェクトがポインターを含んでいる場合、ポインターそのものではなくポインターの指している値が BLOB に渡されます。

BLOB TO VARIABLE コマンドは、これとは逆の操作をするコマンドです。

## コンパイラー

### C\_OBJECT

C\_OBJECT({method ;} variable { ; variable2 ; ... ; variableN})

引数	型	説明
method	メソッド	→ メソッド名
variable	変数	→ 宣言したい変数や引数の名前

C\_OBJECT コマンドは、指定した全ての変数をランゲージオブジェクト型であることを宣言します。

---

*Note* オブジェクト型とは、4D v14 以降のランゲージでサポートされるようになった新しいデータの型です。詳細な情報に関しては、[240P “オブジェクト\(ランゲージ\)”](#) を参照して下さい。

---

このコマンドの第 1 の記法は、オプションの *method* 引数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。この形式はインタプリタのデータベースでも使用できます。

コマンドの第 2 の記法は、オプションの *method* 引数が渡される形式であり、メソッドの結果や引数 (\$0、\$1、\$2 等) をコンパイラ用に事前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、この形式は、インタプリタモードで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： `C_OBJECT(${...})` の記法を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、`C_OBJECT(${5})` という宣言は、4D とコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。

*See also:* [ARRAY OBJECT](#), [オブジェクト \(ランゲージ\)](#)

## データベースメソッド

### On Host Database Event

\$1 → On Host Database Event

引数	型	説明
\$1	倍長整数	← イベントコード

v14 では、新しいデータベースメソッド [On Host Database Event](#) を用いることにより、データベースが開いた時と閉じられた時に 4D コンポーネントにコードを実行させることが出来るようになりました。

*Note* セキュリティ上の理由から、このデータベースメソッドを使用可能にするためには、その実行をホストデータベースで明示的に許可する必要があります。詳細に関しては、[35P “4D components”](#) を参照して下さい。

[On Host Database Event](#) データベースメソッドは、ホストデータベースのコンポーネントとして使用されているデータベースの中でのみ自動的に実行されます (ホストデータベースのプロパティリストで有効にされている必要があります)。このメソッドはホストデータベースの開閉に関するイベントが発生したときに呼び出されます。

イベントを扱うためには、\$1 引数の値をメソッド内で調べて以下の表の新しい定数と比較しなければなりません。この定数は "Database Events" テーマ内にあります。:

定数 (値)	説明

<u>On before host database startup</u> (1)	ホストデータベースはちょうど開かれたところです。ホストデータベースの <i>On Startup</i> データベースメソッドはまだ呼び出されていません。ホストデータベースの <i>On Startup</i> データベースメソッドは、コンポーネントの <i>On Host Database Event</i> データベースメソッドが実行されている間は呼び出されません。
<u>On after host database startup</u> (2)	ホストデータベースの <i>On Startup</i> データベースメソッドが実行を終了したところです。
<u>On before host database exit</u> (3)	ホストデータベースは閉じられようとしているところです。ホストデータベースの <i>On Exit</i> データベースメソッドはまだ呼び出されていない状態です。ホストデータベースの <i>On Exit</i> データベースはコンポーネントの <i>On Host Database Event</i> データベースメソッドが実行されている間は呼び出されません。
<u>On after host database exit</u> (4)	ホストデータベースの <i>On Exit</i> データベースメソッドが実行を終了したところです。

このコマンドにより、4D コンポーネントはホストデータベースのオペレーションに関連したプリファレンスやユーザー情報を読み込んだり保存したりすることが出来ます。

- ▶ On Host Database Event の典型例を紹介しましょう。:

```
// On Host Database Event データベースメソッド
C_LONGINT($1)
```

**Case of**

```
:($1 = On before host database startup)
```

```
// ここにホストデータベースのデータベースメソッド "On Startup"
// の前に実行したいコードを書きます。
```

```
:($1 = On after host database startup)
```

```
// ここにホストデータベースのデータベースメソッド "On Startup"
// の後に実行したいコードを書きます。
```

```
:($1 = On before host database exit)
```

```
// ここにホストデータベースのデータベースメソッド "On Exit"
// の前に実行したいコードを書きます。
```

```
:($1 = On after host database exit)
```

```
//ここにホストデータベースのデータベースメソッド "On Exit"
//の後に実行したいコードを書きます。
```

### End case

## On REST Authentication

On REST Authentication ( \$1; \$2 ; \$3 )→ \$0

引数	型	説明
\$1	テキスト	← ユーザー名
\$2	テキスト	← パスワード
\$3	ブール	← True = ダイジェストモード False = ベーシックモード
\$0	ブール	← True = リクエスト承認 False = リクエスト拒否

新しい [On REST Authentication](#) データベースメソッドは Web サーバーエンジンに送られた REST 要求に対するアクセス権を管理する役割を果たします。データベースメソッドが定義されていると、コマンドは 4D または 4D サーバーが REST 要求を受けた時に自動的に呼び出されます。

このデータベースメソッドは、Wakanda Server と 4D v14 との接続を設定するときに接続をフィルタリングするのが主な目的です。詳細な情報に関しては [121P "On REST Authentication"](#) を参照して下さい。

## 日付と時間

### JavaScript の日付の変換

JavaScript の日付はオブジェクトであるので、それらは他のオブジェクトと同様、JSON 形式を含んだテキストとして 4D に送られてきます。この原理は特に [4D Mobile](#) または [Web エリア](#) を使用するとき適用されます。

JSON 形式の JavaScript Date オブジェクトは ISO 8601 規格に基づき、"2013-08-23T00:00:00Z" のように記述されます。

このテキストは、デベロッパが 4D の日付 (C\_DATE) へと変換してあげる必要があります。方法は二つあります。

- 新しい [JSON Parse](#) コマンドを使用する方法 :

```
C_TEXT($1) // 日付を ISO フォーマットで受け取る
C_DATE($d)
$d:=JSON Parse("\")+$1+"\":Is_date)
```

- Date コマンドを使用する方法 :

```
C_TEXT($1) // 日付を ISO フォーマットで受け取る
C_DATE($d)
$d:=Date($1)
```

二つの方法の違いに注意して下さい。[JSON Parse](#) コマンドは SET DATABASE PARAMETER を使用して設定された変換モードに従うのに対し、Date コマンドはこれに依存しないという事です。Date コマンドを使用した変換は、常にローカルのタイムゾーンを考慮するという事です。

## Time

Time ( *timeValue* ) → Time

引数	型	説明
<i>timeValue</i>	文字列 / 倍長整数	→ 時間として返す値
戻り値	時間	← <i>timeValue</i> 引数で指定された時間

**Time** コマンドは、引数として倍長整数型の値を受け取れるようになりました。: この場合、*timeValue* 引数には 00:00:00 から経過した秒数を渡しません。

- ▶ どんな値でも整数であれば時間型に変換することができます。:

```
vTime:=Time(10000)
// vTime is 02:46:40
vTime2:=Time((60*60)+(20*60)+5200)
// vTime2 is 02:46:40
```

*See also:* [ARRAY TIME](#)

## null 日付の表示

メソッドエディターにおいて、null の日付は年が 4 桁で表示されるようになりました。:

```
!00/00/0000!
```

## デザインオブジェクトアクセス

このテーマ内のコマンドは、4D Pack の AP クリエイトメソッドコマンドのコールを置き換えることができる追加情報を返します。(詳細な情報に関しては [309P “4D Pack から削除されたコマンド”](#) を参照して下さい。)

## FORM GET NAMES

FORM GET NAMES( {aTable ;} arrNames {; filter}{; marker}{; \*} )

引数	型	説明
aTable	テーブル	→ テーブル参照

<code>arrNames</code>	テキスト配列	← フォーム名の配列
<code>filter</code>	テキスト	→ 名前のフィルター
<code>marker</code>	倍長整数	→ 返される最古のバージョンのカウンター ← 最新のカウンター(新しい値)
*	演算子	→ 指定時 = コンポーネントで実行されたとき、 コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

**FORM GET NAMES** コマンドは、4D v14 から任意の `marker` 引数を受け取るようになりました。この引数は、`arrNames` に返されるフォームを、ある時点以降に変更されたものに限定するために使用します。この引数によって、バージョン管理システムの一部として、最後のバックアップ以降に変更されたフォームのみをアップデートすることができます。

この引数は以下の様な原理で動作しています。4D は内部にフォームの更新のカウンターを持っています。フォームが作成されるか保存されるたび、カウンターは増加し、そのカレントの値が作成、または保存されたフォーム内部のカウンターに保存されます。

`marker` 引数を渡すと、コマンドは内部のカウンターが `marker` と同じ若しくはそれより大きい内部カウンターを持つフォームのみが `arrNames` に返されます。また、`marker` に変数を渡すと、コマンドはカウンターの新しい値、つまり最大の値をこの変数に返します。この値を保存し、次に **FORM GET NAMES** コマンドを呼び出すときに使用することによって新しいフォーム、更新されたフォームのみを取り出すことができます。

## METHOD GET CODE

METHOD GET CODE ( path ; code { ; \* } )

このコマンドは、生成されたコードのヘッダーの中に追加された JSON メタデータの中に、メソッドの親フォルダの名前を格納した「フォルダ」属性を追加します。例えば、：

```
// %attributes = {"lang":"en","folder":"Web3"} comment added and reserved by 4D.
```

メソッドに親フォルダがない場合は、このプロパティは表示されません。

## METHOD SET ATTRIBUTE

METHOD SET ATTRIBUTE ( path ; attribType ; attribValue1 ; attribType2 ; attribValue2 ; ... ; attribTypeN ; attribValueN ){ ; \* } )

引数	型	説明
<code>path</code>	テキスト	→ オブジェクトメソッドへのパス
<code>attribType</code>	倍長整数	→ 属性の型

attribValue	ブール / テキスト	→ True = 属性を選択、False = 属性の選択解除 またはフォルダ名
*	演算子	→ 渡した場合 = コンポーネントから実行されたとき、コマンドはホストデータベースへと適用。(これ以外の場合には無視される)

**METHOD SET ATTRIBUTE** コマンドには二つの新しい特徴が追加されています。:

- 4D v14 から追加された「フォルダ」属性を使用することができるようになっていました。(METHOD SET CODE コマンドを参照して下さい。)  
*attribType* 引数に、新しい定数 Attribute folder name (1024) を渡す事によって、この属性を設定することができます。(詳細な情報に関しては "Design Object Access" テーマを参照して下さい。)  
この定数を渡すときには、一緒に *attribValue* 引数にフォルダ名前を渡さなければなりません。:
- このフォルダ名のフォルダが存在するとき、このメソッドはその親フォルダに置かれます。
- フォルダが存在しないとき、このコマンドは親フォルダの階層には何の変更も加えません。
- 空の文字列を渡した場合、このメソッドはルート階層に置かれます。
- 4D v14 では、一度の呼び出しで複数の *attribType*; *attribValue* のペアを渡す事が可能です。
- ▶ *folder* 属性も含めて複数の属性 / 値のペアを設定する場合を考えます。:

**METHOD SET ATTRIBUTE** (*vPath*; Attribute invisible; *vInvisible*;  
Attribute published Web; *v4DAction*; Attribute published SOAP; *vSoap*;  
Attribute published WSDL; *vWSDL*; Attribute shared; *vExported*;  
Attribute published SQL; *vSQL*; Attribute executed on server; *vRemote*;  
Attribute folder name; *vFolder*; \*)

## METHOD SET CODE

**METHOD SET CODE** ( *path* ; *code* { ; \* } )

このコマンドはメタデータの新しい「フォルダ」プロパティを以下の様にサポートします。:

- このプロパティが存在していて、有効なフォルダが存在している場合、このメソッドはそのフォルダの中に置かれます。

- このプロパティが存在しない、もしくは有効なフォルダが存在しない場合は、このコマンドは親フォルダの階層には何の変更も加えません。
- このプロパティが空の文字列を含む場合、メソッドはトップレベルに置かれます。

## ドラッグ&ドロップ

### SET DRAG ICON

SET DRAG ICON(icon{; horOffset{; vertOffset{}})

引数	型	説明
icon	ピクチャ	→ ドラッグ中に表示するアイコン
horOffset	倍長整数	→ カーソルから見てアイコンの左端との水平方向の距離を指定 (>0 は左方向に、<0 は右方向に移動させます。)
vertOffset	倍長整数	→ カーソルから見てアイコンの上端との垂直方向の距離を指定 (>0 は上方向に、<0 は下方向に移動させます。)

新コマンド **SET DRAG ICON** は、*icon* で指定した画像を、プログラムで管理できるドラッグアンドドロップの最中に、カーソルのそばに表示することができます。

このコマンドは On Begin Drag Over のフォームイベント中 (ドラッグ中) しか呼び出すことができません。

*icon* 引数にはドラッグ中に表示させたいピクチャを渡します。サイズは最大で 256x256 ピクセルです。縦か横のどちらかの大きさが 256 ピクセルを超えていた場合、画像は自動的にリサイズされます。

*horOffset* と *vertOffset* ではオフセットの具合をピクセルで指定できます。:

- *horOffset* では、マウスカーソルから見て *icon* で指定した画像の左端がどれだけ水平方向にオフセットしているかを指定します。正の値を渡すとアイコンが左側に、負の値を渡すとアイコンが右側にずれていきます。
- *vertOffset* では、マウスカーソルから見て *icon* で指定した画像の上端がどれだけ垂直方向にオフセットしているかを指定します。正の値を渡すとアイコンが上方向に、負の値を渡すとアイコンが下方向にずれていきます。

引数省略時にはカーソルがアイコンの中央に位置するようになります。

- ▶ フォーム内で、ユーザーが行をドラッグ&ドロップするとラベルを生成することができるようにします。この場合リストボックスのオブジェクトメソッドは以下のようになります。:

**If (Form event=On Begin Drag Over)**

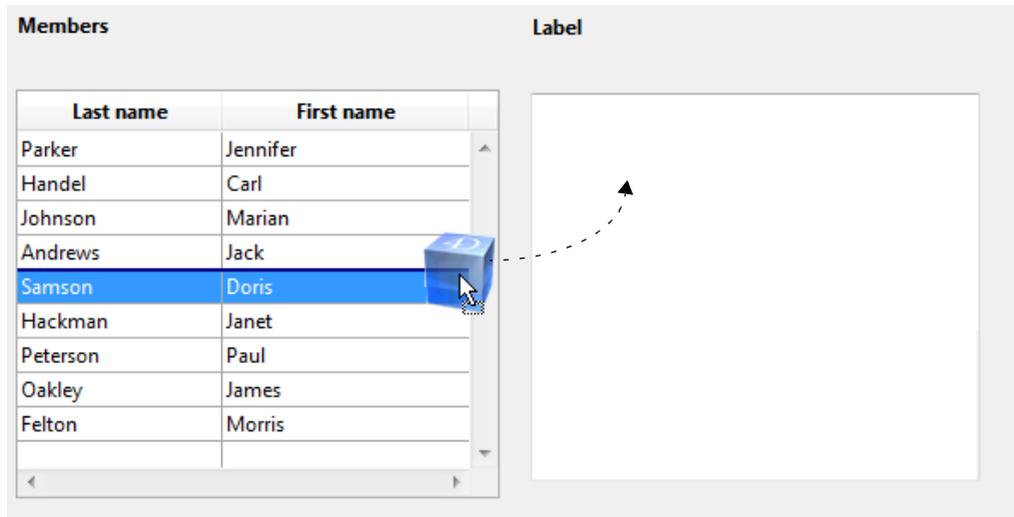
**READ PICTURE FILE(Get 4D folder (Current Resources Folder)+"splash.png";vpict)**

**CREATE THUMBNAIL(vpict;vpict;48;48)**

**SET DRAG ICON(vpict)**

**End if**

行をドラッグすると、以下のように画像が表示されます。



画像とカーソルの位置関係をずらすこともできます。:

**SET DRAG ICON(vpict;0;0)**



## フォームイベント

### フォームイベント

4D v14 ではフォームイベントに二つの新機能が追加されました。

- テキスト (テキスト、日付、時間、数字型) が入る入力可能なフィールドと変数は、v14 より On Clicked と On Double-Clicked イベントが発生する

ようになりました。旧バージョンの 4D では、入力不可のフィールドまたは変数でないこれらのイベントが発生しませんでした。

- "セレクション" 型のリストボックスに関連付けされた入力フォームでも On Open Detail と On Close Detail イベントが発生するようになりました。詳細な情報に関しては、71P "フォームイベント" を参照して下さい。

## フォーム

### Current form name

Current form name → Text

引数	型	説明
----	---	----

このコマンドには指定できる引数がありません。

戻り値	テキスト	← 処理中のカレントプロジェクトフォーム またはカレントテーブルフォーム
-----	------	---

新コマンドの **Current form name** は処理のために定義されたカレントフォームの名前を返します。カレントフォームにはプロジェクトフォームまたはテーブルフォームを指定することが出来ます。

デフォルトで、カレントの処理において **FORM LOAD** コマンドを呼び出していない場合、現在表示されているフォーム、もしくは印刷されたフォームがカレントフォームになります。**FORM LOAD** コマンドを処理中に呼び出した場合、このコマンドで設定されたフォームがカレントフォームになり、**FORM UNLOAD** コマンド (または CLOSE PRINTING JOB コマンド) を呼び出すまでその状態が維持されます。

もし処理によって定義されたカレントフォームがない場合、このコマンドは空の文字列を返します。

- ▶ 入力フォームにおいて以下のコードをボタンの中に入れてみましょう。:

```
C_TEXT($FormName)
$win:=Open form window([Members];"Input";Plain form window)
DIALOG([Members];"Input")
$FormName:=Current form name
// $FormName = "Input"
FORM LOAD([Members];"Drag")
$FormName:=Current form name
// $FormName = "Drag"
//...
```

- ▶ カレントフォームがプロジェクトフォームであればその名前を取得したいという場合、:

```
$PointerTable:=Current form table
If (Nil($PointerTable)) // これがプロジェクトフォームである場合、
  $FormName:=Current form name
  ... // 処理中
End if
```

See also: [FORM LOAD](#)

## FORM LOAD

FORM LOAD( {aTable;} form {; \*})

引数	型	説明
aTable	テーブル	→ ロードするテーブルフォームを指定。(省略時はプロジェクトフォームをロード)
form	文字列	→ 開くフォーム(プロジェクトまたは table) の名前
*	演算子	→ 渡した場合、コンポーネントから実行した場合にホストのデータベースコマンドが適応されず。(それ以外の場合は無視されます。)

*Compatibility note* 以前の 4D では、このコマンドは印刷のテーマ内に [OPEN PRINTING FORM](#) という名前で存在していました。v14 では拡張されています。

[FORM LOAD](#) コマンドはメモリー内に *form* をロードし、以下の動作を行います。

- フォームを使用してデータを印刷する  
この動作は以前のバージョンでの機能と同じものです。この機能を実行するためには [OPEN PRINTING JOB](#) コマンドを使用して印刷タスクが事前に関われている必要があります。[OPEN PRINTING JOB](#) コマンドは内部で [FORM UNLOAD](#) コマンドを呼び出すので、[FORM LOAD](#) コマンドを呼び出す必要が出てくるわけです。その際、*form* は新しいカレント印刷フォームになります。On Load イベントとそれに続いて On Unload イベントが、フォームのオブジェクトメソッドとともに実行されます。  
なお v14 では、このコマンドをテーブルフォームにも適用できるようになりました！

*Compatibility note* 旧バージョンの 4D では、[OPEN PRINTING FORM](#) では *form* 引数に空の文字列を渡す事によってカレントのプロジェクトフォームを閉じることが可能でした。空の文字列を渡すこのシンタックスは v14 からはサポートされず、エラーが返されます。4D v14 では、フォームを閉じるためには [FORM UNLOAD](#) コマンドはまたは [CLOSE PRINTING JOB](#) コマンドを使用しなければならなくなりました。

- フォームの内容を解析する

This new possibility consists in loading an offscreen form for parsing purposes. この動作をするには印刷ジョブと無関係のところで **FORM LOAD** を呼び出すだけです。このときには、フォームイベントは実行されません。**FORM LOAD** コマンドは **FORM GET OBJECTS** や **OBJECT Get type** コマンドと組み合わせることによって、フォームの内容にどんな処理でも行うことができます。これらの動作が終了した後は **FORM UNLOAD** コマンドを使用してメモリーからフォームを解放してあげる必要があります。

どの場合においても、スクリーン上のフォームはロードされたままなので (**FORM LOAD** コマンドに左右されません)、**FORM UNLOAD** を呼び出した後にリロードする必要はないという事に注意して下さい。

コンポーネントからこのコマンドが実行された場合は自動的にコンポーネントフォームをロードします。もし \* 演算子を渡した場合は、ホストデータベースフォームがロードされます。

- ▶ 印刷ジョブ内でプロジェクトフォームを呼び出すには、:

**OPEN PRINTING JOB**

**FORM LOAD** ("print\_form")

// イベントとオブジェクトメソッドの実行

- ▶ 印刷ジョブ内でテーブルフォームを呼び出すには、:

**OPEN PRINTING JOB**

**FORM LOAD** ([People];"print\_form")

// イベントとオブジェクトメソッドの実行

- ▶ フォームの内容を解析してテキスト入力エリアに何らかの処理をするには:

**FORM LOAD** ([People];"my\_form")

// イベントやメソッドを実行することなくフォームを選択

**FORM GET OBJECTS** (arrObjNames;arrObjPtrs;arrPages;\*)

**For** (\$i;1;**Size of array** (arrObjNames))

**If** (**OBJECT Get type** (\*;arrObjNames{\$i})=Object type text input)

// ... 処理

**End if**

**End for**

**FORM UNLOAD**

*See also:* [FORM UNLOAD](#), [Current form name](#), [LISTBOX GET OBJECTS](#)

## FORM UNLOAD

FORM UNLOAD

引数	型	説明
----	---	----

このコマンドには指定できる引数がありません。

新しい **FORM UNLOAD** コマンドは、**FORM LOAD** コマンドを使用してロードされたカレントフォームを、メモリーから解放します。

印刷とは関係のないところで **FORM LOAD** コマンドを使用した場合は **FORM UNLOAD** コマンドを必ず使用しなければなりません。(印刷した場合は内部で CLOSE PRINTING JOB コマンドが呼び出されてカレントフォームが自動的に閉じられます。)

*See also:* **FORM LOAD**

## グラフ

このテーマの中にあるコマンドは 4D v14 からは 4D Chart プラグインをサポートしていません。このプラグインは 4D v14 において廃止され、これ以降使われることはありません。(詳細に関しては 20P “削除された機能” を参照して下さい。)

### GRAPH

GRAPH ( *graphPicture* ; graphNumber ; xLabels { ; yElements } { ; yElements2 ; ... ; yElementsN } )

引数	型	説明
<i>graphPicture</i>	ピクチャ	→ ピクチャ変数
...		

4D v14 では、**GRAPH** コマンドでは第一引数にはピクチャ変数のみ受け取れるようになりました。グラフエリア (4D Chart) を使用するシンタックスは廃止され、サポートされていません。

### GRAPH SETTINGS

GRAPH SETTINGS ( *graphPicture* ; xmin ; xmax ; ymin ; ymax ; xprop ; xgrid ; ygrid ; title { ; title2 ; ... ; titleN } )

引数	型	説明
<i>graphPicture</i>	ピクチャ	→ ピクチャ変数
...		

4D v14 において **GRAPH SETTINGS** コマンドでも第一引数にはピクチャ変数のみ受け取れるようになりました。グラフエリア (4D Chart) を使用するシンタックスは廃止され、サポートされていません。

## GRAPH TABLE

4D v14 以降、GRAPH TABLE コマンドが使用できなくなり、コマンド名も 4D v14 では `_o_GRAPH TABLE` と改名されました。変換されたデータベースでは、このコマンドへの呼び出しを全て削除しておくようにして下さい。

## 階層リスト

*Note* Windows 環境下での階層リストの見た目が変わりました。詳細な情報に関しては、60P “階層リスト” を参照して下さい。

## フォームオブジェクトとの関連付け

4D の v14 では、新しい [OBJECT SET LIST BY REFERENCE](#) コマンドか [OBJECT SET LIST BY NAME](#) コマンド（以前の名称は OBJECT SET CHOICE LIST NAME コマンド）を使用して、階層リストにフォームオブジェクトリスト（選択リスト、指定リスト、除外リスト）を関連付けられるようになりました。旧バージョンでは、デザインモードで作成されたリストしかオブジェクトと関連付けることが出来ませんでした。

## HTTP クライアント

### HTTP Get

HTTP Get ( url; response {;headerNames; headerValues} {;\*}) → Longint

引数	型	説明
...		
response	テキスト   BLOB   ピクチャ   Object	← リクエストの結果
...		

[HTTP SET CERTIFICATES FOLDER](#) コマンドは、4D v14 において以下のものをサポートできるように変更されています。:

- ランゲージオブジェクト（詳細に関しては [240P “オブジェクト（ランゲージ）”](#) を参照して下さい。）
- サーバー証明書

`response` 引数に C\_OBJECT 型の変数を渡して戻り値が "application/json" 等の、何らかの JSON コンテンツ型であった場合、4D は自動的にその JSON を解析して C\_OBJECT オブジェクトを作成します。

コマンドがサーバー証明書を使用し、この証明書が有効でない（失効している、無効である等）場合、このコマンドは 0 を返し、ERROR システム

変数に 901 エラー "*Server certificate invalid*" が返されます。このエラーは ON ERR CALL コマンドのエラーハンドリングメソッドで割り込むことができます。

## HTTP Get certificates folder

HTTP Get certificates folder → Text

引数	型	説明
戻り値	テキスト	← カレントの証明書のフォルダへの完全なパス

新しい [HTTP Get certificates folder](#) コマンドはカレントのクライアント証明書フォルダへの完全なパス名を返します。

デフォルトとして、4D はストラクチャファイルの隣に作成される "ClientCertificatesFolder" というフォルダを使用します (フォルダは必要のある場合のみ作成されます)。[HTTP SET CERTIFICATES FOLDER](#) コマンドを使用してカレントのプロセスにおいてカスタムのフォルダを作成することもできます。

- ▶ 一時的に証明書フォルダを変更したい場合を考えます。:

**C\_TEXT** (\$certifFolder)

\$certifFolder:= **HTTP Get certificates folder** // カレントのフォルダを保存

**HTTP SET CERTIFICATES FOLDER** ("C:/temp/tempCertif/")

... // 何等かのリクエストを実行

**HTTP SET CERTIFICATES FOLDER** (\$certifFolder) // 以前のフォルダへ復元

*See also:* [HTTP SET CERTIFICATES FOLDER](#)

## HTTP Request

HTTP Request (httpMethod; url; contents; response {;headerNames; headerValues} {;\*}) → Longint

引数	型	説明
...		
contents	テキスト   BLOB   ピクチャ   ランゲージオブジェクト	→ リクエストボディの内容
response	テキスト   BLOB   ピクチャ   ランゲージオブジェクト	← レスポンスの内容
...		

**HTTP Request** コマンドは、4D v14 において以下のものをサポートできるように変更されています。:

- ランゲージオブジェクト (詳細に関しては 240P “オブジェクト (ランゲージ)” を参照して下さい。)
- サーバー証明書 (see [HTTP SET CERTIFICATES FOLDER](#))
- *contents* 引数に C\_OBJECT 型のオブジェクトを PUT や POST メソッドで渡した場合、4D は自動的にそれを JSON へとシリアル化 (文字列化) して、リクエストのボディを作成した上でその HTTP ヘッダー内に "Content-Type: application/json" と書き込みます。

*response* 引数に C\_OBJECT 型の変数を渡して戻り値が "application/json" 等の、何らかの JSON コンテンツ型であった場合、4D は自動的にその JSON を解析して C\_OBJECT オブジェクトを作成します。

このコマンドがサーバー証明書を使用し、その証明書が有効でない (失効している、もしくは無効である) 場合、このコマンドは 0 を返し、ERROR システム変数に 901 エラー "Server certificate invalid" が返されます。このエラーは ON ERR CALL コマンドのエラーハンドリングメソッドで割り込むことができます。

- ▶ JSON のレコードをリモートのデータベースに追加するリクエストを書く場合、以下のようになります:

```
C_OBJECT($content)
OB SET($content;"lastname";"Doe";"firstname";"John")
$result:=HTTP Request(HTTP put method;"database.example.com";$content;$response)
```

## HTTP SET CERTIFICATES FOLDER

HTTP SET CERTIFICATES FOLDER ( certificatesFolder )

引数	型	説明
certificatesFolder	テキスト	→ クライアント証明書の名前とパス名

新しい [HTTP SET CERTIFICATES FOLDER](#) コマンドはカレントのプロセスにおける有効なクライアント証明書を変更します。

クライアント証明書フォルダとは、Web サーバーからクライアント証明書の要求を受けた時に 4D がそれを探す場所です。デフォルトでは、[HTTP SET CERTIFICATES FOLDER](#) コマンドが実行されていない限り、4D はストラクチャファイルの隣に作成された "ClientCertificatesFolder" という名前のフォルダを使用します。このフォルダは必要がある場合にのみ作成されます。

4D v14 では、複数のクライアント証明書を使用することが出来るようになりました。

*certificatesFolder* 引数には、クライアント証明書が入っているユーザーによって設定されたフォルダの名前を渡します。アプリケーションのストラクチャファイルからの相対パスを渡す事も出来ますし、絶対パスを渡す事もできます。絶対パスで指定する場合は、システムのシンタックスに沿って以下の様に書かれていなければなりません。:

- (OS X) : Disk:Applications:myserv:folder
- (Windows) C:\Applications\myserv\folder

指定されたフォルダがその場所がない場合、4D は自動的にそのフォルダを作成します。

このコマンドが実行されると、新しいフォルダへのパスは直ちに有効になります (アプリケーションを再起動する必要はありません)。

このコマンドのスコープ (影響が及ぶ範囲) はカレントのプロセスであるため、あるプロセスの中で一度呼び出せば、そのプロセス内の SOAP リクエストおよび HTTP リクエスト全てに対して有効です。複数のプロセスを作成すれば、異なるサーバーに対して同時に複数のリクエストをすることも可能です。

デフォルトの証明書フォルダの場所を使用したい場合は、*certificatesFolder* 引数に空の文字列を渡します。

*certificatesFolder* に渡されたパス名が有効でなかった場合、エラーが発生します。このエラーは ON ERR CALL コマンドのエラーハンドリングメソッドで割り込むことができます。

*See also:* [HTTP Get certificates folder](#)

## JSON

この新しいテーマには、JSON フォーマットのランゲージオブジェクトを生成させて解析するためのいくつかのコマンドがあります。言い換えると、このフォーマットを使用することによって 4D データベース (データとストラクチャ) に Web ブラウザを使ってアクセスすることが出来るようになるのです。

構造化されたオブジェクトの使えるようになったことは 4D の v14 の大きな新機能の一つであり、構造化されたデータの受け渡しを容易にするのが目的です。「JSON」テーマ内のコマンドを使用することによって 4D で

JSON オブジェクトを直接使うことが出来るようになりました。それでいながら、"native" オブジェクト (JSON にインスパイアされた構造をもつオブジェクト) としても扱うことができ、これによりランゲージに用意された他の全てのタイプと受け渡しができるようになっていきます。詳細は 240P “オブジェクト (ランゲージ)” を参照して下さい。

## JSON についての概要

"JSON (JavaScript Object Notation) とは、ECMAScript 表記法に由来する、標準的なテキストベースのデータ表記法である。" (引用:Wikipedia 英語版) JSON は特定のプログラミング言語に依存しませんが、それでいて C++ や JavaScript、Perl 等といった言語の使用者にはなじみ深い慣習を使用しています。データの受け渡し特に特化したフォーマットであると言えるでしょう。

この章では JSON の表記法の原理についての概略を記載しています。JSON の表記法についての完全な情報については、こちらの Web サイトを参照して下さい:

<http://www.json.org/index.html>

## JSON 記法

JSON の記法は以下の原則に従うようにできています。:

- データは、名前と値がペアになるようにできています。
- データはカンマ (,) で区切られています。
- オブジェクトは中カッコ ({} ) によって定義されます。
- 配列は大カッコ ([ ] ) によって定義されます。

## JSON プロパティ

JSON データは、名前 / 値 (もしくはキー / 値) のペアという形で表現されます。名前 / 値のペアの中は、二重引用符 ("" ) で挟まれたフィールド名のあと、コロン (: )、そして同じく二重引用符 ("" ) で挟まれた値が続きます。:

```
"firstName" : "John"
```

これは JavaScript での以下の表記に対応しています。:

```
firstName = "John"
```

プロパティ名では大文字と小文字は区別されることに注意してください。"FirstName" の代わりに "firstName" と表記すると、これは別のデータを意味することになります。

## JSON データ型

JSON では以下の型の値に対応しています。:

型	説明	詳細
---	----	----

文字列	二重引用符 (") とバックスラッシュ (/) 以外ならどんな Unicode 文字も使用できます。 プロパティ名などの値は二重引用符に挟まれる決まりになっています。	制御文字には \ が使用されます。: \" = quotes \\ = backslash \/ = slash \b = backspace \f = form feed \n = line break \r = carriage return \t = tab \u = four hexadecimal digits
数値	整数または浮動小数	数字に関しては C や Java に近いですが、8 進法数と 16 進法数は使用できません。
オブジェクト	{ }	
配列	[ ]	
ブール	true または false	
null	null	

### JSON オブジェクト

JSON オブジェクトは中カッコ ( { } ) によって定義され、必要な数だけ名前 / 値のペアを内包することが出来ます。表記は以下のようになります。

```
:
{ "firstName": "John", "lastName": "Doe" }
```

### JSON 配列

JSON 配列は大カッコ ( [ ] ) によって定義されます。それぞれの配列は、未定数のオブジェクトを内包することができます。:

```
{
  "employees": [
    { "firstName": "John", "lastName": "Doe" },
    { "firstName": "Anna", "lastName": "Smith" },
    { "firstName": "Peter", "lastName": "Jones" }
  ]
}
```

### タイムゾーンのサポート

4D の日付と JSON 感でデータの変換が行われる際、デフォルトで変換を行ったマシンのタイムゾーンに沿って (JavaScript に従って) 変換されます。例えば、フランス (GMT+2) では !23/08/2013! を変換すると "2013-08-22T22:00:00Z" という結果が得られます (逆もまた然りです)。

この機能を変更し、書き出し処理を実行 ( [SET DATABASE PARAMETER](#), [Get database parameter](#) コマンド等を使用) する際にタイムゾーンを考慮しないようにすることもできます。

*Note* 4D/JSON 間の日付の変換についてのより詳細な情報は、[157P “JavaScript の日付の変換”](#) を参照して下さい。

## JSON Parse

JSON Parse( jsonString {; type} ) → Text, Real, Boolean, Pointer, Object

引数	型	説明
jsonString	文字列	→ パース ( 解析 ) する JSON 文字列
type	倍長整数	→ 何の型に変換するかを指定
戻り値	テキスト   実数   ブール   ポインター   オブジェクト	← JSON 文字列から変換された戻り値

**JSON Parse** コマンドは、JSON フォーマットの文字列を解析して、4D のフィールドや変数に保存できる値に変換します。つまり JSON データをデシリアライズします。挙動としては **JSON Stringify** コマンドと逆の挙動をします。

*jsonString* には、解析したい JSON フォーマットの文字列を渡します。この文字列はフォーマットに沿って正しく書かれてる必要があります、そうでない場合にはエラーが発生します。

変換先の型を指定する *type* 引数を省略した場合、4D は自動的に保存先の変数またはフィールドの型へと変換しようと試みます。保存先が変数でかつ未定義の場合は、4D はその型を推測しようとします。

*type* 引数を用いて、変換する型を強制的に指定することもできます。: その場合以下の定数のどれか一つを渡して指定して下さい。定数テーマリストの "Field and Variable Types" にもこの一覧があります。:

定数 ( 値 )	説明
<u>Is real</u> (1)	値の幅は +-10.421e+10
<u>Is text</u> (2)	全ての特殊文字は引用符 ( " ) を含めてエスケープされなければなりません。
<u>Is date</u> (4)	JSON の日付のフォーマットは以下の形式でなければなりません。 "\"YYYY-MM-DDTHH:mm:ssZ\"" コマンドは、4D の日付が GMT ではなくローカル時刻を含むことを考慮します。
<u>Is boolean</u> (6)	true   false
<u>Is longInt</u> (9)	

*Note* ポインターを使用する場合、**JSON Parse** を使用する前に **JSON Stringify** を呼び出さなければなりません。

- ▶ 単純な変換の例 :

```
C_REAL ($r)
$r:=JSON Parse( "42.17" ) // $r = 42,17 (Real)
```

```
C_LONGINT($el)
$el:=JSON Parse("120.13";Is longInt) // $el=120
```

```
C_TEXT ($t)
$t:=JSON Parse( "\"Year 42\"" ;Is text) // $t="Year 42" (text)
```

```
C_OBJECT ($o)
$o:=JSON Parse ( "{ \"name\": \"john\" }" )
// $o = { "name": "john" } (4D object)
```

```
C_BOOLEAN($b)
$b:=JSON Parse("{ \"manager\": true }";Is boolean) // $b=true
```

- ▶ データ型のデータを変換している例 :

```
$test:=JSON Parse("\"1990-12-25T12:00:00Z\"")
// $test=1990-12-25T12:00:00Z
```

```
C_DATE($date)
$date:=JSON Parse("\"2008-01-01T12:00:00Z\"";Is date)
// $date=2008/01/01
```

- ▶ ここでは、[JSON Stringify](#) と [JSON Parse](#) を組み合わせて使用している例を紹介しています。 :

```
C_TEXT ($MyContact)
C_OBJECT($Contact)

// JSON Stringify: conversion of JSON object into a JSON string
$MyContact:=JSON Stringify("{ \"name\": \"Monroe\", \"firstname\": \"Alan\" }")
// $MyContact = "{ \"name\": \"Monroe\", \"firstname\": \"Alan\" }"
// JSON Parse: conversion of JSON string into a JSON object
$Contact:=JSON Parse("{ \"name\": \"Monroe\", \"firstname\": \"Alan\" }")
// $Contact = { "name": "Monroe", "firstname": "Alan" }
```

*See also:* [JSON Stringify](#), [JSON PARSE ARRAY](#)

## **JSON PARSE ARRAY** JSON PARSE ARRAY(jsonString; objArray)

引数	型	説明
jsonString	文字列	→ 解析する JSON 文字列を指定

`objArray` オブジェクト ← JSON 文字列を解析した戻り値を受け取る  
配列 配列を指定

**JSON PARSE ARRAY** コマンドは、JSON 文字列を解析し、その中身を `objArray` の配列に入れます。挙動としては **JSON Stringify array** コマンドと逆の挙動をします。

`jsonString` には、解析したい JSON フォーマットで書かれた文字列を渡します。この文字列はフォーマットに沿って正しく書かれてる必要があります、そうでない場合にはエラーが発生します。

`objArray` には、解析した戻り値を受け取るオブジェクト配列を指定します。

- ▶ ここでは、テーブル内のレコードのフィールドからのデータをオブジェクト配列の中に入れて例を紹介しています。:

**C\_OBJECT** (\$ref)

**ARRAY OBJECT** (\$sel;0)

**ARRAY OBJECT** (\$sel2;0)

**C\_TEXT** (v\_String)

**OB SET** (\$ref;"name";->[Company]Company Name)

**OB SET** (\$ref;"city";->[Company]City)

**While(Not(End selection([Company])))**

\$ref\_company:=**OB Copy** (\$ref;**True**)

**APPEND TO ARRAY**(\$sel;\$ref\_company)

// \$sel{1}={"name":"4D SAS","city":"Clichy"}

// \$sel{2}={"name":"MyComp","city":"Lyon"}

// ...

**NEXT RECORD**([Company])

**End while**

v\_String:=**JSON Stringify array**(\$sel)

// v\_String= [{"name":"4D SAS","city":"Clichy"}, {"name":"MyComp",

// "city":"Lyon"}...]

**JSON PARSE ARRAY**(v\_String;\$sel2)

// \$sel2{1}={"name":"4D SAS","city":"Clichy"}

// \$sel2{2}={"name":"MyComp","city":"Lyon"}

//...

*See also:* [JSON Stringify](#)

## JSON Stringify

JSON Stringify ( value { ; \* } ) → Text

引数	型	説明
value	オブジェクト   オブジェクト配列   文字列   数字   日付   時間	→ JSON 文字列へと変換するデータ
*	演算子	→ 整形フォーマット
戻り値	テキスト	← シリアライズされた文字列を含む JSON テキスト

**JSON Stringify** コマンドは、*value* の引数で指定したオブジェクトを JSON 文字列へと変換します。つまりデータを JSON へとシリアライズします。挙動としては **JSON Parse** コマンドと逆の挙動をします。

シリアライズするデータを *value* へと渡します。データはスカラー形式 (文字列、数字、日付または時間) または 4D オブジェクト (もしくはオブジェクト配列) の形式で表記します。

オブジェクトの場合、全ての型の値を含めることができます (詳細は [171P “JSON データ型”](#) を参照して下さい)。JSON 形式は、以下のルールに従っている必要があります。

- 文字列の値は引用符で囲ってある必要があります。全ての Unicode 文字を使用することが出来ますが、特殊文字はその前にバックスラッシュを付ける必要があります。
- 数字 (+-10.421e+-10 の範囲)
- ブール ("true" または "false" の文字列)
- フィールド、変数、配列へのポインター (ポインターはシリアライズされるときに評価されます)
- 日付 (テキスト型)
- 時間 (実数型)

任意の \* 演算子を渡す事によって、戻り値の文字列にフォーマット文字を含めることができます。このオプションによって、JSON データを Web ページで閲覧するときの表示が改善されます (整形フォーマティング)。

- ▶ スカラー値を変換する場合を考えます。

```
$vc:=JSON Stringify ("Eureka!") // "Eureka!"
$vel:=JSON Stringify (120) // "120"
$vd:=JSON Stringify (!28/08/2013!) // "2013-08-27T22:00:00Z"
$vh:=JSON Stringify (?20:00:00?) // 真夜中より "72000000" 秒
```

- ▶ 特殊文字を含む文字列の変換を考えます。

```
$s:=JSON Stringify("{\"name\":\"john\"}")
// $s="{\"name\":\"john\"}"
$p:=JSON Parse($s)
// $p={"name":"john"}
```

- ▶ 変数へのポインターの使用例 :

```
C_OBJECT ($MyTestVar)
C_TEXT ($name ; $jsonstring )
OB SET ($MyTestVar; "name" ; ->$name) // オブジェクトを定義
// $MyTestVar = {"name":->$name"}
```

```
$jsonstring := JSON Stringify ($MyTestVar)
// $jsonstring = {"name":""}
//...
```

```
$name:="Smith"
$jsonstring := JSON Stringify ($MyTestVar)
// $jsonstring = {"name" : "Smith"}
```

- ▶ 4D オブジェクトのシリアライズの場合 :

```
C_TEXT($varjsonTextserialized)
C_OBJECT($Contact)
OB SET($Contact;"firstname";"Alan")
OB SET($Contact;"lastname";"Monroe")
OB SET($Contact;"age";40)
OB SET($Contact;"phone";"[555-0100,555-0120]")
```

```
$varjsonTextserialized:=JSON Stringify ($Contact)
```

```
// $varjsonTextserialized = {"lastname":"Monroe","phone":["555-0100,
// 555-0120"],"age":40,"firstname":"Alan"}
```

- ▶ 4D オブジェクトのシリアライズを行ったときの、\* 演算子有りとなしの違いを見てみましょう。 :

```
C_TEXT ($MyContact)
C_TEXT ($MyPContact)
C_OBJECT ($Contact;$Children)
OB SET ($Contact;"lastname";"Monroe";"firstname";"Alan")
OB SET ($Children;"firstname";"Jim";"age";"12")
OB SET ($Contact;"children";$Children)
$MyContact:=JSON Stringify ($Contact)
```

```
$MyPContact:=JSON Stringify ($Contact;*)
```

```
 //$MyPContact = {"lastname":"Monroe","firstname":"Alan","children":{"first-  
name":"John","age":"12"}}
```

```
 //$MyPContact = {\n\t"lastname": "Monroe",\n\t"firstname": "Alan",\n\t"chil-  
dren": {\n\t\t"firstname": "John",\n\t\t"age": "12"\n\t}\n}
```

この JSON が Web エリアで表示されると、その違いは明らかです。:

標準フォー  
マット

```
{"Name":"Monroe","firstname":"Alan","children":{"firstname":"John","age":12}}
```

整形フォー  
マット

```
{  
  "Name": "Monroe",  
  "firstname": "Alan",  
  "children": {  
    "firstname": "John",  
    "age": 12  
  }  
}
```

*See also:* [JSON Stringify array](#)

## JSON Stringify array

JSON Stringify array( array {; \*} ) → Text

引数	型	説明
array	テキスト配列、実数配列、 ブール配列、ポインター配列、 オブジェクト配列	→ シリアライズしたいコン テンツを含んだ配列
*	演算子	→ フォーマットの強化
戻り値	テキスト	← JSON配列をシリアライズ した文字列

**JSON Stringify array** コマンドは *array* の 4D 配列をシリアライズされた JSON 配列に変換します。挙動としては **JSON PARSE ARRAY** コマンドと逆の挙動をします。

*array* 引数には、シリアライズするデータを含んだ 4D 配列を渡します。配列の型としては、テキスト、実数、ブール、ポインター、オブジェクト、の 5 つを使うことができます。

任意の \* 演算子を渡す事によって、戻り値の文字列にフォーマット文字を含めることができます。このオプションによって、JSON データを Web ページで閲覧するときの表示が改善されます (整形フォーマティング)。

- ▶ テキスト配列を変換する場合 :

```
C_TEXT($jsonString)
ARRAY TEXT( $ArrayFirstname;2)
$ArrayFirstname{1}:= "John"
$ArrayFirstname{2}:= "Jim"
$jsonString := JSON Stringify array($ArrayFirstname)

// $jsonString = "["John","Jim"]"
```

- ▶ 数字を含んだテキスト配列を変換する場合 :

```
ARRAY TEXT( $phoneNumbers;0)
APPEND TO ARRAY( $phoneNumbers ; "555-0100")
APPEND TO ARRAY( $phoneNumbers ; "555-0120")
$string := JSON Stringify array( $phoneNumbers)
// $string = "["555-0100","555-0120"]"
```

- ▶ オブジェクト配列を変換する場合 :

```
C_OBJECT ($ref_john)
C_OBJECT ($ref_jim)
ARRAY OBJECT($myArray;0)
OB SET ($ref_john; "name" ; "John";"age";35)
OB SET ($ref_jim; "name" ; "Jim";"age";40)
APPEND TO ARRAY($myArray ; $ref_john)
APPEND TO ARRAY($myArray ; $ref_jim)
$jsonString := JSON Stringify array ($myArray)
// $jsonString = "[{"name":"John","age":35},{ "name":"Jim","age":40}]"
```

// 結果を Web ページで見たい場合、  
// 任意の \* 演算子を渡して下さい :

```
$jsonStringPretty := JSON Stringify array ($myArray;*)

[
  {
    "name": "John",
    "age": 35
  },
  {
    "name": "Jim",
    "age": 40
  }
]
```

- ▶ オブジェクト配列内の 4D セレクションを 変換する場合 :

```
C_OBJECT ($jsonObject)
```

**C\_TEXT** (\$jsonString)

```

QUERY ([Company];[Company]Company Name="a@")
OB SET($jsonObject;"company name";->[Company]Company Name)
OB SET($jsonObject;"city";->[Company]City)
OB SET($jsonObject;"date";[Company]Date_input)
OB SET($jsonObject;"time";[Company]Time_input)
ARRAY OBJECT($arraySel;0)

```

```

While(Not (End selection([Company])))
  $ref_value:=OB Copy($jsonObject;True)
  // これらをコピーしない場合、値の中身は空の文字列になります。
  APPEND TO ARRAY($arraySel;$ref_value)
  // それぞれの要素には、選択した値が含まれます。つまり、:
  // $arraySel{1} = // {"company name":"APPLE","time":43200000,"city":
  // "Paris","date":"2012-08-02T00:00:00Z"}
  NEXT RECORD([Company])
End while

```

```

$jsonString:=JSON Stringify array ($arraySel)
// $jsonString = "[{"company name":"APPLE","time":43200000,"city":
// "Paris","date":"2012-08-02T00:00:00Z"},{"company name":
// "ALMANZA",...}]"

```

*See also:* [JSON Stringify](#)

## JSON TO SELECTION

**JSON TO SELECTION**(aTable; jsonObject)

引数	型	説明
aTable	テーブル	→ 4D テーブルを指定
jsonObject	テキスト	→ JSON 内の文字列

**JSON TO SELECTION** コマンドは JSON オブジェクトの中身を aTable のレコードのセレクションへとコピーします。

aTable を呼び出したときに、指定したセレクションが存在していた場合、JSON オブジェクトの要素はオブジェクトの順番とレコードの順番に応じてコピーされます。JSON オブジェクトによって定義された要素の数がカレントセレクション内に存在するレコードより多い場合、新たにレコードが追加されます。レコードは、存在していた場合でも新規に追加した場合でも、自動的に保存されます。

**注意:** **JSON TO SELECTION** コマンドは既存のレコード内の情報を上書きしてしまうため、使用の際には注意が必要です。  
コマンド実行中、保存先のレコードが他の処理などでロックされていた

場合、その中身は変更されません。ロックされたレコードは、システムセットの `LockedSet` 内に保存されています。`JSON TO SELECTION` コマンドを実行し終わったあと、`LockedSet` 内にロックされたレコードが保存されているかどうか検証することが出来ます。

- ▶ `JSON TO SELECTION` コマンドを使用して `[Company]` というテーブルにレコードを加える場合 :

```
C_OBJECT ($Object1;$Object2;$Object3;$Object4)
C_TEXT ($ObjectString)
ARRAY OBJECT($arrayObject;0)
```

```
OB SET($Object1;"ID";"200";"Company Name";"4D SAS";"City";"Clichy")
APPEND TO ARRAY($arrayObject;$Object1)
```

```
OB SET($Object2;"ID";"201";"Company Name";"APPLE";"City";"Paris")
APPEND TO ARRAY($arrayObject;$Object2)
```

```
OB SET($Object3;"ID";"202";"Company Name";"IBM";"City";"London")
APPEND TO ARRAY($arrayObject;$Object3)
```

```
OB SET($Object4;"ID";"203";"Company Name";"MICROSOFT";"City";"New
York")
APPEND TO ARRAY($arrayObject;$Object4)
```

```
$ObjectString:=JSON Stringify array($arrayObject)
```

```
// $ObjectString = "[{"ID":"200","City":"Clichy","Company Name":"4D
// SAS"},{"ID":"201","City":"Paris","Company Name":"AP-
// PLE"},{"ID":"202",
// "City":"London","Company Name":"IBM"},{"ID":"203","City":"New
// York","Company Name":"MICROSOFT"}]"
```

```
JSON TO SELECTION ([Company];$ObjectString)
// 上記のコマンドを実行すると、[Company] テーブルに 4 つ
// レコードを追加し、その中に ID フィールド、
// Company name フィールド、City フィールドが記録されます。
```

*See also:* [Selection to JSON](#)

## Selection to JSON

Selection to JSON ( aTable{; aField1{; aField2;...}}{; template}) → Text

引数	型	説明
aTable	テーブル	→ シリアライズするテーブル

aField1... aFieldN	フィールド	→	シリアライズするフィールドを指定
template	オブジェクト	→	プロパティ名と取り出したいフィールドを指定するポインターをオブジェクトで渡す
戻り値	テキスト	←	シリアライズされた JSON 配列を含む文字列

**Selection to JSON** コマンドは、*aTable* で指定したテーブル内のフィールドの値を JSON 文字列として返します。

*aTable* 引数のみを渡した場合、テーブル内の、全てのフィールドの値のうち、JSON で表現できるものを JSON 文字列で返します。BLOB フィールドとピクチャーフィールドは無視されます。

*aTable* 内の一部のフィールドのみ取り出したい場合、*aField1...aFieldN* 引数か、*template* 引数を使ってその部分を指定することが出来ます。

- *aField1...aFieldN*: 一つ以上のフィールドをこの引数で指定して下さい。ここで指定したフィールドの値のみ JSON 文字列で返されます。
- *template*: 一つ以上の名前 / 値のペアを含んだ 4D オブジェクトを渡して下さい。この値に、取り出したいフィールドを指定するポインターを入れて指定します。(例 3 を参照して下さい。)

▶ 以下のセレクションを JSON 文字列で表現する場合を考えてみましょう :

HierList - Members: 6 of 6				
Last name :	First name :	Address :	City :	Zip Code :
Durant	Mark	25 Park St	Pittsburgh	15205
Smith	John	24 Philadelphia Ave	Dallas	75203
Anderson	Adeline	37 Market St	Cincinnati	45205
Peterson	Paul	32 South Main St	Dallas	75203
Harper	Harry	233 Southport Ave	Cincinnati	45206
Trace	Sandra	332 Court St	Pittsburgh	15205

1) [Members] テーブル内の全てのフィールドの値を取り出す場合 :

`$jsonString := Selection to JSON ([Members])`

```
// $jsonString = [{"LastName":"Durant","FirstName":"Mark","Address":
// "25 Park St","Zip code":"15205","City":"Pittsburgh"},{"LastName":
// "Smith","FirstName":"John","Address":"24 Philadelphia Ave",
// "Zip code":"75203","City":"Dallas"},{"LastName":"Anderson",
// "FirstName":"Adeline","Address":"37 Market St","Zip code":
// "45205","City":"Cincinnati"},...]
```

2) フィールドを指定して、取り出すフィールドを二つだけに限定したい場合 :

```
QUERY ([Members];[Members]LastName="A@")
$jsonString := Selection to JSON ([Members];[Members]LastName;
    [Members]City)
    // $jsonString = [{"LastName":"Anderson","City":"Cincinnati"},
    // {"LastName":"Albert","City":"Houston"}]
```

3) *template* 表記を用いて、一つのフィールドだけを取り出したい場合 :

```
C_OBJECT ($template)
OB SET($template;"LastName";->[Members]LastName)//単一のフィールドを
指定
ALL RECORDS([Members])
$jsonString := Selection to JSON ([Members];$template)
    // $jsonString = [{"LastName":"Durant"},{"LastName":"Smith"},
    {"LastName":"Anderson"},{"LastName":"Albert"},
    {"LastName":"Leonard"},{"LastName":"Pradel"}]
```

*See also:* [JSON TO SELECTION](#)

## リストボックス

### LISTBOX DUPLICATE COLUMN

LISTBOX DUPLICATE COLUMN({\*; }object ; colPosition ; colName ; colVariable ; headerName ; headerVar{; footerName ; footerVar})

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	→ 複製したい列のオブジェクト名 (* 指定時)、または変数 (* 省略時)
positionCol	倍長整数	→ 新しく複製した列の位置
colName	文字列	→ 新しい列の名前
colVariable	配列、フィールド、変数	→ 列の配列変数またはフィールド、変数
headerName	文字列	→ 列のヘッダーのオブジェクト名
headerVar	倍長整数変数	→ 列のヘッダーの変数
footerName	文字列	→ 列のフッターのオブジェクト名
footerVar	倍長整数変数	→ 列のフッターの変数

新しい [LISTBOX DUPLICATE COLUMN](#) コマンドを使用すると、*object* と \* 演算子で指定した列をアプリケーションモードで複製することができます。デザインモードの方のオリジナルのフォームは変更されません。

---

*Note* 列を複製する機能は以前の 4D にもありましたが、デザインモードにおいてフォームエディターのコンテキストメニューの中にある**列複製**コマンドを使用することによってのみ可能、というものでした。

---

複製元となる列においてプロパティリストやオブジェクト管理コマンド (OBJECT SET COLOR 等) を使用して設定されたスタイルオプション (サイズ、背景色、書式等) は、複製先の列にも反映されます。また、フォームのオブジェクトメソッド、スタイルとカラー配列やイベントなどといったものも複製されます。

しかしながら、データソース (リストボックスに定義されたソースにより、配列またはセレクション) に加えてスタイル配列、カラー配列は複製されません。列を複製した際には、デベロッパが新しい列にてそれらのデータを定義しなおさなければなりません。

*object* と \* 演算子を使用して複製する列を指定します。任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で列変数を指定します。

---

*Note* このコマンドは階層リストボックスの最初の列を指定した場合には何もしません。

---

複製した列は、*colPosition* 引数で指定した位置の一つ前に置かれます。*colPosition* が列全体の総数より大きい場合、複製した列は最後の列の後ろに置かれます。

*colName* 引数と *colVariable* 引数には、オブジェクト名と、新しく複製された列の変数を渡します。

- 配列型のリストボックスの場合、変数名は列に表示される配列です。
- セレクション型のリストボックスの場合、*colVariable* 引数で指定したフィールドまたは変数の値が列に表示されます。これはリストボックスに関連付けられたセレクションのレコードごとに評価された値となります。この型の内容をしようするためには、リストボックスの「データソース」プロパティがカレントセレクションか命名セレクションに設定されている必要があります。

繰り返しになりますが、オリジナルの列のデータソースまでは複製されないことにご注意ください。複製された列に対し、新たにソースとなる変数、配列、フィールドを設定してあげる必要があります。

*headerName* 引数と *headerVariable* 引数には、新たに作成する列のヘッダーのオブジェクト名と変数を渡します。

同様に、*footerName* 引数と *footerVariable* 引数にオブジェクト名と変数を渡してあげることで新しい列のフッターを設定することもできます。  
*footerVariable* 引数省略時には 4D はフォーム変数を割り当てます。

**Note** オブジェクト名は、フォーム内において固有でなければなりません。  
*colName* 引数、*headerName* 引数、*footerName* 引数などで渡す名前がこれまでに使用されていないことを確認して下さい。重複していた場合、複製は行われず、エラーが発生します。

このコマンドはフォーム表示中に使用される必要があります。通常、フォームの On Load イベント内か、ユーザーのアクションへの反応 (On Clicked イベント) にて使用されます。

- ▶ 以下のような配列型のリストボックスの中で、"First Name" の列を複製して入力できるようにしたい場合の例を考えましょう。:

**Members**

Last name	First name	City
Durant	Mark	Pittsburgh
Smith	John	Dallas
Anderson	Adeline	Cincinnati
Peterson	Paul	Dallas
Harper	Harry	Cincinnati
Trace	Sandra	Pittsburgh

Add Middle Name

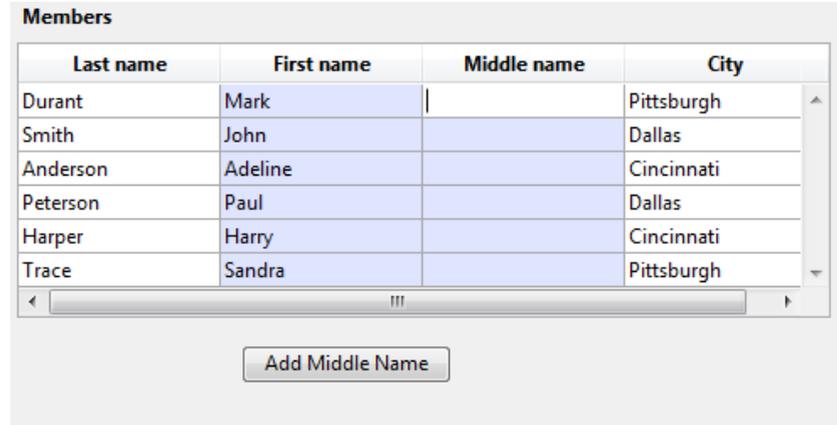
ボタンのコードは以下のようになります。:

```

ARRAY TEXT(arrFirstNames2;Records in table([Members]))
LISTBOX DUPLICATE COLUMN(*;"column2";3;"col2bis";arrFirstNames2;
  "FirstNameA";vHead2A)
OBJECT SET TITLE(*;"FirstNameA";"Middle Name")
EDIT ITEM(*;"col2A";0)

```

ボタンをクリックすると、リストボックスは以下のようになります。:



See also: [LISTBOX MOVE COLUMN](#)

## LISTBOX Get array

LISTBOX Get array({\*; }arrType) → Pointer

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数 (* 省略時)
arrType	倍長整数	→ 配列のタイプ
戻り値	ポインター	← プロパティと関連付けられた配列に対する ポインター

Note このコマンドは配列型のリストボックスに対してのみ有効です。

この新しいコマンド [LISTBOX Get array](#) は *object* と \* 演算子で指定されたリストボックスまたはリストボックスの列に対して、スタイル配列またはカラー配列を返します。

スタイル、カラー、背景色の配列は、デザインモードのプロパティリストを使用するか、新しいコマンド [LISTBOX SET ARRAY](#) 使用することで配列型のリストボックスと関連付けすることができます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で変数を指定します。対象がリストボックスなのか列なのかを、*object* 引数で指定します。

*arrType* 引数には、取得したいプロパティの配列の型を渡します。"List box" テーマ内にある以下の定数を使用することができます。:

定数	型	値
Listbox font color array	倍長整数	0
Listbox background color array	倍長整数	1
Listbox style array	倍長整数	2

戻り値は以下のどれかになります。:

- 要求されたプロパティの配列がそのリストボックスまたは列に関連付けされていない場合、Nil が返ってきます。
  - ユーザーによって定義されたプロパティの配列へのポインター
  - [LISTBOX SET ROW COLOR](#) や [LISTBOX SET ROW FONT STYLE](#) コマンドを使用して設定されたプロパティの配列へのポインター
- ▶ 典型的な使用例を紹介します。:

```
vPtr:=LISTBOX Get array(*;"MyLB";Listbox font color array)
// "MyLB" というリストボックスに関連付けされた
// フォントカラー配列を返します。
```

```
vPtr:=LISTBOX Get array(*;"Col4";Listbox style array)
// "Col4" リストボックスの列に関連付けされた
// フォントスタイル配列を返します。
```

*See also:* [LISTBOX SET ARRAY](#)

## LISTBOX GET OBJECTS

LISTBOX GET OBJECTS ( {\*; }object ; arrObjectNames )

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 ( 文字列 ) 省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数 (* 省略時)
arrObject-Names	テキスト配列	← リストボックスを構成するサブオブジェクト名 ( ヘッダー、列、フッター )

新しい [LISTBOX GET OBJECTS](#) コマンドは、*object* と \* 演算子で指定したリストボックスを構成するオブジェクトの、それぞれの名前を含んだ配列を返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で変数を指定します。この場合、文字列ではなく変数参照を渡します。

`arrObjectNames` には、コマンドからの配列を受け取るテキスト配列を渡します。配列には、オブジェクト名が以下の規則に従って表示順に返されます。

```
nameCol1
headerNameCol1
footerNameCol1
nameCol2
headerNameCol2
footerNameCol2
...
```

配列には、表示非表示に関わらず、全ての列に関して（列のフッターを含む）オブジェクト名が返されます。

このコマンドは `FORM GET OBJECTS` と `OBJECT Get type` コマンドを使用してフォームを解析する際に有用です。必要に応じてリストボックスのサブオブジェクトの名前を取得するために使用することができます。

- ▶ フォームをロードして、そのフォームに含まれる全てのオブジェクトの一覧を取得したい場合を考えます。

```
FORM LOAD ("MyForm")
ARRAY TEXT (arrObjects;0)
FORM GET OBJECTS (arrObjects)
ARRAY LONGINT (ar_type;Size of array (arrObjects))
ARRAY TEXT (ar_typeName;Size of array (arrObjects))
For ($i;1;Size of array (arrObjects))
  ar_type{$i}:=OBJECT Get type (*;arrObjects{$i})
  If (ar_type{$i}=Object type listbox)
    ARRAY TEXT (arrLBOBJECTS;0)
    LISTBOX GET OBJECTS (*;arrObjects{$i};arrLBOBJECTS)
  End if
End for
FORM UNLOAD
```

See also: [FORM LOAD](#)

## LISTBOX Get row color

LISTBOX Get row color({\*; }object ; row{; colorType) → Longint

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名（文字列）省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名（* 指定時）、または変数（* 省略時）

row 倍長整数 → 列番号  
 colorType 倍長整数 → Listbox font color (デフォルトの値) または  
Listbox background color  
 戻り値 倍長整数 ← カラーの値

*Note* このコマンドは配列型のリストボックスに対してのみ有効です。

この新しい LISTBOX Get row color コマンドは *object* と \* 演算子で指定したリストボックス内の行または単一のセルの色を返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で変数を指定します。

対象がリストボックスなのか列なのかを、*object* 引数で指定します。:

- *object* がリストボックス全体を指定している場合、コマンドは指定された行の色を返します。
- *object* がリストボックス内の列を指定している場合、コマンドは単一のセルの色を返します。

*row* 引数にはカラーを取得したい行を番号で指定します。

*Note* このコマンドは行の表示 / 非表示の状態は無視します。

*colorType* 引数には、行の背景色を取得したいのかフォントカラーを取得したいのかによって、Listbox background color または Listbox font color の定数のどちらかを渡します。省略時は、フォントカラーが返されます。

**注意:** 65P “スタイルとカラーにおける優先順位” に説明のある通り、行に割り当てられたカラーであっても、それが全てのセルに配色されるとは限りません。(以下の例を参照のこと)

- ▶ LISTBOX SET ROW COLOR コマンドの例題で用いた以下のリストボックスについての戻り値は、以下のようになります。:

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

vColor := LISTBOX Get row color(\*;"Col5";3)

vColor2 := LISTBOX Get row color(\*;"List Box";3)

vColor3 := LISTBOX Get row color(\*;"List Box";Listbox background color)

// vColor contains 0xFFFF00 (yellow)

// vColor2 contains 0x00FF (blue)

```
// vColor3 contains 0x00FF0000 (red)
```

See also: [LISTBOX SET ROW COLOR](#)

## LISTBOX Get row font style

LISTBOX Get row font style ({\*; }object ; row) → Longint

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数 (* 省略時)
row	倍長整数	→ 列の番号
戻り値	倍長整数	← スタイルの値

Note このコマンドは配列型のリストボックスに対してのみ有効です。

[LISTBOX Get row font style](#) は、*object* と \* 演算子によって指定されたリストボックス内の行または単一のセルのフォントスタイルを返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で変数を指定します。

対象がリストボックスなのか列なのかを、*object* 引数で指定します。:

- *object* がリストボックス全体を指定している場合、コマンドは指定された行のフォントスタイルを返します。
- *object* がリストボックス内の列を指定している場合、コマンドは単一のセルのフォントスタイルを返します。

*row* 引数にはフォントスタイルを取得したい行を番号で指定します。

Note このコマンドは行の表示 / 非表示の状態は無視します。

**注意:** 65P “[スタイルとカラーにおける優先順位](#)” に説明のある通り、行に割り当てられたフォントスタイルであっても、それが全てのセルに割り当てられるとは限りません。(以下の例を参照のこと)

- ▶ **LISTBOX SET ROW FONT STYLE** の例題で用いたリストボックスについての戻り値は、以下のようになります。:

text	text	text	text	text	text
text	text	text	text	text	text
<u>text</u>	<u>text</u>	<u>text</u>	<u>text</u>	<b>text</b>	<u>text</u>
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vStyle := LISTBOX Get row font style (*;"Col5";3)
vStyle2 := LISTBOX Get row font style(*;"List Box";3)
// vStyle contains 1 (Bold)
// vStyle2 contains 6 (Italic + Underline)
```

*See also:* **LISTBOX SET ROW FONT STYLE**

## LISTBOX MOVE COLUMN

LISTBOX MOVE COLUMN({\*; }object ; colPosition )

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	→ 移動したい列のオブジェクト名 (* 指定時)、 または変数 (* 省略時)
colPosition	倍長整数	→ 列の新しい位置

新しい **LISTBOX MOVE COLUMN** コマンドを使用すると、*object* と \* 演算子で指定した列をアプリケーションモードで移動させることができます。デザインモードの方のオリジナルのフォームは変更されません。

*Note* 列を移動させることは以前の 4D でもすでに可能でした。ただしそれは「ドラッグしない列」でないものを、マウスでドラッグできるというものでした。

*object* と \* 演算子を使用して移動する列を指定します。任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で列変数を指定します。

指定した列は、colPosition 引数で指定した位置の一つ前に移動されます。colPosition が列全体の総数より大きい場合、指定した列は最後の列の後ろに移動されます。

*Note* このコマンドは階層リストボックスの最初の列を指定した場合には何もしません。

このコマンドには列が「ドラッグしない列数」や「スクロールしない列数」で指定されているかどうかに関わってきます。つまり、「ドラッグしない列数」に含まれる場合は列を移動させることはできません。

ユーザーによって列が移動された場合と異なり、このコマンドは On Column Moved イベントを生成しません。

- ▶ リストボックスの 2 列目と 3 列目を入れ替えたい場合：

**LISTBOX MOVE COLUMN**(\*;"column2";3)

*See also:* [LISTBOX DUPLICATE COLUMN](#)

## LISTBOX SET ARRAY

LISTBOX SET ARRAY({\*; }object ; arrType ; arrPtr)

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名（文字列）省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名（* 指定時）、または変数（* 省略時）
arrType	倍長整数	→ 配列のタイプ
arrPtr	ポインタ	→ プロパティに関連付ける配列を指定

*Note* このコマンドは配列型のリストボックスに対してのみ有効です。

この新しいコマンドは *object* と \* 演算子で指定されたリストボックスまたはリストボックスの列に対して、スタイル配列またはカラー配列を関連付けます。

これまでの 4D で配列型のリストボックスにスタイル、文字色、背景色の配列を関連付けるためには、デザインモードのプロパティリストを使用するしかありませんでした。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で変数を指定します。対象がリストボックスなのか列なのかを、*object* 引数で指定します。：

*arrType* 引数にはリストボックスまたは列に関連付けたい配列の種類を、“List box” のテーマ内にある以下の定数を渡すことによって指定します。：

定数	型	値
Listbox font color array	倍長整数	0
Listbox background color array	倍長整数	1
Listbox style array	倍長整数	2

*arrPtr* 引数には、制御したいプロパティを制御するための配列に対するポインタを渡します (フォントカラー、背景色あるいはフォントスタイル)。

- ▶ 4 列目のフォントカラー配列を 10 列目にも使いたいという場合を考えます。:

```
// 4 列目で使用している配列に対するポインタを取得します。
$Pointer:=LISTBOX Get array(*;"Col4";Listbox font color array)
// 存在するかどうかをチェックします
If(Not(Nil($Pointer)))
    // 10 列目へ適用します。
    LISTBOX SET ARRAY(*;"Col10";Listbox font color array;$Pointer)
End if
```

See also: [LISTBOX Get array](#)

## LISTBOX SET ROW COLOR

LISTBOX SET ROW COLOR({\*; }object ; row ; color{; colorType})

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数 (* 省略時)
row	倍長整数	→ 列番号
color	倍長整数	→ RGB カラー
colorType	倍長整数	→ <a href="#">Listbox font color</a> ( デフォルトの値 ) または <a href="#">Listbox background color</a>

Note このコマンドは配列型のリストボックスに対してのみ有効です。

この新しいコマンドは配列リストボックス内で、*object* 引数または \* 演算子で指定された行またはセルの色を設定します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で列変数を指定します。

対象がリストボックスなのか列なのかを、*object* 引数で指定します。

- *object* がリストボックスを指定するとき、コマンドは其中で指定した行全体に反映されます。
- *object* がリストボックスの列を指定するとき、コマンドはその列内の、指定された行にある単一のセルに対して反映されます。

*row* 引数には指定した列を番号で指定し、*color* 引数で反映させたい色を指定して下さい。

Note このコマンドは列の表示 / 非表示の状態は無視します。

`color` 引数には、指定したい色を RGB カラー値で渡します。RGB カラーについての詳細は、OBJECT SET RGB COLORS コマンドの詳細を参照して下さい。一つ上の行の配色をそのまま使用したい場合には、`color` 引数に `Listbox inherited` 定数を渡します。詳細については、67P “スタイルとカラーの継承” を照して下さい。

`colorType` 引数には、行の背景色を指定したいのかフォントカラーを指定したのかによって、`Listbox background color` または `Listbox font color` のどちらかの定数を渡します。省略時には、指定した色はフォントカラーとして使用されます。

このコマンドは、列またはリストボックスに対して定義されていた可能性のあるカラー配列内の値を変更します。この配列が未定義だった場合、このコマンドは、`LISTBOX Get array` コマンドを使用してアクセス可能な配列を動的に作成します。

他のリストボックスプロパティ（全体のプロパティ、列のカラー配列など）によって、このコマンドと異なる色が指定された場合、4D 内の優先順位に応じて処理されます。この優先順位の詳細に関しては、65P “スタイルとカラーにおける優先順位” を参照して下さい。

- ▶ 配列型のリストボックス内において、行全体と単一のセルの色を指定したい場合を考えましょう。:

// 単一のセルのフォントカラー（黄色）の定義

```
LISTBOX SET ROW COLOR(*;"Col5";3;0x00FFFF00)
```

// 3 行目の背景色とフォントカラーの定義

// 背景色は赤、フォントカラーは青

```
LISTBOX SET ROW COLOR(*;"ListBox";3;0x00FF0000;
```

```
  Listbox background color)
```

```
LISTBOX SET ROW COLOR(*;"List Box";3;0x000000FF)
```

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

See also: `LISTBOX Get row color`

## LISTBOX SET ROW FONT STYLE

LISTBOX SET ROW FONT STYLE({\*; }object ; row ; style)

引数	型	説明
*	演算子	→ 指定時、Object はオブジェクト名（文字列） 省略時、Object は変数
object	フォームオブジェクト	→ オブジェクト名（* 指定時）、 または変数（* 省略時）
row	倍長整数	→ 列番号
style	倍長整数	→ フォントスタイル

*Note* このコマンドは配列型のリストボックスに対してのみ有効です。

この新しいコマンドは配列リストボックス内で、*object* 引数または \* 演算子で指定された行全体または単一のセルの色を設定します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数で列変数を指定します。対象がリストボックスなのか列なのかを、*object* 引数で指定します。:

- *object* がリストボックスを指定するとき、コマンドは其中で指定した行全体に反映されます。
- *object* がリストボックスの列を指定するとき、コマンドはその列内の、指定された行にある単一のセルに対して反映されます。

*row* 引数には指定した行を番号で指定し、*style* 引数で反映させたいスタイルを指定して下さい。

*Note* このコマンドは列の表示 / 非表示の状態は無視します。

*style* 引数には、指定したいスタイルの値を渡します。指定する際には "Font Style" テーマ内にある、以下の定数のどれか一つ、あるいはその組み合わせを指定して下さい。:

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

リストボックスや列にフォントスタイル配列が設定されている場合、指定された列の要素に関してのみ変更されます。言い換えると、この場合コマンドを実行するのはフォントスタイル配列の要素を変更するのと同等の効果があります。

リストボックスや列にフォントスタイル配列が何も設定されていない場

合、このコマンドが使用されたときに作成されます。この配列には [LISTBOX Get array](#) を使用することによってアクセス可能です。

他のリストボックスプロパティ（一般のプロパティ、行スタイル配列など）によって、このコマンドと異なるフォントスタイルが指定された場合、4D 内の優先順位に応じて処理されます。この優先順位の詳細に関しては、[65P “スタイルとカラーにおける優先順位”](#) を参照して下さい。

*Note* リストボックス全体に対するスタイルの指定より単一の列に対するスタイルの方が優先されるので、リストボックス全体に対してこのコマンドは、列に対するスタイル配列が設定されていない場合のみ有効です。

- ▶ ある配列リストボックスに以下のスタイルを適用する場合を考えます。：
  - フォントスタイル配列がリストボックス全体に適用されているとします。(ArrGlobalStyle)
  - フォントスタイル配列が、5 行目に適用されているとします。(ArrCol5Style)
  - 他の列にはスタイル配列は適用されていません。

**LISTBOX SET ROW FONT STYLE(\*;"Col5";3;Bold)**

// ArrCol5Style{3}:=Bold と同様の効果

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	<b>text</b>	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

5 列目の 3 行目

**LISTBOX SET ROW FONT STYLE(\*;"List Box";3;Italic+Underline)**

// ArrGlobalStyle{3}:=Italic+Underline と同様の効果

text	text	text	text	text	text
text	text	text	text	text	text
<u>text</u>	<u>text</u>	<u>text</u>	<u>text</u>	<b>text</b>	<u>text</u>
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

2 つ目のコマンドの後、3 行目のセルは下線付きのイタリックになりましたが、5 列目のセルだけは太字のままになっています（列スタイル配列の方がリストボックス配列より優先されるからです）。

*See also:* [LISTBOX Get row font style](#)

## メッセージ

### DISPLAY NOTIFICATION

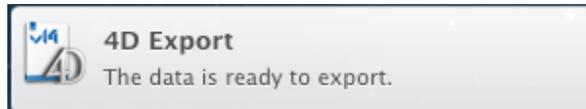
DISPLAY NOTIFICATION ( title ; text { ; duration } )

4D の v14 では、OS X(バージョン 10.8) 環境下において、**DISPLAY NOTIFICATION** コマンドを使用することが出来るようになりました。このコマンドを使用すると、画面の右上隅に可動式の小さな通知ウィンドウを表示します。

Apple の仕様により、この通知はアプリケーションが前面に来ていないときのみ表示されるという点にご注意ください。ただし画面上に表示されなかったとしても "notification center" のリストには表示されます。

- ▶ OS X 環境下において、以下のように記述することが出来ます。:

**DISPLAY NOTIFICATION** ("4D Export";"The data is ready to export.")



## オブジェクトプロパティ

このテーマは、4D の v14 では「オブジェクト(フォーム)」に名称が変更されました。変更されたコマンドや関数に関する詳細な情報に関しては、197P「[オブジェクト\(フォーム\)](#)」を参照して下さい。

## オブジェクト(フォーム)

*Compatibility note* "オブジェクト(フォーム)" テーマは、以前の 4D では "オブジェクトプロパティ" と呼称されていたものです。さらに、混乱を避けるために、以前は "Form objects" という名称だった定数テーマは 4D v14 では "Form Objects (Properties)" に名前が変わっています。

### New commands

#### GET STYLE SHEET INFO

GET STYLE SHEET INFO(styleSheetName ; font ; size ; styles)

引数	型	説明
styleSheetName	テキスト	→ スタイルシート名

font	テキスト	←	文字のフォント
size	倍長整数	←	フォントサイズ
styles	倍長整数	←	スタイルの値

**GET STYLE SHEET INFO** コマンドは、*styleSheetName* 引数で指定したスタイルシート名のカレントの設定を返します。

*styleSheetName* 引数には、デザインモードで定義されたスタイルシート名を渡します。自動スタイルシートを指定するためには、"Font Styles" テーマ内にある Automatic style sheet 定数を使用して下さい。:

定数	型	値
Automatic style sheet	テキスト	"__automatic__"

*font* 引数には、カレントのプラットフォームのスタイルシートと関連付けられているフォントの名前がコマンドから返されます。

*size* 引数には、カレントのプラットフォームのスタイルシートと関連付けられている文字サイズがポイント数でコマンドから返されます。

*styles* 引数には、カレントのプラットフォームのスタイルシートと関連付けられているスタイルに対応した値がコマンドから返されます。返された値に対応するスタイルは、以下の表のとおりで、"Font Styles" テーマ内にあります。:

定数	型	値
Bold	倍長整数	1
Bold and Italic	倍長整数	3
Bold and Underline	倍長整数	5
Italic	倍長整数	2
Italic and Underline	倍長整数	6
Plain	倍長整数	0
Underline	倍長整数	4

このコマンドが正しく実行された場合には、OK システム変数は 1 に変更されます。そうでない場合 (例えば *styleSheetName* 引数で指定したスタイルシートが存在しないなど) は、OK システム変数は 0 に設定されます。

- ▶ 自動スタイルシートのカレント設定を調べたい場合を考えます。:

**C\_LONGINT**(\$size;\$style)

**C\_TEXT** (\$font)

**GET STYLE SHEET INFO** (Automatic style sheet;\$font;\$size;\$style)

*See also:* [LIST OF STYLE SHEETS](#), [OBJECT SET STYLE SHEET](#)

## LIST OF STYLE SHEETS

LIST OF STYLE SHEETS(arrStyleSheets)

引数	型	説明
arrStyleSheets	テキスト配列	← アプリケーション内で定義されたスタイルシートの名前

新コマンド [LIST OF STYLE SHEETS](#) はアプリケーションのスタイルシートを *arrStyleSheets* という配列に返します。

*arrStyleSheets* テキスト配列が未定義だった場合、この配列はコマンドによって自動的に作成されます。配列のサイズは、定義されているスタイルシートの数に応じて自動的に決められます。

コマンド実行後、配列の各要素にはスタイルシートの名前が格納されます。これらの名前はスタイルシートエディターと同様にアルファベット順にソートされます。配列の第1要素には自動スタイルシートを意味する "\_\_automatic\_\_" が必ず入ります。

- ▶ アプリケーション内に、以下のようにスタイルシートが定義されていたとします。:



下のコードを実行すると、戻り値の配列の要素には以下の様に値が格納されています。:

**LIST OF STYLE SHEETS(\$arrStyles)**

```
// $arrStyles{1} には "__automatic__"
// $arrStyles{2} には "Buttons"
// $arrStyles{3} には "Input_fields"
// $arrStyles{4} には "Default"
// $arrStyles{5} には "Labels"
// $arrStyles{6} には "Variables"
```

*See also:* [GET STYLE SHEET INFO](#)

## OBJECT Get action

OBJECT Get action( { \* ; } object) → Text

引数	型	説明
----	---	----

- \* 演算子 → 指定時: object はオブジェクト名 (文字列)  
省略時: object は変数またはフィールド
- object フォームオブジェクト → オブジェクト名 (\* 指定時)、  
または変数やフィールド (\* 省略時)
- 戻り値 テキスト ← 関連付けられた標準アクション

新しい **OBJECT Get action** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトと関連付けられた標準アクションのコードを返します。

オブジェクトに対する標準アクションは、デザインモードにおいてプロパティリストで設定するか、**OBJECT SET ACTION** コマンドという新しいコマンドを使用することによって設定できます。

このコマンドはオブジェクトに関連付けられた標準アクションのコードを格納した文字列を返します。返された値と対応する定数は "Text Values for Associated Standard Action" テーマ内にあります。以下の通りです。:

定数	型	値
Object Refresh current URL action	テキスト	"39"
Object Show Clipboard action	テキスト	"23"
Object Add subrecord action	テキスト	"14"
Object Goto page action	テキスト	"15"
Object Cancel action	テキスト	"1"
Object Undo action	テキスト	"17"
Object Stop loading URL action	テキスト	"40"
Object Paste action	テキスト	"20"
Object Copy action	テキスト	"19"
Object Cut action	テキスト	"18"
Object MSC action	テキスト	"36"
Object Last record action	テキスト	"6"
Object Last page action	テキスト	"11"
Object Clear action	テキスト	"21"
Object Previous record action	テキスト	"4"
Object Next record action	テキスト	"3"
Object Edit subrecord action	テキスト	"12"
Object Open back URL action	テキスト	"37"
Object Open next URL action	テキスト	"38"
Object Previous page action	テキスト	"9"
Object Next page action	テキスト	"8"
Object First record action	テキスト	"5"
Object action Première page	テキスト	"10"
Object Database Settings action	テキスト	"32"
Object Quit action	テキスト	"27"
Object Redo action	テキスト	"31"

定数	型	値
Object Return to Design mode action	テキスト	"35"
Object Automatic splitter action	テキスト	"16"
Object Delete record action	テキスト	"7"
Object Delete subrecord action	テキスト	"13"
Object Test Application action	テキスト	"26"
Object Select all action	テキスト	"22"
Object Accept action	テキスト	"2"
Object No standard action	テキスト	"0"

- ▶ フォーム内にあるオブジェクトのうち、まだ何もアクションが関連付けられていないものに関して全て「キャンセル」のアクションを関連付けたい場合を考えます。:

```
ARRAY TEXT($arrObjects;0)
```

```
FORM GET OBJECTS($arrObjects)
```

```
For ($i;1;Size of array($arrObjects))
```

```
  If (Object Get action(*;$arrObjects{$i})=Object No standard action)
```

```
    OBJECT SET ACTION(*;$arrObjects{$i};Object Cancel action)
```

```
  End if
```

```
End for
```

*See also:* [OBJECT SET ACTION](#)

## OBJECT Get border style

OBJECT Get border style ( { \* ; } object ) → Longint

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
戻り値	倍長整数	← 境界線スタイル

新しい [OBJECT Get border style](#) コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトの境界線スタイルを返します。

オブジェクトの境界線スタイルはデザインモードのプロパティリストを使用するか、新コマンド [OBJECT SET BORDER STYLE](#) を使用することで設定できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

コマンドは、境界線スタイルに対応する値を返します。返される値は "Form objects (Properties)" テーマの中にあります。以下の通りです。:

定数 ( 値 )	説明
<u>Border None</u> (0)	オブジェクトは境界線を持っていません。
<u>Border Double</u> (5)	オブジェクトの境界線は二重線 (1 ピクセル離れた 2 本の 1pt の線) です。
<u>Border Plain</u> (1)	オブジェクトの境界線は連続した一本の 1pt の線です。
<u>Border Raised</u> (3)	オブジェクトの境界線は浮き上がったような 3D エフェクトです。
<u>Border Sunken</u> (4)	オブジェクトの境界線は沈み込んだような 3D エフェクトです。
<u>Border System</u> (6)	オブジェクトの境界線はシステムのグラフィック仕様に沿ったものです。
<u>Border Dotted</u> (2)	オブジェクトの境界線は 1pt の点線です。

*See also:* [OBJECT SET BORDER STYLE](#)

## OBJECT Get context menu

OBJECT Get context menu (`{* ;} object`) → Boolean

引数	型	説明
*	演算子	→ 指定時: <code>object</code> はオブジェクト名 (文字列) 省略時: <code>object</code> は変数またはフィールド
<code>object</code>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
戻り値	ブール	← True = コンテキストメニューは使用可能 False = コンテキストメニューは使用不可

新しい [OBJECT Get context menu](#) コマンドは `object` 引数と \* 演算子で指定されたオブジェクトの「コンテキストメニュー」のカレントの状態を返します。

コンテキストメニューのオプションは、デザインモードのプロパティリストを使用するか、[OBJECT SET CONTEXT MENU](#) という新しいコマンドを使用して設定することができます。

任意の \* 演算子を渡した場合、`object` 引数でオブジェクト名を文字列で指定します。省略時には `object` 引数でフィールドまたは変数を指定します。

`object` で指定したオブジェクトにおいてコンテキストメニューが使用可能になっている場合は True を、そうでない場合には False を返します。

*See also:* [OBJECT SET CONTEXT MENU](#)

## OBJECT Get data source

OBJECT Get data source ( { \* ; } object ) → Pointer

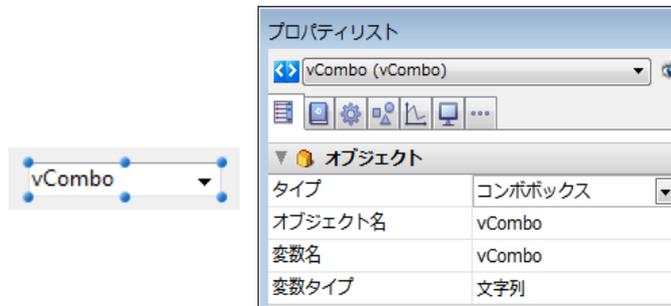
引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
戻り値	ポインター	← オブジェクトのカレントデータソースに対するポインター

新しい **OBJECT Get data source** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのカレントのデータソースを返します。

オブジェクトのデータソースはデザインモードのプロパティリストを使用するか、新コマンド **OBJECT SET DATA SOURCE** を使用することで定義できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

- ▶ フォーム内に以下の様に定義されたコンボボックスがあるとします。 :



以下のコードを実行すると、 :

```
$vPtr := OBJECT Get data source (*;"vCombo")
//vptr の値は ->vCombo
```

*See also:* **OBJECT SET DATA SOURCE**

## OBJECT GET EVENTS

OBJECT GET EVENTS({ \* ; } object; arrEvents)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド

```

object   フォーム      → オブジェクト名または "" (* 指定時)
          オブジェクト  または変数やフィールド (* 省略時)
arrEvents 倍長整数配列 ← 有効化されたイベントの配列

```

新しい **OBJECT GET EVENTS** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのカレントの設定を取得します。

フォームイベントはプロパティリストを通して有効化 / 無効化することができます。また、同一のプロセスにおいてなら **OBJECT SET EVENTS** コマンドを使用して設定することも可能です。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。フォーム自体のイベントの設定を取得したい場合は、任意の \* 演算子を渡したうえで、*object* 引数に空の文字列 "" を渡します。こうすることでカレントフォームをこれで指定します。

*Note* テーブルに関連付けられたサブフォームのイベントを取得したい場合、オブジェクト名を使用する記法しか使用できません。

*arrEvents* 引数には、倍長整数配列を渡します。この配列はコマンドが実行されたときに自動的にリサイズされ、オブジェクトまたはフォームの中で有効化されている、定義済みのイベントかカスタムイベントのコードを全て受け取ります。この値と対応する定数は "Form Events" テーマ内にあります。

オブジェクトまたはフォームに対してメソッドが何も関連付けられていない場合は、*arrEvents* は空で返される点に注意して下さい。

- ▶ イベントを2つ有効化して、オブジェクトのイベントのリストを取得したい場合を考えます。:

```
ARRAY LONGINT($ArrCurrentEvents;0)
```

```
ARRAY LONGINT($ArrEnabled;2)
```

```
$ArrEnabled{1}:=On Header Click
```

```
$ArrEnabled{2}:=On Footer Click
```

```
OBJECT SET EVENTS(*;"Col1";$ArrEnabled;Enable events others unchanged)
```

```
OBJECT GET EVENTS(*;"Col1";$ArrCurrentEvents)
```

*See also:* **OBJECT SET EVENTS**

## OBJECT Get indicator type

OBJECT Get indicator type ( { \* ; } object) → Longint

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数

object フォームオブジェクト → オブジェクト名 (\* 指定時)、  
または変数 (\* 省略時)

戻り値 倍長整数 ← インジケータータイプ

新しい **変更されたコマンド** コマンドは、*object* 引数と \* 演算子で指定されたサーモメーターのカレントのインジケータータイプを返します。

インジケータータイプは、デザインモードのプロパティリストを使用するか、新コマンド **OBJECT SET INDICATOR TYPE** を使用することによって定義できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

返された値は、"Form objects (Properties)" テーマ内にある以下の定数と対応しています。:

定数 ( 値 )	説明
<u>Barber shop</u> (2)	連続したアニメーションを表示するバー
<u>Progress bar</u> (1)	標準の進捗バー
<u>Asynchronous progress bar</u> (3)	連続したアニメーションを表示する、回転型のインジケータータイプ

*See also:* **OBJECT SET INDICATOR TYPE**

## OBJECT Get list reference

OBJECT Get list reference( { \* ; } object{ ; listType } ) → ListRef

引数	型	説明
*	演算子	→ 指定時: <i>object</i> はオブジェクト名 ( 文字列 ) 省略時: <i>object</i> は変数またはフィールド
<i>object</i>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
<i>listType</i>	倍長整数	→ リストの種類: 選択リスト、指定リスト、 除外リスト
戻り値	ListRef	← リストの参照番号

新しい **OBJECT Get list reference** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトと関連付けされた階層リストの参照番号 (*ListRef*) を返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*listType* 引数を省略した場合、コマンドは自動的にオブジェクトに割り当てられた選択リストの参照番号を返します。また、*listType* 引数に "Form objects (Properties)" テーマ内にある以下の定数を渡すことにより、指定リストと除外リストの参照番号を取得することが出来ます。:

定数 ( 値 )	説明
<u>Choice list</u> (0)	選択できる値のリストを返します。(プロパティリスト内の「選択リスト」)(デフォルトの動作)
<u>Required list</u> (1)	入力できる値のリストを返します。(プロパティリスト内の「指定リスト」)
<u>Excluded list</u> (2)	入力できない値のリストを返します。(プロパティリスト内の「除外リスト」)

*listType* で指定した種類の階層リストが関連付けされていない場合、コマンドは 0 を返します。

*See also:* [OBJECT SET LIST BY REFERENCE](#), [OBJECT Get list name](#)

## OBJECT GET MAXIMUM VALUE

OBJECT GET MAXIMUM VALUE({\*; }object ; maxValue)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名(*指定時)、または変数やフィールド(*省略時)
maxValue	日付   時間   整数   倍長整数   64 bit 整数   実数   フロート	← オブジェクトのカレントの最大値

新しい [OBJECT GET MAXIMUM VALUE](#) コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのカレントの最大値を *maxValue* 変数に返します。

最大値のプロパティは、デザインモードのプロパティリストを使用するか、新コマンド [OBJECT SET MAXIMUM VALUE](#) を使用することによって定義できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*See also:* [OBJECT SET MAXIMUM VALUE](#), [OBJECT GET MINIMUM VALUE](#),

## OBJECT GET MINIMUM VALUE

OBJECT GET MINIMUM VALUE({\*; }object ; minValue)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
minValue	日付   時間   整数   倍長整数   64 bit 整数   実数   フロート	← オブジェクトのカレントの最小値

新しい **OBJECT GET MINIMUM VALUE** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのカレントの最少値を *minValue* 変数に返します。

最小値のプロパティは、デザインモードのプロパティリストを使用するか、新コマンド **OBJECT SET MINIMUM VALUE** を使用することによって定義できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*See also:* **OBJECT SET MINIMUM VALUE**

## OBJECT Get multiline

OBJECT Get multiline ( {\*; } object ) → Longint

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
戻り値	倍長整数	← 複数行の状態

新しい **OBJECT Get multiline** コマンドは *object* 引数と \* 演算子で指定されたオブジェクトの「複数行」のオプションのカレントの状態を返します。

「複数行」のオプションはデザインモードのプロパティリストを使用するか、**OBJECT SET MULTILINE** という新しいコマンドを使用して設定することができます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

返ってきた値は "Form objects (Properties)" テーマ内にある以下の定数と対応しています。:

定数 ( 値 )	説明
<u>Multiline Auto</u> (0)	単独行のエリアでは、行に表示しきれない単語は切り落とされ、改行はされません。 複数行のエリアでは、改行が行われます。
<u>Multiline No</u> (2)	改行は禁止されます。テキストは必ず単独行として表示されます。文字列かテキストフィールドか変数に改行が含まれていたとしても、改行は行われません。
<u>Multiline Yes</u> (1)	単独行のエリアでは、テキストは最初の改行までか、単語全体を表示できる最後の単語までが表示されます。その後改行が挿入されるので、キーを押すことによってエリアの内容をスクロールすることができます。 複数行のエリアでは、自動で改行が行われます。

*Note* **OBJECT Get multiline** コマンドを、複数行のオプションがないオブジェクトに対して使用した場合、コマンドは 0 を返します。

*See also:* **OBJECT SET MULTILINE**

## OBJECT Get placeholder

**OBJECT Get placeholder** ( { \* ; } object ) → Text

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 ( 文字列 ) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
戻り値	テキスト	← オブジェクトと関連付けられたプレースホルダーテキスト

新しい **OBJECT Get placeholder** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトと関連付けられたプレースホルダーのテキストを返します。もし指定したオブジェクトにプレースホルダーのテキストが何も関連付けされていない場合は空の文字列を返します。

プレースホルダーテキストは、プロパティリストを使用するか、**OBJECT SET PLACEHOLDER** コマンドを使用することで定義できます。詳細な情報に関しては、53P “プレースホルダーテキスト” を参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

プレースホルダーにプロパティリストを通じて `xliff` 参照が定義されていた場合、コマンドは参照値ではなく、`":xliff:resname"` という原文をそのまま返します。

- ▶ あるフィールドのプレースホルダーのテキストを取得したい場合を考えます。:

```
$txt:=OBJECT Get placeholder ([People]LastName)
```

*See also:* [OBJECT SET PLACEHOLDER](#)

## OBJECT GET PRINT VARIABLE FRAME

OBJECT GET PRINT VARIABLE FRAME({\*; }object ; variableFrame{; fixedSubform})

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
variableFrame	ブール	← True = 可変長フレームを使用 False = 固定長フレームを使用
fixedSubform	倍長整数	← 固定長フレーム時のサブフォームに対するオプション

新しい [OBJECT GET PRINT VARIABLE FRAME](#) コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのフレーム印刷のオプションの状態を取得します。

フレーム印刷のプロパティはプロパティリストか、[OBJECT SET PRINT VARIABLE FRAME](#) という新しいコマンドを使用することで設定できます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*variableFrame* 引数には、フレーム印刷のオプションの状態がブールで返されます。True の場合は「可変」を、False の場合は「固定」を意味します。

*object* で指定したオブジェクトがサブフォームで、フレーム印刷が「固定」に設定されている場合に限り、コマンドは *fixedSubform* 引数に固定フレーム印刷時のオプションの状態を返します。返される値は "Form objects (Properties)" テーマ内にあります。以下の通りです。:

定数 ( 値 )	説明
----------	----

<u>Print Frame fixed with truncation</u> (1)	サブフォームのエリア内に収まるレコードのみを印刷します。フォームは一度しか印刷されず、印刷されないレコードは無視されます。
<u>Print Frame fixed with multiple records</u> (2)	フレームのサイズが変わりませんが、全てのレコードが印刷されるまでフォームは複数回印刷されます。

*See also:* [OBJECT SET PRINT VARIABLE FRAME](#)

## OBJECT Get style sheet

OBJECT Get style sheet( { \* ; } object) → Text

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
戻り値	テキスト	← スタイルシート名

新しい [OBJECT Get style sheet](#) コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトと関連付けられたスタイルシート名を返します。

スタイルシートはデザインモードにおいてプロパティリストで設定することができます。または、セッションの間においては、[OBJECT SET STYLE SHEET](#) コマンドを使用して設定することもできます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

コマンドから以下のどれかが返されます。:

- スタイルシート名
- 自動スタイルシートが設定されていた場合は定数 "\_\_automatic\_\_" が返されます。
- スタイルシートが何も設定されていない場合は空の文字列 ("") が返されます。

コマンドで複数のオブジェクトを指定した場合、コマンドが最初に見つけたオブジェクトのスタイルシートのみが返されます。

*See also:* [OBJECT SET STYLE SHEET](#)

## OBJECT Get text orientation

OBJECT Get text orientation( { \* ; } object) → Longint

引数	型	説明
----	---	----

- \* 演算子 → 指定時 : object はオブジェクト名 ( 文字列 )  
省略時 : object は変数またはフィールド
- object フォームオブジェクト → オブジェクト名 ( \* 指定時 )、  
または変数やフィールド ( \* 省略時 )
- 戻り値 倍長整数 ← テキストの回転角度

新しい **OBJECT Get text orientation** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのテキストに適用されたカレントの方向を返します。

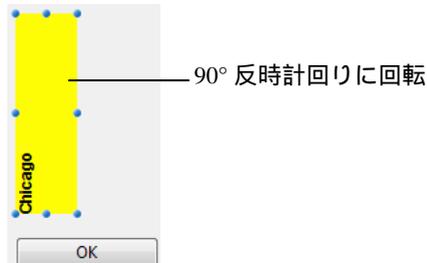
「方向」のオプションは、デザインモードのプロパティリストを使用するか (76P “**テキストの回転**” を参照して下さい)、**OBJECT SET TEXT ORIENTATION** という新しいコマンドを使用して設定することができます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

返ってきた値は "Form objects (Properties)" テーマ内にある以下の定数と対応しています。∴

定数 ( 値 )	説明
<u>Orientation 0°</u> (0)	回転なし ( 初期値 )
<u>Orientation 90° right</u> (90)	テキストは時計回りに 90° 回転
<u>Orientation 180°</u> (180)	テキストは時計回りに 180° 回転
<u>Orientation 90° left</u> (270)	テキストは反時計まわりに 90° 回転

- ▶ 以下のような、フォームエディターで "90° 反時計回り" が適用されているオブジェクトがあるとき、：



```
$vOrt:=OBJECT Get text orientation (*;"myText")//$vOrt=270
```

フォームを実行した際、以下のコマンドを実行すると、：

```
OBJECT SET TEXT ORIENTATION (*;"myText";Orientation 180°)
```

... オブジェクトは以下のように表示されます。:



```
$vOrt:=OBJECT Get text orientation (*;"myText") // $vOrt=180
```

*See also:* [OBJECT SET TEXT ORIENTATION](#)

## OBJECT Get three states checkbox

OBJECT Get three states checkbox( { \* ; } object) → Boolean

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
戻り値	ブール	← True = スリーステートチェックボックス False = 標準のチェックボックス

新しい [OBJECT Get three states checkbox](#) コマンドは、*object* 引数と \* 演算子で指定されたチェックボックスの「スリーステート」の現在の状態を返します。

スリーステートのプロパティは、プロパティリストを使用するか、同一プロセスにおいてであれば [OBJECT SET THREE STATES CHECKBOX](#) を呼び出して設定することができます。

*See also:* [OBJECT SET THREE STATES CHECKBOX](#)

## OBJECT Get type

OBJECT Get type( { \* ; } object) → Longint

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数 (* 省略時)
戻り値	倍長整数	← オブジェクトのタイプ

新しい **OBJECT Get type** コマンドは、カレントフォームにおいて *object* 引数と \* 演算子で指定されたオブジェクトのタイプを返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。線や四角といった静的なオブジェクトを処理する場合にはこの記法を使用しなければなりません。

\* 演算子省略時には *object* 引数でフィールドまたは変数を指定します。

*Note* このコマンドを複数のオブジェクトに対して適用した場合、最背面にあるオブジェクトのタイプが返されます。

返された値は "Form Object Types" テーマ内にある以下の定数と対応しています。:

定数	型	値
Object type unknown	倍長整数	0
Object type static text	倍長整数	1
Object type static picture	倍長整数	2
Object type text input	倍長整数	3
Object type picture input	倍長整数	4
Object type radio button field	倍長整数	5
Object type hierarchical list	倍長整数	6
Object type listbox	倍長整数	7
Object type listbox header	倍長整数	8
Object type listbox column	倍長整数	9
Object type listbox footer	倍長整数	10
Object type combobox	倍長整数	11
Object type popup drop down	倍長整数	12
Object type hierarchical popup	倍長整数	13
Object type picture popup	倍長整数	14
Object type push button	倍長整数	15
Object type 3D button	倍長整数	16
Object type highlight button	倍長整数	17
Object type invisible button	倍長整数	18
Object type picture button	倍長整数	19
Object type button grid	倍長整数	20
Object type group	倍長整数	21
Object type radio button	倍長整数	22
Object type 3D radio button	倍長整数	23
Object type picture radio button	倍長整数	24
Object type checkbox	倍長整数	25
Object type 3D checkbox	倍長整数	26

Object type progress indicator	倍長整数	27
Object type dial	倍長整数	28
Object type ruler	倍長整数	29
Object type groupbox	倍長整数	30
Object type rectangle	倍長整数	31
Object type line	倍長整数	32
Object type rounded rectangle	倍長整数	33
Object type oval	倍長整数	34
Object type matrix	倍長整数	35
Object type splitter	倍長整数	36
Object type tab control	倍長整数	37
Object type plugin area	倍長整数	38
Object type subform	倍長整数	39
Object type web area	倍長整数	40

### OBJECT Is styled text OBJECT Is styled text( { \* ; } object) → Boolean

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
戻り値	ブール	← オブジェクトがマルチスタイルテキストであれば True、そうでなければ False

新しい **OBJECT Is styled text** コマンドは、*object* 引数と \* 演算子にて指定したオブジェクトで、「マルチスタイル」のオプションにチェックが入っていた場合に **True** を返すというものです。

「マルチスタイル」オプションは、複数のスタイルバリエーションを一つのエリアで使用可能にするものです。詳細な情報に関しては、*Design Reference* マニュアルを参照して下さい。

「マルチスタイル」のオブジェクトは、"**スタイル付テキスト**" テーマ内のコマンドを使うことによってプログラムで管理することもできます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

- ▶ あるフォーム内に、二つの異なるオブジェクトによって表現されたフィールドがあり、片方のオブジェクトは「マルチスタイル」のプロパ

ティにチェックがされていて、もう片方にはチェックがされていません。  
このとき、以下の様なコードでこれを判別することが出来ます。:

```
$Style:=OBJECT Is styled text(*;"Styled_text")
// True が返されます (「マルチスタイル」がチェックされています)
```

```
$Style:=OBJECT Is styled text(*;"Plain_text")
// False が返されます (「マルチスタイル」はチェックされていません)
```

## OBJECT SET ACTION

OBJECT SET ACTION({\*; }object; action)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
action	テキスト	→ 関連付けるアクション

新しい **OBJECT SET ACTION** コマンドはカレントプロセスにおいて、*object* 引数と \* 演算子で指定されたオブジェクトの標準アクションを変更します。

これまでのバージョンの 4D では、標準アクションはプロパティリストを使用することでしか設定できませんでした。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*action* 引数には、オブジェクトと関連付けたい標準アクションを表すコードを文字列で渡します。"Text Values for Associated Standard Action" テーマ内にある、以下の定数のどれか一つを渡すことができます。:

定数	型	値
Object Refresh current URL action	テキスト	"39"
Object Show Clipboard action	テキスト	"23"
Object Add subrecord action	テキスト	"14"
Object Goto page action	テキスト	"15"
Object Cancel action	テキスト	"1"
Object Undo action	テキスト	"17"
Object Stop loading URL action	テキスト	"40"
Object Paste action	テキスト	"20"
Object Copy action	テキスト	"19"
Object Cut action	テキスト	"18"
Object MSC action	テキスト	"36"
Object Last record action	テキスト	"6"

定数	型	値
Object Last page action	テキスト	"11"
Object Clear action	テキスト	"21"
Object Previous record action	テキスト	"4"
Object Next record action	テキスト	"3"
Object Edit subrecord action	テキスト	"12"
Object Open back URL action	テキスト	"37"
Object Open next URL action	テキスト	"38"
Object Previous page action	テキスト	"9"
Object Next page action	テキスト	"8"
Object First record action	テキスト	"5"
Object First page action	テキスト	"10"
Object Database Settings action	テキスト	"32"
Object Quit action	テキスト	"27"
Object Redo action	テキスト	"31"
Object Return to Design mode action	テキスト	"35"
Object Automatic splitter action	テキスト	"16"
Object Delete record action	テキスト	"7"
Object Delete subrecord action	テキスト	"13"
Object Test Application action	テキスト	"26"
Object Select all action	テキスト	"22"
Object Accept action	テキスト	"2"
Object No standard action	テキスト	"0"

- ▶ ボタンの標準アクションを確定 (OK) に設定したい場合、以下のように記述します。:

**OBJECT SET ACTION**(\*;"bValidate";Object Accept action)

*See also:* [OBJECT Get action](#)

## OBJECT SET BORDER STYLE

**OBJECT SET BORDER STYLE** ({\*; }object ; borderStyle)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
borderStyle	倍長整数	→ 境界線スタイル

新しい **OBJECT SET BORDER STYLE** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトの境界線スタイルを変更します。

「境界線スタイル」のプロパティは、オブジェクトの外枠の見た目を管理します。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*borderStyle* 引数には、オブジェクトに適用したい境界線スタイルを指定する値を渡します。"Form objects (Properties)" テーマ内にある、以下の定数のどれかを使用することができます。:

定数 ( 値 )	説明
<u>Border None</u> (0)	オブジェクトは境界線を持ちません。
<u>Border Double</u> (5)	オブジェクトの境界線は二重線 (1 ピクセル離れた 2 本の 1pt の線) になります。
<u>Border Plain</u> (1)	オブジェクトの境界線は連続した一本の 1pt の線になります。
<u>Border Raised</u> (3)	オブジェクトの境界線は浮き上がったような 3D エフェクトになります。
<u>Border Sunken</u> (4)	オブジェクトの境界線は沈み込んだような 3D エフェクトになります。
<u>Border System</u> (6)	オブジェクトの境界線はシステムのグラフィック仕様に沿ったものになります。
<u>Border Dotted</u> (2)	オブジェクトの境界線は 1pt の点線になります。

*See also:* [OBJECT Get border style](#)

## OBJECT SET CONTEXT MENU

OBJECT SET CONTEXT MENU({\*; }object; contextMenu)

引数	型	説明
*	演算子	→ 指定時: <i>object</i> はオブジェクト名 (文字列) 省略時: <i>object</i> は変数またはフィールド
<i>object</i>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
<i>contextMenu</i>	ブール	→ True = コンテキストメニュー有効化 False = コンテキストメニュー無効化

新しい [OBJECT SET CONTEXT MENU](#) コマンドは、カレントのプロセスにおいて、*object* 引数と \* 演算子で指定されたオブジェクトの標準コンテキストメニューの関連付けを有効または無効にします。

コンテキストメニューは、テキスト型の入力エリア、Web エリアとピクチャーに存在するオプションです。標準的なアクションメニューをこれ

らのオブジェクトに対して型に応じて関連付けすることができます。たとえばテキストオブジェクトに対してコピー・ペーストが使用できるかどうかなどです。詳細に関しては、*Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*contextMenu* 引数には、コンテキストメニューを有効にしたい場合には True を、無効にしたい場合には False を渡します。

*See also:* [OBJECT Get context menu](#)

## OBJECT SET COORDINATES

OBJECT SET COORDINATES ({\*; }object ; left ; top{; right ; bottom})

引数	型	説明
*	演算子	→ 指定時 : <i>object</i> はオブジェクト名 (文字列) 省略時 : <i>object</i> は変数またはフィールド
<i>object</i>	倍長整数	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
<i>left</i>	倍長整数	→ オブジェクトの左端の絶対座標 (ピクセル)
<i>top</i>	倍長整数	→ オブジェクトの上端の絶対座標 (ピクセル)
<i>right</i>	倍長整数	→ オブジェクトの右端の絶対座標 (ピクセル)
<i>bottom</i>	倍長整数	→ オブジェクトの下端の絶対座標 (ピクセル)

*Note* このコマンドは、既存の OBJECT MOVE コマンドに第 2 引数の \* を渡した時と同じ動作をします。

新しい [OBJECT SET COORDINATES](#) コマンドは、カレントのプロセスにおいて、*object* 引数と \* 演算子で指定されたオブジェクトの位置 (またはそれに伴うサイズ) を変更します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*left* と *top* 引数には、フォーム内の、*object* 引数で指定したオブジェクトの、絶対座標を渡します。

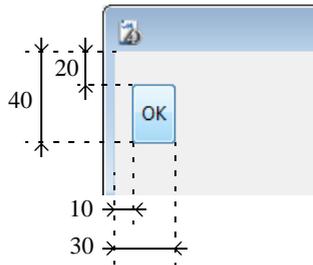
*right* と *bottom* 引数にも絶対座標を渡すことで、オブジェクトの右下隅の位置を指定することもできます。この隅の位置が *left* と *top* 引数で指定された隅に矛盾する場合、オブジェクトは自動的にリサイズされます。

*Note* オブジェクトのサイズを変更せずに位置だけを変更したい場合は、既存の OBJECT MOVE コマンドの使用が推奨されます。

このコマンドは以下の様な場合においてのみ機能します。:

- レコード編集の入力フォーム
  - DIALOG コマンドを使用して表示したフォーム
  - MODIFY SELECTION か DISPLAY SELECTION フォームを使用して表示された出力フォームのヘッダーとフッター
  - 印刷中のフォーム
- ▶ 以下の宣言は、"button\_1" のオブジェクトを、(10,20) (30,40) の座標に表示します。:

**OBJECT SET COORDINATES**(\*;"button\_1";10;20;30;40)



*See also:* OBJECT MOVE

## OBJECT SET DATA SOURCE

OBJECT SET DATA SOURCE ({\*; }object ; dataSource)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
dataSource	ポインター	→ オブジェクトの新しいデータソースへのポインター

新しい **OBJECT SET DATA SOURCE** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトのデータソースを変更します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

データソースとは、フォームを実行した際に、オブジェクトに表示される値を持っているフィールドまたは変数のことです。デザインモードでは、通常データソースはプロパティリスト内でソースとソースフィールド(フィールド)または変数名(変数)として定義されています。:

▼ Objects	
Type	Field
Object Name	Field5
▼ Data Source	
Source	Employees
Source Field	Last name

データソース (フィールド)

▼ オブジェクト	
タイプ	ボタン
オブジェクト名	Button
変数名	vB1
タイトル	Couleurs

データソース (変数)

*Note* 4D の v14 では、ポップアップ / ドロップダウンリストかコンボボックス型のオブジェクトにリスト項目の値またはリスト項目の参照番号を関連付けることができます。(72P “[ポップアップメニューやコンボボックスとフィールド・変数の関連付け](#)” を参照して下さい)

フォーム内のすべてのデータソースはこのコマンドで変更することができます。データソースを変更することによって、アプリケーションの一貫性が損なわれないように気を付けてください。

リストボックスの場合は、以下の点に注意して下さい。:

- データソースを変更する場合はリストボックスの型に注意する必要があります。たとえば、配列型のリストボックスの列のデータソースとして、フィールドを指定することはできない、ということです。
- セレクション型のリストボックスの場合、リストボックスオブジェクトそのもののデータソースを読み取ったり変更したりはできません。セレクション型のリストボックスの変数は、データソースではなくて内部参照として扱っているからです。
- このコマンドは主に配列型のリストボックスに対して使用されるものです。選択型のリストボックスに対しては、このコマンドの代わりに LISTBOX SET COLUMN FORMULA コマンドを使用することができます。

変更不可能なデータソースに対してこのコマンドが使用された場合、何も起こりません。

- ▶ 入力エリアに対してデータソースを変更する場合、:

```
C_POINTER($ptrField)
$ptrField:=Field(3;2)
OBJECT SET DATA SOURCE (*;"Input";$ptrField)
```

*See also:* [OBJECT Get data source](#)

## OBJECT SET EVENTS

OBJECT SET EVENTS({\*; }object; arrEvents ; mode)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名または "" (* 指定時) または変数やフィールド (* 省略時)
arrEvents	倍長整数配列	→ 設定したいイベントの配列
mode	倍長整数	→ arrEvents 引数で定義されたイベントの起動モード

新しい **OBJECT SET EVENTS** コマンドは、カレントのプロセスにおいて、*object* 引数と \* 演算子で指定されたフォームまたはオブジェクトのフォームイベントの設定を変更します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。フォーム自身のイベントの設定を定義する場合は、任意の \* 演算子を渡したうえで *object* 引数に空の文字列 "" を渡します。こうすることでカレントフォームを指定します。

*Note* もしテーブルに関連したサブフォームのイベントを変更したい場合にはオブジェクト名を指定する記述法のみ有効です。

*arrEvents* 引数には、変更したい定義済みもしくはカスタムのフォームイベントのリストを倍長整数配列で渡します。(このとき *mode* 引数を使って、指定したイベントを有効にするか無効にするかを指定できます。) 定義済みのイベントを指定するには、*arrEvents* 配列の要素の中に以下の定数を渡します。これらの定数は "Form Events" のテーマ内にあります。:

定数	型	値
On Delete Action	倍長整数	58
On Activate	倍長整数	11
On Display Detail	倍長整数	8

On Outside Call	倍長整数	10
On Plug in Area	倍長整数	19
On After Keystroke	倍長整数	28
On After Edit	倍長整数	45
On After Sort	倍長整数	30
On Before Keystroke	倍長整数	17
On Before Data Entry	倍長整数	41
On Mac Toolbar Button	倍長整数	55
On Close Box	倍長整数	22
On Page Change	倍長整数	56
On Load Record	倍長整数	40
On URL Resource Loading	倍長整数	48
On Clicked	倍長整数	4
On Header Click	倍長整数	42
On Arrow Click	倍長整数	38
On Long Click	倍長整数	39
On Footer Click	倍長整数	57
On Collapse	倍長整数	44
On Begin URL Loading	倍長整数	47
On Begin Drag Over	倍長整数	46
On Mouse Enter	倍長整数	35
On Picture Scroll	倍長整数	59
On Column Moved	倍長整数	32
On Row Moved	倍長整数	34
On Expand	倍長整数	43
On Drop	倍長整数	16
On Deactivate	倍長整数	12
On Data Change	倍長整数	20
On Double Clicked	倍長整数	13
On Header	倍長整数	5
On URL Loading Error	倍長整数	50
On Close Detail	倍長整数	26
On URL Filtering	倍長整数	51
On End URL Loading	倍長整数	49
On Mouse Leave	倍長整数	36
On Getting Focus	倍長整数	15
On Drag Over	倍長整数	21
On Printing Detail	倍長整数	23
On Printing Footer	倍長整数	7
On Printing Break	倍長整数	6
On Unload	倍長整数	24
On Menu Selected	倍長整数	18
On Timer	倍長整数	27
On Bound Variable Change	倍長整数	54
On Selection Change	倍長整数	31

On Open Detail	倍長整数	25
On Open External Link	倍長整数	52
On Losing Focus	倍長整数	14
On Resize	倍長整数	29
On Column Resize	倍長整数	33
On Window Opening Denied	倍長整数	53
On Mouse Move	倍長整数	37
On Validate	倍長整数	3

このリストでは On Load (1) イベントが含まれていないことに注意して下さい。コマンドを実行した際には既にイベントが発生されてしまっているため、ここでは定義されていません。

*arrEvents* 引数にはカスタムイベントに対応する値を渡すこともできます。この際には、負の値を使用することが推奨されます。(CALL SUBFORM CONTAINER コマンドのドキュメントを参照して下さい。)

*mode* 引数には、配列の要素に対する全体的な処理の適用方法を定める値を渡します。"Form objects (Properties)" テーマ内にある、以下の定数のどれかを渡して指定して下さい。:

定数 ( 値 )	説明
<u>Enable events disable others</u> (0)	<i>arrEvents</i> に指定された全てのイベントは有効になり、他は全て無効化されます。
<u>Enable events others unchanged</u> (1)	<i>arrEvents</i> に指定された全てのイベントは有効になり、他は何も変更されません。
<u>Disable events others unchanged</u> (2)	<i>arrEvents</i> に指定された全てのイベントは無効化され、他は何も変更されません。

**OBJECT SET EVENTS** コマンドを使用する際、*object* 引数で指定したオブジェクトでサポートされていない型のイベントの設定をしようとした場合、このイベントは無視されます。

オブジェクトが **OBJECT SET EVENTS** コマンドを呼び出した後に複製された場合は、有効 / 無効の設定も複製先に引き継がれます。

- ▶ いくつかのリストボックスオブジェクトに対して 3 つのフォームイベントを有効にし、他を全て無効にする場合、:

```
ARRAY LONGINT($MyEventsOnLB;3)
```

```
$MyEventsOnLB {1} := On After Sort
```

```
$MyEventsOnLB {2} := On Column Moved
```

```
$MyEventsOnLB {3} := On Column Resize
```

```
OBJECT SET EVENTS(*,"MyLB@"; $MyEventsOnLB;
```

```
  Enable events disable others) // 指定した 3 つのイベントを有効にし、他は全て無効
```

- ▶ いくつかのリストボックスオブジェクトに対して3つのフォームイベントを無効にし、他は何も変更をしない場合、:

```

ARRAY LONGINT($MyEventsOnLB;3)
$MyEventsOnLB {1} := On After Sort
$MyEventsOnLB {2} := On Column Moved
$MyEventsOnLB {3} := On Column Resize
OBJECT SET EVENTS(*;"MyLB@"; $MyEventsOnLB;
  Disable events others unchanged) // 指定した3つのイベントのみ無効

```

- ▶ あるオブジェクトのフォームイベントを有効にし、他は何も変更をしない場合、:

```

ARRAY LONGINT($MyEventsOnLB;1)
$MyEventsOnLB {1} := On Column Moved
OBJECT SET EVENTS(*;"Col1"; $MyEventsOnLB;Enable events others un-
changed)
  // 指定したイベントのみ有効

```

- ▶ フォームのイベントを全て無効にする場合、:

```

ARRAY LONGINT($MyFormEvents;0)
OBJECT SET EVENTS(*;""; $MyFormEvents;Enable events disable others)
  // 全てのイベントが無効

```

- ▶ フォームのイベントを一つだけ無効にし、他は何も変更しない場合、:

```

ARRAY LONGINT($MyFormEvents;1)
$MyFormEvents{1}:=On Timer
OBJECT SET EVENTS(*;""; $MyFormEvents;Disable events others un-
changed)
  // 指定した一つのイベントのみ無効

```

*See also:* [OBJECT GET EVENTS](#)

## OBJECT SET INDICATOR TYPE

**OBJECT SET INDICATOR TYPE**({\*; }object ; indicator)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォーム オブジェクト	→ オブジェクト名(*指定時)、 または変数やフィールド(*省略時)
indicator	倍長整数	→ インジケータータイプ

新しい [OBJECT SET INDICATOR TYPE](#) コマンドは、カレントプロセスにおいて *object* 引数と \* 演算子で指定されたサーモメーターオブジェクトのインジケータータイプを変更します。

インジケータタイプは、サーモメータの表示方法を定義します。詳細な情報に関しては、*Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*indicator* 引数には、表示したいインジケータの種類を渡します。"Form objects (Properties)" のテーマ内にある、以下の定数の中から選ぶことができます。:

定数 ( 値 )	説明
<u>Barber shop</u> (2)	連続したアニメーションを表示するバー
<u>Progress bar</u> (1)	標準の進捗バー
<u>Asynchronous progress bar</u> (3)	連続したアニメーションを表示する、回転型のインジケータ

*See also:* [変更されたコマンド](#)

## OBJECT SET LIST BY REFERENCE

OBJECT SET LIST BY REFERENCE({\*; }object{; listType}; list)

引数	型	説明
*	演算子	→ 指定時: <i>object</i> はオブジェクト名 ( 文字列 ) 省略時: <i>object</i> は変数またはフィールド
<i>object</i>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
<i>listType</i>	倍長整数	→ リストの種類: 選択リスト、指定リスト、 除外リスト
<i>list</i>	ListRef	→ リストの参照番号

新しい **OBJECT SET LIST BY REFERENCE** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトに関連付けられたリストを、*list* 引数で指定した階層リストに設定または置換します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*listType* 引数を省略した場合、コマンドは自動的にソースとなる選択リストを定義します。*listType* 引数では設定したいリストのタイプを指定することができます。"Form objects (Properties)" テーマ内にある以下の定数のどれかを渡して下さい。:

定数 ( 値 )	説明
<u>Choice list</u> (0)	リストには選択できる値が表示されます。(プロパティリストの選択リストを選択した場合と同じです。定数省略時にはこれが適用されます)

<u>Required list</u> (1)	入力可能な値のみ指定します。(プロパティリストで指定リストを選択した場合と同じです。)
<u>Excluded list</u> (2)	入力不可な値のみ指定します。(プロパティリストで除外リストを選択した場合と同じです。)

*list* 引数には、オブジェクトに関連付けたい階層リストの参照番号を渡します。このリストは、New list コマンド、Copy list コマンド、または Load list コマンドを使用してすでに生成されている必要があります。

オブジェクトとリストの関連付けを解除するためには、解除したいリストの種類を *listType* 引数で指定し、*list* 引数に 0 を渡します。

*Note* リストの関連付けを解除しても、リスト参照はメモリーからは消去されません。リストが不要になった時には CLEAR LIST コマンドを使用して下さい。

このコマンドは、変数またはフィールドに関連付けられたポップアップまたはコンボボックスに対して使用すると興味深い挙動をします (72P “[ポップアップメニューやコンボボックスとフィールド・変数の関連付け](#)” を参照して下さい)。この場合、関連付けは動的に行われ、リストの変更は全てフォームへと反映されます。オブジェクトが配列に関連付けられたとき、リストは配列へとコピーされ、リストへのいかなる変更も自動的に不可になります (最後の例を参照して下さい)。

- ▶ 単純な選択リスト (デフォルトのリスト型) をテキストフィールドへと関連付ける場合を考えます。

**vCountriesList:=New list**

**APPEND TO LIST**( vCountriesList ; "Spain" ; 1)

**APPEND TO LIST**( vCountriesList ; "Portugal" ; 2)

**APPEND TO LIST**( vCountriesList ; "Greece" ; 3)

**OBJECT SET LIST BY REFERENCE** ([Contact]Country;vCountriesList)

- ▶ "vColor" というリストを単純な選択リストとして "DoorColor" ポップアップリストに関連付けます。:

**vColor:=New list**

**APPEND TO LIST** (vColor ; "Blue" ; 1)

**APPEND TO LIST** (vColor ; "Green" ; 2)

**APPEND TO LIST** (vColor ; "Red" ; 3)

**APPEND TO LIST** (vColor ; "Yellow" ; 4)

**OBJECT SET LIST BY REFERENCE**(\*;"DoorColor";Choice list;vColor)

- ▶ 更に、"vColor" のリストを、"WallColor" というコンボボックスに関連付けする場合を考えます。このコンボボックスは入力可能なので、"black"

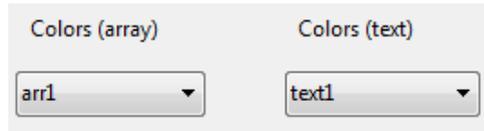
"purple" と言った色が入力されないようにしたい場合、これらの色を "vReject" というリストに入れて除外します。:

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;vColor)
vReject:=New list
APPEND TO LIST( vReject ; "Black" ; 1)
APPEND TO LIST( vReject ; "Gray" ; 2)
APPEND TO LIST( vReject ; "Purple" ; 3)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;vReject)
```

- ▶ これらのリストの関連付けを解除する場合、:

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Required list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;0)
```

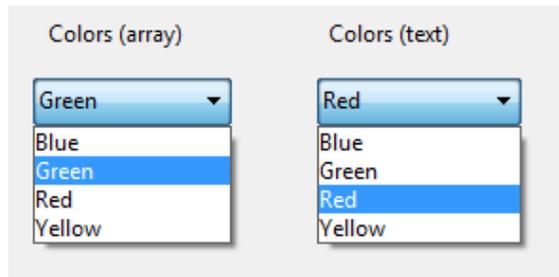
- ▶ この例では、このコマンドがテキスト配列に関連付けられたポップアップメニューに適用されたときと、テキスト変数に関連付けられたポップアップメニューに適用されたときの違いについて説明します (v14 の新機能)。フォーム内に二つのポップアップメニューがあるとします。:



これらのポップアップメニューの中身は <>vColor list を使用して設定されています (中身はカラーの値です)。以下のコードがフォームをロードした際に実行されるとします。

```
ARRAY TEXT (arr1;0) //arr1 pop up
C_TEXT (text1) //text1 pop up
OBJECT SET LIST BY REFERENCE (*;"arr1";<>vColor)
OBJECT SET LIST BY REFERENCE (*;"text1";<>vColor)
```

実行すると、両メニューとも同じ値を表示します:

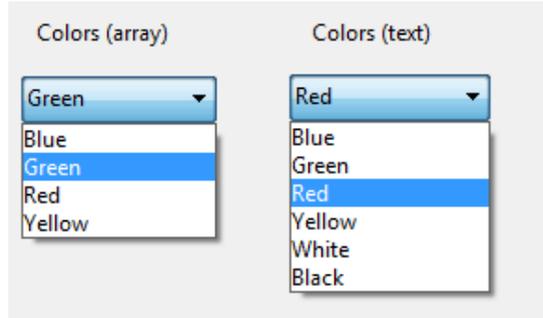


次に、ボタンなどを使用して以下のコードを実行します。

APPEND TO LIST (<>vColor;"White";5)

APPEND TO LIST (<>vColor;"Black";6)

テキストフィールドと関連付けられていたメニューのみが (動的な参照によって) 更新されます。:



配列によって管理されたポップアップに関連付けられたリストを更新するためには、OBJECT SET LIST BY REFERENCE コマンドを再度使用してリストの中身をコピーする必要があります。

See also: [OBJECT Get list reference](#)

## OBJECT SET PLACEHOLDER

OBJECT SET PLACEHOLDER({\*; }object ; placeholderText)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
placeholder-Text	テキスト	→ オブジェクトに関連付けるプレースホルダーテキスト

新しい [OBJECT SET PLACEHOLDER](#) コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトにプレースホルダーテキストを関連付けます。詳細な情報に関しては、53P “[プレースホルダーテキスト](#)” を参照して下さい。

プロパティリストを通じてプレースホルダーが既に関連付けられていた場合、そのテキストはカレントプロセスにおいて *placeholderText* 引数で指定した値で置換されます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*placeholderText* 引数には、オブジェクトが空欄の時に表示させたいヘルプテキストか指示を渡します。

*Note* **OBJECT SET PLACEHOLDER** コマンドでは、プレースホルダーに *xliff* 参照を挿入することはできません。挿入する場合にはプロパティリストを通じてプレースホルダーを定義して下さい。

このコマンドは、変数、フィールド、コンボボックスのフォームオブジェクトにしか使用できません。プレースホルダーには文字列かテキストの値を関連付けることが出来ます。また、「ヌルのときブランクにする」プロパティがあるフォームオブジェクトに関しては、日付か時刻のデータと関連付けることができます。

- ▶ コンボボックスにプレースホルダーとして「Search」というテキストを表示したい場合、：

**OBJECT SET PLACEHOLDER** (\*;"search\_combo";"Search")



*See also:* [OBJECT Get placeholder](#)

## OBJECT SET PRINT VARIABLE FRAME

**OBJECT SET PRINT VARIABLE FRAME**({\*; }object ; variableFrame{; fixedSubform})

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
variableFrame	ブール	→ True = 可変長フレーム印刷 False = 固定長フレーム印刷
fixedSubform	倍長整数	→ サブフォームを固定サイズで印刷する際のオプション

新しい **OBJECT SET PRINT VARIABLE FRAME** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトの印刷時可変のプロパティを変更します。

このプロパティはテキスト型・ピクチャー型の変数とフィールド、そしてサブフォームにのみ存在します。サブフォームの場合は更に固定長印刷のオプションも存在します。詳細な情報に関しては、*Design Reference*

のマニュアルを参照して下さい。このコマンドは、印刷時可変のプロパティがないオブジェクトに対しては何もしません。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*variableFrame* 引数はブール型の引数です。True を渡した場合、オブジェクトは可変長フレームで印刷されます。False を渡した場合、オブジェクトは固定長フレームで印刷されます。

任意の *fixedSubform* 引数では、*object* でサブフォームを指定していて *variableFrame* 引数に False を渡した場合のみ、追加のオプションを設定できます (他の場合には無視されます)。このオプションではサブフォームの固定長フレーム印刷モードを定義できます。"Form objects (Properties)" テーマにある、以下の定数のどちらかを渡して下さい。:

定数 ( 値 )	説明
<u>Print Frame fixed with truncation</u> (1)	サブフォームのエリア内に収まるレコードのみを印刷します。フォームは一度しか印刷されず、印刷されないレコードは無視されます。
<u>Print Frame fixed with multiple records</u> (2)	フレームのサイズが変わりませんが、全てのレコードが印刷されるまでフォームは複数回印刷されます。

*See also:* [OBJECT GET PRINT VARIABLE FRAME](#)

## OBJECT SET MAXIMUM VALUE

OBJECT SET MAXIMUM VALUE({\*; }object ; max Value)

引数	型	説明
*	演算子	→ 指定時 : <i>object</i> はオブジェクト名 (文字列) 省略時 : <i>object</i> は変数またはフィールド
<i>object</i>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
<i>max Value</i>	日付   時間   整数   倍長整数   64 ビット整数   実数   フロート	→ オブジェクトの最大値

新しい [OBJECT SET MAXIMUM VALUE](#) コマンドはカレントプロセスにおいて *object* 引数と \* 演算子で指定されたオブジェクトの最大値を変更します。

「最大値」プロパティは、数、日付、時間のデータ種類に対して適用することができます。詳細な情報に関しては、*Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*maxValue* 引数には、カレントプロセスにおいてオブジェクトに対して適用したい最大値を渡します。このとき、渡す最大値はオブジェクトの型と対応している必要があります。そうでない場合には、エラー 18"Field types are incompatible" が返されます。

*See also:* [OBJECT SET MINIMUM VALUE](#), [OBJECT GET MINIMUM VALUE](#)

## OBJECT SET MINIMUM VALUE

OBJECT SET MINIMUM VALUE({\*; }object ; minValue)

引数	型	説明
*	演算子	→ 指定時 : <i>object</i> はオブジェクト名 (文字列) 省略時 : <i>object</i> は変数またはフィールド
<i>object</i>	フォームオブジェクト	→ オブジェクト名 (* 指定時)、または変数やフィールド (* 省略時)
<i>minValue</i>	日付 時間  整数  倍 長整数  64 bit 整数   実数   フロート	→ オブジェクトの最小値

新しい [OBJECT SET MINIMUM VALUE](#) コマンドはカレントプロセスにおいて *object* 引数と \* 演算子で指定されたオブジェクトの最小値を変更します。

「最小値」プロパティは、数、日付、時間のデータ種類に対して適用することができます。詳細な情報に関しては、*Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*minValue* 引数には、カレントプロセスにおいてオブジェクトに対して適用したい最小値を渡します。このとき、渡す最小値はオブジェクトの型と対応している必要があります。そうでない場合には、エラー 18"Field types are incompatible" が返されます。

[OBJECT GET MINIMUM VALUE](#)

**OBJECT SET  
MULTILINE**

OBJECT SET MULTILINE({\*; }object ; multiline)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
multiline	倍長整数	→ 複数行プロパティの状態

新しい **OBJECT SET MULTILINE** コマンドは、*object* 引数と \* 演算子で指定されたオブジェクトの「複数行」のプロパティを変更します。

複数行のプロパティはテキストエリアの表示と印刷に関係する二つの状態を管理します。単独行の最後に置かれる単語の表示と、改行の自動挿入です。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。このプロパティが存在しないオブジェクトに対してコマンドを適用した場合、何も起こりません。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*multiline* 引数には、設定を決める値を渡します。値としては "Form objects (Properties)" テーマ内にある、以下の定数を使用することができます。:

定数 ( 値 )	説明
<u>Multiline Auto</u> (0)	単独行のエリアでは、行に表示しきれない単語は切り落とされ、改行はされません。 複数行のエリアでは、改行が行われます。
<u>Multiline No</u> (2)	改行は禁止されます。テキストは必ず単独行として表示されます。文字列かテキストフィールドか変数に改行が含まれていたとしても、改行は行われません。
<u>Multiline Yes</u> (1)	単独行のエリアでは、テキストは最初の改行までか、単語全体を表示できる最後の単語までが表示されます。その後改行が挿入されるので、キーを押すことによってエリアの内容をスクロールすることができます。 複数行のエリアでは、自動で改行が行われます。

- ▶ 入力エリアにおいて、改行を禁止したい場合、:

**OBJECT SET MULTILINE**(\*;"vEntry";Multiline No)

*See also:* [OBJECT Get multiline](#)

## OBJECT SET STYLE SHEET

OBJECT SET STYLE SHEET({\*; }object; styleSheetName)

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
styleSheet-Name	テキスト	→ スタイルシート名

新しい **OBJECT SET STYLE SHEET** コマンドは、カレントのプロセスにおいて、*object* 引数と \* 演算子で指定されたオブジェクトに関連付けられたスタイルシートを変更します。スタイルシートを変更すると、フォント、フォントサイズ、そしてフォントスタイルが変更されます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*styleSheetName* 引数の中には、*object* 引数で指定したオブジェクトに適用するスタイルシートの名前等を渡します。渡せる値は以下の通りです。:

- 既存のスタイルシート名 (指定したスタイルシートが存在しない場合は、ON ERR CALL コマンドで割り込み可能なエラーが返されます)
- "Font Styles" テーマ内にある、[Automatic style sheet](#) 定数 (指定すると、自動スタイルシートが適用されます)
- 空の文字列 ("") (指定すると、適用していたスタイルシートが解除されます。)

*object* 引数で指定したオブジェクトにデザインモードですでにスタイルシートが関連付けられている場合、このコマンドを呼び出すことによって、カレントプロセス中はスタイルシートが変更されます。

セッション中、XXXST SET TEXT コマンド、ST SET ATTRIBUTES コマンド、OBJECT SET FONT コマンド等をオブジェクトに使用してフォントやフォントサイズを変更した場合、スタイルシートへの参照はオブジェクトから自動的に削除されます。元からあったスタイルシートと同じ設定を適用しようとした場合でも同様に削除されます。

しかしながら、例えば ST SET ATTRIBUTES XXX コマンドや OBJECT SET FONT STYLE コマンド等を使用してスタイル(太字、イタリック等)のみを変更した場合、スタイルシートへの参照は削除されず、これらのプロパティはセッションの間スタイルシートへと追加されます。

**See also:** [GET STYLE SHEET INFO](#), [LIST OF STYLE SHEETS](#), [OBJECT Get style sheet](#)

**OBJECT SET TEXT ORIENTATION**

OBJECT SET TEXT ORIENTATION ({\*; }object ; orientation)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
orientation	倍長整数	→ オブジェクトの方向を決める値

新しい **OBJECT SET TEXT ORIENTATION** コマンドは、カレントプロセスにおいて *object* 引数と \* 演算子で指定されたオブジェクトの方向を変更します。

フォームエディター内にある方向のプロパティは、テキストエリアを回転させることができます。このプロパティとは異なり、**OBJECT SET TEXT ORIENTATION** コマンドはオブジェクトの中身を回転させ、オブジェクトそのものを回転させるわけではありません。詳細な情報に関しては、76P “**テキストの回転**” を参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。スタティックなテキストと、入力不可能な変数とフィールドのみ回転させることができます。このプロパティをサポートしないオブジェクトに対してコマンドを使用した場合、コマンドは何もしません。詳細な情報に関しては、76P “**回転可能なオブジェクト**” を参照して下さい。

*orientation* 引数には、*object* で指定したオブジェクトに適用したい方向の絶対値を渡します。“Form objects (Properties)” テーマ内にある、以下の定数のどれかのみ使用可能です。:

定数 ( 値 )	説明
<u>Orientation 0°</u> (0)	回転しません ( デフォルトの動作 )
<u>Orientation 90° right</u> (90)	テキストを時計回りに 90° 回転させます。
<u>Orientation 180°</u> (180)	テキストを時計回りに 180° 回転させます。
<u>Orientation 90° left</u> (270)	テキストを反時計回りに 90° 回転させます。

*Note* 上記の値に対応する方向のみサポートされています。これ以外の値を渡した場合、その値は無視されます。

- ▶ フォームの内の変数に、270° の回転をさせたい場合、:

**OBJECT SET ENTERABLE**(\*;"myVar";False)

// 変数が入力可能である場合は必須のコマンド

**OBJECT SET TEXT ORIENTATION** (\*;"myVar";Orientation 90° left)

*See also:* [OBJECT Get text orientation](#)

## OBJECT SET THREE STATES CHECKBOX

OBJECT SET THREE STATES CHECKBOX({\*; }object; threeStates)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
threeStates	ブール	→ True = スリーステートチェックボックス False = 標準のチェックボックス

新しい [OBJECT SET THREE STATES CHECKBOX](#) コマンドは、カレントのプロセスにおいて、*object* 引数と \* 演算子で指定されたチェックボックスのスリーステートプロパティの状態を変更します。

スリーステートオプションは、チェックボックスにて「どちらでもない」という新しい状態の使用を可能にします。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。このコマンドは変数と関連付けられたチェックボックスに対してのみ適用され、チェックボックスとして現れるブール型のフィールドには使用できません。

*threeStates* 引数には、スリーステートモードをオンにしたい場合には True を、オフにしたい場合は False を渡します。

*See also:* [OBJECT Get three states checkbox](#)

## 変更されたコマンド

### OBJECT Get choice list name

このコマンドは 4D の v14 では、[OBJECT Get list name](#) という名前になり、機能が拡張されました。そちらを参照して下さい。

### OBJECT Get list name

OBJECT Get list name( {\*; } object{; listType} ) → Text

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド



## OBJECT SET CHOICE LIST NAME

このコマンドは 4D の v14 では、**OBJECT SET LIST BY NAME** という名前になり、機能が拡張されました。

## OBJECT SET FONT

OBJECT SET FONT ( {\*; }object ; font)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
font	文字列	→ フォントの <i>名前</i>

**OBJECT SET FONT** コマンドは 4D v14 より、*font* 引数ではフォント名 (文字列) のみを受け取るように変わりました。フォント番号 (倍長整数) の使用はサポートされていません。

## OBJECT SET LIST BY NAME

OBJECT SET LIST BY NAME({\*; }object{; listType}; list)

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
listType	倍長整数	→ リストの種類 : 選択リスト、指定リスト、 除外リスト
list	文字列	→ デザインモードで定義された選択リストの 名前を指定、もしくはリストを解除する ために "" を指定

*Note* 以前の 4D では、このコマンドは **OBJECT SET CHOICE LIST NAME** という名前でした。

**OBJECT SET LIST BY NAME** コマンドを使用すると、*object* 引数と \* 演算子で指定されたオブジェクトと関連付けされた全てのタイプのリスト (選択リスト、指定リスト、除外リスト) を設定または変更できるようになりました。

*listType* 引数には、"Form objects (Properties)" テーマ内にある以下の定数を渡し、リストの種類を指定します。:

定数 (値)	説明

<u>Choice list</u> (0)	選択できる値のリスト。(プロパティリスト内の「選択リスト」)(デフォルト)
<u>Required list</u> (1)	入力できる値のリスト。(プロパティリスト内の「指定リスト」)
<u>Excluded list</u> (2)	入力できない値のリスト。(プロパティリスト内の「除外リスト」)

この引数を省略した場合、自動的に 0 (Choice list) を渡したものとみなされます。

このコマンドでは更に、オブジェクトと関連付けられたリストの、関連づけを解除できるようになりました。解除するためには *list* 引数に空の文字列 ("") を渡して下さい。

- ▶ "color\_choice" というリストを、"DoorColor" という単純なポップアップ / ドロップダウンリストと関連付ける場合を考えます。:

```
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list;"color_choice")
// この場合、第 3 引数 (定数) は省略することが出来ます。
```

- ▶ "color\_choice" というリストを "WallColor" というコンボボックスと関連付けたい場合を考えます。コンボボックスは入力可能なので、"black" や "purple" と書いた色が入力されないようにしたい場合、これらの色を "excl\_colors" というリストに入れて以下の様に除外します。:

```
OBJECT SET LIST BY NAME(*;"WallColor";Choice list;"color_choice")
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list;"excl_colors")
```

- ▶ これらのリストの関連付けを解除したい場合、:

```
// 選択リストの解除
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list; "")
// 入力が許可されていない値 (除外リスト) の解除
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list; "")
```

*See also:* [OBJECT Get list name](#)

## OBJECT SET RGB COLORS

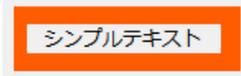
```
OBJECT SET RGB COLORS({*; }object ; foregroundColor ; backgroundColor
; altBackgrndColor)
```

"SET RGB COLORS" テーマ内に、透明の背景色を設定する新しい定数が追加されました。:

定数	型	値
<u>Background color none</u>	倍長整数	-16

この定数は `backgroundColor` 引数と `altBackgrndColor` 引数にしか渡すことができません。

- ▶ 背景色を透明に、フォントカラーを明るい色に設定する場合があります。  
:



**OBJECT SET RGB COLORS**(\*;"myVar";Light shadow color;Background color none)



*See also:* [OBJECT GET RGB COLORS](#)

## 汎用コマンドとマルチスタイルエリアの関係性

4D v14 では、OBJECT SET RGB COLORS や OBJECT SET FONT STYLE といった汎用コマンドとマルチスタイルエリアとの関係性が新しくなりました。以前のバージョンの 4D では、これらのコマンドのどれかを実行すると、エリア内に挿入されたカスタムのスタイルタグを全て変更してしまいました。

4D v14 からは、デフォルトのプロパティ（とデフォルトのタグで保存されたプロパティ）のみが変更されるようになりました。カスタムのスタイルタグはそのままの状態を維持します。

例えば、以下の様なマルチスタイルエリアにデフォルトのタグが保存されていた場合を考えます。:

This is the word red

このエリアのプレーンテキストは以下のようになります。:

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">This is the word <span style="color:#D81E05">red</span></span>
```

以下のコードを実行した場合、

**OBJECT SET COLOR**(\*;"myArea";-(Blue+(256\*Yellow)))

4D v14 では、赤文字の部分はそのまま赤文字として残ります。

**4D v14**

This is the word red

**v14 以前のバージョンの 4D**

This is the word red

**4D v14**

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#0000FF">This is the word
<span style="color:#D81E05">red</span></span>
```

**v14 以前のバージョンの 4D**

```
<span style="font-family:'Segoe UI';font-size:9pt;text-align:left;font-weight:normal;font-style:normal;text-decoration:none;color:#0000FF;"><span style="background-color:#FFFFFF">This is the word red</span></span>
```

これが適用される汎用コマンドは、以下の 5 つです。

- OBJECT SET RGB COLORS
- OBJECT SET COLOR
- OBJECT SET FONT
- OBJECT SET FONT STYLE
- OBJECT SET FONT SIZE

マルチスタイルエリアにおいては、汎用コマンドはデフォルトのスタイルを設定するためだけに使用されるべきです。データベースの実行中にスタイルを管理するためには、「[スタイル付テキスト](#)」テーマ内のコマンドを使用することが推奨されます。

## オブジェクト (ランゲージ)

この新しいテーマにはオブジェクトフォーム内のデータを作成、操作するためのコマンドが含まれています。この新機能によって、4D の v14 では、構造化されたデータをサポートするどんなアプリケーションともデータのやりとりができるようになります。

4D オブジェクトは新しい **C\_OBJECT** ("コンパイラー" テーマ) と **ARRAY OBJECT** ("配列" テーマ) を使用して作成・初期化することができます。

### 4D オブジェクトの構造

4D のネイティブなオブジェクトの構造は、よくある「値 / 名前」というペア (連想配列) に基づいています。これらのオブジェクトの記法は JSON のそれをもとにしていますが、完全に同じというわけではありません。

---

*Note* JSON オブジェクトを扱うためには新しい "**JSON**" テーマ内にあるコマンドを使用して下さい。

---

プロパティ名は必ずテキストで表現されます。(例: "Name")

プロパティ値は以下のどれかの型で表現されます。:

- 番号 (実数、整数、等)

- テキスト
- 配列 (テキスト、実数、ブール、オブジェクト、ポインター)
- null
- ブール
- ポインター ([JSON Stringify](#) コマンドを使用して評価されたか、コピーされたときにポインターとして保存されます。)
- 日付 ("\"YYYY-MM-DDTHH:mm:ssZ\"") というフォーマットで表現されます。)
- オブジェクト (オブジェクトを代入する階層の数に制限はありません)。

## OB Copy

OB Copy(object{; resolvePtrs}) → Value

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
resolvePtrs	ブール	→ True = ポインターを解決 False または省略時 = ポインターを解決しない

戻り値      ランゲージオブジェクト   ← 複製されたオブジェクト

**OB Copy** コマンドは *object* 引数で指定したオブジェクトと同一のプロパティ、オブジェクト内オブジェクト、値を内包した、完全なコピーのオブジェクトを返します。

*object* 引数で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して定義されている必要があります。

指定したオブジェクトがポインター型の値を格納している場合、複製先にもポインターが格納されます。もしくは、*resolvePtrs* 引数に **True** を渡すことで、複製時に値を解決させることもできます。この場合、オブジェクト内で値を指定しているポインターは解決され、解決済みの値が使用されます。

- ▶ 単純な値を格納しているオブジェクトを複製する場合があります。:

```
C_OBJECT($Object)
C_TEXT($JsonString)
```

```
ARRAY OBJECT($arraySel;0)
ALL RECORDS([Product])
While (Not(End selection([Product])))
  OB SET($Object;"id";[Product]ID_Product)
  OB SET($Object;"Product Name";[Product]Product_Name)
```

```

OB SET($Object;"Price";[Product]Price)
OB SET($Object;"Tax rate";[Product]Tax_rate)
$ref_value:=OB Copy($Object) // 直接複製
APPEND TO ARRAY($arraySel;$ref_value)
    // $ref_value の中身は $Object の中身と完全に同じ
NEXT RECORD([Product])

```

**End while**

```

    // $ref_value の中身

```

```

$JsonObject:=JSON Stringify array($arraySel)

```

- ▶ ポインターを格納しているオブジェクトを複製する場合、:

```

C_OBJECT($ref)

```

```

OB SET($ref;"name";->[Company]name) // ポインターを含むオブジェクト

```

```

OB SET($ref;"country";->[Company]country)

```

```

ARRAY OBJECT($sel;0)

```

```

ARRAY OBJECT($sel2;0)

```

```

ALL RECORDS([Company])

```

```

While(Not(End selection([Company])))

```

```

    $ref_comp:=OB Copy($ref) // ポインターを解決せずに複製

```

```

        // $ref_comp={ "name":->[Company]name, "country":->[Company]Country" }

```

```

    $ref_comp2:=OB Copy($ref;True) // 解決済みのポインターを含んだコピー

```

```

        // $ref_comp2={ "name":"4D SAS", "country":"France" }

```

```

    APPEND TO ARRAY($sel;$ref_comp)

```

```

    APPEND TO ARRAY($sel2;$ref_comp2)

```

```

    NEXT RECORD([Company])

```

**End while**

```

$JsonObject:=JSON Stringify array($sel)

```

```

$JsonObject2:=JSON Stringify array($sel2)

```

```

    // $Object = [{ "name":"","country":"" }, { "name":"","country":"" }, ... ]

```

```

    // $Object2 = [{ "name":"4D SAS", "country":"France" }, { "name":"4D,
Inc", "country":"USA" }, { "name":"Catalan", "country":"France" } ... ]

```

## OB Is defined

**OB Is defined**(object{; property}) → Boolean

**引数**

**型**

**説明**

object

ランゲージオブジェクト

→

構造化されたオブジェクト

property	テキスト	→ 指定時にはプロパティを チェック、省略時にはオブ ジェクトをチェック
戻り値	ブール	← <i>property</i> 省略時 : <i>object</i> が定義 済みの場合は True を、それ以 外は False を返す <i>property</i> 指定時 : <i>property</i> が定 義済みの場合は True を、それ 以外は False を返す

**OB Is defined** コマンドは、*object* 引数で指定されたオブジェクト、または *property* 引数で指定されたプロパティが定義済みであれば **True** を、そうでない場合には **False** を返します。

*object* 引数で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されたものでなければなりません。

*property* 引数を省略した場合、コマンドはオブジェクトが定義済みかどうかを調べます。オブジェクトの中身が初期化されていれば定義済みであるとみなされます。

---

*Note* 定義済みのオブジェクトでも空である場合もあります。オブジェクトが空もしくは未定義であるかどうかを調べる場合には、**OB Is empty** コマンドを使用して下さい。

---

*property* 引数を渡した場合、コマンドは指定されたプロパティがオブジェクト内に存在するかどうかをチェックします。 *property* 引数では、大文字と小文字は区別されることに注意して下さい。

- ▶ オブジェクトの初期化を調べる記法 :

```
C_OBJECT($object)
$def:=OB Is defined($object)//$object は未定義なので $def=false になります。
```

```
OB SET($object;"Name";"Martin")
OB REMOVE($object;"Name")
$def2:=OB Is defined($object)//$object は空 ({} ) ではあるものの定義済みなので $def2=true になります。
```

- ▶ プロパティが存在しているかどうかを調べます。 :

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
//...
If(OB Is defined($ref;"age"))
```

//...

**End if**

このテストは以下のコマンドと同等です。:

**If(OB Get type(\$Object;"name") # Is undefined)***See also:* [OB Is empty](#)

## OB Is empty

OB Is empty(object) → Boolean

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
戻り値	ブール	← <i>object</i> が空か未定義のときには True、それ以外の場合には False

**OB Is empty** コマンドは *object* 引数で指定したオブジェクトが空、もしくは未定義の場合に **True** を、オブジェクトが定義（初期化）され、少なくとも一つの要素を含んでいる場合には **False** を返します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

- ▶ **OB Is defined** コマンド同様、このコマンドもオブジェクトの中身によって異なる結果を返します。ここではその例を紹介します。:

**C\_OBJECT(\$ref)**\$empty:=**OB Is empty**(\$ref) //True\$def:=**OB Is defined**(\$ref) //False**OB SET**(\$ref;"name";"Susie";"age";4)

//\$ref="{ "name": "Susie", "age": 4 }"

\$empty:=**OB Is empty**(\$ref) //False\$def:=**OB Is defined**(\$ref) //True**OB REMOVE**(\$ref;"name")**OB REMOVE**(\$ref;"age")\$empty:=**OB Is empty**(\$ref) //True\$def:=**OB Is defined**(\$ref) //True*See also:* [OB Is defined](#)

## OB Get

OB Get(object ; property{; type}) → Value

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト

property テキスト → 情報を取得したいプロパティ名  
 type 倍長整数 → 値を変換したい型  
 戻り値 テキスト、実数、ブール、 ← プロパティのカレントの値  
 ポインター、日付、時間、  
 ランゲージオブジェクト

**OB Get** コマンドは、*object* 引数で指定したオブジェクトの、*property* 引数で指定したプロパティのカレントの値を返します。任意の *type* 引数を指定することによってそこで指定した型へと値を変換します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、情報を取得したいプロパティのラベルを渡します。*property* 引数では、大文字と小文字は区別されることに注意して下さい。

特に指定がなければ、4D はプロパティの値を本来の型のまま返します。このとき、*type* 引数を使用することによって返ってくる値の型を強制的に変換することができます。この場合、*type* 引数には以下の定数のどれか一つを渡して下さい。これらの定数は "Field and Variable Types" テーマ内にあります。(4D の v14 では、このテーマには新しい定数が追加されています。):

定数 ( 値 )	説明
<u>Is real</u> (1)	
<u>Is text</u> (2)	
<u>Is date</u> (4)	
<u>Is boolean</u> (6)	
<u>Is longInt</u> (9)	
<u>Is integer</u> (8)	
<u>Is integer 64 bits</u> (25)	
<u>Is string var</u> (24)	
<u>Is time</u> (11)	
<u>Is object</u> (38)	4D の v14 からの新定数
<u>Object array</u> (39)	4D の v14 からの新定数
<u>Is JSON null</u> (255)	4D の v14 からの新定数

コマンドは *property* 引数で指定されたプロパティの値を返します。いくつかのデータの型がサポートされています。以下のことに注意して下さい:

- ポインターはそのまま返されますが、**JSON Stringify** コマンドを使用することによってポインターが指してる値に変換することができます。
- 日付は "\"YYYY-MM-DDTHH:mm:ssZ\"" というフォーマットで返されます。

- 実数の値において、小数点はかならずピリオド (".") になります。
- 時刻は数字で表現されます。(4D では秒単位の、JSON ではミリ秒単位の数字になります。)

- ▶ テキスト型の値を取得する場合は考えます。:

```
C_OBJECT($ref)
C_TEXT($FirstName)
OB SET($ref;"FirstName";"Harry")
$FirstName:=OB Get($ref;"FirstName")
// $FirstName = "Harry" (text)
```

- ▶ 実数型の値を取得して、倍長整数へと変換する場合は考えます。:

```
OB SET ($ref ; "age" ; 42)
$age:=OB Get($ref ; "age") // $age は実数型 ( デフォルト )
$age:=OB Get ($ref ; "age" ; Is longInt) // $age を倍長整数型に変換
```

- ▶ オブジェクトの中の値を取得する場合は考えます。:

```
C_OBJECT($ref1;$ref2)
OB SET($ref1;"LastName";"Smith") // $ref1={"LastName":"Smith"}
OB SET($ref2;"son";$ref1) // $ref2={"son":{"LastName":"Smith"}}
$son:=OB Get($ref2;"son") // $son={"LastName":"john"} (object)
$sonsName:= OB Get($son ; "name") // $sonsName="john" (text)
```

- ▶ 従業員の年齢を二度修正したい場合は考えます。:

```
C_OBJECT($ref_john;$ref_jim)
OB SET($ref_john;"name";"John";"age";35)
OB SET($ref_jim;"name";"Jim";"age";40)
APPEND TO ARRAY($myArray;$ref_john) // オブジェクト配列を作成
APPEND TO ARRAY($myArray;$ref_jim)
// John の年齢を 35 から 25 へと修正
OB SET($myArray{ 1 }; "age";25)
// "John" の age の値を上書き
For($i;1;Size of array($myArray))
  If(OB Get($myArray{$i};"name")="John")
    OB SET($myArray{$i};"age";36) // 25 から 36 へと修正
    // $ref_john={"name":"John","age":36}
  End if
End for
```

- ▶ ISO 形式のデータを日付型で取り出す場合は考えます。:

```
C_OBJECT($object)
C_DATE($birthday)
C_TEXT($birthdayString)
```

```

OB SET($object;"Birthday";"1990-12-25T12:00:00Z")
$birthdayString:=OB Get ($object;"Birthday")
// $birthdayString="1990-12-25T12:00:00Z"
$birthday:=OB Get ($object;"Birthday";Is date)
// $birthday=25/12/90

```

- ▶ 入れ子にされたオブジェクトを使用することもできます。:

```

C_OBJECT($ref1;$child;$children)
C_TEXT($childName)
OB SET($ref1;"FirstName";"John";"LastName";"Monroe")
//{"FirstName":"john","LastName":"Monroe"}
OB SET($children;"children";$ref1)
$child:=OB Get($children;"children")
//$son = {"FirstName":"John","LastName":"Monroe"} (object)
$childName:=OB Get($child;"LastName")
//$childName = "Monroe" (text)
// または
$childName:=OB Get(OB Get($children;"children");"LastName")
// $childName = "Monroe" (text)

```

## OB GET ARRAY

OB GET ARRAY(object ; property ; array)

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ 情報を取得したいプロパティ名
array	テキスト配列、実数配列、 ブール配列、オブジェクト 配列、ポインター配列	← プロパティの値の配列

**OB GET ARRAY** コマンドは、*object* 引数で指定されたランゲージオブジェクト内の、*property* 引数で指定されたプロパティの値を配列として取り出し、*array* 引数で指定した配列の中へ戻します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、値を取得したいプロパティのラベルを渡します。  
*property* 引数では、大文字と小文字は区別されることに注意して下さい。

- ▶ **OB SET ARRAY** コマンドの例示で定義された以下のオブジェクトにおいて、

<b>\$Children</b>	["Children":[{"name":"Richard","age":7},{"name":"Susan","age":4}...]
<b>Children</b>	[{"name":"Richard","age":7},{"name":"Susan","age":4},{"name":"J...]
<b>[0]</b>	{"name":"Richard","age":7}
<b>[1]</b>	{"name":"Susan","age":4}
<b>[2]</b>	{"name":"James","age":3}
<b>age</b>	3
<b>name</b>	"James"

以下の値を取得したい場合は以下のようになります。:

```
ARRAY OBJECT($result;0)
```

```
OB GET ARRAY ($Children;"Children";$result)
```

式	値
<b>\$arrChildren</b>	3 エレメント
<b>\$arrChildren</b>	0
<b>\$arrChildren{0}</b>	undefined
<b>\$arrChildren{1}</b>	{"nom":"Richard","age":7}
<b>age</b>	7
<b>nom</b>	"Richard"
<b>\$arrChildren{2}</b>	{"name":"Susan","age":4}
<b>age</b>	4
<b>name</b>	"Susan"
<b>\$arrChildren{3}</b>	{"name":"James","age":3}

- ▶ 配列の第一要素の値を変更したい場合、以下のようになります。:

```
// "age" の値を変更:
```

```
ARRAY OBJECT($result;0)
```

```
OB GET ARRAY($Children; "Children" ; $result)
```

```
OB SET($result{1}; "age" ; 25)
```

*See also:* **OB SET ARRAY**

## OB GET PROPERTY NAMES

```
OB GET PROPERTY NAMES(object ; arrProperties{; arrTypes})
```

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
arrProperties	テキスト配列	← プロパティ名
arrTypes	倍長整数配列	← プロパティの型

**OB GET PROPERTY NAMES** コマンドは、*object* 引数で指定されたランゲージオブジェクト内に含まれるプロパティの名前を、*arrProperties* へと返します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*arrProperties* 引数にはテキスト配列を渡します。配列が存在しない場合、コマンドが自動的に作成してリサイズします。

また、任意の *arrTypes* 引数に倍長整数配列を渡すこともできます。この場合、*arrProperties* 内の個々の要素に関して、プロパティに保存された値の型を *arrTypes* に返します。返される値は、"Field and Variable Types" テーマ内にある以下の定数のどれかになります。(4D の v14 では、このテーマには新しい定数が追加されています。):

定数 ( 値 )	説明
<u>Is real</u> (1)	
<u>Is text</u> (2)	
<u>Is undefined</u> (5)	
<u>Is boolean</u> (6)	
<u>Is string var</u> (24)	
<u>Is object</u> (38)	4D の v14 からの新定数
<u>Object array</u> (39)	4D の v14 からの新定数
<u>Is JSON null</u> (255)	4D の v14 からの新定数

- ▶ オブジェクトが空でないかどうかをテストしたい場合を考えます。:

```

ARRAY TEXT(arrNames;0)
ARRAY LONGINT(arrTypes;0)
C_OBJECT($ref_richard)
OB SET($ref_richard;"name";"Richard";"age";7)
OB GET PROPERTY NAMES($ref_richard;arrNames;arrTypes)
  //arrNames{1}="name", arrNames{2}="age"
  //arrTypes{1}=2, arrTypes{2}=1
If(Size of array(arrNames)#0)
  //...
End if

```

- ▶ オブジェクト配列の要素を使用する場合を考えます。:

```

C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrayChildren;0)

OB SET($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY($arrayChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4)
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James";"age";3)
APPEND TO ARRAY($arrayChildren;$ref_james)

```

## OB GET PROPERTY NAMES (\$arrayChildren{1};\$arrNames;\$arrTypes)

式	値
<ul style="list-style-type: none"> <li>▾ \$arrayChildren{1} <ul style="list-style-type: none"> <li>▾ age</li> <li>▾ name</li> </ul> </li> <li>▾ \$arrNames <ul style="list-style-type: none"> <li>▾ \$arrNames</li> <li>▾ \$arrNames{0}</li> <li>▾ \$arrNames{1}</li> <li>▾ \$arrNames{2}</li> </ul> </li> <li>▾ \$arrTypes <ul style="list-style-type: none"> <li>▾ \$arrTypes</li> <li>▾ \$arrTypes{0}</li> <li>▾ \$arrTypes{1}</li> <li>▾ \$arrTypes{2}</li> </ul> </li> </ul>	<pre>{   "name": "Richard",   "age": 7 }</pre> <p>7</p> <p>"Richard"</p> <p>2 エレメント</p> <p>0</p> <p>""</p> <p>"name"</p> <p>"age"</p> <p>2 エレメント</p> <p>0</p> <p>0</p> <p>2</p> <p>1</p>

## OB Get type

OB Get type(object ; property) → Longint

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ 読み出したいプロパティ名
戻り値	倍長整数	← プロパティの値のタイプ

**OB Get type** コマンドは、*object* 引数で指定されたランゲージオブジェクト内の、*property* 引数で指定されたプロパティと関連付けられた値の型を返します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、値の型を取得したいプロパティのラベルを渡します。*property* 引数では、大文字と小文字は区別されることに注意して下さい。

コマンドは、指定した値の型を示す倍長整数の値を返します。返される値は、"Field and Variable Types" テーマ内にある以下の定数のどれかになります。(4D の v14 では、このテーマには新しい定数が追加されています。)

:

定数 ( 値 )	説明
<u>Is real</u> (1)	
<u>Is text</u> (2)	
<u>Is undefined</u> (5)	
<u>Is boolean</u> (6)	
<u>Is string var</u> (24)	
<u>Is object</u> (38)	4D の v14 からの新定数

<a href="#">Object array (39)</a>	4D の v14 からの新定数
<a href="#">Is JSON null (255)</a>	4D の v14 からの新定数

- ▶ よくある値の型を取得する場合があります。:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
$type:=OB Get type($ref;"name") // $type は 2 を返します。
$type2:=OB Get type($ref;"age") // $type2 は 1 を返します。
```

*See also:* [OB GET PROPERTY NAMES](#)

## OB REMOVE

OB REMOVE(object ; property )

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ 削除したいプロパティの名前

**OB REMOVE** コマンドは、*object* 引数で指定されたランゲージオブジェクト内の、*property* 引数で指定されたプロパティを削除します。削除した際には、そこにあったカレント値も削除されます。

*object* で指定するオブジェクトは、[C\\_OBJECT](#) コマンドを使用して作成されている必要があります。

*property* 引数には、削除したいプロパティのラベルを渡します。  
*property* 引数では、大文字と小文字は区別されることに注意して下さい。

- ▶ オブジェクトから、"age" のプロパティを削除する場合があります。:

```
C_OBJECT ($Object)
OB SET($Object;"name";"smith";"age";42;"client";True)
// $Object={"name":"smith","age":42,"client":true}
OB REMOVE($Object;"age")
// $Object={"name":"smith","client":true}
```

## OB SET

OB SET(object ; property ; value{; property2; value2; ...; propertyN; valueN})

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ 設定したいプロパティの名前
value	テキスト、実数、日付、ブール、ポインター、Null、ランゲージオブジェクト	→ プロパティの新しい値

**OB SET** コマンドは *object* 引数で指定されたランゲージオブジェクト内のプロパティ / 値のペアを作成、もしくは修正します (一つでも複数でも指定可)。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、作成または修正したいプロパティのラベル (名前) を渡して下さい。オブジェクト内に指定されたプロパティが存在する場合、その値は指定した値で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。

*property* 引数では、大文字と小文字は区別されることに注意して下さい。

*value* 引数には、プロパティに設定したい値を渡して下さい。渡せる値としては複数の型がサポートされています。渡す際には以下のことに注意して下さい。:

- ポインタを渡した場合、それはそのままの形で保存されます。値を解決するためには **JSON Stringify** コマンドを使用します。
  - 日付を渡す場合には、"*\*"YYYY-MM-DDTHH:mm:ssZ*\*" というフォーマットで表現されている必要があります。
  - ランゲージオブジェクトを渡した場合、コマンドはオブジェクトの参照を使用し、実際にコピーを作成するわけではありません。
- ▶ オブジェクトを作成し、テキスト型のプロパティを追加する場合を考えます。:

```
C_OBJECT($Object)
OB SET($Object ; "FirstName" ; "John" ; "LastName" ; "Smith")
// $Object = { "FirstName": "John", "LastName": "Smith" }
```

- ▶ オブジェクトを作成し、ブール型のプロパティを追加する場合を考えます。:

```
C_OBJECT($Object)
OB SET($Object ; "LastName"; "smith"; "age"; 42; "client"; True)
// $Object = { "LastName": "smith", "age": 42, "client": true }
```

- ▶ プロパティを修正する場合、:

```
// $Object = { "FirstName": "John", "LastName": "Smith" }
OB SET($Object ; "FirstName"; "Paul")
// $Object = { "FirstName": "Paul", "LastName": "Smith" }
```

- ▶ プロパティを追加する場合、:

```
// $Object = { "FirstName": "John", "LastName": "Smith" }
```

```
OB SET($Object ; "department";"Accounting")
// $Object = {"FirstName":"Paul","LastName":"Smith","department":"Accounting"}
```

- ▶ プロパティの名前を変更する場合、:

```
C_OBJECT($Object)
OB SET($Object ; "LastName";"James";"age";35)
// $Object = {"LastName":"James","age":35}
OB SET($Object ; "FirstName";OB Get($Object ; "LastName"))
// $Object = {"FirstName":"James","LastName":"James","age":35}
OB REMOVE($Object ; "LastName")
// $Object = {"FirstName":"","age":35}
```

- ▶ ポインターを使用する場合、:

```
// $Object = {"FirstName":"Paul","LastName":"Smith"}
C_TEXT($LastName)
OB SET($Object ; "LastName";->$LastName)
// $Object = {"FirstName":"Paul","LastName":->$LastName}
$jsonString:=JSON Stringify($Object)
// $jsonString="{\"FirstName\":\"Paul\",\"LastName\":\"\"}"
$LastName:="Wesson"
$jsonString:=JSON Stringify($Object)
// $jsonString="{\"FirstName\":\"Paul\",\"LastName\":\"Wesson\"}"
```

- ▶ オブジェクトを使用する場合、:

```
C_OBJECT($ref_smith)
OB SET($ref_smith ; "name" ; "smith")
C_OBJECT($ref_emp)
OB SET($ref_emp ; "employee" ; $ref_smith)
$json_string := JSON Stringify($ref_emp)
// $ref_emp = {"employee":{"name":"smith"}} (object)
// $json_string = "{\"employee\":{\"name\":\"smith\"}}\" (string)
```

値をプログラム実行中に変えることもできます。:

```
OB SET($ref_smith ; "name" ; "smyth")
// $ref_smith = {"employee":{"name":"smyth"}}
$string := JSON Stringify ($ref_emp)
// $string = "{\"employee\":{\"name\":\"smyth\"}}"
```

*See also:* [OB SET NULL](#)

## OB SET ARRAY

OB SET ARRAY(object ; property ; array)

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ 設定したいプロパティ名
array	テキスト配列、実数配列、 ブール配列、オブジェクト配列、ポインター配列	→ プロパティに保存したい配列

**OB SET ARRAY** コマンドは、*object* 引数で指定されたランゲージオブジェクト内の、*property* 引数で指定されたプロパティと関連付けたい配列を *array* 引数で指定して定義します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、作成または修正したいプロパティのラベル（名前）を渡して下さい。オブジェクト内に指定されたプロパティが存在する場合、その値は指定した値で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。

*property* 引数では、大文字と小文字は区別されることに注意して下さい。

*array* 引数には、プロパティの値として関連付けたい配列を渡して下さい。渡せる配列の型としては複数の型がサポートされています。

---

*Note* 二次元配列を使用することはできません。

---

- ▶ テキスト配列を使用する場合を考えます。:

```
C_OBJECT($Children)
ARRAY TEXT($arrChildren;3)
$arrChildren{1}:="Richard"
$arrChildren{2}:="Susan"
$arrChildren{3}:="James"
```

```
OB SET ARRAY($Children;"Children";$arrChildren)
// Value of $Children = {"Children":["Richard","Susan","James"]}
```

- ▶ 配列に要素を追加する場合を考えます。:

```
ARRAY TEXT($arrText;2)
$arrText{1}:="Smith"
$arrText{2}:="White"
C_OBJECT($Employees)
OB SET ARRAY($Employees;"Employees";$arrText)
APPEND TO ARRAY($arrText;"Brown") //4D 配列へと追加
```

```
// $Employees = {"Employees":["Smith","White"]}
```

```
OB SET ARRAY($Employees;"Employees";$arrText)
```

```
// $Employees = {"Employees":["Smith","White","Brown"]}
```

- ▶ テキスト配列の一要素を配列として使用する場合を考えます。:

```
// $Employees = {"Employees":["Smith","White","Brown"]}
```

```
OB SET ARRAY ($Employees ;"Manager";$arrText{1})
```

```
// $Employees = {"Employees":["Smith","White","Brown"],"Manager":["Smith"]}
```

- ▶ オブジェクト配列を使用する場合を考えます。:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
```

```
ARRAY OBJECT($arrChildren;0)
```

```
OB SET ($ref_richard;"name";"Richard";"age";7)
```

```
APPEND TO ARRAY($arrChildren;$ref_richard)
```

```
OB SET($ref_susan;"name";"Susan";"age";4)
```

```
APPEND TO ARRAY($arrChildren;$ref_susan)
```

```
OB SET($ref_james;"name";"James";"age";3)
```

```
APPEND TO ARRAY($arrChildren;$ref_james)
```

```
// $arrChildren {1} = {"name":"Richard","age":7}
```

```
// $arrChildren {2} = {"name":"Susan","age":4}
```

```
// $arrChildren {3} = {"name":"James","age":3}
```

```
OB SET ARRAY($Children;"Children";$arrChildren)
```

```
// $Children = {"Children":[{"name":"Richard","age":7},{"name":"Susan",
```

```
// "age":4},{"name":"James","age":3}]}
```

オブジェクトはデバッガ内では以下の様に表示されます。:

<pre> \$Children ├── Children │   ├── [0] │   ├── [1] │   └── [2] │       ├── age │       └── name </pre>	<pre> {"Children":[{"name":"Richard","age":7},{"name":"Susan","age":4}, [{"name":"Richard","age":7},{"name":"Susan","age":4},{"name":"J... {"name":"Richard","age":7} {"name":"Susan","age":4} {"name":"James","age":3} 3 "James" </pre>
---	--

*See also:* [OB GET ARRAY](#)

**OB SET NULL**

OB SET NULL(object ; property)

引数	型	説明
object	ランゲージオブジェクト	→ 構造化されたオブジェクト
property	テキスト	→ null 値を適用したいプロパティ名

**OB SET NULL** コマンドは、*object* 引数で指定されたランゲージオブジェクトに **null** の値を保存します。

*object* で指定するオブジェクトは、**C\_OBJECT** コマンドを使用して作成されている必要があります。

*property* 引数には、**null** の値を保存したいプロパティのラベル(名前)を指定します。オブジェクト内に指定されたプロパティが存在する場合、その値は **null** で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。

*property* 引数では、大文字と小文字は区別されることに注意して下さい。

- ▶ Lea の "age" というプロパティに **null** を入れる場合を考えます。:

```
C_OBJECT($ref)
OB SET($ref;"name";"Lea";"age";4)
// $ref = {"name":"Lea","age":4}
...
OB SET NULL($ref;"age")
// $ref = {"name":"Lea","age":null}
```

*See also:* **OB SET**

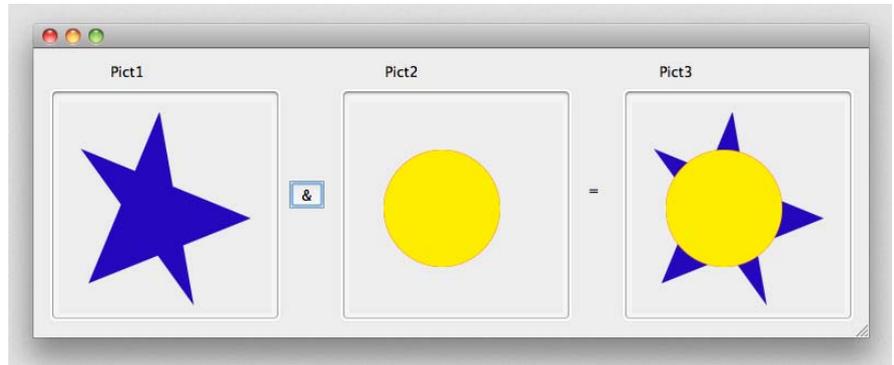
## ピクチャ演算子

4D の v14 では、ピクチャ演算モジュールのアップデートに伴い、& 演算子と | 演算子の振る舞いに変更されています。

### "&" 演算子 (排他的論理和)

& 演算子は、XOR 重ねではなく、一つのピクチャの上にもう一つのピクチャを重ねるといった動作をするように変更されました。:

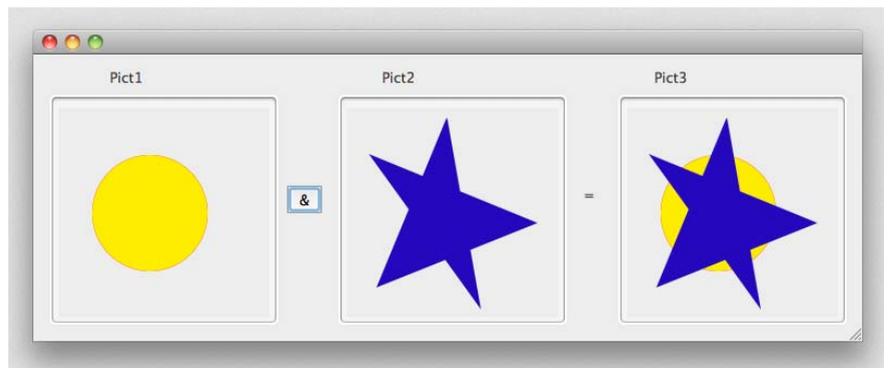
Pict3 := Pict1 & Pict2



この演算子は以下のコマンドと同じ結果をもたらします。:

**COMBINE PICTURES**( pict3 ; pict1 ; Superimposition ; pict2)

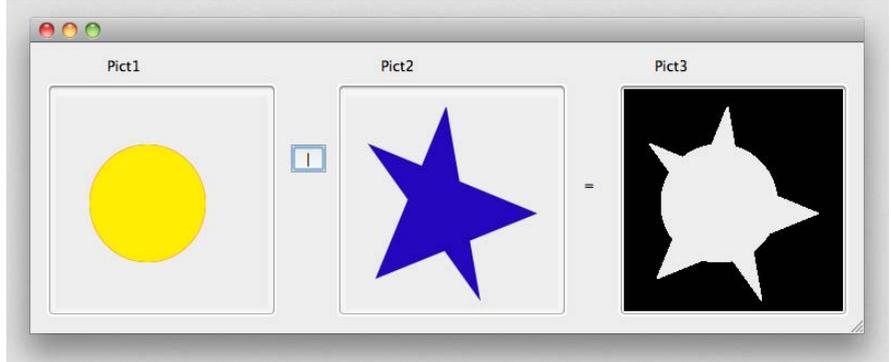
Pict1&Pict2 と Pict2&Pict1 は、異なる結果になることに注意して下さい。:



"|" 演算子 (包括的論理和)

"|" 演算子は OR 重ねではなく、同じサイズの二つの画像を重ね合わせたときの、マスクの結果を返すように変更されました。

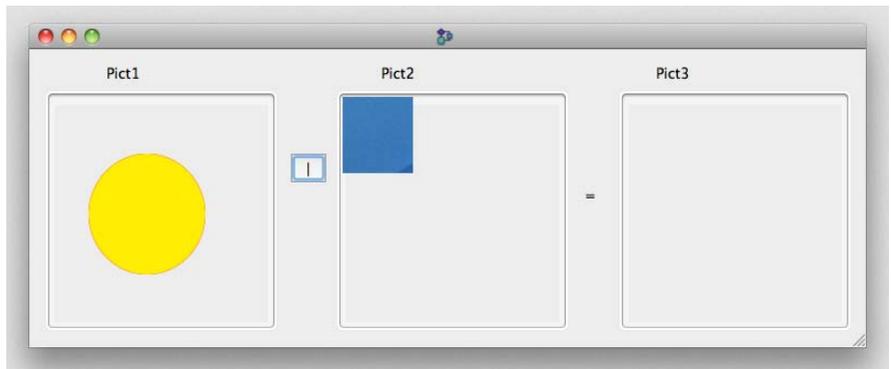
Pict3 := Pict1 | Pict2



この演算子は以下のコマンドと同じ結果をもたらします。:

`$equal:=Equal pictures(Pict1 ; Pict2 ; Pict3)`

このとき、Pict1 と Pict2 は全く同じ寸法でなければならないことに注意して下さい。2つの画像が異なる寸法である場合、Pict1 | Pict2 の結果は空のピクチャとなります。:



## 印刷

### OPEN PRINTING FORM

4D v14 では、このコマンドは "フォーム" テーマに移されています。それに加えて、名前が `FORM LOAD` に変更され、挙動も変更されています。詳細についてはこのコマンドの説明の部分を参照して下さい。

# プロセス

## Execute on server

Execute on server ( procedure ; stack {; name {; param {; param2 ; ... ; paramN}}};\* ) -> 戻り値

*param* に 4D オブジェクト (**C\_OBJECT**) を渡した場合、もしくは戻り値として指定した場合、サーバーに対しては UTF-8 の JSON がやりとりされるようになります。

**C\_OBJECT** オブジェクトがポインターを含んでいた場合、ポインターそのものではなく、ポインターの指している値が送られます。

# クイックレポート

## QR REPORT

QR REPORT ( {aTable ;} document {; hierarchical {; wizard {; search {; *methodName*{; \*}}}} )

引数	型	説明
aTable	テーブル	→ レポートを作成するテーブル、省略時はデフォルトテーブル
document	文字列	→ ロードするクイックレポートドキュメント
hierarchical	ブール	→ True = リレートする n テーブルを表示 False または省略 = 表示しない (デフォルト)
wizard	ブール	→ True = ウィザード・ボタンを表示 False または省略 = 表示しない (デフォルト)
search	ブール	→ True = 検索ツールとマスターテーブル選択を表示する False または省略 = 表示しない (デフォルト)
<i>methodName</i>	文字列	→ 呼び出すメソッド名
*	演算子	→ プリントダイアログボックスを表示しない

4D v14 では、**QR REPORT** コマンドに新たに任意の *methodName* 引数が渡せるようになりました。この引数はメニューアイテムから選択するかボタンをクリックしてクイックレポートエディターのコマンドが呼び出された際に実行される 4D プロジェクトメソッドを指定します。

この引数を **QR REPORT** コマンドに渡すことは、QR ON COMMAND をクイックレポートエディターウィンドウにて使用する事と同等の効果があります (QR ON COMMAND は組み込みエリアの中でしか動きません)。

この新しい引数は、特にクイックレポートで使用される文字セットを変更するために有用です。

この *methodName* 引数で指定したメソッドは、引数を二つ受け取ります。

- \$1 にはエリア参照が格納されます ( 倍長整数 )。
- \$2 には選択されたコマンド数が格納されます ( 倍長整数 )。

*Note* データベースをコンパイラーを使用してコンパイルする場合、例え使用しないとしても \$1 引数と \$2 引数を明示的に宣言する必要があります。

ユーザーが最初に選択したコマンドを実行したい場合、以下の宣言を *methodName* で指定したメソッドで使用して下さい。

```
QR EXECUTE COMMAND($1;$2).
```

*methodName* 引数が空の文字列 ("") または省略されていた場合、メソッドは何も呼び出されず、**QR REPORT** の標準オペレーションが適用されます。

- ▶ 呼び出したクイックレポート内の文字コードを **QR REPORT** を使用して Mac Roman に変換する場合があります。

```
QR REPORT([MyTable];Char(1);False;False;False;"myCallbackMeth")
```

*myCallbackMeth* メソッドはレポートが生成されたときにそれを変換します。

```
C_LONGINT ($1;$2)
```

```
If ($2=qr cmd generate) // もしレポートを生成したら
```

```
  C_BLOB ($myblob)
```

```
  C_TEXT ($path;$text)
```

```
  QR EXECUTE COMMAND ($1;$2) // コマンドを実行
```

```
  QR GET DESTINATION ($1;$type;$path) // 出力先を取得
```

```
  DOCUMENT TO BLOB ($path;$myblob)
```

```
    //UTF-8 を使用したテキストに変換
```

```
  $text:=Convert to text ($myblob;"UTF-8")
```

```
    //MacRoman を使用
```

```
  CONVERT FROM TEXT ($text;"MacRoman";$myblob)
```

```
    // 変換済みのレポートを返す
```

```
  BLOB TO DOCUMENT ($path;$myblob)
```

```
Else // それ以外の場合はコマンドを実行
```

```
  QR EXECUTE COMMAND ($1;$2)
```

```
End if
```

## QR SET DESTINATION

```
QR SET DESTINATION ( area ; type {; specifics} )
```

4D v14から4D Chartがサポートされなくなったので(20P “削除された機能”を参照して下さい。)、クイックレポートの出力先として4D Chartは選択

できなくなりました。これに伴い定数 `qr 4D Chart area` は `_O qr 4D Chart area` と改名され、今後使用されるべきではありません。

## スペルチェッカー

スペルチェッカーインターフェースとスペルチェッカー機能は、4D v14 にてより改善されています。特に、OS X のシステムスペルチェッカーに対するサポートが強化されています（しかしながら、コーディアルスペルチェッカーはもうサポートされていません）。詳細に関しては、86P “[スペルチェック](#)” を参照して下さい。

スペルチェッカーテーマ内の以下のコマンドにも、変更があります。

---

*Note* OS X 環境下において、[SET DATABASE PARAMETER](#), [Get database parameter](#) コマンドを使用することによってスペルチェックに使用する辞書を指定できるようになりました。

---

### SPELL CHECK TEXT

SPELL CHECK TEXT ( text ; errPos ; errLength ; checkPos ; arrSuggest )

OS X 環境下においてシステムのスペルチェックが使用されているとき、このコマンドは文法修正まではサポートしません。

### SPELL GET DICTIONARY LIST

SPELL GET DICTIONARY LIST ( langID ; langFiles ; langNames )

このコマンドはコーディアル辞書への参照は返さなくなりました。

### SPELL SET CURRENT DICTIONARY

SPELL SET CURRENT DICTIONARY {( dictionary )}

- 4D v14 ではコーディアル辞書はサポートされなくなっています。デフォルトでは 4D は以下の辞書を使用します。：
  - Windows 環境下においてはハンスペル辞書を使用。
  - OS X 環境下においてのみシステムのスペルチェッカーを使用。
- 互換性のために、*dictionary* 引数にコーディアルの辞書を指定する数 ("Dictionaries" テーマ内にある値または定数) を渡す事は可能です。ただしその場合、使用する辞書は内部でハンスペル辞書（または OS X でのシステムの辞書）へとリダイレクトされます。どの場合においても、[SPELL GET DICTIONARY LIST](#) コマンドと [SPELL](#)

Get current dictionary コマンドはコーディアル辞書への参照は返さないという点に注意して下さい。

- このコマンドは引数として言語コード「BCP 47」や「ISO 639-1」、「ISO 639-2」の言語コードを受け取ることができます。例えば、BCP 47 のランゲージコード "en" はアメリカ英語を意味し、"en-GB" はイギリス英語を意味します。これらのコードはシステム内部でカレントの辞書の対応するコードへと変換されます。

## 文字列

### String

String ( expression {; format {; addTime}} ) → String

null の日付を渡した時、Internal date short フォーマットを使用することにより、年が 4 桁の 0 表示される、以下の様な結果が返されます。:

```
00/00/0000
```

以前のバージョンの 4D では、年は 2 桁の 0 で表示されていました。

## ストラクチャーアクセス

### PAUSE INDEXES

PAUSE INDEXES ( aTable )

引数	型	説明
aTable	テーブル	→ インデックスを停止するテーブル

**PAUSE INDEXES** コマンドは *aTable* のインデックス管理を主キーのインデックスを除いて全て一時的に停止します。

このとき、インデックスはデータ (4DIndx ファイル) やデータベースのストラクチャ (\_USER\_INDEXES システムテーブル) から消去されてしまいうわけではありませんが、停止されてしまったことによりそれ以降更新されることはありません。インデックス管理が無効になっている時には、*aTable* 上での操作 (クエリ、並べ替え、レコードの追加・修正・削除等) はインデックスを使用しません。

このコマンドは、複数のインデックスを持つテーブルに大量のデータをインポートしようとしたり、変更しようとしたりする際に便利です。何故なら 4D は、レコードが保存されたときには毎回インデックスを更新するので、その操作にとっても時間がかかる可能性があるからです。事前に

インデックスを無効化しておくことで、操作にかかる時間を大幅に短縮することが出来るかもしれません。

操作が終わった後に再びインデックスを再開させるときには、新しい **RESUME INDEXES** コマンドを *aTable* に対して使用します。

*Note* CREATE INDEX コマンド DELETE INDEX とを使用することで似たような結果を得ることが出来ますが、その場合は *aTable* 内の各インデックスを個別に指定する必要があります。

**PAUSE INDEXES** をテーブルに対して使用し、**RESUME INDEXES** を使わないままにそのデータベースを閉じた場合でも、インデックスはデータベースを再度開いた場合に自動的に全て再構築されます。

*Note* このコマンドは 4D リモートから実行することはできません。

- ▶ 大量のデータをインポートするためのメソッドの例：

```
PAUSE INDEXES([Articles])
IMPORT DATA("HugeImport.txt") // ここでインポート
RESUME INDEXES([Articles])
```

*See also:* **RESUME INDEXES**

## RESUME INDEXES

RESUME INDEXES ( *aTable* {; \*} )

引数	型	説明
<i>aTable</i>	テーブル	→ インデックスを再開するテーブル
*	演算子	→ 渡した場合、非同期モード

**RESUME INDEXES** コマンドは *aTable* のインデックスが **PAUSE INDEXES** コマンドによって停止されていた場合、そのインデックスを全て再開させます。もし *aTable* のインデックスが停止されていなかった場合は、このコマンドは何もしません。

大体的場合、このコマンドを実行することにより、*aTable* のインデックスが再構築されます。

任意の \* 演算子を渡した場合、インデックスの再構築は非同期モードにて行われます。これはつまり、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るということです。この演算子を省略した場合、再構築が終わるまでメソッドの実行はブロックされます。

**RESUME INDEXES** コマンドは、4D サーバーか、ローカルの 4D でしか呼び出すことができません。このコマンドがリモートの 4D のマシンから呼

び出された場合、エラー 10513 が発生します。このエラーは ON ERR CALL コマンドに実装されたメソッドによって割り込むことができます。

See also: [PAUSE INDEXES](#)

## スタイル付テキスト

新しい "Styled Text" テーマには、マルチスタイルのテキストエリア (またの名をリッチテキストエリアとも言います) を管理・変更するためのコマンドや関数が用意されています。

マルチスタイルテキストに関するコマンドは、以前は "オブジェクトプロパティ" 内にありましたが、4D の v14 からこの新しいテーマに移動となり、"ST" という接頭辞がつけました。(285P "改名されたコマンド" を参照して下さい)。またこのテーマにはいくつかの新しいコマンドも追加されています。

リッチテキストを宣言するためには、プロパティリストの中の「マルチスタイル」にチェックを入れる必要があることに注意して下さい。4D の v14 では、リッチテキスト導入に伴う新しいタグや属性が追加されています。詳細な情報に関しては 57P "マルチスタイルエリア" を参照して下さい。

---

*Note* 新しい **OBJECT Is styled text** コマンド ("オブジェクト (フォーム)" テーマ) を使用するとテキストエリアがマルチスタイルモードであるかどうかを調べることができます。

---

## 新コマンド

### ST COMPUTE EXPRESSIONS

ST COMPUTE EXPRESSIONS({\*; }object{; startSel{; endSel{}})

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点

**ST COMPUTE EXPRESSIONS** コマンドは、*object* 引数で指定されたテキストフィールドまたはテキスト変数のダイナミック 4D 式を更新します。

マルチスタイルテキストエリアで使用されている 4D 式の詳細に関しては、276P “[ST INSERT EXPRESSION](#)” を参照して下さい。

このコマンドは *object* で指定したオブジェクト内の 4D 式の結果を、カレントの内容に応じて更新してそれを表示します。例えば、挿入された 4D 式が時刻であった場合、表示される時刻は [ST COMPUTE EXPRESSIONS](#) コマンドを使用するたびに変更されます。4D 式は以下のときにも更新されます。:

- 挿入されたとき
- オブジェクトがロードされたとき
- [ST FREEZE EXPRESSIONS](#) コマンドにおいて、2 番目の \* を渡して 4D 式が固定化されたとき

[ST COMPUTE EXPRESSIONS](#) コマンドは SPAN タグも含めてスタイル付テキストを変更しません。*object* で指定されたオブジェクト内に表示された標準テキストのみ変更します。処理された値はスタイル付テキストの中には保存されず、参照のみが保存されます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

指定されたオブジェクトはフォーカスになっている必要はありません。しかしながら、オブジェクトはフォームに含まれている必要があります。そうでない場合には [ST COMPUTE EXPRESSIONS](#) コマンドは何もしません。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、[ST COMPUTE EXPRESSIONS](#) コマンドは指定された範囲内の 4D 式のみ変更します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、*startSel* の位置からテキストの終わりまでの全ての 4D 式が変更されます。
- *startSel* と *endSel* の両方を省略した場合、指定されたオブジェクト内の全ての 4D 式は変更されます。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
----------	----

<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

- ▶ テキストの選択範囲に含まれる参照を更新する場合があります。:

**ST COMPUTE EXPRESSIONS** (\*;"myText";ST Start highlight;ST End highlight)

*See also:* [ST FREEZE EXPRESSIONS](#)

## ST FREEZE EXPRESSIONS

ST FREEZE EXPRESSIONS({\*; }object{; startSel{; endSel}{; \*})

引数	型	説明
*	演算子	→ 指定時: <i>object</i> はオブジェクト名(文字列) 省略時: <i>object</i> は変数またはフィールド
<i>object</i>	ランゲージ オブジェクト	→ オブジェクト名(*指定時)、 または変数やフィールド(*省略時)
<i>startSel</i>	倍長整数	→ 選択範囲の開始地点
<i>endSel</i>	倍長整数	→ 選択範囲の終了地点
*	演算子	→ 渡した場合、固定化する前に 4D 式を更新 します。

**ST FREEZE EXPRESSIONS** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内の 4D 式の内容を固定化します。これにより、ダイナミックな 4D 式はスタティックなテキストへと変換され、オブジェクトに関連付けられた参照は解除されます。

マルチスタイルテキストエリアの中で使用される 4D 式の詳細については、[276P “ST INSERT EXPRESSION”](#) を参照して下さい。

**ST FREEZE EXPRESSIONS** コマンドは、変更された 4D 式の値を保存しません。この操作は特に、オブジェクトをマルチスタイルエリア外で使用する

る（エクスポート、ディスクファイルへの保存、印刷等）際に必要になります。なぜなら、エリアのそのものには、4D 式への参照しか保存されていないからです。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグや参照は文字数としてはカウントされません。

- *startSel* と *endSel* の両方を渡した場合、**ST FREEZE EXPRESSIONS** コマンドは指定された範囲内の 4D 式のみ固定化します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、*startSel* の位置からテキストの終わりまでの全ての 4D 式が固定化されます。
- *startSel* と *endSel* の両方を省略した場合、指定されたオブジェクト内の全ての 4D 式が固定化されます。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

デフォルトでは、4D 式は固定化される直前に再評価されるわけではありません。4D 式を評価しなおして固定化したい場合には、2 番目の \* 演算子を渡します。

- ▶ テキストの冒頭にカレントの時刻を挿入し、レコードに保存する前に固定化する場合を考えます。:

```
// テキストの冒頭に時刻を挿入
ST INSERT EXPRESSION (*;StyledText_t;"Current time";1)
// フォーマットを固定
ST FREEZE EXPRESSIONS (*;StyledText_t;1)
```

*See also:* [ST INSERT EXPRESSION](#), [ST COMPUTE EXPRESSIONS](#)

## ST Get content type

ST Get content type ( { \* ; } object { ; startSel { ; endSel { ; startBlock { ; endBlock } } } ) → Longint

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点
startBlock	倍長整数	← 選択範囲内の、同一のタイプの開始地点
endBlock	倍長整数	← 選択範囲内の、同一のタイプの終了地点
戻り値	倍長整数	← 内容のタイプ

[ST Get content type](#) コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で見つかったコンテンツのタイプを返します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集時のテキストに関する情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。

省略時には *object* 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、[ST Get content type](#) コマンドは指定された範囲内に限りコンテンツを評価します。

- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、コマンドは *startSel* の位置からテキストの終わりまでの範囲内のコンテンツのみ評価されます。
- *startSel* と *endSel* の両方を省略した場合、現在選択されている範囲のコンテンツのみが評価されます。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

任意の *startBlock* 引数と *endBlock* 引数は、オブジェクト内、もしくはオブジェクトの中で選択された範囲の中で、タイプが連続する位置を探し、その最初と最後の文字を返します。例えば、選択範囲内に 4D 式に続いて標準テキストが含まれていた場合、*startBlock* と *endBlock* はそれぞれ 4D 式の開始地点と終了地点を返します。この操作は選択範囲内の全てを処理するためにループさせることができます。

コマンドは、選択範囲内を調べた結果特定できたタイプを示す値を返します。返される値は、"Multistyle Text" テーマ内にある、以下のどれかになります。:

定数 ( 値 )	説明
<u>ST Plain Type</u> (0)	選択範囲にはテキストのみ含まれ、参照はありません。
<u>ST Url Type</u> (1)	選択範囲には URL 参照のみ含まれます。
<u>ST Expression Type</u> (2)	選択範囲には 4D 式の参照のみ含まれます。

<u>ST Mixed Type</u> (3)	選択範囲には、少なくとも異なる2つ以上のタイプが含まれます。
<u>ST Unknown tag type</u> (4)	選択範囲には、不明な型のタグのみ含まれます。
<u>ST User type</u> (5)	選択範囲には、カスタムの参照のみ含まれます。

- ▶ エリア内で選択されたコンテンツのタイプによって表示されるコンテキストメニューを変えたい場合を考えます。

#### Case of

: (Form event=On Clicked)

// 選択された範囲を取得します

**GET HIGHLIGHT** (\*;"myText";startSel;endSel)

**If** (Contextual click & (Macintosh control down=False))

// コンテキストメニューを呼び出し

**If** (startSel=endSel) // コンテンツが選択されていない場合

// 一部のコマンドのみ有効にします。

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;2)

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;4)

**ENABLE MENU ITEM**(<>menu\_STYLEDTEXT;6)

...

**Else** // コンテンツのタイプを取得

CT\_Texttype:=**ST Get content type** (\*;"myText";startSel;endSel)

**Case of** // 異なるタイプによって処理を変える

: (CT\_Texttype=ST Url Type)

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;6)

**ENABLE MENU ITEM**(<>menu\_STYLEDTEXT;7)

...

: (CT\_Texttype=ST Expression Type)

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;6)

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;7)

...

**Else**

**ENABLE MENU ITEM**(<>menu\_STYLEDTEXT;6)

**DISABLE MENU ITEM**(<>menu\_STYLEDTEXT;7)

...

**End case**

**End if**

**GET MOUSE**(\$xCoord;\$yCoord;\$ButtonState)

\$AlphaVar:=**Dynamic pop up menu**(<>menu\_STYLEDTEXT;"",  
\$xCoord;\$yCoord)

startSel:=-3

endSel:=-3

**End if**

...  
End case

## ST Get expression

ST Get expression( { \* ; } object{ ; startSel{ ; endSel} } ) → Text

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点
戻り値	テキスト	← 4D 式の内容

**ST Get expression** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で選択されている範囲の中で、最初に見つけた 4D 式を返します。

このコマンドは、オブジェクト内に挿入された 4D 式の内容を返します。(結果は、例えば "mymethod" や "[table1]field1" 等になります) 4D 式の値は返されません。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関する情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。

省略時には *object* 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、**ST Get expression** コマンドは指定された範囲内に限り 4D 式を探します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、コマンドは *startSel* の位置からテキストの終わりまでの範囲内の 4D 式を探します。
- *startSel* と *endSel* の両方を省略した場合、指定されたオブジェクトが編集集中であれば、そのとき選択されている範囲のテキスト内を探します。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます (ただし *endSel* が 0 である場合を除く)。

選択範囲の中に 4D 式が何もない場合、コマンドは空の文字列を返します。

- ▶ ダブルクリックイベントがあると、4D 式が実際にあるかどうかをチェックし、あった場合にはユーザーがそれを変更できるようにその値を取得したダイアログボックスを表示する、という場合について考えます。

Case of

: (Form event=On Double Clicked)

**GET HIGHLIGHT** (\*;"StyledText\_t";startSel;endSel)

**If** (ST Get content type (\*;"StyledText\_t";startSel;endSel)=ST Expression Type)

vExpression:=**ST Get expression** (\*;"StyledText\_t";startSel;endSel)

\$winRef:=**Open form window** ("Dial\_InsertExpr";

Movable form dialog box;Horizontally Centered;

Vertically Centered;\*)

**DIALOG** ("Dial\_InsertExpr")

**If** (OK=1)

**ST INSERT EXPRESSION** (\*;"StyledText\_t";vExpression;startSel;

endSel)

**HIGHLIGHT TEXT** (\*;"StyledText\_t";startSel;endSel)

**End if**

**End if**

**End case**

- ▶ ユーザーリンクがクリックされたときに 4D メソッドを実行したいという場合：

**Case of**

: (Form event=On Clicked)

// セレクションを取得

**HIGHLIGHT TEXT**(\*;"myText";startSel;endSel)

**If** (startSel#endSel) // 選択された範囲があるならば

// コンテンツの型を取得

\$CT\_type:=**ST Get content type**(\*;"myText";startSel;endSel)

**If** (\$CT\_type=**ST User type**) // これはユーザーリンクであるならば

*MyMethod* //4D メソッドを実行

**End if**

**End if**

**End case**

*See also:* [ST INSERT EXPRESSION](#)

## ST GET OPTIONS

ST GET OPTIONS({\*; }object; option; value{; option2; value2; ...; optionN; valueN})

引数	型	説明
*	演算子	→ 指定時：object はオブジェクト名 (文字列) 省略時：object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
option	倍長整数	→ 取得したいオプション
value	倍長整数	← オプションのカレントの値

**ST GET OPTIONS** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で動作中のオプションのカレントの値を取得します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集時のテキストに関する情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。

省略時には *object* 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

*option* 引数には、取得したいオプションを指定するコードを渡します。コマンドはそのオプションのカレントの値を *value* 引数に返します。option

引数には "Multistyle Text" テーマ内にある、以下の定数を使用することができます。:

定数 ( 値 )	説明
<u>ST Expressions Display Mode</u> (1)	以下のどちらかを値として渡します: <u>ST Values</u> (0): 4D 式の計算された値を表示 ( デフォルトの動作 ) <u>ST References</u> (1): 4D 式のソースを文字列として表示

▶ **ST SET OPTIONS** コマンドの例示も参照して下さい。

*See also:* **ST SET OPTIONS**

## ST GET URL

ST GET URL({\*; }object ; urlText ; urlAddress{ ; startSel{ ; endSel{ })

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 ( 文字列 ) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
urlText	テキスト	← リンクの表示テキスト
urlAddress	テキスト	← URL アドレス
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点

**ST GET URL** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で見つかった最初の URL の表示テキストとアドレスを返します。

表示テキストと URL アドレスは、それぞれ *urlText* 引数と *urlAddress* 引数に返されます。もし選択範囲内に URL が含まれない場合は、これらの引数には空の文字列が返されます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関する情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。

省略時には *object* 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタ

イルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、**ST GET URL** コマンドは指定された範囲内に限り URL を探します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、コマンドは *startSel* の位置からテキストの終わりまでの範囲内に限り URL を探します。
- *startSel* と *endSel* の両方を省略した場合、現在選択中の範囲内に限り URL を探します。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

- ▶ ダブルクリックイベントがあると、URL が実際にあるかどうかをチェックし、あった場合にはユーザーがそれを変更できるようにその値を取得したダイアログボックスを表示する、という場合について考えます。

#### Case of

: (Form event=On Double Clicked)

**GET HIGHLIGHT** (\*;"StyledText\_t";startSel;endSel)

**If (ST Get content type** (\*;"StyledText\_t";startSel;endSel)=ST Url Type)

//URL

**ST GET URL**(\*;"StyledText\_t";vTitle;vURL;startSel;endSel)

\$winRef:=**Open form window** ("Dial\_InsertURL";

Movable form dialog box;Horizontally Centered;

```

Vertically Centered;*)
SET WINDOW TITLE ("URL settings")
DIALOG ("Dial_InsertURL")
If (OK=1)
    ST INSERT URL (*;"StyledText_t";vTitle;vURL;startSel;endSel)
    HIGHLIGHT TEXT (*;"StyledText_t";startSel;startSel+1)
End if
End if
End case

```

*See also:* [ST INSERT URL](#)

## ST INSERT EXPRESSION

ST INSERT EXPRESSION({\*; }object ; expression{; startSel{; endSel})

引数	型	説明
*	演算子	→ 指定時 : object はオブジェクト名 (文字列) 省略時 : object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
expression	テキスト	→ 挿入したい 4D 式と (任意の) フォーマット
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点

[ST INSERT EXPRESSION](#) コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内に、*expression* 引数で指定された 4D 式への参照を挿入します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*expression* 引数には、オブジェクト内にて評価したい 4D 式を渡します。有効な 4D 式とは、値を返す文字列です。*expression* 引数には、フィールド、変数、4D コマンド、値を返す宣言、プロジェクトメソッド等を指定することができます。

4D 式は、引用符 ("" ) で囲まれている必要があります。

---

*Note* *expression* 引数にピクチャ型の変数を渡すことはできません。

---

*expression* で指定した 4D 式から返ってきた値がキャリッジリターンとタブを含んでいた場合、4D 式が入っているオブジェクトに合わせてそのテキストを表示します。例えばこのとき、キャリッジリターンは改行として扱われます。

*expression* 引数に、任意のフォーマット情報を渡すことによって、4D 式のフォーマット形式を指定することができます。この場合、以下の書式で指定する必要があります。:

```
"String(value;format)"
```

*value* 引数には 4D 式そのものを渡し、*format* 引数には適用したい書式を渡します。*format* には以下のどれかを指定することができます。:

- 数字に対しては、(存在するかしないかに関わらず)いかなる表示形式も指定することができます。例えば、"###,###" のような形です。
- 日付に対しては、存在する日付のフォーマットを指定する数字を渡すことができます。このとき、"Date display formats" テーマ内にある定数 (System date short 等) を使用することができます。
- 時刻に対しては、存在する時刻のフォーマットを指定する数字を渡すことができます。このとき、"Time Display Formats" テーマ内にある定数 (System time short 等) を使用することができます。

例えば、以下の様な形になります。:

```
"String([Table_1]Field_1;System date short)"
```

特に何も指定しなければ、*expression* の *values* がマルチスタイルテキストエリアに表示されます。また、**ST SET OPTIONS** コマンドを使用することによって強制的に参照を表示させることもできます。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、**ST INSERT EXPRESSION** コマンドは指定された範囲内のテキストを 4D 式からの結果で置換します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、*startSel* の位置からテキストの終わりまでの全てのテキストが 4D 式からの結果で置換されます。
- *startSel* と *endSel* の両方を省略した場合、指定されたオブジェクトが編集集中であれば、そのとき選択されている範囲のテキストが 4D 式からの結果で置換されるか、範囲が選択されていなければカーソルのある位置に挿入されます。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

- ▶ 選択したテキストをプロジェクトメソッドの結果で置き換える場合を考えます。:

**ST INSERT EXPRESSION** (\*;"myText";"MyMethod";ST Start highlight;ST End highlight)

*See also:* [ST INSERT URL](#), [ST Get expression](#)

## ST INSERT URL

ST INSERT URL({\*; }object ; urlText ; urlAddress{; startSel{; endSel{}})

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 ( 文字列 ) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 ( * 指定時 )、 または変数やフィールド ( * 省略時 )
urlText	テキスト	→ リンクの表示テキスト
urlAddress	テキスト	→ URL アドレス
startSel	倍長整数	→ 選択範囲の開始地点
endSel	倍長整数	→ 選択範囲の終了地点

**ST INSERT URL** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内に、URL のリンクを挿入します。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

*urlText* 引数には、オブジェクト内で表示される、リンクの表示テキストを渡します。例えば、"4D Web Site" や "Follow this link for more information" などのテキストラベルです。"http://www.4d.com" のような、アドレスそのものを指定することもできます。

*urlAddress* 引数には、"http://www.4D.com" のように、ブラウザに表示させたい Web サイトの完全なアドレスを渡します。

任意の *startSel* 引数と *endSel* 引数はオブジェクト内のテキストの選択範囲を指定します。*startSel* と *endSel* は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- *startSel* と *endSel* の両方を渡した場合、**ST INSERT URL** コマンドは指定された範囲内のテキストのみ *urlText* で指定したテキストと置換します。
- *startSel* のみを指定した場合、もしくは *endSel* の値がオブジェクト内の総文字数より大きい場合は、*startSel* の位置からテキストの終わりまでの全ての文字が *urlText* で置換されます。
- *startSel* と *endSel* の両方を省略した場合、指定されたオブジェクトが編集集中であればそのとき選択されている範囲のテキストが 4D 式からの結果で置換されるか、範囲が選択されていない場合はカーソルのある位置に 4D 式が挿入されます。

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(*)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(*)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

(\*) これら二つの定数を使用する際には、*object* 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

*Note* もし *startSel* 引数が *endSel* 引数より大きい場合、コマンドは何も行わず、OK 変数は 0 に設定されます。(ただし *endSel* が 0 である場合を除く)

リンクは挿入されれば既に有効になっています。ラベルを、Windows では **Ctrl+ クリック**、Mac OS X では **Command+ クリック** することによって既定のブラウザで *urlAddress* で指定したページを開くことができます。

- ▶ オブジェクト内で選択されたテキストを、4D のウェブサイトへのリンクで置き換えたい場合を考えます。:

```
vTitle := "4D Web Site"
```

```
vURL := "http://www.4d.com/"
```

```
ST INSERT URL (*,"myText";vTitle;vURL;ST Start highlight;ST End highlight)
```

*See also:* [ST INSERT EXPRESSION](#)

## ST SET OPTIONS

ST SET OPTIONS({\*; }object; option; value{; option2; value2; ...; optionN; valueN})

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
option	倍長整数	→ 設定したいオプション
value	倍長整数	→ オプションの新しい値

**ST SET OPTIONS** コマンドは、*object* 引数で指定されたスタイル付テキストフィールドまたはテキスト変数に関する様々なオプションを変更することができます。

任意の \* 演算子を渡した場合、*object* 引数でオブジェクト名を文字列で指定します。省略時には *object* 引数でフィールドまたは変数を指定します。

修正したいオプションを指定する値を *option* 引数に、新しく設定したい値を *value* に渡します。このどちらの引数にも、"Multistyle Text" テーマ内にある以下の定数を使用することができます。:

定数 ( 値 )	説明
<u>ST Expressions Display Mode</u> (1)	以下の値を渡すことができます: <u>ST Values</u> (0): 4D 式の、参照された値を表示します。(デフォルトの動作) <u>ST References</u> (1): 4D 式の、ソースコードの原文を文字列として表示します。

値を表示

Current time:	14:39:10
Field contents:	Bravo

ソースを表示

Current time:	String(Current time)
Field contents:	[Table_1]Comment

- ▶ 以下のコードは、エリアの表示モードを切り替えます。:

```

ST GET OPTIONS (*;"StyledText_t";ST Expressions Display Mode;$exprValue)
If ($exprValue=1)
    ST SET OPTIONS (*;"StyledText_t";ST Expressions Display Mode;
                    ST Values)
Else
    ST SET OPTIONS (*;"StyledText_t";ST Expressions Display Mode;
                    ST References)
End if

```

*See also:* [ST GET OPTIONS](#)

## 修正されたコマンド

### ST Get plain text

ST Get plain text ( { \* ; } object { ; refMode } ) → Text

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名 (文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時)、 または変数やフィールド (* 省略時)
refMode	倍長整数	→ テキスト内の参照を扱うモード
戻り値	テキスト	← タグを含まないテキスト

*Compatibility note* このコマンドは旧バージョンの 4D では OBJECT Get plain text と呼ばれていたものです。( [285P “改名されたコマンド”](#) を参照して下さい。)

[ST Get plain text](#) コマンドは、4D の v14 より、新たに *object* 引数で指定されたオブジェクト内で見つけた参照の返し方を指定する、任意の *refMode* 引数を受け取れるようになりました。

*refMode* 引数には、"Multistyle Text" テーマ内にある、以下の定数のどれか一つまたは組み合わせた定数を渡すことができます。:

定数 ( 値 )	説明
<u>ST References as spaces</u> (0)	それぞれの参照は、ノンブレイクスペース文字として返されます ( デフォルトの動作 )
<u>ST References as values</u> (1)	4D 式は参照していた値が返され、URL 参照は表示テキストが返されます。
<u>ST References as sources</u> (2)	4D 式は参照の原文が文字列として返され、URL 参照は表示テキストが返されます。
<u>ST URL as labels</u> (4)	「こちらのサイトまでどうぞ」といったような URL の表示ラベルが返されます ( フォームのデフォルトの機能 )。
<u>ST URL as links</u> (8)	"http://www.4d.com" のようにリンクが返されます。
<u>ST User links as labels</u> (16)	ユーザーリンクの表示ラベルが返されます ( フォームのデフォルトの機能 )。
<u>ST User links as links</u> (32)	ユーザーリンクの中身が返されます。
<u>ST Tags as plain text</u> (64)	タグのラベルがプレーンテキストとして返されます。例えば '
<u>ST Tags as XML code</u> (128)	タグの XML コードがプレーンテキストで返されます。例えば '
<u>ST Text displayed with 4D Expression values</u> (85)	評価済みの状態の 4D 式を含んだ、フォームに表示されているテキストを返します。既定の定数の組み合わせ 1+4+16+64 に相当します。
<u>ST Text displayed with 4D Expression sources</u> (86)	オリジナルの文字列の状態の 4D 式を含んだ、フォームに表示されているテキストを返します。既定の定数の組み合わせ 2+4+16+64 に相当します。

*Note* *refMode* 引数にどの値を渡しても ( どのモードで扱っても ) 標準テキストに変化はありません。言い換えると、*refMode* を使って違いが現れるのはテキストに参照が含まれていた場合のみです。

- ▶ "MyZone" というマルチスタイルエリアの中にある以下の文字列について考えます。:

```
<span>It is now <span style="-d4-ref:'Current time:C178'"> </span> <a href="http://www.4d.com">Go to the 4D site</a> or <span style="-d4-ref-user:'openW'">Open a window</span></span>
```

文字列は以下の様に表示されます。



以下のコードを実行した場合、:

```
$txt := ST Get plain text (*;"myArea";ST References as spaces)
// $txt = "It is now or " (spaces)
$txt := ST Get plain text (*;"myArea";ST 4D Expressions as values)
// $txt = "It is now 15:48:19 or "
$txt := ST Get plain text (*;"myArea";ST 4D Expressions as sources)
// $txt = "It is now Current time or "
$txt := ST Get plain text (*;"myArea";ST URL as links)
// $txt = "It is now http://www.4d.com or "
$txt := ST Get plain text (*;"myArea";ST Text displayed with 4D Expression values)
// $txt = "It is now 15:48:19 Go to the 4D site or Open a window"
$txt := ST Get plain text (*;"myArea";ST Text displayed with 4D Expression sources)
// $txt = "It is now Current time Go to 4D site or Open a window"
$txt := ST Get plain text(*;"myArea";ST User links as labels)
// $txt = "It is now or Open a window"
$txt := ST Get plain text(*;"myArea";ST User links as links)
// $txt = "It is now or openW"
```

## 行末の自動標準化

データベースで扱われるテキストがより多くのプラットフォームで互換性があることを保証するために、v14以降、4Dは自動的に行末を標準化し、単一文字 ('\r' (キャリッジリターン)) 分のスペースを確保するようにしました。この標準化は、マルチスタイルテキストまたは標準テキストを内包しているフォームオブジェクト (変数またはフィールド) まで適用されます。ネイティブでない行末、または複数の文字の組み合わせ ('\r\n' 等) は単一の文字 '\r' として認識されます。

XML 標準 (マルチスタイルテキストフォーマット) と適合するため、マルチスタイルテキストコマンドも、オブジェクトに関連付けられていないテキスト変数の行末を標準化することに注意して下さい。

この原理により、マルチスタイルテキストコマンドや、HIGHLIGHT TEXT のようなコマンドをマルチプラットフォームのコンテキストで簡単に使用できるようになります。しかしながら、異なるソースからのテキストを扱う処理をする場合には、このことを考慮に入れなければなりません。

## 定数を使用して選択範囲を指定

4D v14 では、選択範囲を自動的に指定するために *startSel* 引数と *endSel* 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります。:

定数 ( 値 )	説明
<u>ST Start highlight</u> (-1000)	オブジェクト内のカレントのテキストの選択範囲の、最初の文字を指定します。(オブジェクトシンタックスのみ)
<u>ST End highlight</u> (-1001)	オブジェクト内のカレントのテキストの選択範囲の、最後の文字を指定します。(オブジェクトシンタックスのみ)
<u>ST Start text</u> (1)	オブジェクト内のテキストの最初の文字を指定します。
<u>ST End text</u> (0)	オブジェクト内のテキストの最後の文字を指定します。

これらの定数は、既存のものを含め、マルチスタイルのテキストを管理するコマンド全てで使用することができます。: ST SET ATTRIBUTES, ST GET ATTRIBUTES, ST SET TEXT そして ST Get text です。(これらのコマンドは 4D v14 で改名されています。詳細に関しては、[285P “改名されたコマンド”](#) を参照して下さい)。

ST Start highlight と ST End highlight 定数は、オブジェクトシンタックスでのみ有効であるという点に留意して下さい (*object* 引数に \* 引数とオブジェクト名を渡す必要があります)。

- ▶ フォーム内に [Table\_1] というスタイル付テキストフィールドが表示されているとします。オブジェクトにはマルチスタイルプロパティがあり、"StyledText\_t" という名前がついています。選択されているテキストと、太字のスタイルに関する情報を取得したい場合を考えます。オブジェクト名を使うかフィールド参照を使用するかによって 2 通りのやりかたがあります。

- オブジェクト名を使用する方法:

```
$text:=ST Get text (*;"StyledText_t";ST Start highlight;ST End highlight)
ST GET ATTRIBUTES (*;"StyledText_t";ST Start highlight;ST End highlight;Attribute bold style;$bold)
```

- フィールド名を使用する方法 :

```
GET HIGHLIGHT ([Table_1]StyledText;$Begin_1;$End_1)
$text:=ST Get text ([Table_1]StyledText;$Begin_1;$End_1)
ST GET ATTRIBUTES ([Table_1]StyledText;$Begin_1;$End_1;Attribute bold
style;$bold)
```

## 改名されたコマンド

以下のコマンドは旧バージョンの 4D では「オブジェクトプロパティ」テーマ内に存在していたものです。4D の v14 では、これらは新しい「スタイル付きテキスト」テーマ内に移動され、「OBJECT」の代わりに「ST」接頭辞が付けられました。[ST Get plain text](#) コマンドを除き、これらのコマンドの動作そのものは変更されていません。

4D v14 での名前	以前の名前 (4D v13.x)
<a href="#">ST Get plain text</a>	OBJECT Get plain text
ST SET ATTRIBUTES	OBJECT SET STYLED TEXT ATTRIBUTES
ST GET ATTRIBUTES	OBJECT GET STYLED TEXT ATTRIBUTES
ST SET TEXT	OBJECT SET STYLED TEXT
ST Get text	OBJECT Get styled text
ST SET PLAIN TEXT	OBJECT SET PLAIN TEXT

## システムドキュメント

### COPY DOCUMENT

COPY DOCUMENT ( sourceName ; destinationName{; newName}{; \* } )

引数	型	説明
sourceName	文字列	→ コピーするファイルやフォルダーのパス名
destination-Name	文字列	→ ファイルやフォルダーコピーの配置先名またはパス名
newName	文字列	→ コピーしたファイルやフォルダーの名前
*	演算子	→ 存在する場合、既存のドキュメントを上書きする

**COPY DOCUMENT** コマンドは、複製されたドキュメントのコピー先の名前（ファイルまたはフォルダー）を指定する任意の *newName* 引数を取るようになりました。

この引数がファイルを複製するときには渡された場合には、*destinationName* 引数に渡された名前があった場合にそれを上書きします。

- ▶ The following examples illustrate how the new parameter works (examples under Windows):

```
COPY DOCUMENT( "folder1\\name1" ; "folder2\\")
// "folder2/name" ファイルを作成します
```

```
COPY DOCUMENT( "folder1\\name1" ; "folder2\\" ; "new")
// "folder2/new" ファイルを作成します
```

```
COPY DOCUMENT( "folder1\\name1" ; "folder2\\name2")
// "folder2/name2" ファイルを作成します
```

```
COPY DOCUMENT( "folder1\\name1" ; "folder2\\name2" ; "new")
// "folder2/new" ファイルを作成します (name2 は無視されます)
```

```
COPY DOCUMENT( "folder1\\" ; "folder2\\")
// "folder2/folder1/" フォルダを作成します
```

```
COPY DOCUMENT( "folder1\\" ; "folder2\\" ; "new")
// "folder2/new/" フォルダを作成します
```

## Document to text

Document to text( fileName {; charSet{; breakMode{}}) → Text

引数	型	説明
fileName	文字列	→ ドキュメント名またはドキュメントへのパス
charSet	テキスト   倍長整数	→ 文字コードを名前または番号で指定
breakMode	倍長整数	→ 改行モードを指定
戻り値	テキスト	← ドキュメントからのテキスト

**COPY DOCUMENT** コマンドを使用すると、ディスク上のファイルの内容を直接 4D のテキスト変数やテキストフィールドに取り込むことが出来ます。

*fileName* には、読み込みたいファイルのファイル名かファイルへのパスを渡します。ファイルがディスク上に存在しない場合は、エラーが発生します。*fileName* には、次の 3 つの渡し方ができます。:

- "myFile.txt" の様に、ファイル名のみを渡す方法。: この場合、指定するファイルはアプリケーションのストラクチャファイルと同じフォルダ内になければなりません。

- アプリケーションのストラクチャファイルからの相対パスを指定する方法。:Windows 環境では "\\docs\\myFile.txt" のように、Mac OS X 環境では ":docs:myFile.txt" のように指定します。
- 絶対パスを指定する方法。:Windows 環境では "c:\\app\\docs\\myFile.txt" のように、Mac OS X 環境では "MacHD:docs:myFile.txt" のように指定します。

*charSet* には、ファイルの中身を読みだす際に使用する文字コードを指定します。文字コードの一般的な名前 (“ISO-8859-1” や “UTF-8” など) を文字列として渡すこともできますし、MIBEnum ID を倍長整数として渡すこともできます。4D で使用できる文字コードの一覧については、4D のランゲージリファレンス内の CONVERT FROM TEXT コマンドの頁を参照して下さい。

指定したドキュメントが BOM(バイトオーダーマーク) を含んでいる場合は、*charSet* で指定された文字コードは無視され、BOM で指定された文字コードが優先されます。

指定したドキュメントに BOM がなく、かつ *charSet* 引数が省略されていた場合、4D 日本語版では Mac/Windows 両プラットフォームとも Shift\_JIS が使用されます。

*breakMode* 引数には、ドキュメント内の改行モードを指定する倍長整数を渡します。使用できる定数は以下の通りです (これらの定数は System Documents テーマ内にもあります)。

定数 ( 値 )	説明
<u>Document unchanged</u> (0)	何も処理をしません
<u>Document with native format</u> (1)	改行は OS のネイティブフォーマットに変換されます。Mac OS 環境では CR( キャリッジリターン ) に、Windows 環境では CRLF( キャリッジリターン+ラインフィード ) に変換されます。( デフォルトの動作 )
<u>Document with CRLF</u> (2)	改行は Windows フォーマット (CRLF、 キャリッジリターン+ラインフィード) へと変換されます。
<u>Document with CR</u> (3)	改行は Mac OS フォーマット (CR、 キャリッジリターン) へと変換されます。
<u>Document with LF</u> (4)	改行は Unix フォーマット (LF、 ラインフィード) へと変換されます。

*breakMode* 引数省略時には、改行コードは自動的にプラットフォームのネイティブフォーマット (1) になります。

*Note* これら二つの新コマンドでは OK 変数は更新されません。処理を完了できなかった場合にはエラーが発生します。このエラーは ON ERR CALL コマンドに実装されたメソッドによって割り込むことができます。

- ▶ 以下のタブ区切りのテキスト文書に対して、：

```
id name price vat
3 4D Tags 99 19.6
```

以下のコードを実行した場合、：

```
$Text:=Document to text("products.txt")
```

結果は以下のようになります。：

```
// $Text = "id\name\price\vat\r\n3\t4D Tags\t99 \t19.6"
// \t = tab
// \r = CR
```

*See also:* [TEXT TO DOCUMENT](#)

## TEXT TO DOCUMENT

TEXT TO DOCUMENT( fileName; text{ {}; charSet{ }; breakMode{ } })

引数	型	説明
fileName	文字列	→ ドキュメント名またはドキュメントへのパス
text	テキスト	→ ドキュメントに保存したいテキスト
charSet	テキスト   倍長 整数	→ 文字コードを名前または番号で指定
breakMode	倍長整数	→ 改行モードを指定

[TEXT TO DOCUMENT](#) コマンドを使用すると、*text* で指定したテキストをディスク上のファイルに直接書き込むことができます。

*fileName* には書き込みたいファイルのファイル名またはパスを渡します。存在しないファイル名の場合はコマンドが新規にファイルを作成します。既存のファイルを指定した場合には元の内容は消去された上で上書きされます。ただし、既存のファイルが既に開かれている場合には、中身がロックされ、エラーが発生します。

*fileName* では、次の3つの渡し方ができます。：

- "myFile.txt" のように、ファイル名のみを渡す方法。：この場合、ファイルはアプリケーションのストラクチャファイルと同じフォルダに存在しなければならぬか、存在しない場合同じフォルダに作成されます。

- アプリケーションのストラクチャファイルからの相対パスを指定する方法。:Windows 環境では "\\docs\\myFile.txt" のように、Mac OS X 環境では ":docs:myFile.txt" のように指定します。
- 絶対パスを指定する方法。:Windows 環境では "c:\\app\\docs\\myFile.txt" のように、Mac OS X 環境では "MacHD:docs:myFile.txt" のように指定します。

もしユーザーにファイル名や場所を指定させたい場合は、Open document コマンドか Create document コマンドを *Document* システム変数とともに使用して下さい。

*Note* デフォルトでは、このコマンドで作成されたドキュメントには拡張子はありません。拡張子をつける場合には *fileName* で拡張子を含めて指定するか、SET DOCUMENT TYPE コマンドを使用する必要があります。

*text* には、ディスクに書き込むテキストを渡します。リテラル定数か、4D のテキストフィールドかテキスト変数を指定することが出来ます。

*charSet*, には、ファイルの中身を書き出す際に使用する文字コードを指定します。文字コードの一般的な名前 (“ISO-8859-1” や “UTF-8” など) を文字列として渡すこともできますし、MIBEnum ID を倍長整数として渡すこともできます。4D で使用できる文字コードの一覧については、4D のランゲージリファレンス内の CONVERT FROM TEXT コマンドの頁を参照して下さい。

BOM( バイトオーダーマーク ) が文字コードに存在する場合、4D はそれをドキュメント内に挿入します。

文字コードを指定しない場合には、4D は "UTF\_8" が BOM とともに使用されます。

*breakMode* 引数には、ドキュメントに保存する前に改行モードを指定する倍長整数を渡します。使用できる定数は以下の通りです (これらの定数は System Documents テーマ内にもあります) 。 :

定数 ( 値 )	説明
<u>Document unchanged</u> (0)	何も処理をしません
<u>Document with native format</u> (1)	改行は OS のネイティブフォーマットに変換されます。Mac OS 環境では CR( キャリッジリターン ) に、Windows 環境では CRLF( キャリッジリターン + ラインフィード ) に変換されます。( デフォルトの動作 )
<u>Document with CRLF</u> (2)	改行は Windows フォーマット (CRLF、 キャリッジリターン + ラインフィード) へと変換されます。

<u>Document with CR</u> (3)	改行は Mac OS フォーマット (CR、キャリッジリターン) へと変換されます。
<u>Document with LF</u> (4)	改行は Unix フォーマット (LF、ラインフィード) へと変換されます。

*breakMode* 引数省略時には、改行コードは自動的にプラットフォームのネイティブフォーマット (1) で処理されます。

*Note* デフォルトでは、このコマンドで作成されたドキュメントには拡張子はありません。拡張子をつける場合には *fileName* で拡張子を含めて指定するか、SET DOCUMENT TYPE コマンドを使用する必要があります。

- ▶ このコマンドの使用例は以下のようになります。:

```
TEXT TO DOCUMENT ("myTest.txt";"This is a test")
TEXT TO DOCUMENT ("myTest.xml";"This is a test")
```

- ▶ ユーザーにファイルの作成場所を指定させる場合には以下のような例になります。:

```
$MyTextVar:="This is a test"
ON ERR CALL( "IO ERROR HANDLER")
$vhDocRef := Create document("")
// ドキュメントは ".txt" 拡張子で保存されます。
// この場合、.txt 拡張子は常にファイル名に追加され、変更することはできません。
If(OK=1) // もしドキュメントが正常に作成された場合は、
  CLOSE DOCUMENT($vhDocRef) // ドキュメントを閉じる
  TEXT TO DOCUMENT(Document; $MyTextVar)
  // ドキュメントを作成
Else
  // エラーマネージメント
End if
```

*See also:* [COPY DOCUMENT](#)

## システム環境

### 変更されたコマンド

#### FONT LIST

```
FONT LIST(fonts {; listType | *})
```

引数	型	説明
fonts	テキスト配列	→ フォント名の配列

*listType* | \* 倍長整数 | \* → 取得したいフォントリスト、または \* を渡すことによって Mac OS においてフォント名を返す

**FONT LIST** コマンドを使用すると、様々な種類のフォントリストを取得することができます。そのためには、*listType* 引数に、新しい "Font Type List" 定数を渡します。使用できる定数は以下の通りです。

定数 ( 値 )	説明
<u>System fonts</u> (0)	<i>fonts</i> に全てのシステムフォントがリストとして返されます。(旧バージョンでの動作と同様です。) <i>listType</i> 省略時のデフォルトの動作です。
<u>Favorite fonts</u> (1)	<i>fonts</i> にお気に入りのフォント (マシン上で最もよく使われているフォント) のリストが返されます。
<u>Recent fonts</u> (2)	<i>fonts</i> に最近使われたフォント (4D のセッション中に使用されたフォント) のリストが返されます。このリストは特に <b>マルチスタイルエリア</b> で使用されます。

\* 演算子は以前のバージョンに引き続き、同様にご使用いただけます。

- ▶ 最近使用したフォントのリストを取得したい場合、以下のように記述します。:

**FONT LIST**(\$arrFonts;Recent fonts)

## Font name

Font name ( fontNumber ) → Function result

*Compatibility note* このコマンドは廃止予定であり、今後使用されるべきではありません。v14 ではこのコマンドは互換性のために残されており、今後のバージョンにおいてサポートされなくなります。

## Font number

Font number ( fontName ) → Function result

*Compatibility note* このコマンドは廃止予定であり、今後使用されるべきではありません。v14 ではこのコマンドは互換性のために残されており、今後のバージョンにおいてサポートされなくなります。

## 新コマンド

### OPEN COLOR PICKER

OPEN COLOR PICKER {(textOrBackground)}

引数	型	説明
textOrBackground	倍長整数	→ 0 または省略時 = テキストのカラーを指定 1 = テキストの背景色を指定

**OPEN COLOR PICKER** コマンドは、システムのカラーピッカーのダイアログボックスを表示します。

*Note* これは Windows 環境下ではモーダルダイアログですが、OS X ではモーダルではありません。

オブジェクトの「**ピッカーの仕様を許可**」オプションがチェックされていた場合、ユーザーがダイアログボックスから色を選択して確定させると、その色がフォーカスされたオブジェクト内の、選択されたテキストに適用されます（詳細は 59P “[カラーピッカーとフォントピッカーの使用を許可](#)” を参照して下さい）。

*textOrBackground* 引数に 0 を渡すか省略した場合、選択された色はテキストに適用されます。*textOrBackground* 引数に 1 を渡した場合、選択された色は背景色に適用されます。

カラーが変更された場合、フォームイベントの On After Edit がこのオブジェクトに発生します。

*See also:* [OPEN FONT PICKER](#)

### OPEN FONT PICKER

OPEN FONT PICKER

引数	型	説明
		このコマンドには指定できる引数がありません。

**OPEN FONT PICKER** コマンドはシステムのフォントピッカーダイアログを表示します。

*Note* これは Windows 環境下ではモーダルダイアログですが、OS X ではモーダルではありません。

オブジェクトの「**ピッカーの仕様を許可**」オプションがチェックされていた場合、ユーザーがダイアログボックスからフォントを選択して確定させると、そのフォントがフォーカスされているオブジェクト内の選択

されたテキストに適用されます。(詳細は 59P “カラーピッカーとフォントピッカーの使用を許可” を参照して下さい)。

フォントが変更された場合、フォームイベントの On After Edit がこのオブジェクトに発生します。

*See also:* [OPEN COLOR PICKER](#)

## SET RECENT FONTS

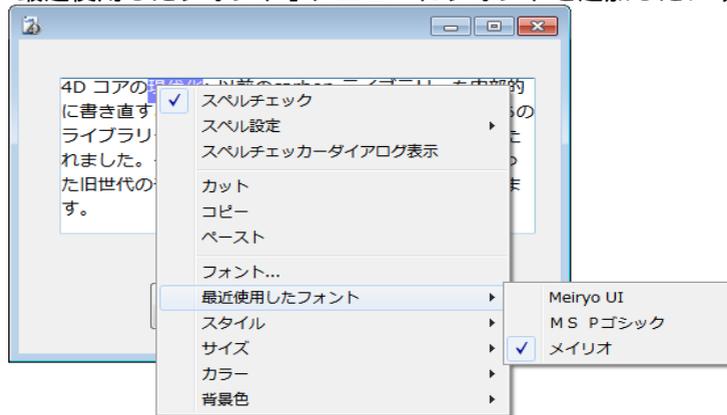
SET RECENT FONTS(fontsArray)

引数	型	説明
fontsArray	テキスト配列	→ フォント名の配列

**SET RECENT FONTS** コマンドを使用すると、コンテキストメニューの中の「最近使用したフォント」のリストを変更することができます。

このメニューにはセッション中、最後に選択したフォントも含まれます。これは特に、[マルチスタイルエリア](#) で使用されます。

- ▶ 「最近使用したフォント」メニューにフォントを追加したい場合、



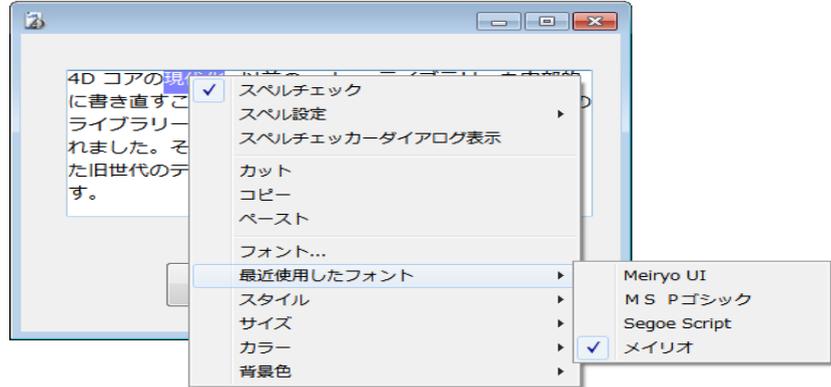
以下のコードを実行してください。:

```

ARRAY TEXT($arrRecent;0)
FONT LIST($arrRecent;2)
APPEND TO ARRAY($arrRecent;"Segoe Script")
SET RECENT FONTS($arrRecent)

```

その結果メニューはこうなります。:



## FONT LIST

# ツール

## Generate digest

Generate digest ( param ; algorithm ) → Function result

*algorithm* 引数において新しい定数が使用できるようになりました。

定数 ( 値 )	説明
4D digest (2)	4D の内部アルゴリズムを使用

この新しい定数によって 4D 内部で使用されているものと同じアルゴリズムを使用し、4D パスワードの有用性をチェックできるようになります。この機構は特に [On REST Authentication](#) データベースメソッドを使用する場合などに有効です。

## GET ACTIVITY SNAPSHOT

GET ACTIVITY SNAPSHOT ( { \* ; } arrActivities ) | ( { \* ; } arrUUID ; arrStart ; arrDuration ; arrInfo { ; arrSubOp } )

引数	型	説明
*	演算子	→ 渡した場合、サーバーの状態を取得
arrActivities	オブジェクト配列	← オペレーションの詳細な情報
arrUUID	テキスト配列	← オペレーションの UUID
arrStart	テキスト配列	← オペレーションの開始時刻
arrDuration	倍長整数配列	← オペレーションの所要時間 ( ミリ秒単位 )
arrInfo	テキスト配列	← オペレーションを説明するラベル
arrSubOp	オブジェクト配列	← subOperations プロパティ

新しい **GET ACTIVITY SNAPSHOT** コマンドは、4D data 上で進行中の操作の詳細を記載した配列を返します。これらの操作は、通常進捗ウィンドウが表示されます。

この情報は、もっとも時間のかかっているオペレーションまたは頻繁に使用されているオペレーション（ライトキャッシュやフォーミュラの実行など）のスナップショットを取るのにつかわれます。

---

*Note* **GET ACTIVITY SNAPSHOT** コマンドによって返された情報は、4D Server のリアルタイムモニターのページに表示されているデータと同じです。（33P “リアルタイムモニター” を参照して下さい。）

---

このコマンドではシンタックスを使用することができます。:

- オブジェクト配列のみを使用するシンタックス XXX
- 複数の配列を使用するシンタックス

デフォルトとして、**GET ACTIVITY SNAPSHOT** はローカルに実行されている操作のみ処理します (4D シングルユーザー、4D サーバーまたはリモートモードの 4D など)。

それに加え、リモートモードの 4D では、サーバーで実行されている操作のスナップショットを取ることもできます。そのためには第一引数として、\* 演算子を渡して下さい。サーバーのデータはローカルに復元されません。

\* 演算子は、コマンドが 4D サーバーや 4D シングルユーザー上で実行された場合には無視されます。

- **第一シンタックス** : **GET ACTIVITY SNAPSHOT** ( { \* ; } arrActivities )

この記法では、リアルタイムモニターの全てのオペレーションが構造化された形式で 4D オブジェクト配列 ( *arrActivities* 引数で指定 ) に返されます。配列の各要素は、以下の様に構築されたオブジェクトになっています。:

```
progressIndicator :{
  ID: string
  StartTime: 12:15:20:0054
  Duration: 501
  Title: String,
  sub-operations: [
    {
      Title: string,
      Start time: string,
      Duration: number (milliseconds)
    },
  ],
}
```

```

    {
      Title: string,
      Start time: string,
      Duration: number (milliseconds)
    }
  ]
}

```

- **第二シンタックス**: GET ACTIVITY SNAPSHOT ( { \* ; } arrUUID ; arrStart ; arrDuration ; arrInfo { ; arrSubOp } )

この記法では、リアルタイムモニターの全てのオペレーションが、同期した複数の配列に返されます ( オペレーションがあるたび、全ての配列に要素が追加されていきます )。返される配列は以下の通りです。:

- *arrUUID*: それぞれのオペレーションに対する UUID が保存されます。
  - *arrStart*: それぞれのオペレーションの開始時刻が、"dd/mm/yyyy - hh:mm:ss" というフォーマットで格納されます。(テキスト)
  - *arrDuration*: それぞれのオペレーションの所要時間がミリ秒単位で格納されます。
  - *arrInfo*: それぞれのオペレーションを説明するラベルが保存されます。
  - *arrSubOp* ( 任意 ): この配列の要素には、"subOperations" プロパティを格納するオブジェクトが入っています。このプロパティの値はカレントオペレーションのサブ処理を全て含んだ object 配列となっています。カレントのオペレーションにサブ処理が何もない場合、*subOperations* の値は空の配列となります。
- ▶ 4D か 4Dサーバーにおける個別のプロセスにおいて以下のメソッドを実行した場合、下図のようにオペレーションのスナップショットを返します。:

```

ARRAY TEXT (arrUUID;0)
ARRAY TEXT (arrStart;0)
ARRAY LONGINT (arrDuration;0)
ARRAY TEXT (arrInfo;0)

```

#### Repeat

```

GET ACTIVITY SNAPSHOT (arrUUID;arrStart;arrDuration;arrInfo)
If (Size of array (arrUUID)>0)
  TRACE // デバッガを呼び出し
End if
Until (False) // 無限ループ

```

- 以下の様な配列が返されます。:

Expression	Value
arrUUID	6 elements
arrUUID	0
arrUUID(0)	""
arrUUID(1)	"1858EB64D281A7429E2012CEEE78499A"
arrUUID(2)	"078BF19381C0184EA26F1D4235A89CE6"
arrUUID(3)	"AE18AFA94268424FBFB157554751E05"
arrUUID(4)	"715C5B028868E44BAC4062AB08507601"
arrUUID(5)	"BEF371C7CFF45946A6B5FE3488692BD4"
arrUUID(6)	"5C51ED352AF1344AA3895D8236D9EC25"
arrStart	6 elements
arrStart	0
arrStart(0)	""
arrStart(1)	"12/3/2013 - 11:32:34"
arrStart(2)	"12/3/2013 - 11:32:35"
arrStart(3)	"12/3/2013 - 11:32:34"
arrStart(4)	"12/3/2013 - 11:32:34"
arrStart(5)	"12/3/2013 - 11:32:34"
arrStart(6)	"12/3/2013 - 11:32:35"
arrDuration	6 elements
arrDuration	0
arrDuration(0)	0
arrDuration(1)	5747
arrDuration(2)	5388
arrDuration(3)	5752
arrDuration(4)	5653
arrDuration(5)	5959
arrDuration(6)	5466
arrInfo	6 elements
arrInfo	0
arrInfo(0)	""
arrInfo(1)	"Array to selection: 9 of 100"
arrInfo(2)	"Loading data"
arrInfo(3)	"Array to selection: 7 of 100"
arrInfo(4)	"Sequential searching on Companies: 11 of 98167 records"
arrInfo(5)	"Deleting records: 6 of 10"
arrInfo(6)	"Sequential searching on Companies: 13 of 98167 records"

## ユーザーインターフェース

### SHOW TOOL BAR, HIDE TOOL BAR

4D の v14 では、このコマンドは何の効果も持たないように変更されました。

これまでの 4D では、これらのコマンドはアプリケーションモード時の自動ツールバーを管理していましたが、v14 ではサポートされていません。

### Tool bar height

その変更に伴い、このコマンドはアプリケーションモードで呼び出された時には常に 0 を返すように変更されています。

## ユーザー & グループ

### Validate password

Validate password (*user*; password {; *digest*} ) → Text

引数	型	説明
<i>user</i>	Longint   String	→ ユニークなユーザ ID またはユーザ名
password	String	→ パスワード
<i>digest</i>	Boolean	→ ダイジェストパスワード = True プレーンテキストパスワード (デフォルト) = False
Function result	Boolean	← True = 有効なパスワード False = 無効なパスワード

4D v14 では、[Validate password](#) コマンドに二点の変更があります。

- 第一引数 (*user*) に直接ユーザ名 (文字列) を渡せるようになりました。
- 任意の第三引数 *digest* が追加されました。この引数は *password* 引数がプレーンテキストか、ハッシュドパスワードであるか (ダイジェストモード) を指定します。
  - True を渡した場合、*password* はハッシュドパスワードを含むということを表します (ダイジェストモード)。
  - False を渡すかこの引数を省略した場合、*password* はプレーンテキストのパスワードを含むという事を表します。

これらの変更は、特に新しい [On REST Authentication](#) データベースメソッドを使用している場合に有効です。

- ▶ [On REST Authentication](#) データベースメソッドで、(データベースの 4D ユーザを使用して) 接続リクエストをテストしたい場合、以下のように記述します。

```
$0 := Validate password($1; $2; $3)
```

## Web Area

### WA Evaluate JavaScript

WA Evaluate JavaScript ( { \* ; } object; jsCode {; *type*} ) → Function result

引数	型	説明
*	Operator	→ 指定時、object はオブジェクト名 (文字列) 省略時、object は変数

object	Form object	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
jsCode	文字列	→	JavaScript コード
type	倍長整数	→	結果を変換する型を指定
戻り値	テキスト、実数、日付、時間、ブール、ポインター、ランゲージオブジェクト	←	評価の結果

*Compatibility note* このコマンドは以前のバージョンの 4D では WA Execute JavaScript と呼ばれていました。

4D v14 では、[WA Evaluate JavaScript](#) コマンドは文字列以外の型の結果を返すことが出来るようになりました。

**重要：**この新機能は Web エリアが統合された **Web Kit** をレンダリングエンジンとして使用している場合に限り使用可能です。

特に指定しなければ、コマンドは値を文字列として返します。任意の *type* 引数を用いることによって、戻ってくる値の型を指定することができます。使用できる定数の一覧は以下の通りで、"Field and Variable Types" テーマ内にあります。(4D v14 では新しい定数が追加されています。):

Constant (value)	Comments
<a href="#">Is real</a> (1)	
<a href="#">Is text</a> (2)	
<a href="#">Is date</a> (4)	
<a href="#">Is boolean</a> (6)	
<a href="#">Is longInt</a> (9)	
<a href="#">Is integer</a> (8)	
<a href="#">Is integer 64 bits</a> (25)	
<a href="#">Is string var</a> (24)	
<a href="#">Is time</a> (11)	ミリ秒で表現された数字
<a href="#">Is object</a> (38)	4D の v14 からの新定数
<a href="#">Object array</a> (39)	4D の v14 からの新定数
<a href="#">Is JSON null</a> (255)	4D の v14 からの新定数

- ▶ 以下に、受け取った値の変換を含む例をいくつか紹介します。
  - HTML ファイル内にある JavaScript ファンクションの例を考えます。:
 

```
<!DOCTYPE html>
<html>
```

```

<head>
  <script>
    function evalLong(){
      return 123;
    }
    function evalText(){
      return "456";
    }
    function evalObject(){
      return {a:1,b:"hello world"};
    }
    function evalDate(){
      return new Date();
    }
  </script>
</head>
<body>
  TEST PAGE
</body>
</html>

```

- 4D フォームメソッドでは以下の様に記述します。:

#### If (Form event=On Load)

**WA OPEN URL** (\*;"Web Area";"C:\\myDatabase\\index.html")

#### End if

- その後、4D からの JavaScript コードを以下の様に評価します。:

```

$Eval1:=WA Evaluate JavaScript (*;"Web Area";"evalLong()";Is longInt)
// $Eval1 = 123
// $Eval1 = 型が省略されていた場合は "123"
$Eval2:=WA Evaluate JavaScript (*;"Web Area";"evalText()";Is string var)
// $Eval2 = "456"
$Eval3:=WA Evaluate JavaScript (*;"Web Area";"evalObject()";Is object)
// $Eval3 = {"a":1,"b":"hello world"}
$Eval4:=WA Evaluate JavaScript (*;"Web Area";"evalDate()";Is date)
// $Eval4 = 06/21/13
// $Eval4 は型が省略されていた場合は "2013-06-21T14:45:09.694Z"

```

## WA EXECUTE JAVASCRIPT FUNCTION

WA EXECUTE JAVASCRIPT FUNCTION ( { \* ; } object; jsFunction ; result / \* { ;  
param } { ; param2 ; ... ; paramN } )

引数	型	説明
*	演算子	→ 指定時、object はオブジェクト名 (文字列) 省略時、object は変数

object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
jsFunction	文字列	→	実行される JavaScript 関数の名前
result   *	変数	→	関数が結果を返さない場合 *
		←	または関数結果 (返される場合)
param	文字列、数値、日付、オブジェクト	→	関数に渡す引数

**WA EXECUTE JAVASCRIPT FUNCTION** コマンドは、入力・出力、どちらの引数においても文字列以外の型もサポートするようになりました。文字列の他に数値、日付、そしてオブジェクトの型が使用できます。

**重要**：この新機能は Web エリアが統合された Web Kit をレンダリングエンジンとして使用している場合に限り使用可能です。

- ▶ 以下の例において、"getCustomerInfo" という JavaScript ファンクションは番号 ID を引数として受け取り、結果をオブジェクトとして返します。：

```
C_OBJECT ($Result)
```

```
C_LONGINT ($ID)
```

```
$ID:=1000
```

```
WA EXECUTE JAVASCRIPT FUNCTION (*,"WA";"getCustomerInfo";$Result;$ID)
```

## WA SET PREFERENCE

```
WA SET PREFERENCE ( { * ; } object ; selector ; value )
```

**WA SET PREFERENCE** コマンドは 4D v14 から *selector* 引数に新しい定数を受け取ることが出来るようになりました。：

定数 ( 値 )	説明
<code>wa enable Web inspector (100)</code>	Web エリア内での Web インспекタの表示を許可します。

デバッガを Web エリアの中で表示できるようになったのも、4D v14 の大きな特徴の一つです。詳細に関しては、95P “Web インспекターへのアクセス” を参照して下さい。

## Web サーバー

### WEB SEND TEXT

```
WEB SEND TEXT ( htmlText { ; type } )
```

引数	型	説明
----	---	----

htmlText	テキスト	→ Web ブラウザに送られる HTML テキストフィールドまたは変数
type	テキスト	→ <i>MIME 型</i>

**WEB SEND TEXT** コマンドには 4D v14 より、*type* 引数という新しい任意のテキスト型の引数を渡せるようになりました。

---

*Note* この引数は以前は使用できたブール型の *noContext* 引数にとって変わるものです。 *noContext* 引数は 4D v12 より廃止予定でした。

---

*type* 引数には、送られるテキストの MIME 型を指定することができます。省略時にはデフォルトとして "テキスト /html" 型が使用されます。これは旧バージョンの 4D においても同様です。

MIME 型のより詳細な情報に関しては、WEB SEND BLOB コマンドの詳細を参照して下さい。 .

## Web サービス (クライアント)

4D v14 では、SOAP 1.2 に対するサポートが拡張されています。Web サービスにはサーバー証明書もつきました。( [HTTP SET CERTIFICATES FOLDER](#) を参照して下さい)。

### WEB SERVICE CALL

WEB SERVICE CALL ( accessURL ; soapAction ; methodName ; nameSpace { ; complexType { ; \* } } )

コマンドがサーバー証明書を使用しようとしたときにその証明書が有効でない ( 失効しているか、無効である ) とき、OK システム変数は 0 に設定され、901 エラー "*Server certificate invalid*" が返されます。このエラーは ON ERR CALL コマンドに実装されたメソッドによって割り込むことができます。

### WEB SERVICE Get info

WEB SERVICE Get info ( infoType ) → Function result

このコマンドを使用するときに SOAP プロトコルから返されたエラーのメッセージが一部変更されています。 :

- 「クライアント側のエラー」は改名されて「『送信側』のエラー」になりました。
- 「サーバー側のエラー」は改名されて「『送信側』のエラー」になりました。

# ウィンドウ

- Window types (OS X)** v14 では、Mac OS X でのウィンドウ管理が変更になりました。:
- 通常はコンポジットモードが使用されます。
  - それに加えて、新しい「フルスクリーンモード」というオプションが追加されました。

**コンポジットモード** v14 では、全てのウィンドウがコンポジットモードで作成されます。  
(20P “[ノンコンポジットモードのウィンドウ \(Mac OS\)](#)” も参照して下さい。)

結果的に、“Open Window” と “Open Form Window” テーマで使用されていた Compositing mode と Compositing Mode form window という定数には “\_O\_” 接頭辞 がつき、サポートされないこととなりました。

## Full screen button

OS X 環境下の v14 では、ドキュメントタイプのウィンドウにおいて新しいオプション「フルスクリーンモード」が追加されています。これを使用すると、ウィンドウの右上隅に「全画面ボタン」が追加されます。



全画面への切り替えスイッチのアイコン

ユーザーがこのアイコンをクリックすると、ウィンドウはフルスクリーンモードになり、メインツールバーが自動的に隠されます。

このオプションを使用するには Has full screen mode Mac という定数を Open window コマンド、Open form window コマンド、Open external window コマンドにて *type* の引数に渡します。

- ▶ OS X 環境下でフルスクリーンボタンを持ったウィンドウを作成するには:

```
$win := Open form window ([Interface];"User_Choice";Plain form window + Has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

*Note* Windows 環境下ではこのオプションは何もしません。

## 改名された定数

技術的推移に伴い、以下の定数は 4D v14 では改名されています。:

4D v14 での名前	4D v13.x での以前の名前	詳細
_O_qr 4D Chart area	qr 4D Chart area	4D Chart が廃止になったことによる改名 (詳細は <a href="#">20P “削除された機能”</a> を参照のこと)

## 4D Internet Commands

4D IC SMTP コマンドは 4D v14 にてアップデートされ以下の様な点が改善されました。:

- デフォルトで Unicode が使用できるようになりました
- ID と HTML の管理
- 新しい [SMTP\\_MessageID](#) コマンド

*Note* 4D Internet Commands プラグインは 64-bit 版の v14 においても供給されています。詳細に関しては、[16P “Windows 環境下における 64-bit プラグインに対するサポート”](#) を参照して下さい。

## Unicode の使用

4DIC の全ての SMTP コマンドにおいて、*subject* と *body* フィールドにてデフォルトで UTF-8 が使用されるようになりました。今日、ほぼすべての E メールクライアントはこの文字コードに対応しています。

この新機能により、SMTP コマンドの使用が簡単になり、SMTP\_Charset や SMTP\_SetPrefs と行ったコマンドを使用しなくてもよくなります。

## SMTP\_Attachment

SMTP\_Attachment ( smtp\_ID ; fileName ; encodeType ; deleteOption ; attachmentID ) → Function result

引数	型	説明
smtp_ID	倍長整数	→ メッセージ参照
fileName	テキスト	→ 添付するファイル名
encodeType	整数	→ 0 = エンコードなし ( データフォークのみ送信 ) ±1 = BinHex ±2 = Base64 ( データフォークのみ送信 ) ±3 = AppleSingle、±4 = AppleDouble ±5 = AppleSingle と Base64 ±6 = AppleDouble と Base64 ±7 = UUEncode

<code>deleteOption</code>	整数	→ 0 = 既存のリストに追加 1 = 全ての添付を <i>filename</i> で置き換え 2 = この添付を削除
<code>attachmentID</code>	テキスト	→ 添付の ID (HTML メッセージの場合のみ)
戻り値	整数	← エラーコード

**SMTP\_Attachment** コマンドでは `attachmentID` という新しい引数を受け取るようになりました。これは添付ファイルを、メッセージ本文内にある HTML タグ `<img src=\"cid:ID\">` で定義された参照と関連付けための ID です。

これにより、添付ファイルの中身 (例えばピクチャなど) を E メールクライアント上のメッセージ内に表示することができるようになりました。

この機能は HTML メッセージにおいてのみサポートされています。また最終的に表示される結果は E メールクライアントごとに異なる可能性があることに注意して下さい。

- ▶ ピクチャを内包する HTML メッセージを送信する場合を考えます。:

```

$error:=$SMTP_New ($smtp_id)
$error:=$SMTP_Host ($smtp_id;"smtp.gmail.com")
$error:=$SMTP_From ($smtp_id;"henry@gmail.com")
$error:=$SMTP_ReplyTo ($smtp_id;"replies@gmail.com")
$error:=$SMTP_Subject ($smtp_id;"HTML Test & picture included")
$error:=$SMTP_To ($smtp_id;"john@4d.com";1)
$error:=$SMTP_Body ($smtp_id;"<html><B><I>Hello world in bold!</I>
</B> <img src=\"cid:myID123\">(normal text)</HTML>";4)
$error:=$SMTP_Attachment ($smtp_id;"c:\\temp\\tulips.jpg";2;0;
"myID123")
$error:=$SMTP_Auth ($smtp_id;"henry@gmail.com";"*****")
$error:=$SMTP_Send ($smtp_id;1)
$error:=$SMTP_Clear ($smtp_id)

```

## SMTP\_Body

SMTP\_Body ( smtp\_ID ; msgBody ; option ) → Function result

引数	型	説明
<code>smtp_ID</code>	倍長整数	→ メッセージ参照
<code>msgBody</code>	テキスト	→ メッセージ本文 (UTF-8 をデフォルトで使用)
<code>option</code>	整数	→ 0 = 置き換え (msgBody が空でない場合) 1 = 削除、2 = 追加、4 = HTML テキスト (デフォルトでは標準テキストを使用)
Function result	整数	← エラーコード

`SMTP_Body` コマンドでは、4D v14 より、メッセージを HTML フォーマットで送信することが出来るようになりました。そのためには、`option` 引数において値 4 を渡すだけです。この値を省略した場合、メッセージはデフォルトで標準テキストを使用して送信されます。

HTML でメッセージを送信するのと置き換えのオプションを組み合わせる場合、二つの値を足した値を渡します。例えば、メッセージを削除して HTML テキストで送信する場合には、1+4 で 5 を `option` 引数に渡します。

それに加え、4D v14 では、メッセージ本文 (`msgBody`) はデフォルトで UTF-8 を使用してエンコードされます。

*See also:* [SMTP\\_Subject](#)

## SMTP\_MessageID

`SMTP_MessageID ( smtp_ID ; message_ID; option )` → Function result

引数	型	説明
<code>smtp_ID</code>	倍長整数	→ メッセージ参照
<code>message_ID</code>	テキスト	→ メッセージの固有の ID
<code>option</code>	整数	→ 0 = 追加、1 = 置き換え、2 = 削除
戻り値	整数	← エラーコード

新しい `SMTP_MessageID` コマンドは、`smtp_ID` で指定されたメッセージのヘッダーに "Message-Id" フィールドを作成します。この固有の ID はもっぱらフォーラムやメーリングリストにおいて使用されます。一般的に、メールサーバーは送信するメッセージには自動的にこのヘッダーを追加します。このコマンドを使用することによってその中身を定義することができます。

`smtp_ID` 引数には `SMTP_New` コマンドを使用して作成したEメールのIDを渡します。

`message_ID` 引数には、メッセージに関連付けたい ID を渡します。渡す内容に関しては通常特に制限はありませんが、一般的には、"(文字または数字)@(ドメイン名)" という書式で書かれることが多いです (例えば "abcdef.123456@4d.com" など)。ただし、Gmail などの一部のメールサーバーでは、カスタムのメッセージ ID ヘッダーを認識せず、上記の書式に沿っていない場合には置き換える、という点に注意して下さい。

`option` 引数では、既存の `message_ID` 引数を引き続き使用するかそれを削除するかを指定することができます。:

- 0 (省略時のデフォルト値) を渡した場合、message\_ID 引数の内容は既存の内容に追加されます。
  - 1 を渡した場合、message\_ID 引数の内容で既存の内容を置き換えます。
  - 2 を渡した場合、既存の内容はメッセージから削除されます。
- ▶ 以下の例では、[Admin] テーブル内のそれぞれのレコードに対して固有の "message\_ID" ヘッダーを持ったメッセージが送信されます。:

```

$error:=SMTP_New ($smtp_id)
$error:=SMTP_Host ($smtp_id;"infoserv.com")
$error:=SMTP_From ($smtp_id;"info@infoserv.com")
$error:=SMTP_Subject ($smtp_id;"General statistics")
FIRST RECORD ([Admins])
For ($i;1;Records in selection ([Admin]))
    $error:=SMTP_Body ($smtp_id;$Stats)
    $error:=SMTP_To ($smtp_id;[Admin]Email;1)
    // Replaces the"A" header with a new value
    $error:=SMTP_MessageID ($smtp_id;[Admin]ID+"@infoserv.com";1)
    // Use of the admin ID
    $error:=SMTP_Send ($smtp_id)
    NEXT RECORD ([Admin])
End for
$error:=SMTP_Clear ($smtp_id)

```

## SMTP\_QuickSend

SMTP\_QuickSend ( hostName ; msgFrom ; msgTo ; subject ; message { ; sessionParam } { ; port } { ; userName ; password } ) → Function result

引数	型	説明
hostName	文字列	→ ホスト名または IP アドレス
msgFrom	テキスト	→ 送信元 MailAddress または AddressList
msgTo	テキスト	→ 送信先 MailAddress または AddressList
subject	テキスト	→ 件名 ( <i>UTF-8 をデフォルトで使用</i> )
message	テキスト	→ メッセージ本文 ( <i>UTF-8 をデフォルトで使用</i> )
sessionParam	倍長整数	→ 0 または省略 = SSL を使用しないが、変更可 1 = SSL を使用 2 = SSL を絶対使用しない (変更不可) 4 = HTML テキストを SSL を使用せずに送信 5 = HTML テキストを SSL を使用して送信
port	倍長整数	→ 使用するポート番号
userName	テキスト	→ 認証に使用するユーザー名
password	テキスト	→ 認証に使用するパスワード
戻り値	整数	← エラーコード

`SMTP_QuickSend` コマンドは、4D v14 より HTML フォーマットでメッセージを作成・送信できるようになりました。そのためには、`sessionParam` 引数において以下の定数を渡すだけです。:

- 4 = HTML フォーマットのメッセージをスタンダードモードで送信 (SSL は使用しない)
- 5 = HTML フォーマットのメッセージを SSL モードで送信

これに加え、4D v14 ではデフォルトとして、`message` 本文と `subject` が UTF-8 でエンコードされるようになりました。

- ▶ メッセージを、SSL を使用して HTML で送信する場合を考えます。:

```
$Host:="smtp.gmail.com"  
$ToAddress:="john@4d.com"  
$FromAddress:="harry@gmail.com"  
$Subject:="HTML Message"  
$Message:="Let's meet at <b>Joe's Coffee Shop</b>!"  
$Param:=5 // HTML with SSL  
$Port:=465 // SSL port of gmail  
$User:="harry@gmail.com"  
$Password:="xyz&!&@"  
$Error:=SMTP_QuickSend($Host;$FromAddress;$ToAddress;$Subject;$Message;$Param;$Port;$User;$Password)
```

## SMTP\_Subject

`SMTP_Subject ( smtp_ID ; subject ; option )` → Function result

4D v14 ではデフォルトとして、`subject` 引数で指定するメッセージの件名は UTF-8(Unicode) でエンコードされます。

*See also:* [SMTP\\_Body](#)

## 4D Pack

新機能などの技術的な理由により、4D Pack v14 ではいくつかのコマンドが削除されています。

---

*Note* 4D Pack プラグインは 64-bit 版の 4D v14 においてもご利用いただけます。詳細な情報に関しては、[16P “Windows 環境下における 64-bit プラグインに対するサポート”](#) を参照して下さい。

---

## 4D Pack から削除されたコマンド

4D Pack v14 で削除されたコマンドと推奨されるそれらの代替案は以下の一覧表にまとめてあります。:

削除されたコマンド	推奨される代替案
AP AVAILABLE MEMORY	<a href="#">GET MEMORY STATISTICS</a> (v14において修正あり)
AP CLOSE HELP	廃止予定のコマンド - Windows Vista 以降、Windows ヘルプアプリケーション (WinHlp32.exe) は Windows に組み込まれていません。
AP HELP INDEX	
AP HELP ON HELP	
AP HELP ON KEY	
AP Create method	<a href="#">METHOD SET CODE</a> ("myMeth";vCode;*)
AP Modify method	<a href="#">METHOD SET ATTRIBUTE</a> ("myMeth";vInvisible;2;v4DAction;3;vWebService;4;vWSDL;5;vExported;7;vSQL;8;vRemote;1024;vFolderName;*)
AP Does method exist	<a href="#">METHOD GET NAMES</a> (\$arrNames;"myMeth") \$exists:=( <b>Size of array</b> (\$arrNames)>0) // -> メソッドが存在する場合は True
AP Get picture type	"ピクチャー" テーマ内のコマンド
AP Get templates	廃止予定のコマンド
_AP External clock	新しい <a href="#">TimePicker Widget</a> (4D v14)
AP SET CLOCK	
AP Rect Dragger	<a href="#">SET DRAG ICON</a> (4D v14)
AP Timestamp to GMT	\$date:= <b>String (Current date;ISO Date GMT; Current time)</b> // 戻り値は e.g. "2013-05-06T12:19:23Z" のようになります
AP SET PARAM	廃止予定のコマンド - これらのコマンドの第二引数は使用されていません。
AP GET PARAM	

## TimePicker Widget

TimePicker ウィジェットには時間データを表示するための新しいオブジェクトが用意されています。このウィジェットを使用することによってフォーム内に動的な時計を表示させることなどが出来るようになります。

二つの新しいオブジェクトが使用可能です。:

- "アナログ時計" (*TimeDisplay*):



- " デジタル時計 " (*TimeDisplayLCD*) :



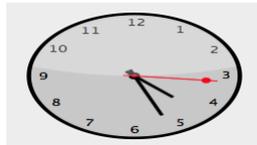
フォーム内に時間表示エリアを挿入するための方法は二つあります。:

- 4D のオブジェクトライブラリから時間表示エリアオブジェクトを挿入する方法
- サブフォームエリアを作成して、その詳細フォームプロパティに "TimeDisplay" または "TimeDisplayLCD" 割り当てる方法

その後、プロパティリストにてオブジェクト名と、そのサブフォームに関連付けたい変数名を定義します。変数に名前を与えないことにより、フォームローカル変数の利点を生かすこともできます。

## Clock

clock ウィジェットは SVG で描画されているので、ベクターパスをもち、それ故にアプリケーションモードでは自在に変形することが出来ます。(デザインモードではサイズは固定です。):



この時計は "Resources" フォルダの第一レベルにある "clock.svg" を置き換えることにより削除したりデベロッパ自身の作成したオブジェクトで置き換えたりすることができます。

## カレントの時刻を表示する

デフォルトでは、clock ウィジェットに関連付けられている変数は実数型です。この場合、ウィジェットは自動的に現在の時刻を表示し、時計として機能します。

### Compatibility note

この機能により、4D Pack の `_AP EXTERNAL CLOCK` エリア (v14 では使用不可) を使用して生成した時計を置き換えることができます。

表示される時刻にはオフセットをを適用することもできます。秒数に換算したオフセットをウィジェットに関連付けられている変数に渡して下さい。例えば、3600 = 時計を 1 時間進める、-1800 = 時計を 30 分遅らせる、等です。

更に、以下の様な機能もあります。:

- 時計の秒針は、[TimePicker DISPLAY SECOND HAND](#) メソッドを使用することにより表示したり隠したりすることが出来ます。
- 時計は、時刻に応じて "昼間モード" と "夜間モード" を切り替えます。:



昼間と夜間の時刻の範囲はそれぞれ、  
 8:00:00 -> 19:59:59 = 昼間  
 20:00 -> 07:59:59 = 夜間  
 となっています。

**静止した時刻を表示する** clock ウィジェットは指定した静止時刻を表示するように設定することができます。そのためには、関連付けられた変数を "時間" 型であると宣言します。それにより、以下のことが出来るようになります。:

- 関連付けた変数に C\_TIME コマンドを使用する
- プロパティリスト内の変数タイプで時間を指定する

これらの場合、時計は時間変数の値の時刻を表示します。

▶ 時計に 10:10:30 と表示させたい場合を考えます。:

`C_TIME (myvar) // myvar はウィジェットの変数の名前`  
`myvar:=?10:10:30?`



## デジタル時計

新しい "digital clock" ウィジェットは時刻を液晶ディスプレイまたは LED スクリーンのように表示させることが出来ます。:

03:04:15 PM

このウィジェットは SVG で描画されているので、表示される画像を劣化させることなく変形させることができます。ウィジェットは透明なので色のあるオブジェクトに重ねて色を変えることも出来ます。:



### カレントの時刻または静止した時刻を表示

"Clock" ウィジェットと同じように、デジタル時計もカレントの時刻を動的に表示したり静止した時刻を表示したりすることが出来ます。

- カレントの時刻を表示するためにはウィジェットのサブフォームのオブジェクトに実数型の変数を関連付けます (デフォルト)。ウィジェットは自動的にカレントの時刻を表示します。
- 静止した時刻を表示するためには、ウィジェットのサブフォームに時間型の変数を関連付けます。ウィジェットは変数の値を表示します。

"clock" モードでは、表示される時刻にはオフセットを適用することもできます。秒数に換算したオフセットをウィジェットに関連付けられている変数に渡して下さい。例えば、3600 = 時計を 1 時間進める、-1800 = 時計を 30 分遅らせる、等です。

コンポーネントメソッドには表示のオプションに関する様々なメソッドが用意されています。これらのメソッドには "TimePicker LCD" の接頭辞がついています。

### TimePicker DISPLAY SECOND HAND

TimePicker DISPLAY SECOND HAND ( *objectName*; *secondHand* )

引数	型	説明
<i>objectName</i>	テキスト	→ サブフォームオブジェクト名
<i>secondHand</i>	ブール	→ True (デフォルト) = 秒針を表示 False = 秒針を非表示

**TimePicker DISPLAY SECOND HAND** コマンドは *objectName* で指定したサブフォームオブジェクトの、秒針の表示・非表示を設定します (ただし clock ウィジェットに限る)

デフォルトでは、秒針は表示されています。日表示するためには、このコマンドを呼び出して *secondHand* 引数に False を渡します。

## TimePicker LCD DISPLAY AMPM

TimePicker LCD DISPLAY AMPM ( *objectName*; *amPm* )

引数	型	説明
<i>objectName</i>	テキスト	→ サブフォームオブジェクト名
<i>ampm</i>	ブール	→ True = AM/PM を表示、False = 非表示

**TimePicker LCD DISPLAY AMPM** コマンドは、*objectName* で指定したサブフォームオブジェクトの、AM/PM 表示を表示・非表示を設定します。この文字は時計が 12 時間モードのときに午前か午後を区別するために表示されています。(TimePicker LCD SET MODE を参照して下さい。)

デフォルトでは、この文字は表示されています。*amPm* 引数に **False** を渡すことによって非表示にすることができます。

- ▶ AM/PM の文字表示を非表示にしま。:

**TimePicker LCD DISPLAY AMPM** ("Subform1";False)

03 20 04

*See also:* TimePicker LCD SET MODE

## TimePicker LCD DISPLAY SECONDS

TimePicker LCD DISPLAY SECONDS ( *objectName*; *seconds* )

引数	型	説明
<i>objectName</i>	テキスト	→ サブフォームオブジェクト名
<i>seconds</i>	ブール	→ True = 秒数を表示、False = 非表示

**TimePicker LCD DISPLAY SECONDS** コマンドは、*objectName* で指定したサブフォームオブジェクトの、秒数表示の表示・非表示を設定します。(ただしデジタル時計に限る)

デフォルトでは、秒数は表示されています。*seconds* 引数に **False** を渡す事によって非表示にすることができます。

## TimePicker LCD SET COLOR

TimePicker LCD SET COLOR ( *objectName*; *color* {;*colorG*; *colorB*} )

引数	型	説明
<i>objectName</i>	テキスト	→ サブフォームオブジェクト名
<i>color</i>	倍長整数	→ RGB カラーの値 (4 bytes) または他の引数が渡された場合には赤成分の値 (0..255)
<i>colorG</i>	倍長整数	→ 緑成分の値 (0..255)
<i>colorB</i>	倍長整数	→ 青成分の値 (0..255)

**TimePicker LCD SET COLOR** コマンドは、*objectName* で指定したサブフォームオブジェクトの、数字の色を設定します。(ただしデジタル時計に限る)

このコマンドには二つのシンタックスが使用できます。

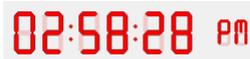
- *color* 引数のみを指定する場合、(0x00RRGGBB) のフォーマットに適合する 4 バイトの倍長整数を渡します。(桁番号は右から左へ数えられ、0 から 3 とナンバリングされています):

バイト	詳細
3	絶対 RGB カラーを指定するためには 0 でなければなりません。
2	カラーの赤成分 (0..255)
1	カラーの緑成分 (0..255)
0	カラーの青成分 (0..255)

- それ以外に、3 つの引数を渡す事もできます。: *color*, *colorG* そして *colorB* の 3 つです。この場合、それぞれの引数が 0 から 255 の間の数字である必要があり、それぞれが RGB カラーの要素となります。
- ▶ 数字の表示を赤に変更する場合があります。:

**TimePicker LCD SET COLOR** ("Subform1";0x00FF0000)

// 別の記法 : TimePicker LCD SET COLOR ("Subform1";255;0;0)



## TimePicker LCD SET MODE

TimePicker LCD SET MODE ( *objectName*; mode )

引数	型	説明
<i>objectName</i>	テキスト	→ サブフォームオブジェクト名
<i>mode</i>	倍長整数	→ 12 = 時間を 12 時間モードで表示 24 = 時間を 24 時間モードで表示

**TimePicker LCD SET MODE** コマンドは *objectName* で指定したサブフォームオブジェクトの、12 時間モードまたは 24 時間モードの表示モードを設定します。(ただしデジタル時計に限る)

デフォルトでは、オブジェクトは 12 時間モードで表示されています。*mode* 引数に 24 を渡す事によって 24 時間モードに切り替えることが出来ます。この場合、一般的に言って AM/PM 表示を同時に非表示すると良いです。(TimePicker LCD DISPLAY AMPM コマンドを参照して下さい。)

- ▶ 24 時間表示モードへと切り替えて、同時に AM/PM 表示を非表示にする場合を考えます。:

*TimePicker LCD SET MODE* ("Subform1";24)

*TimePicker LCD DISPLAY AMPM* ("Subform1";False)

A digital LCD display showing the time 15:40:23 in a black, segmented font on a light gray background. The digits are arranged in a standard 12-hour digital clock format with colons separating the hours, minutes, and seconds.

*See also:* [TimePicker LCD DISPLAY AMPM](#)

