4D v15 - アップグレードリファレンス(標準版)

4D v15へようこそ。このマニュアルでは、この4Dの新しいメジャーリリースで追加された機能と実装を全て説明しています。

4D v15は、ハードウェア面においてもソフトウェア面においても4Dアプリケーションを新しいテクノロジーへと統合するプロセスの重要なステップを担っている一方、4Dデベロッパコミュニティにおいて期待されていたたくさんの機能が提供されています:

- 新しい"Object"型フィールド
- 複数のの最適(SQLエンジン経由の検索と並べ替えられた配列内の検索、新しいServerNet ネットワークレイヤー、強化されたネットワークセキュリティ、拡張されたUnicodeのサポート、etc.)
- 4Dコードをローカル設定に関係なく使用できるようになる新機能(メソッドにおけるUnicodeモードとインターナショナルモード)
- 新テクノロジーへの幕開け: Wakanda アプリケーションに接続するための4D Mobile、社内ディレクトリに接続するためのLDAP、XSLTテクノロジー(廃止予定)にとって代わる新しい4Dタグ
- フォームと、リストボックスなどのオブジェクトを管理するための新しいコマンドとファンクション(これらは、モダンなデザイン、ダイナミックなインターフェースの作成をアシストします)
- 統合された編集とワードプロセスを可能にする新しい4D Write Pro オブジェクト
- コードとクエリをモニター・監視するために追加された機能
- 新しいOS X用64-ビット版: 4D v15でOS X用64-ビット版に対応するのは**4D Server** (プレビュー) だけではなく、**4D Internet Commands**、**4D Pack**、**4D For OCI**、**4D ODBC Pro** そして **4D ODBC Driver** です。これにより、Appleの最も強力なマシンを使用したフルな生産環境を配布することが[#/note]ビルドすることが可能になります。(OS X用64-ビット版 4D Server はプレビューリリースとして提供されるため、運用には推奨されないことに留意ください)

Note: 4Dの継続的なデリバリープログラム(R-リリース)のメンバーの方であれば、これらのアップデートを一足早くお使いの4Dアプリケーションに取り入れる機会がありました。また、このバージョンは新たなリリースの一つでもあります。例えば、もしv14 R5をお使いだった場合、このマニュアル内で"v14 Rx"というスタンプが付いた新機能については既にお持ちですし、それらは以前のR-バージョンのアップグレードマニュアルにも記載がありました。こういった方に関しては、**4D v15 - アップグレードリファレンス(Rリリース版)** の方のマニュアルをお読みいただく方が早いでしょう。

デザインモード

- コマンドラインインターフェースの拡張
- オブジェクトフィールドデータ型
- 重複不可のフィールドではインデックスが必須に
- フォーム
- 廃止予定のコマンドは非表示に
- メソッドでのUnicode
- メソッドエディターでのEnglish-US設定
- デバッガーとランタイムエクスプローラー
- クエリエディター
- 組み込みアプリ内のデフォルトのデータファイル

コマンドラインインターフェースの拡張

4D v14 R3 から導入

以前はOS Xでのみサポートされていたアプリケーションコマンドラインパーサーが改善され、OS XとWindows、両方で使用できるようになりました。

Windowsでも、例えば以下の様に記述する事ができるようになります:

```
%HOMEPATH%\Desktop\4D\4D.exe --structure %HOMEPATH%\Documents\myBase.4dbase --data
%HOMEPATH%\Documents\data\myData.4DD
```

オブジェクトフィールドデータ型

4D v15以降、新しい**オブジェクト**フィールド型が4Dデータベースエンジンによってサポートされるようになりました。オブジェクトフィールドデータ型を使用すると、以下のようなことが可能になります:

- データファイル内部にオブジェクトを保存する
- 動的にオブジェクトの属性を追加、編集、または削除する
- 属性でオブジェクトをクエリする
- オブジェクトの値を読み込む/書き出す、等

実装に関する注意: 一部の機能では、現在オブジェクトをフィールドをサポートしていません(以下の **現状での制約** の章を参照して下さい)。

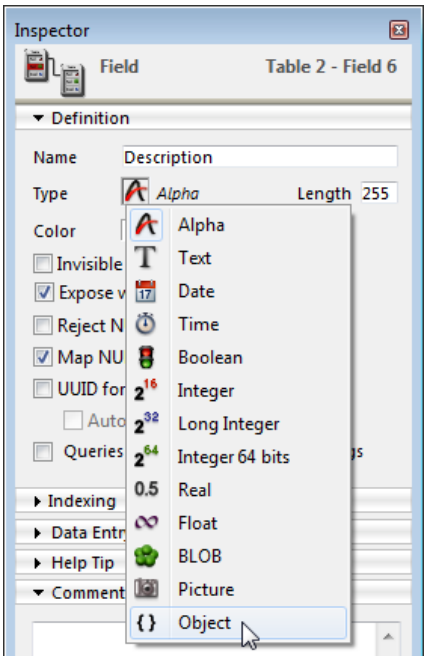
何故オブジェクトフィールドを使用するのか？

オブジェクトフィールド型を使用すると、スキーマレスでダイナミックなフィールドを定義することができます。これらのオブジェクトフィールドは"ユーザー定義の"、または"カスタムの"フィールドとみなすこともできます。4D v15では、スキーマデータモデルとスキーマレスデータモデルの二つの選択肢があります。どちらの場合においても、インデックス付きの速いクエリを実行することができます。

また、オブジェクトフィールドはデータモデルを単純化することもできることに注意して下さい。例えば、よくある"Contacts"というテーブルについて、一つのオブジェクトフィールドがあるだけで、可能性がある、ありとあらゆる値をのために無数にフィールドを作成する必要がなくなります(またそのようなフィールドは90%の割合で結局のところ使用されません)。情報モデルは必要に応じてオン・ザ・フライで作成することが可能です。

オブジェクトフィールド定義

新しいオブジェクトフィールドデータ型は、他のデータ型と同じようにストラクチャーエディターのインスペクターを使用して設定することができます:



4D オブジェクトフィールドは異なる種類の属性/値のペアを事前にデータスキーマを定義することなく保存することができます。保存されたデータストラクチャーは、異なるレコード間において必ずしも同じとは限りません。例えば、[Person]Address オブジェクトフィールドは都市や国等に応じて、異なる属性を含めることができます:

```
record1= {"street1":"Cotton Treasure Grounds", "street2":"Place Corners", "state":"MD",...} record2=
{"street1":"Umber Road", "Number":"28", "state":"MO",...}
```

4Dオブジェクトのストラクチャーは、"属性/値"ペアの標準原理に基づいています。これらのオブジェクトのシンタックスはJSON記法を基にしておりますが、全て準拠しているという訳でもありません:

- 属性の**名前**は常にテキストです。例:"Name"等
- 属性の**値**は、以下の型のどれかを取ります:
  - 数(実数、整数、等)
  - テキスト
  - 配列(テキスト、実数、倍長整数、整数、ブール、オブジェクト、ポインター)
  - null
  - ブール
  - ポインター(保存時はポインターとして保存されますが、**JSON Stringify** コマンド使用時、または複製の際には評価されます)
  - 日付("YYYY-MM-DDTHH:mm:ssZ" フォーマット)
  - オブジェクト(オブジェクトは複数の階層へと保存することが可能です)

警告: 属性名は大文字と小文字を区別するという点に注意して下さい。

オブジェクトフィールドのサイズは2GBが上限です。オブジェクトフィールドを内包しているレコードを扱っている場合、オブジェクト全体がメモリーへと読み込まれます。テキスト、ピクチャー、BOBフィールドと同じように、オブジェクトフィールドは(レコード内かそうでないかに関わらず)データファイル内に保存することができますし、データファイル外に保存することもできます。このオプションについてはデザインリファレンスマニュアルの**データをデータファイル外に保存** の章において詳細な説明があります。

オブジェクトフィールドはインデックスを付けることができます(**自動**オプションのみ)。つまり、全ての属性パスは自動的にインデックスされるという事です。また、**非表示**にすることもできますし、**4D Mobileサービスで公開**の属性を使用することもできます。その一方で、オブジェクトフィールドは**重複不可**を設定することはできません。
オブジェクトフィールドを管理するためには4Dの**オブジェクト(ランゲージ)** コマンドを使用します。オブジェクトフィールドからデータを読み出し、あるいは書き込みするためには**OB Get** と **OB SET** コマンドを使用します。また、**OB SET ARRAY** と **OB GET ARRAY** コマンドを使用することによって配列を属性として保存・読み出しができます。

オブジェクトフィールドはテキストベースであるために、オブジェクトの中身は4Dフォーム内においてはデフォルトでJSONフォーマットされたテキストとして表示されます(以下の章を参照して下さい)。
注: JSONオブジェクトを扱うためには、"**JSON**" テーマ内のコマンドを使用して下さい。

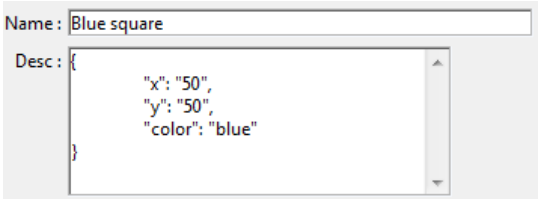
フォーム内にてオブジェクトフィールドを表示・入力する

デフォルトで、オブジェクトフィールドは4D フォーム内ではテキストエリアとして表示されます。これらのエリアのうち、オブジェクトデータは、未定義か、またはJSONテキストとしてフォーマットされている必要があります。そうでない場合には、エラーが返されます。

例えば、[Rect]Descというフィールドをオブジェクトフィールドとして定義した場合、以下のように書くことができます:

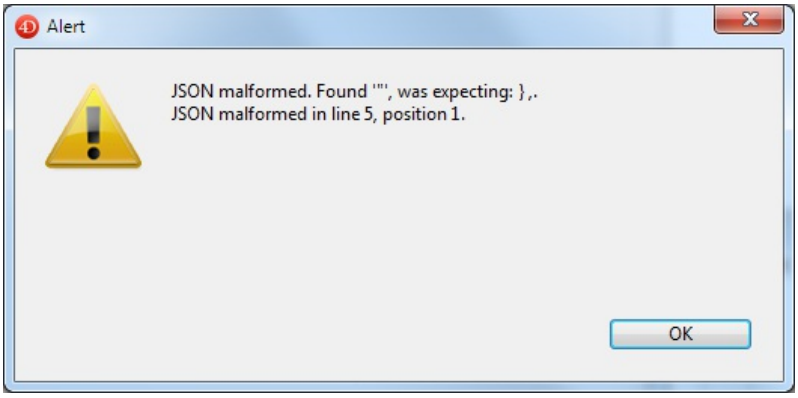
```
CREATE RECORD([Rect])
[Rect]Name:="Blue square"
OB SET([Rect]Desc;"x";"50";"y";"50";"color";"blue")
SAVE RECORD([Rect])
```

[Rect]Descフィールドがフォーム内に含まれているとき、以下の内容が表示されます:



テキストフィールド内にて表示されている値を直接編集することもできますし、標準のオブジェクト記法を使用してオブジェクトデータを直接入力することもできます。**[Tab]** キーを押すと、オブジェクトは自動的にJSON形式にフォーマットされます:

しかしながら、直接編集する際には細心の注意を払って行うべきです。何故なら、誤った場所に入った空白や記号などは、JSON解析エラーとなり、編集したデータは保存されないからです:



一般的には、**オブジェクト(ランゲージ)** と **JSON**コマンドを通してオブジェクトフィールドの中身を管理する方がより正確です。

オブジェクトフィールド値の取得と設定

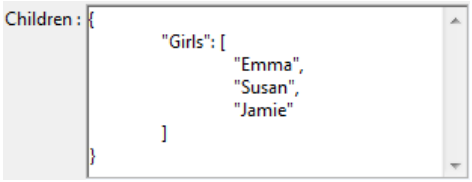
オブジェクト(ランゲージ)テーマ内の全てのコマンドは第一引数(*object*)としてオブジェクトフィールドを受け入れるようになりました。

標準のランゲージオブジェクトの様に、オブジェクトフィールド値は**オブジェクト(ランゲージ)**テーマのコマンドを使用して管理することができます。例えば:

```
 // 値を設定するには
OB SET([Persons]Identity_OB;"First Name";$firstName)
OB SET([Persons]Identity_OB;"Last Name";$lastName)

 // 値を取得するには
$firstName:=OB Get([Persons]Identity_OB;"First Name")
$lastName:=OB Get([Persons]Identity_OB;"Last Name")
```

配列もまたサポートされます。例えば:

```
ARRAY TEXT($arrGirls;3)
$arrGirls{1}:="Emma"
$arrGirls{2}:="Susan"
$arrGirls{3}:="Jamie"
OB SET ARRAY([Persons]Children;"Girls";$arrGirls)
```



4Dランゲージ内におけるオブジェクトフィールドのサポート

4Dランゲージは、オブジェクトフィールドをサポートするようにアップデートされました。具体的には以下のような点です:

- **オブジェクト(ランゲージ)** テーマのコマンドは全てオブジェクトフィールドを第一引数(*object*)として受け入れるようになりました。
- オブジェクトフィールドでのクエリを管理するために、新しいコマンドが追加されました。このコマンドは別の章にて詳細な説明があります。

- **SELECTION TO ARRAY**、**SELECTION RANGE TO ARRAY** と **ARRAY TO SELECTION** コマンドは、**ARRAY OBJECT** バインディングを通してオブジェクトフィールドをサポートします。
- **Selection to JSON** と **JSON TO SELECTION** コマンドはオブジェクトフィールドをサポートします。その際、自動的にJSONフォームに変換されます。
  しかしながら、以下の4D式は曖昧であるという点に注意して下さい:

  ```
  Selection to JSON([aTable];objectField)
  ```

  以下の様に解釈することが可能です:
  - *table*のテーブルのカレントセレクション内の*objectField* のフィールドの全ての値からJSONを生成
  - 'objectField' のカレントレコード値をテンプレートしてJSONを生成
  この場合4Dは最初の解釈を使用した振る舞いを行います。こちらの方が最も一般的に使用されている解釈だからです。
- **Old** と **Modified** コマンドはオブジェクトフィールドをサポートします。
- **SET FIELD VALUE NULL** はオブジェクトフィールドの中身を消去します。
- **GET FIELD PROPERTIES** は Is object を返すようになりました。
- **PROCESS 4D TAGS** は、4D形式のオブジェクトフィールドのみサポートします。

ただし、技術的な理由から、一部のコマンドはオブジェクトフィールドをサポートしない点に注意して下さい。これらのコマンドは"現状での制約"のセクションにまとめられています。

式とオブジェクトフィールド

オブジェクトフィールドはフォーミュラの中で使用することができます(標準のフォーミュラエディタ、または**EXECUTE FORMULA** コマンドを使用します)。しかしながら、このコンテキストにおいては、オブジェクトフィールドは以下のコマンドの使用でしか扱う事ができません:

- **OB Get**
- **OB Is empty**
- **OB Is defined**
- **OB Get type**

例えば、以下のクエリフォーミュラを実行することができます:

```
OB Get([Rect]Desc;"color")="blue"
```

現状での制約

ほとんどの4D標準の機能は、オブジェクトフィールドのデータの方に限らずサポートします。しかしながら、アプリケーション内のいくつかの特殊な部分においては、現在のリリースにおいてはオブジェクトフィールドに未対応な部分もあります。これらの部分は将来のリリースにおいてアップデートされ、徐々に使用可能になって行く予定です。

4Dフォーミュラで一部サポートするもの

以下の機能またはコマンドは、4Dフォーミュラを通して部分的にオブジェクトフィールドをサポートします:

機能/コマンド
クエリエディター
並べ替えエディター
読み込み/書き出しエディター(4Dアプリケーションフォーマットに対しては自動的にサポートされますが、テキストフォーマットについてはフォームを使用する必要があります)
4D Write
4D Write Pro
4D View
4D Tags
PROCESS 4D TAGS

サポートされないもの

以下の4Dの機能やコマンドは、オブジェクトフィールドをサポートしません:

機能/コマンド

クイックレポートエディター

ラベルエディター

Graphs

PHP

SDK プラグイン

SQL データ定義ランゲージ(CREATE TABLE)

SQL データ操作ランゲージ(SELECT, INSERT, UPDATE)

SQL EXPORT DATABASE と **SQL EXPORT SELECTION**

DISTINCT VALUES

RELATE ONE

RELATE MANY

ORDER BY

SCAN INDEX

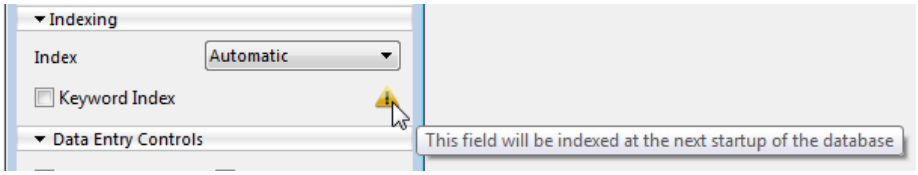
重複不可のフィールドではインデックスが必須に

4D v14 R4 から導入

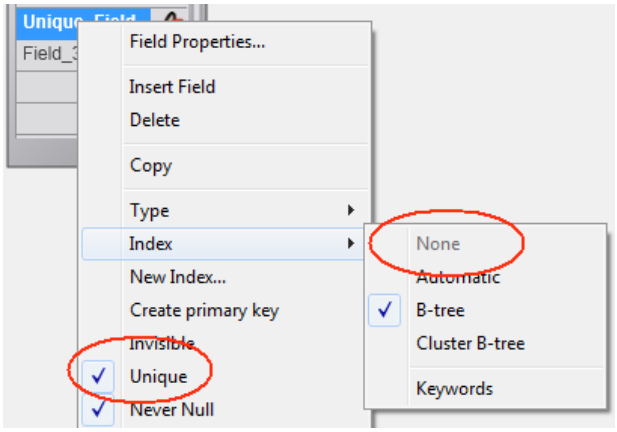
4Dでは、**重複不可**属性を持つフィールドはインデックスされる必要があります。ストラクチャーエディターにおいて、インデックスを持たない重複不可属性のフィールドを定義することはできなくなりました。以前のリリースでは、メンテナンスのためにそのような設定を維持することは可能でした。

これに伴いストラクチャーエディターも更新されました:

- フィールドに対して**重複不可**属性を選択したとき、まだインデックスがない場合に、インデックスが作成されることが視覚的に表示されるようになりました:



- 重複不可フィールドに対して、インデックスを削除することができなくなりました(そのようなフィールドに対しては、**なし**または削除オプションが使用できなくなりました):



重複不可のフィールドからインデックスを削除したい場合には、**重複不可**属性を先に解除する必要があります。

使用不可のピクチャーフォーマットに対しての新アイコン

4D v14 R3 から導入

マシン上でレンダリングできないフォーマットで保存されているピクチャーに対しては新しいアイコンが表示されるようになりました。その対応していないフォーマットの拡張子は、アイコンの下部に書かれています:

このアイコンは、これらのピクチャー形式が使用されている個所には全て使用されます:

FirstName :	LastName :	Photo :
Elizabeth	Smith	
Gerry	Mc Namara	
Henry	Portier	

これは、ピクチャーがローカルでは表示または処理することができないが、他のマシン上で表示するときのために保存しておけるということを意味します。これは 例えば、Windowsプラットフォーム上でのPDFピクチャーや、Max OS X版の64 bit 4D ServerでのPICTベースのピクチャーなどが該当します(詳細は **OS X用4D Serverの64ビット版について(プレビュー)**を参照して下さい)。

新しい角の半径プロパティ

4D v14 R4 から導入

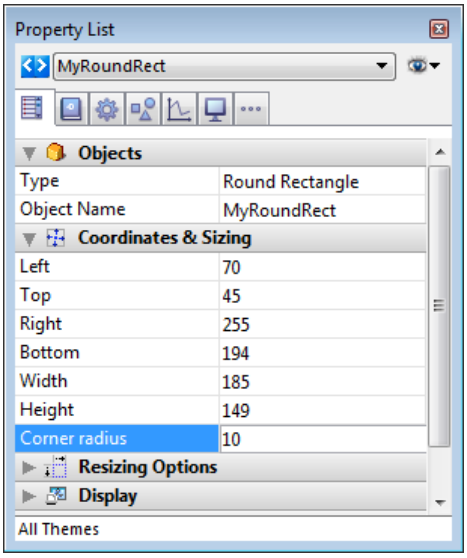
角の丸い四角オブジェクトの角の半径を設定できるようになりました。このプロパティはそれぞれの丸い角の半径を定義します。

注: 以前のリリースでは、このプロパティはカスタマイズ不可でした(常に5ピクセル)。

このプロパティはフォームエディターのプロパティ―リスト内の新しい**角の半径**オプション(以下参照)を通して、または二つの新しいコマンドを使用することによって設定可能です(**OBJECT SET CORNER RADIUS** と **OBJECT Get corner radius**を参照して下さい)。

角の半径

角の半径プロパティは、角の丸い四角オブジェクトのプロパティリスト内の、"座標とサイズ"テーマ内にあります:



デフォルトでは、角の丸い四角に対しての半径の値は5ピクセルとなっています。このプロパティを変更することによって、独自の形をもった角の丸い四角を描画することができます:

最小値は0で、この場合は通常の角が丸くない四角が描画されます。
最大値は四角のサイズに応じて動的に計算されます。この値は長方形の短辺の半分を超えることはできません。

フォームオブジェクトに対してより長い名前の適用

4D v14 R5 から導入

フォームオブジェクト名には、**255 バイト** までの名前を付ける事ができるようになりました。以前のリリースでは、これらの名前は31バイトまでに制限されていました。

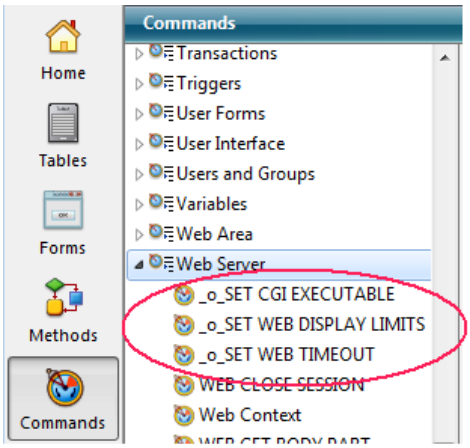
より長い名前をフォームオブジェクト名に使用することができるようになったおかげで、"xxxx_Button" や "xxx_Mac"など、特定の命名ルールに則った名前を定義・適用しやすくなりました。

廃止予定のコマンドは非表示に

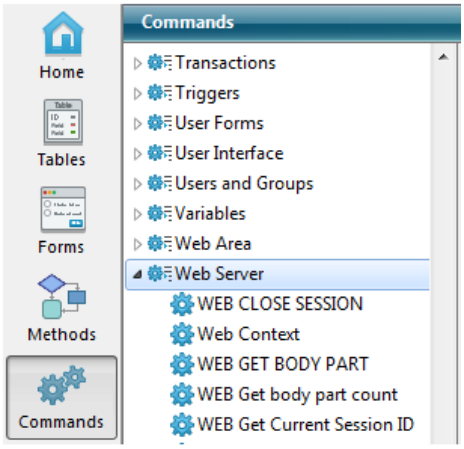
4D v15以降、全ての廃止予定(または廃止された)コマンドはシステム的にコマンド名の先頭に"_o_"がつけられ、4Dのリストとサジェスチョンリスト内に表示されなくなります。

注: 4D v15で改名された廃止予定のコマンドの一覧は、"**廃止予定のコマンドの改名**"のセクションにて確認することができます。

以 前のリリースにおいては、廃止予定のコマンドは必ずしもコマンド名が改変されていたわけではなく、通常のコマンドと並んで一覧に表示されていたため、混乱 の原因となる可能性がありました。4D v15では、廃止予定になっていないコマンドのみコマンドの一覧から選択することができます:



4D v14



4D v15

関係するリストは以下の箇所です:

- エクスプローラー/"コマンド"タブ
- メソッドエディターのサジェスチョンリスト(コマンドの一覧と、テーマごとのコマンドの一覧)、それからタイプアヘッドポップアップ
- フォーミュラエディターサジェスチョンリスト

もちろん、廃止予定のコマンドが既存のコードの中で勝手に削除されていることはありません。先頭に"_o_"がつけられた上で、サポートされている限りはドキュメントにある通りの振る舞いを続けます。必要であれば、メソッド内でその名前をタイプするだけで廃止予定のコマンドを使用することもで きます。これらは引き続き正常に解釈されます:

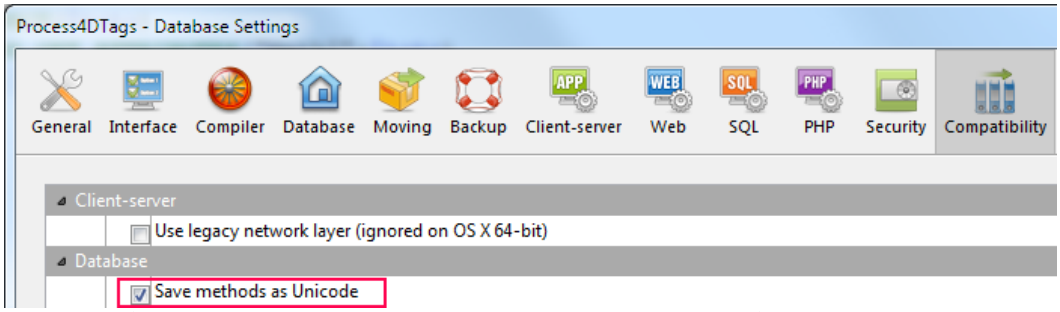
```
4    _o_SET CGI EXECUTABLE
```

しかしながら、廃止予定のコマンドを新しいコードの中で使用することは推奨されていないという点に注意して下さい(**廃止予定の機能と削除された機能**マニュアルを参照して下さい)。

メソッドでのUnicode

4D v15から、4Dメソッドコードの文字列をUnicodeで保存できるようになりました:

- 4D v15以降で作成されたデータベースに関しては、メソッドコードは自動的にUnicodeで保存されます。
- それより古いバージョンから変換されたデータベースに関しては、新しい**メソッドをUnicodeで保存**のデータベース互換性設定によってUnicodeモードへと切り替えることができます:



注: このオプションを変更した場合、それが有効化されるためにはアプリケーションを再起動する必要があります。このオプションはどの時点においてもチェックを入れる・外す事ができます。変更した後に保存したメソッドのみその影響を受けます。

メソッドをUnicodeで保存する理由

これまでのリリースにおいては、4Dメソッドコードの文字(式、変数名、メソッド名、コメント等)はカレントのローカルエンコーディングを使用して保存されていました。このエンコーディングは、異国のデベロッパ間で4Dコードをやり取りする場合に特に問題を引き起こす可能性がありました。例えば、フランスのデベロッパがアクセント記号がつかわれている4Dコードを書き、そのデータベースをイギリスのデベロッパに送った場合、これらのアクセントは失われてしまいます。日本語版で書かれたコードに関しても深刻な問題が起こる可能性がありました。

メソッドをUnicodeで保存することで、これらのタイプの問題は全て解決できるうえに、特殊な文字を使用した4Dコードの交換を可能にします。

Unicodeモードオプションは、既存のデータベースにおいても速やかに有効化されることが推奨されます(国際的な環境で作業している場合にはなおさらです)。

実装に関する注意: この新機能はランゲージとその解釈において適用されます。一部の4Dエディタウィンドウ(プロパティリスト等)においては、以前カレントのローカル設定が使用されるので、特定の文字列が正しく表示されない可能性があります。しかしながら、これはコードの実行には影響しません。

互換性

Unicodeでの4Dメソッドの保存(4D v15で新規に作成されたデータベース、または**メソッドをUnicodeで保存**オプションが選択された変換されたデータベース)はユーザーには見えない機能です。

しかしながら、内部的な変更の結果、4D コードでの**ポインター表示**は大幅に変更されました。"新しい"ポインターは内部的により最適化され、二次元配列の要素などの拡張された機能をサポートします(**RESOLVE POINTER**を参照して下さい)。以前コンパイルされたコンポーネントとプラグインとの互換性を維持するために、(単に既存の型を置き換えるのではなく)新しいポインターデータ型が4Dランゲージに追加されました。この新しいタイプはアプリケーション内において見えないように管理されます。しかしながら、以下の点に注意する必要があります:

- **RESOLVE POINTER** は、一次元配列に対しては、第四引数に0ではなく-1を返すようになりました。
- **Get pointer** は不正な変数名と余分な空白に対しての振る舞いが変わりました。

メソッドエディターでのEnglish-US設定

4D v15より、4Dメソッドエディターは、4Dのバージョンやローカルのシステム設定値に関わらず、デフォルトで国際的な"English-US"言語を使用します。この新しい特徴により、4Dアプリケーション間でのコード解釈を妨げうるリージョン間での差異(例えば日付フォーマットなど)を全て統一します。またフランス語版の4Dであっても、コマンドと定数は常に"English-US"で書かれます。

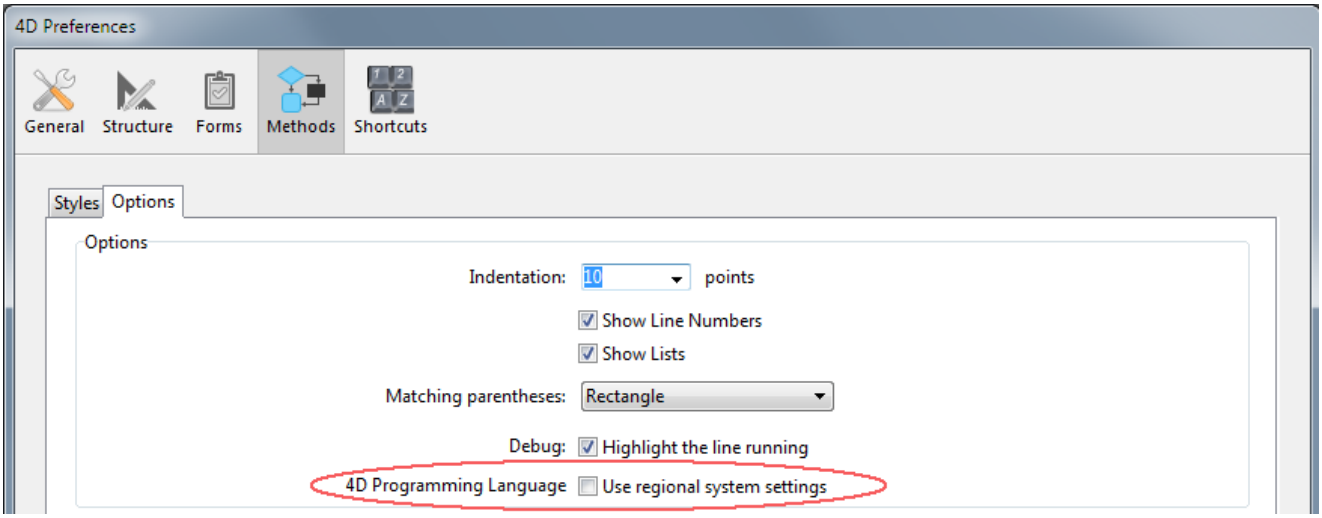
この新しいデフォルト設定により、4Dデベロッパには主に二つの利点があります:

- 国、地域と言語の設定、使用している4Dのバージョンに関わらず、4Dデベロッパー間でのコードの共有が簡単になります。4Dメソッドは単純なコピー/ペースト、またはテキストファイルへの保存だけで、互換性の問題なく交換することが可能になりました。
- また、ソースコントロールツールに4Dメソッドを含めることが可能になりました。これは一般的にリージョン設定や言語とは関係なく書き出される必要があるものです。

この新しい設定は4D設定ダイアログボックス内の新しいオプションにより無効化することもできます。

新しい設定オプション

新しい**地域特有のシステム設定を使う**オプションは、4D環境設定ダイアログボックス内のメソッド/オプションタブを有効化/無効化することができ、"国際的な"コーディングを可能にするものです:



このオプションがチェックされていない(4D v15でのデフォルト設定)場合、カレントの4Dアプリケーションの4Dメソッド内にてEnglish-US設定が使用されます。このオプションがチェックされていた場合以前の4Dのバージョン同様、カレントの4Dアプリケーションの4Dメソッド内ではリージョン設定が使用されます。

このオプションを変更した場合、変更が有効になるためには4Dアプリケーションを再起動する必要があります。

Englis-US設定では何が変わるのか?

新しいEnglish-US設定の採用により、メソッドの書き方に影響がでる可能性があります。これは開発モードで書かれるコードに加え、配付されたアプリケーションでのフォーミュラにも関係します。

この新しいモードにおいては、コードは以下のルールに従う必要があります:

- 実数値の小数点は、全てのバージョンに置いてピリオド(".") が使用されます(例えばフランス語版で一般的に使用されていたカンマ(",")は使用されません)。
- 日付定数は全てのバージョンにおいてISOフォーマットに準拠する必要があります。
- コマンド名と定数名は英語である必要があります(これはフランス語版の4Dにのみ影響します。他の言語ではもともとこのルールが適用されていました)。

注: メソッドエディターには、必要に応じて不正な入力を自動的に修正するメカニズムが搭載されています。

以下の一覧は4D v15と以前のバージョンのコードでの違いをまとめたものです:

	メソッド/フォーミュラでのコード例
4D v15 (デフォルトモードの全てのバージョン)	a:=12.50 b:=!2013-12-31! Current date
4D v14 または 4D v15 (設定がチェックされている、USバージョン等)	a:=12.50 b:=!12/31/2013! Current date
4D v14 または 4D v15 (設定がチェックされている、フランス語版)	a:=12,50 b:=!31/12/2013! Date du jour

注: 4D の以前のバージョンと、設定がチェックされている4D v15においては、実数と日付のフォーマットはシステム設定によって変化します。

フランス語版における変更点

一部の特定の変更は、フランス語版の4Dにのみ影響します。このバージョンにおいては、ランゲージ(コマンド名と定数名)とオンラインヘルプ、そしてドキュメントにおいて、フランス語の名前を使用してきました。

より詳細な情報は、このマニュアルのフランス語版にまとめてあります。

互換性に関する問題

以前のバージョンから変換されたアプリケーションに付いては、4D v15のでの4DコードのEnglisu-US設定を使用するために変更を加えなければならない場合があります。例えば**EXECUTE FORMULA** など、コードがオンザフライで解釈され、トークナイズされていない場合には問題が起こる可能性があります。これは4Dの開発モードだけではなく、配付された製品(リモートモードの4Dまたは組み込まれた4Dアプリケーション)にも関係する問題です。

以下の一覧は4D v15での影響を言語ごとにまとめたものです(ここに記載のないその他のバージョンは、US版と同じように影響を受けます):

機能	4Dの言語設定がUSの場合の影響	4Dの言語設定がFRの場合の影響	4Dの言語設定がDEの場合の影響
フォーミュラエディターの適用	フォーミュラ内: 日付フォーマットのみ	フォーミュラ内: コマンド言語(US)と日付/時刻/小数点	フォーミュラ内: 日付フォーマット+小数点
クイックレポート	フォーミュラ内: 日付フォーマットのみ	フォーミュラ内: コマンド言語(US)と日付/時刻/小数点	フォーミュラ内: 日付フォーマット+小数点
4D Write	フォーミュラ内: 日付フォーマットのみ	フォーミュラ内: コマンド言語(US)と日付/時刻/小数点	フォーミュラ内: 日付フォーマット+小数点
4D View	なし	フォーミュラ内(4Dコマンドを使用する場合、4D Viewコマンドは含めない): コマンドランゲージ(US)	なし
PROCESS 4D TAGS	日付フォーマットのみ	コマンド(Cxxxを使用していない場合) 日付/時刻/小数点	日付フォーマット+小数点
EXECUTE FORMULA	日付フォーマットのみ	コマンド言語(US)と日付/時刻/小数点	日付フォーマット+小数点
METHOD GET CODE/METHOD SET CODE	日付フォーマットのみ	コマンド言語(US)と日付/時刻/小数点	日付フォーマット+小数点

必要があれば、新しい**地域特有のシステム設定を使う**オプションをチェックすることで4D v14の振る舞いへと戻せる、ということに注意して下さい。

配付について

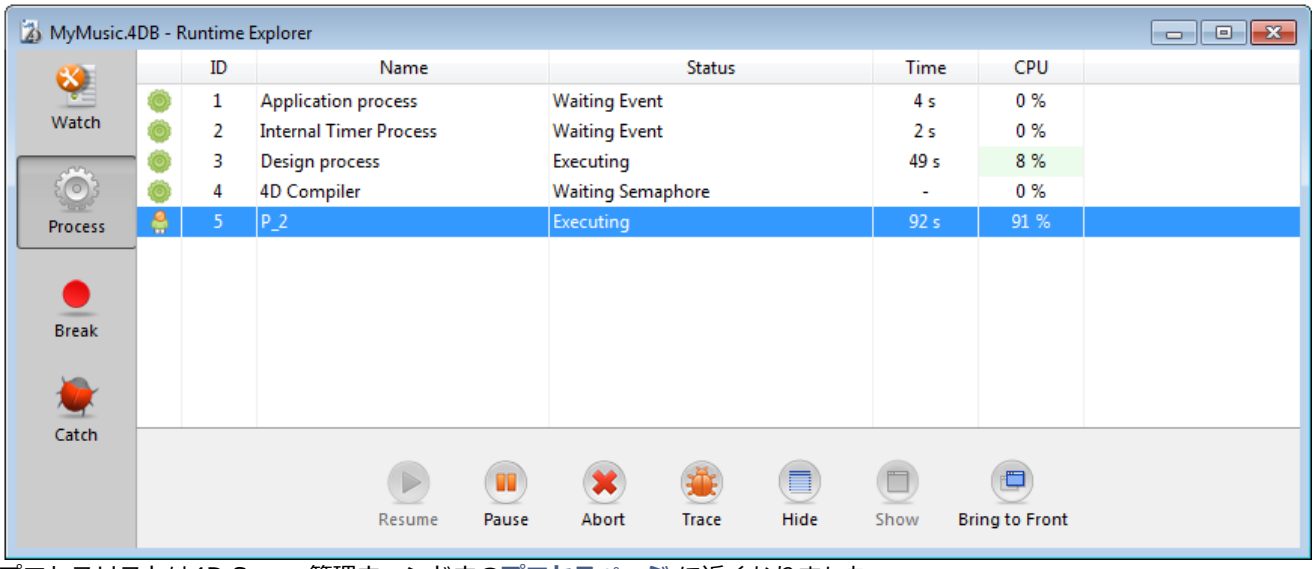
この設定はマシンごとにローカルに保存されるため、リージョン設定を使用したい場合には、4Dアプリケーションを実行するコンピューターにおいてそれぞれ設定しなければなりません。組み込みアプリのコンテキストに置いては、それぞれのマシンにおいて4D v15設定ファイルを編集した上で、"use_localized_language"キーを"true"へと設定しなければなりません。

注: リージョン設定を配付されたアプリケーションにおいて使用するためのソリューションは、このマニュアルのフランス語版において提供されています。

プロセスページ

4D v14 R3 から導入

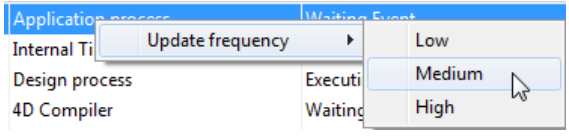
プロセスページのインターフェースは変更され、異なる情報を表示するようになりました:



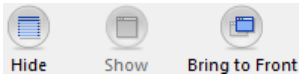
プロセスリストは4D Server管理ウィンドウの**プロセスページ** に近くなりました。

ID、プロセス名、状態、そして実行時間の値は、以前のリリースと同じ様に変わらず表示されます。以下の情報が更新されています:

- CPU: グラフィカルビューはなくなりました。その代り、背景の赤色のグラデーションがプロセスのアクティビティレベルを表します。
- それぞれのプロセスはアイコンで表示されるようになりました。色と形がプロセスの型を表します:
  - アプリケーションサーバー
  - SQL サーバー
  - DB4D サーバー(データベースエンジン)
  - Web サーバー
  - SOAP サーバー
  - プロテクトされた4Dクライアントプロセス(接続している4Dの開発プロセス)
  - メインの4Dクライアントプロセス(接続している4Dのメインプロセスまたはコラボラティブプロセス。クライアントマシンで作成されたプロセスのサーバー上と同じ)
  - 4D クライアントベースプロセス(4Dクライアントプロセスと対等なプロセス。対応する4Dクライアントプロセスを管理するためのプリエンプティブプロセス)
  - スペアプロセス(以前の"4D クライアントデーターベースプロセス")
  - 4Dクライアントプロセス(接続した4Dで実行中のプロセス)
  - ストアドプロシージャー(接続した4Dによって起動されたプロセスでサーバーで実行中のもの)
  - Web メソッド(例えば4DACTIONによって起動されたもの)
  - SOAP メソッド(Webサービスによって起動されたもの)
  - SQL メソッド(SQLクエリによって起動されたもの)
- コンテキストメニューを使用して更新頻度を設定することが出来るようになりました(行を右クリックして下さい):



画面下部のツールバーのボタンも新しくなりました。以前のリリースにあった”オプション”ボタン(ドロップダウンリストを表示)は3つの個別のボタンになりました:

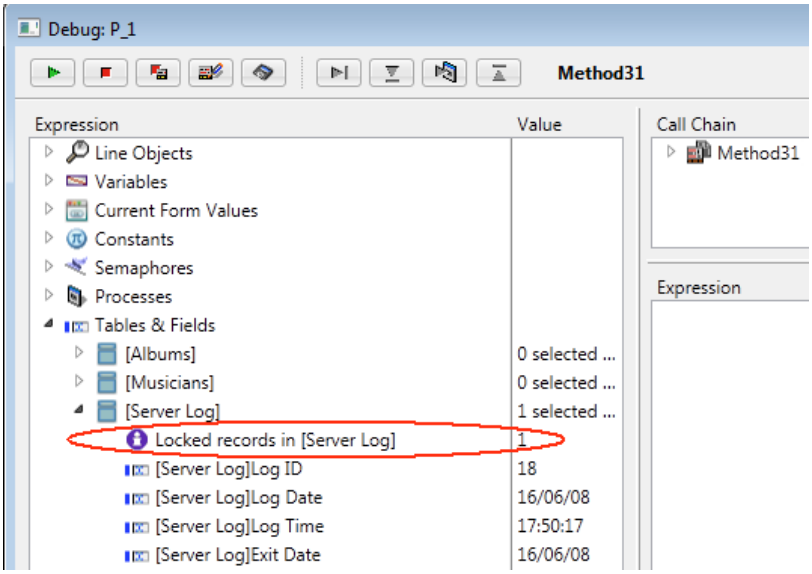


これらのボタンはタスクのセレクションに影響します。R3より、複数のタスクを同時に選択できるようになりました (**Shift+クリック** で連続したセレクション、**Command/Ctrl+クリック** で個別のセレクション)。

ロックされたレコードの数

4D v14 R4 から導入

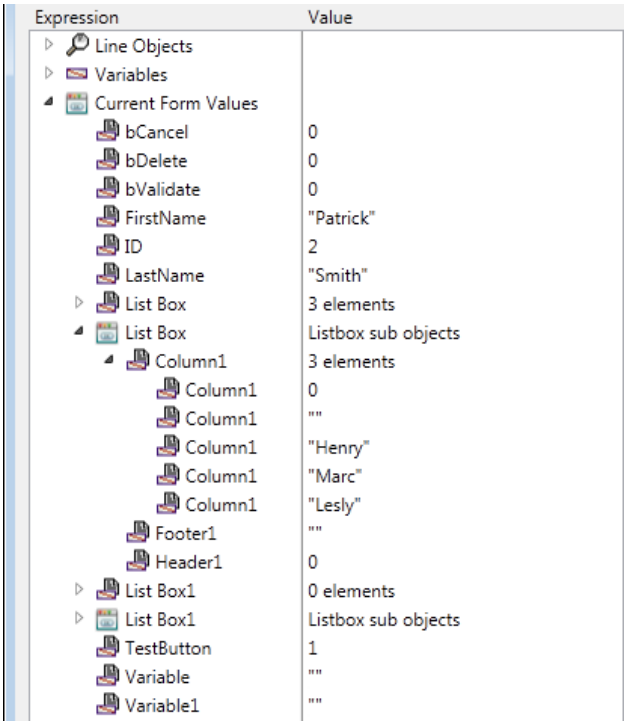
各テーブルでいくつのレコードがロックされているかという数は、ランタイムエクスプローラーのウォッチ画面と、4Dデバッガーの中に表示されるようになりました。この情報は、4Dスタンドアロンと、4D Serverにおいて使用可能です。**4D デバッガ:**



カレントのフォームオブジェクト値

4D v14 R4 から導入

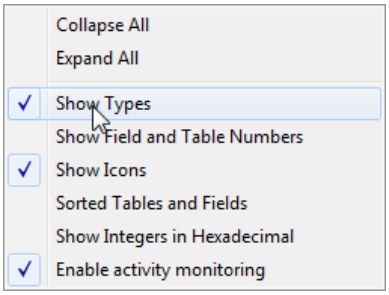
新しい**カレントフォーム値** リストがデバッガーとランタイムエクスプローラーに表示されるようになりました。このリストには、カレントフォームに含まれる動的なオブジェクトそれぞれの名前と、それに関連付けられた変数の値が含まれています:



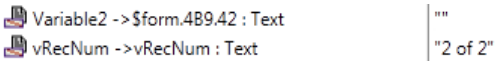
リストボックス配列などの一部のオブジェクトは、二つの別個のアイテム(オブジェクト自身の変数とそのデータソース)として表示させることができます。

新しいリストは、フォームにおいてダイナミック変数を多数使用した場合に特に有効です。フォームオブジェクト名を使用すればそれぞれのダイナミック変数を識別するのは容易だからです。

以前のバージョンの4Dでは、この情報は変数/プロセスリスト中で、ダイナミック変数が内部名(*$form.9.8* 等)で表示されている形で閲覧可能でした。ダイナミック変数はこのリストには表示されなくなり、**カレントフォーム値**のリストの中にのみ表示されるようになりました。ダイナミック変数の内部名は、コンテキストメニューの**型を表示**を選択することで表示させることができます:



ダイナミック変数の名前は、フォーム内にて例えば"$form.4B9.42"と表示されます:



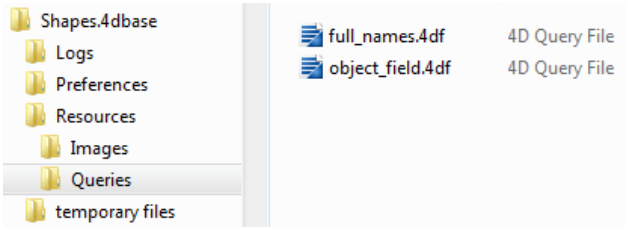
クエリエディター

定義済みのクエリ

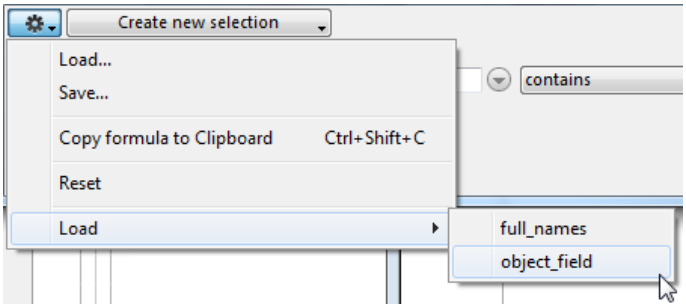
4D v15のクエリエディターでは**定義済みクエリ**をサポートします。ちょうど標準の保存されたクエリのように、定義済みクエリには全ての条件を含む完全なクエリ定義が格納されており、いつでもクエリエディターに読み込むことができます。定義済みのクエリは配付されたアプリケーションにも埋め込むことができ、クエリエディターの新しいサブメニュー内に表示されます。

定義済みのクエリを定義するためには、以下の様にします:

1. データベースの"Resources"フォルダ内に"Queries"サブフォルダを作成します。
2. 定義済みクエリとして使用したい保存されたクエリファイル(.4df)をこのフォルダに追加します:



Resources/Queriesフォルダ内に、カレントテーブルに関係する .4dfクエリファイルが少なくとも一つあるとき、新しい**読み込み >**項目がクエリエディターの編集メニューの最後に追加され、全ての定義済みクエリにサブメニュー項目としてアクセスできるようになります:



サブメニュー項目を選択すると、それに対応するクエリがクエリエディターへと読み込まれます。

リマインダ: サブメニュー内にはカレントテーブルに関係するクエリファイルのみが表示されます。

日本語版ユーザーへの対応

クエリエディターは、特に日本語でのクエリをより正確に扱うためにアップデートされました。具体的には、条件と最近使用したクエリがより自然な語順で表示されるようになりました。

これらの変更は日本語版の4Dに対してのみ影響します。

組み込みアプリ内のデフォルトのデータファイル

4D v15において、組み込みアプリケーション(スタンドアロン版とクライアント-サーバー版の組み込みアプリケーション)のデフォルトのデータファイルの管理が変更されました:

- デベロッパは"デフォルトのデータファイル"をアプリケーションのビルドの段階で定義できるようになりました。
- 組み込みアプリの初回起動時に、"デフォルトのデータファイル"が検知された場合、それは4Dによって自動的に読み込み専用モードで開かれます。

これらの新機能によってデベロッパは、組み込みアプリケーションの初回起動時から、より細かくデータファイルの作成または開くことを管理できるようになります。具体的には以下の様なケースが想定されます:

- 新しい、またはアップデートされた組み込みアプリケーションを起動する際に、"データファイルを開く"4Dダイアログボックスが表示されることを回避することができます。例えばOn Startupデータベースメソッド内にて、デフォルトデータファイルが開かれたかどうかを検知し、ローカルのデータファイルを作成または選択する独自のコードまたはダイアログを実行することができます。
- 読み込み専用のベーシックなデータを含んだ組み込みアプリの配付が可能になります(例えばデモアプリケーション用)。

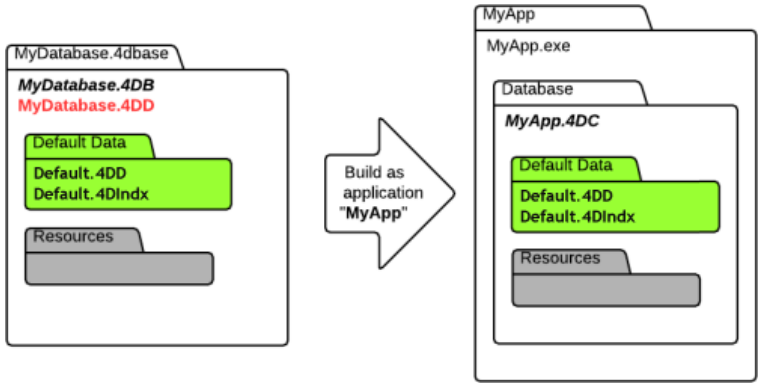
デフォルトのデータフォルダを定義する

4D v15 では組み込みアプリに簡単にデフォルトのデータファイルを収録することが出来るようになりました。これにより、特別なダイアログボックスを表示させることなく、エンドユーザーのマシンにアプリケーションをインストールまたはアップデートさせることができるようになります。デフォルトのデータファイルを定義するためには、以下の用にします:

- デベロッパはデフォルトのデータファイルを、データベースパッケージ(4dbase)内のデフォルトのフォルダ内に保存する必要があります:
  - デフォルトのデータフォルダ名は"Default Data"である必要があります。
  - デフォルトのデータファイル名は"Default.4DD" である必要があり、デフォルトのデータフォルダ内に保存されている必要があります。

  デフォルトのデータファイルには全ての必要なファイルもそろっている必要があります: インデックス、ジャーナル、外部BLOB、、、等です。
  (デベロッパが責任をもって有効なデフォルトデータファイルを用意して下さい)
- アプリケーションがビルドされたとき、アプリケーションのビルドプロセスが、このデフォルトデータフォルダを組み込みアプリ内に統合します。

以下はこの新機能を図示した画像です:



デフォルトのデータファイルが初回起動時に検知された場合、データファイルは自動的に読み込み専用モードで開かれるので、任意のオペレーションが実行できるようになります。

新しい起動シーケンス

組み込みアプリの初回起動時、4Dは有効なデータファイルを選択しようとします。以前のリリースでは、デフォルトデータ

ファイルが提供されていない場合(.4DCファイルと同名のファイルが同階層に無い場合)、標準の"データファイルを開く"ダイアログボックスが表示されていました(これによりユーザーはアプリケーションフォルダ内にデータファイルを作成することができました)。

4D v15では、この起動シーケンスに新しいステップが追加され、これによりデベロッパは新しいデフォルトデータフォルダ機能(上記参照のこと)を利用できるようになります。ステップ2では、アプリケーションは標準の"データファイルを開く"ダイアログボックスを表示することなく開かれ、デベロッパは適切なカスタムのコードを実行できます。

組み込みアプリ起動時の新しい起動シーケンスは以下の通りです:

1. 4D は最後に開かれたデータファイルを開こうとします。
2. 見つからない場合、*4D*は*.4DC*ファイルの隣にあるデフォルトのデータフォルダ内のデータファイルを読み込み専用モードで開こうとします*(4D v15*の新機能*)*
3. 見つからない場合、4Dはストラクチャーファイルの隣にある、同名のデフォルトのデータファイルを開こうとします。
4. 見つからない場合、4Dは標準の"データファイルを開く"ダイアログボックスを表示します。

ランゲージ

- 4D 変換タグ
- Application version
- DELAY PROCESS
- FORM GET OBJECTS
- FORM GET PROPERTIES
- GET ACTIVITY SNAPSHOT
- Get database parameter と SET DATABASE PARAMETER
- GET FIELD PROPERTIES
- Get pointer
- Is licence available
- LDAP 新コマンドテーマ
- MAXIMIZE WINDOW
- METHOD GET CODE and METHOD SET CODE
- OBJECT GET COORDINATES
- On Scroll フォームイベント
- OPEN URL (OPEN WEB URL の新しい名前)
- PROCESS 4D TAGS
- QR Get text property と QR SET TEXT PROPERTY
- RESOLVE POINTER
- TRANSFORM PICTURE
- WA GET PREFERENCE と WA SET PREFERENCE
- WEB GET OPTION と WEB SET OPTION
- WEB SET HTTP HEADER
- XSLT コマンドは廃止予定に
- サブレコード変換
- ツールバーフォームウィンドウ
- リストボックスにてダイナミック変数でカラムを作成
- リストボックスのカラムでオブジェクト配列を使用 (4D View Pro)
- 実数値の、有効桁数の縮小
- 廃止予定のコマンドの改名
- Clickcount
- EXPORT STRUCTURE
- Find in sorted array
- Get database measures
- Get locked records info
- IMPORT STRUCTURE
- LISTBOX GET CELL COORDINATES
- METHOD GET ATTRIBUTES
- METHOD SET ATTRIBUTES
- MOBILE Return selection
- OBJECT Get corner radius
- OBJECT SET CORNER RADIUS
- OPEN DATABASE
- QUERY BY ATTRIBUTE
- WEB Is server running

4D 変換タグ

*4D v14 R4*から導入

4D 変換タグ(以前の名前は*4D HTML タグ*)は拡張されました:

- 新しい4DEVAL タグが追加されました。
- 4DLOOP タグは引数として4D式と配列へのポインターを受け取れるようになりました。

注: PROCESS 4D TAGS コマンドも拡張されています。詳細な情報に関しては、**PROCESS 4D TAGS** の章を参照して下さい。

4DEVAL (新しいタグ)

シンタックス: *<!--#4DEVAL VarName-->* または *<!--#4DEVAL 4DExpression-->*

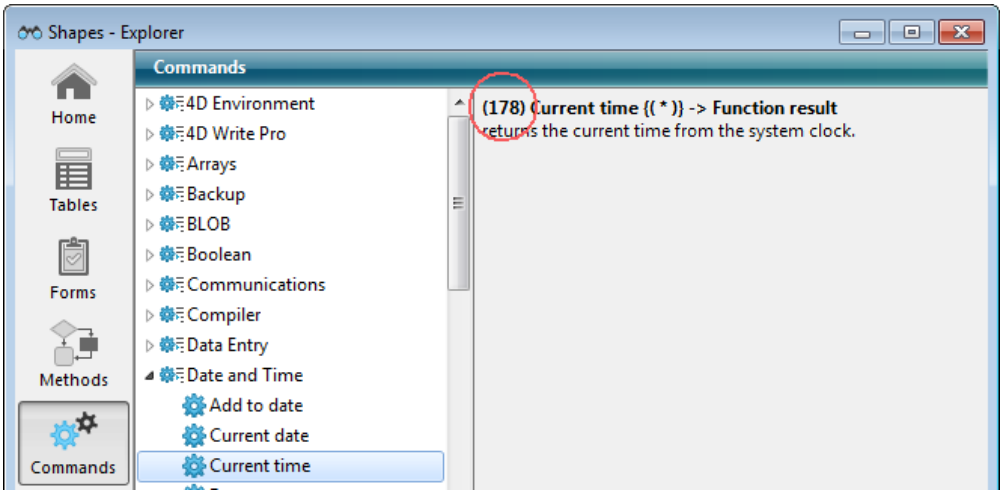
新しい4DEVALを使用すると、変数や4D式を挿入できます。

既存の4DHTMLタグのように、4DEVALタグはテキストを返す際にHTML特殊 文字をエスケープしません。しかしながら、4DHTMLや4DTEXTと異なり、4DEVALは有効な4D宣言であればどれでも実行することができます (値を返さない割り当てや式も含まれます)。
例えば、以下の様なコードを実行することができます:

```
$input:="<!--#4DEVAL a:=42-->" //割り当て
$input:=$input+"<!--#4DEVAL a+1-->" //計算
PROCESS 4D TAGS($input;$output)
 //$output = "43"
```

4Dコマンドやファンクションを、式として直接使用することができます。この場合、*4DExpression* 引数にコマンド番号エスケープコードを渡すことが推奨されます。これによってどの言語版の4Dを使用しても、また将来の4Dリリースにおいてコマンド名が変わっても式は正常に評価されるからです。その際使用すべきシンタックスは、"*<command_name>:C<command_number>* です。例えば、現在時刻**Current time**を呼び出したいときには、"**Current time:C178**"と表記します。

注: コマンド番号はエクスプローラーの**コマンド**ページにて表記があります:



評価エラーの場合、挿入されたテキストは“<!--#4DEVAL expression--> : ## error # error code” の形式になります。

注: Webリクエスト経由で4Dメソッドに4DEVALを用いる場合には、メソッドプロパティにて"4D タグや URL(4DACTION...)からの利用を許可"オプションがチェックされている必要があります。この点についてのより詳細な説明は、**Connection Security** の章を参照して下さい。

4DLOOP

4DLOOP タグは、二つの新しい条件を受け取れるようになりました(他は*table*、*array* そして*method* です): それは**4D 式**と

ポインター配列です。

- <!--#4DLOOP 4DExpression-->
  このシンタックスを使用すると、4DLOOPタグは4D式がTrueを返すまでループを繰り返します。式部分には、無限ループを避けるために毎ループに評価されるべき変数部分を含んだ、有効なブール式が必要です。

例えば、以下のコードは:

```
<!--#4DEVAL $i:=0--> <!--#4DLOOP ($i<4)--> <!--#4DEVAL $i--> <!--#4DEVAL $i:=$i+1--> <!--#4DENDLOOP-->
```

以下の結果を生成します:

```
0
1
2
3
```

- <!--#4DLOOP pointerArray-->
  この場合、4DLOOPタグは配列に対して用いたのと同様に振る舞います: 配列の各要素ごとにループを作成します。カレントの配列の要素はコードの一部が繰り返されるたびに増加していきます。

このシンタックスは、配列ポインターを**PROCESS 4D TAGS** コマンドの引数として渡す場合に有効です。

例:

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
PROCESS 4D TAGS($input;$output;"elements = ";->$array)
 // $output = "elements = hello world"
```

Application version

Application version {(buildNum {; *})} -> 戻り値

説明

*4D v14 R2*から導入

4Dの新しい"R"リリースをサポートするため、**Application version** コマンド("**4D環境**" テーマ)で返される値が更新されました。

原則として、"R"リリースの場合、"R"番号がリリース番号となり、リビジョン番号は常に"0"となります。この原理は長いバージョン番号でも短いバージョン番号でも適用されます。

短いバージョン番号での例です:

```
value:=Application version // 短いバージョン番号
```

バージョン	返り値	
4D v14 R2	"1420"	
4D v14 R3	"1430"	
4D v14.1	"1401"	4D v14の最初のバグフィックスリリース
4D v14.2	"1402"	4D v14の2つ目のバグフィックスリリース

長いバージョン番号での例です:

```
value:=Application version(*) // 長いバージョン番号
```

バージョン	返り値
4D v14 beta R2	"B0011420"
4D v14 final R3	"F0011430"
4D v14.1 beta	"B0011401"

例題

コマンドから返されたアプリケーションの短いバージョン番号の値を使用して4Dアプリケーションのリリース名を表示したい場合を考えます。以下の様に書くことが出来ます:

```
C_LONGINT($Lon_build)
C_TEXT($Txt_info;$Txt_major;$Txt_minor;$Txt_release;$Txt_version)

$Txt_version:=Application version($Lon_build)

$Txt_major:=$Txt_version[[1]]+$Txt_version[[2]] //バージョン番号、この場合は14
$Txt_release:=$Txt_version[[3]] //Rx
$Txt_minor:=$Txt_version[[4]] //.x

$Txt_info:="4D v"+$Txt_major
If($Txt_release="0") //4D v14.x
   $Txt_info:=$Txt_info+Choose($Txt_minor#"0";"."+$Txt_minor;"")

Else //4D v14 Rx
   $Txt_info:=$Txt_info+" R"+$Txt_release
End if
```

DELAY PROCESS

DELAY PROCESS (process ; duration)

引数	型		説明
process	倍長整数	->	プロセス番号
duration	実数	->	ticks数で表現された遅延時間

説明

*4D v14 R3*から導入

DELAY PROCESS command コマンドは、*duration* 引数にて実数型の値を受け入れるようになりました(以前のリリースでは、倍長整数型の値のみを受け入れ可能でした)。

時間の単位は変わらずtick(1 tick = 1/60 秒)ですが、プロセスの実行を遅らせる時間に1tick以下の時間を指定できるようになりました。例えば、0.5を*duration* に渡した場合、プロセスは1/2tickだけ、つまり1/120 秒だけ遅れます。

FORM GET OBJECTS

FORM GET OBJECTS (objectsArray {; variablesArray {; pagesArray}} {; * | *formPageOption*})

引数	型		詳細
objectsArray	文字列配列	<-	フォームオブジェクト名
variablesArray	ポインター配列	<-	オブジェクトに関連付けられた変数またはフィールドへのポインター
pagesArray	整数配列	<-	各オブジェクトのページ番号
* | *formPageOption*	演算子 | 倍長整数	->	* 指定時=カレントページまで減らす、または *1=Form current page, 2=Form all pages, 4=Form inherited*

説明

*4D v14 R2*から導入

FORM GET OBJECTS コマンド("**フォーム**" テーマ)は新しい *formPageOption* 引数を受け取るようになりました。これにより、オブジェクトをどこから取得したいのかというフォームの部分を指定することができるようになります。

デフォルトでは、*formPageOption* 引数(と* 演算子)が省略された場合継承されたオブジェクトを含むすべてのページからのオブジェクトが返されます(以前の4Dのリリースと同じ挙動です)。

コマンドのスコープを限定するためには*formPageOption* に値を渡します。以下の値のうちどれか一つ(またはそれらの組み合わせ)を使用することが出来ます。これらの値は"**Form Objects (Access)**" テーマ内にあります:

定数	型	値	詳細
Form current page	倍長整数	1	カレントページの全てのオブジェクトを返します。0ページ目も含めますが、継承されたオブジェクトは除きます。
Form all pages	倍長整数	2	全てのページの全てのオブジェクトを返しますが、継承されたオブジェクトは除きます。
Form inherited	倍長整数	4	継承されたオブジェクトのみを返します。

注: * 演算子を渡す事はForm current page+Form inherited の組み合わせを渡す事と等価です。* 演算子を使用したシンタックスは、今後は使用されるべきではありません。

例題 1

継承されたフォームのオブジェクトも含めて(もしあれば)、全てのページの情報を取得したい場合:

```
  //開いているフォーム
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray)
```

または:

```
  //ロードしたフォーム
FORM LOAD([Table1];"MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages+Form inherited)
```

例題 2

カレントページに関する情報だけを取得し、ロードされたフォームのページ0と継承されたフォームオブジェクトも(もしあれば)含めたい場合:

```
FORM LOAD("MyForm")
```

```
FORM GOTO PAGE(2)
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form current page+Form inherited)
```

例題 3

継承されたフォーム内の全てのオブジェクトの情報が(もしあれば)取得したい場合(ただい、もし継承されたフォームがない場合には空の配列が返されます):

```
FORM LOAD("MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form inherited)
```

例題 4

0ページ目のオブジェクトも含め、4ページ目のオブジェクトの情報を取得し、継承されたフォームオブジェクトに関しては(もしあれば)除外したい場合:

```
FORM LOAD([Table1];"MyForm")
FORM GOTO PAGE(4)
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form current page)
```

例題 5

全てのページのオブジェクトの情報を取得し、継承されたフォームオブジェクトに関しては(もしあれば)除外したい場合:

```
FORM LOAD([Table1];"MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages)
```

FORM GET PROPERTIES

FORM GET PROPERTIES ({{aTable*{;formName ;}* width ; height {; numPages {; fixedWidth {; fixedHeight {; title}}}})

---

*4D v14 R2*から導入
formName parameter引数が任意となりました。この引数が省略された場合、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へと適用されます。
参照*:* **FORM GET PROPERTIES**

GET ACTIVITY SNAPSHOT

GET ACTIVITY SNAPSHOT (arrActivities | arrUUID ; arrStart ; arrDuration ; arrInfo {; *arrDetails*}{; *})

引数	型		説明
arrActivities | arrUUID	オブジェクト配列、テキスト配列	<-	オペレーションの完全な詳細(オブジェクト配列)または オペレーションのUUID(テキスト配列)
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	<-	オペレーションの経過時間(秒)
arrInfo	テキスト配列	<-	詳細
arrDetails	オブジェクト配列	<-	コンテキストの詳細と サブオペレーション(あれば)
*	演算子	->	渡された場合、サーバーのアクティビティを取得

新しいプロパティ

*4D v14 R3*から導入

このコマンドは、4D Server管理ウィンドウのReal Time Monitor (RTM)ページの進化に伴い、追加の情報を返すようになりました(**リアルタイムモニタリングページ** を参照して下さい)。

追加の情報は、二つの新しいプロパティ内に返されます:**dbContextInfo** と **dbOperationDetails** です。このプロパティはどちらのシンタックスにも有効です。

GET ACTIVITY SNAPSHOT(arrActivites{; *})

このシンタックスでは、新しいプロパティは*arrActivities* オブジェクトの第1階層にあります:

```
[
    {
        "message":"xxx",
        "maxValue":12321,
        "currentValue":63212,
        "interruptible:0,
        "remote":0,
        "uuid":"deadbeef",
        "taskId":xxx,
        "startTime":"2014-03-20 13:37:00:123",
        "duration":92132,
        "dbContextInfo":{
            "task_id": xxx,
            "user_name": myName,
            "host_name": HAL,
            "task_name": "CreateIndexLocal",
            "client_uid": "DE4DB33F33F"
            "user4d_id ": 1,
            "client_version ": 123456
        },
        "dbOperationDetails":{
            table: "myTable"
            field: "Field_1"
        },
        "subOperations":[...]
    },...
]
```

GET ACTIVITY SNAPSHOT(arrUUID;arrStart;arrDuration;arrInfo{;arrDetails}{; *})

このシンタックスでは、新しいプロパティは最後の引数としてあらわれ、分かりやすさのために*arrDetails* と改名されていま

す。この配列の構造は以下のようになっています:

```
[
    {
        "dbContextInfo":{...} //以下を参照の事
        "dbOperationDetails":{...} //以下を参照の事
        "subOperations":[...]
    },...
]
```

説明

dbContextInfo

このプロパティは、オペレーションがデータベースエンジンによって取り扱われた際に情報を管理するオブジェクトです。これには以下の様なプロパティが含まれます:

- *host_name* (文字列): オペレーションを開始したホストの名前
- *user_name* (文字列): オペレーションを開始したセッションの4Dユーザー名
- *task_name* (文字列): オペレーションを開始したプロセス名
- *task_id* (数値): オペレーションを開始したプロセスのID番号
- *client_uid* (文字列): 任意。オペレーションを開始したクライアントのUUID
- *is_remote_context* (ブール、0 または 1): 任意。データベースオペレーションがクライアントによって開始されたのか(値1)、ストアドプロシージャーを通してサーバーからから開始されたのか(値0)を表します。
- *user4d_id* (数値): クライアント側のカレントの4DユーザーのID番号
- *client_version* (文字列): アプリケーションの4Dエンジンのバージョンを表す4桁の数字。 **Application version** コマンドで返されるものと同じ。

client_uid と *is_remote_context* は、クライアント/サーバーモードでのみ使用可能です。*client_uid* はクライアントマシンからデータベースオペレーションが開始された場合のみ返されます。

dbOperationDetails

このプロパティはオペレーションがデータベースエンジン使用した場合にのみ返されます(例えばクエリや並べ替えなどが相当します)。これは特定のアクティビティ情報を定義するオブジェクトです。返されるプロパティは、実行されたデータベースオペレーションのタイプによります。具体的には、プロパティには以下が含まれます:

- *table* (文字列): オペレーションに関連したテーブルの名前
- *field* (文字列): オペレーションに関連したフィールドの名前
- *queryPlan* (文字列): クエリの内部プラン

Get database parameter と SET DATABASE PARAMETER

Get database parameter ({aTable ;} *selector* {; stringValue}) -> 戻り値

SET DATABASE PARAMETER ({aTable ;} *selector* ; value)

説明

Get database parameter と **SET DATABASE PARAMETER** コマンドにおいて、新しい*selector* 引数が使えるようになりました:

定数	型	値
Use legacy network layer	倍長整数	87

- **スコープ**: ローカルモードの4Dおよび 4D Server
- **2セッション間で設定を保持**: Yes
- **説明**: クライアント/サーバー通信における旧式ネットワークレイヤーの状態を設定、または取得します。旧式ネットワークレイヤーは4D v14 R5以降廃止予定となり、*ServerNet* ネットワークレイヤーへと積極的に置き換えられるべきです。*ServerNet* は今後の4Dリリースにおいて、将来のネットワークの進化の利益を享受できるようにするために必須要項となる予定です。互換性の観点から、既存のアプリケーションのスムーズな移行のために旧式ネットワークレイヤーは引き続きサポートはされます(v14 R5以前の4Dから変換されたアプリケーションにおいてはデフォルトで使用されます)。この引数に1を渡すと、クライアント/サーバー通信において旧式ネットワークレイヤーが使用されます(同時に*ServerNet* は無効化されます)。0を渡すと、旧式ネットワークレイヤーが無効化されます(同時に*ServerNet* が使用されます)。
  このプロパティは、データベース設定の"互換性"ページの"旧式ネットワークレイヤーを使用する"のオプションを使用することによっても設定することができます(**新しいServerNetネットワークレイヤー** を参照して下さい)。この章では、以降の戦略についてのディスカッションを見る事もできます。**新しいServerNetネットワークレイヤー**をなるべく早く有効化することが推奨されます。
  この引数の設定が使用されるためには、アプリケーションの再起動が必要です。この引数は*ServetNet のみ*をサポートするOS X用の4D Server v14 R5 64-bit では使用できません(値は常に0を返します)。
- **取り得る値**: 0 または 1 (0 = 旧式ネットワークレイヤーを使用しない、1 = 旧式ネットワークレイヤーを使用する)
- **デフォルト値**: 4D v14 R5 以降で作成されたデータベースにおいては0、4D v14 R4以前から変換されたデータベースにおいては1。

説明

*4D v14 R3*から導入

Get database parameter と **SET DATABASE PARAMETER** コマンドにおいて、新しい*selector* 引数を使用することが出来るようになりました:

定数	型	値
SQL Server Port ID	倍長整数	88

- **スコープ**: 4D ローカル、4D Server
- **2セッション間で設定を保持**: Yes
- **詳細**: 4Dローカル/4D Server と 4DビルトインのSQLサーバー間で使用されるTCPポート番号を設定または取得します。デフォルトでは、この値は19812 に設定されています。TCPポート番号はデータベース設定ダイアログボックス内の"SQL"ページでも設定する事ができます。このセレクターが使用される と、データベース設定はこれに従って書き換えられ、閉じて再起動した後も保持されます。
- **取り得る値**: 0 -> 65535
- **デフォルト値**: 19812

説明

*4D v14 R5*から導入

Get database parameter と **SET DATABASE PARAMETER** コマンドにおいて、新しい*selector* 引数が使えるようになりました:

定数	型	値
Use legacy network layer	倍長整数	87

- **スコープ**: ローカルモードの4Dおよび 4D Server
- **2セッション間で設定を保持**: Yes
- **説明**: クライアント/サーバー通信における旧式ネットワークレイヤーの状態を設定、または取得します。旧式ネットワークレイヤーは4D v14 R5以降廃止予定となり、*ServerNet* ネットワークレイヤーへと積極的に置き換えられるべきです。*ServerNet* は今後の4Dリリースにおいて、将来のネットワークの進化の利益を享受できるようにするために必須要項となる予定です。互換性の観点から、既存のアプリ ケーションのスムーズな移行のために旧式ネットワークレイヤーは引き続きサポートはされます(v14 R5以前の4Dから変換されたアプリケーションにおいてはデフォルトで使用されます)。この引数に1を渡すと、クライアント/サーバー通信において旧式 ネットワークレイヤーが使用されます(同時に*ServerNet* は無効化されます)。0を渡すと、旧式ネットワークレイヤーが無効化されます(同時に*ServerNet* が使用されます)。
  こ のプロパティは、データベース設定の”互換性”ページの”旧式ネットワークレイヤーを使用する”のオプションを使用することによっても設定することができ ます(**新しいServerNetネットワークレイヤー** を参照して下さい)。この章では、以降の戦略についてのディスカッションを見る事もできます。[#title id="3809"/]をなるべく早く有効化することが推奨されます。
  この引数の設定が使用されるためには、アプリケーションの再起動が必要です。この引数は*ServetNet のみ*をサポートするOS X用の4D Server 64-bit では使用できません(値は常に0を返します)。
- **取り得る値**: 0 または 1 (0 = 旧式ネットワークレイヤーを使用しない、1 = 旧式ネットワークレイヤーを使用する)
- **デフォルト値**: 4D v14 R5 以降で作成されたデータベースにおいては0、4D v14 R4以前から変換されたデータベースにおいては1。

GET FIELD PROPERTIES

GET FIELD PROPERTIES (fieldPtr | tableNum {; fieldNum}; *fieldType* {; fieldLength {; indexed {; unique {; invisible}}}})

説明

4D v15 以降、フィールドに新しいObject型が使用できるようになりました。詳細な情報に関しては、 **オブジェクトフィールドデータ型** の章を参照して下さい。

結果として、このコマンドは*fieldType* 引数に対し、"Field and Variable Types"定数からの新しい値を返す事ができるようになりました:

定数	型	値
Is object	倍長整数	38

Get pointer

Get pointer (varName) -> 戻り値

説明

Get pointer は4D v15において、新しいポインター型をサポートするために書きなおされました(詳細については **メソッドでのUnicode** を参照して下さい)。以下の変更点に注意して下さい:

- 式を含め、二次元配列へのポインターがサポートされるようになりました:

```
$pt:=Get pointer("a{1}{2}")
 //$pt=->a{1}{2}
$pt2:=Get pointer("atCities"+"{2}{6}")
 //$pt2=->atCities{}{6}
```

- 無効な変数名はエラー77("変数名が不正です。")を生成するようになりました。以前のリリースでは、そういった変数名も使用可能でした。例えば:

```
$pt:=Get pointer("123") //v15以前の4Dでは使用可能
 // 4D v15 以降のバージョンでは拒否されます
```

- 余分な空白もサポートされるようになりました:

```
$pt:=Get pointer(" a ") //4D v15では使用可能: "->a"
 // v15より前の4Dでは使用不可
```

Is licence available

Is license available {(license)} -> 戻り値

引数	型		説明
license	倍長整数	->	ライセンスの有効性テストを行う機能
戻り値	ブール	<-	その機能が利用可能な場合はTrue、それ他の場合はFalse

説明

*4D v14 R3*から導入

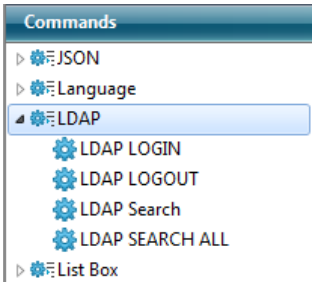
Is licence available コマンドは、"Is License Available"定数テーマに追加された二つの新しい定数を受け入れるようになりました:

定数	型	値
4D Mobile License	倍長整数	808464439
4D Mobile Test License	倍長整数	808465719

注意: 4D Developer Proには、デフォルトでテスト目的のライセンスが3つ付属してきます。

LDAP 新コマンドテーマ

4D v15では新しいLDAP機能が追加され、これにより4Dアプリケーションををを使用して既存の会社のディレクトリへと接続することができます。新しい**LDAP**コマンドはLDAPという名前の新しいテーマに追加されています:



これらのコマンドの詳細については、このアップグレードマニュアル内の専用の"**カンパニーディレクトリ(LDAP)**"セクションにて説明があります。

MAXIMIZE WINDOW

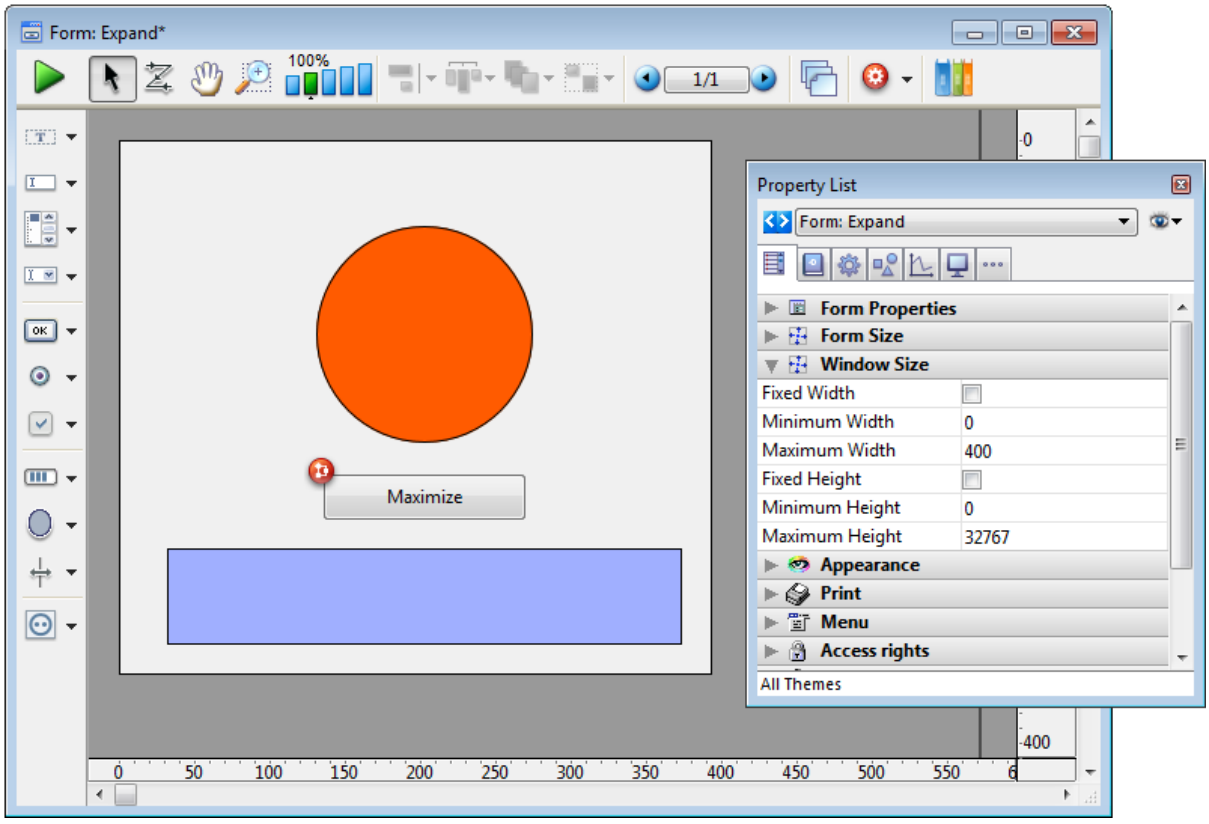
*4D v14 R3*から導入

Windowsにおける **MAXIMIZE WINDOW** コマンドの挙動が一部変更になりました。サイズに制約のあるウィンドウ(例えばフォームウィンドウ)に適用された際には以下の様に挙動が変わります:

- サ イズ制約がターゲットサイズと競合しない場合、ウィンドウサイズはこれまでのリリース同様”最大化”の状態に設定されます(つまりウィンドウはマルチ・ドキュメント・インタフェース(MDI)の親 ウィンドウのサイズと一致するようにリサイズされ、タイトルバーと枠は非表示になり、タイトルバーのボタン(最小化、復元、閉じる)はアプリケーションメニューバーの右側に置かれます)。
- しかし、どれか一つでもサイズ制約に引っかかる(例えばMDIウィンドウ幅が100でフォームウィン ドウ幅の最大値が80であった)場合には、ウィンドウは"最大化"の状態には設定されず、その制約された範囲内での最大のサイズへとリサイズされます。こ のサイズはMDIサイズまたは制約サイズによって決定されます。
  この新しい挙動によって、サイズ制約付きのウィンドウがリサイズされたときにもインターフェースは一様に保たれます。

例題

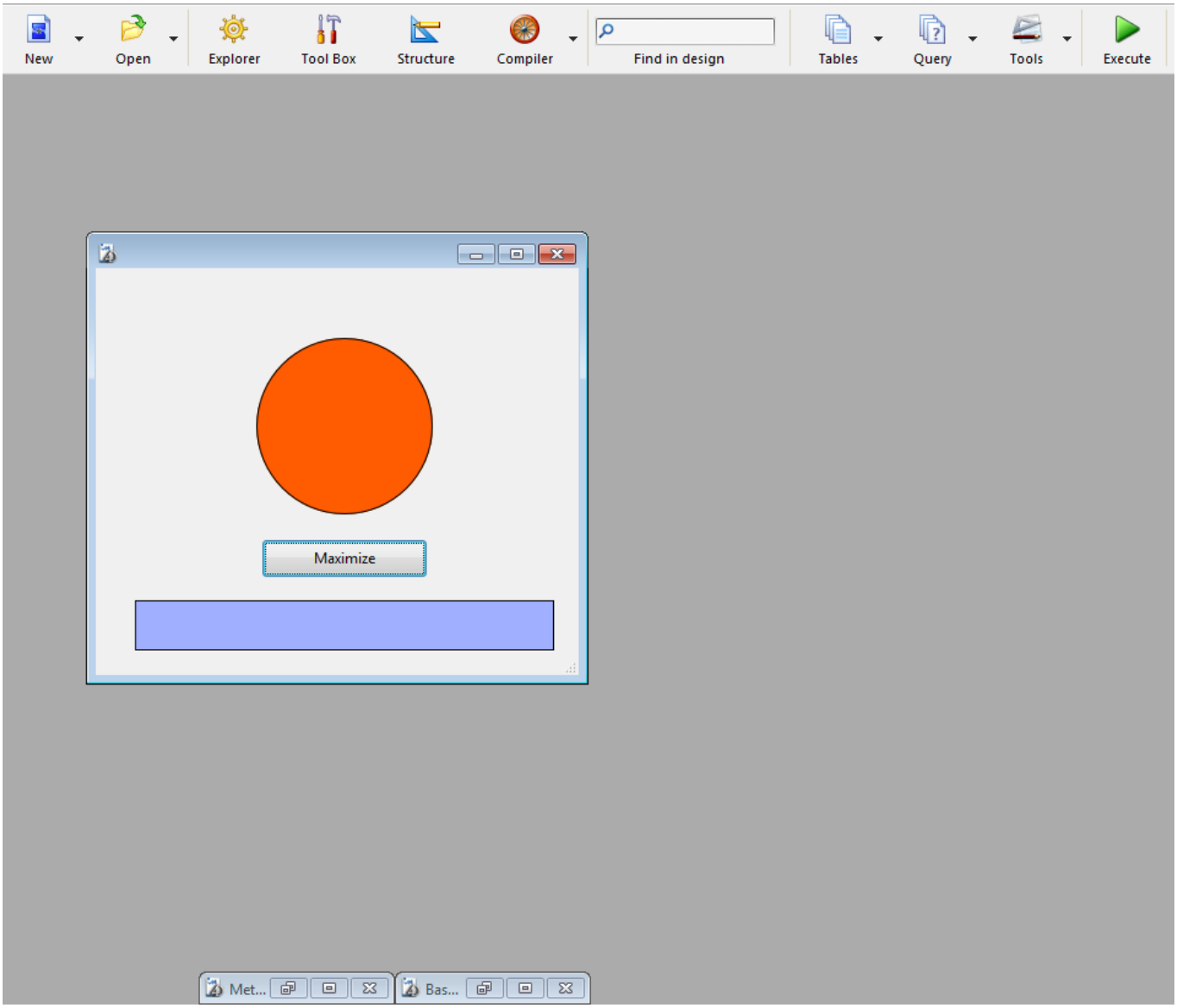
以下のフォームはサイズ制約付きで定義されています(最大幅=400):



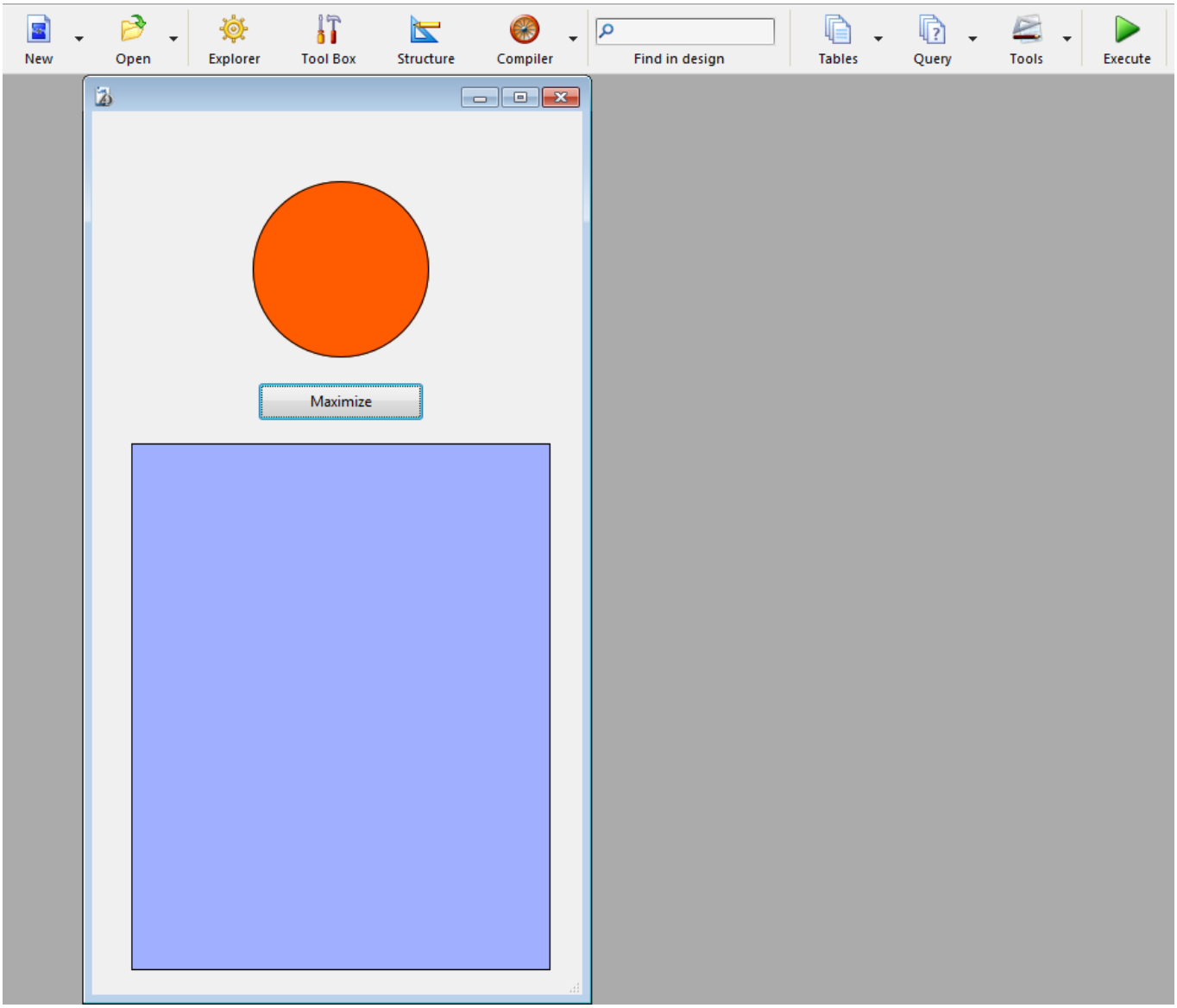
ボタンのコードには以下のように書かれています:

```
MAXIMIZE WINDOW(Current form window)
```

この状態において、ユーザーがボタンをクリックすると:



... ウィンドウは"最大化"はされず、高さだけが以下の様に引き伸ばされます:


New
Open
Explorer
Tool Box
Structure
Compiler
Find in design
Tables
Query
Tools
Execute
Maximize

METHOD GET CODE and METHOD SET CODE

```
METHOD GET CODE ( path ; code {; *} )
METHOD SET CODE ( path ; code {; *} )
```

説明

METHOD GET CODEと**METHOD SET CODE** コマンドはどちらも新しい属性metadataをサポートするようになりました。これにより、4D Mobile 関連のプロパティを取得・設定できるようになりました。

"published4DMobile"という名前の新しい属性が%attribute ラインに追加され、含めることができるようになりました:

```
// %attribute = { "published4DMobile": { "table": "aTableName", "scope": "table" } }
```

"scope"というプロパティは以下の値を受け取ることができます: "none"、"table"、"currentRecord" または"currentSelection"

サポートされる属性は以下の通りです:

```
{     "invisible" : false, // 表示状態であればtrue     "publishedWeb" : false,  // 4D tags と URLを通して使用可能であればtrue     "publishedSoap": false,  // Webサービスとして提供されていればtrue     "publishedWsdl": false,  // WSDLで公開されていればtrue     "shared" : false,  // コンポーネントとホストデータベースで共有されていればtrue     "publishedSql" : false,  // SQLを通して利用可能であればtrue     "executedOnServer" : false, // サーバー側で実行されていればtrue     "published4DMobile" : { "scope": "table",  // "none" または "table" または "currentRecord" または "currentSelection"         "table": "aTableName"  // scope が "none" 以外の場合には表示される     } }
```

注: "published4DMobile" 属性については、"table"の値が存在しない、または"scope" が有効でない場合には、これらの属性は無視されます。

OBJECT GET COORDINATES

OBJECT GET COORDINATES ({* ;} object ; left ; top ; right ; bottom)

説明

*4D v14 R5*から導入

テーマ: オブジェクト(フォーム)

OBJECT GET COORDINATES コマンドは特定の親のリストボックスオブジェクトだけではなく、リストボックスのパーツ(カラム、ヘッダーまたはフッター)の座標を返す事ができるようになりました。

以前のリリースでは、このコマンドは適用されたリストボックスのパーツに関わらず、常に親のリストボックスの座標を返していましたが、現在では*object* 引数で参照しているオブジェクトがリストボックスヘッダー、カラム、またはフッターサブオブジェクトの場合、返される座標は指定されたそれらのサブオブジェクトのものとなります。

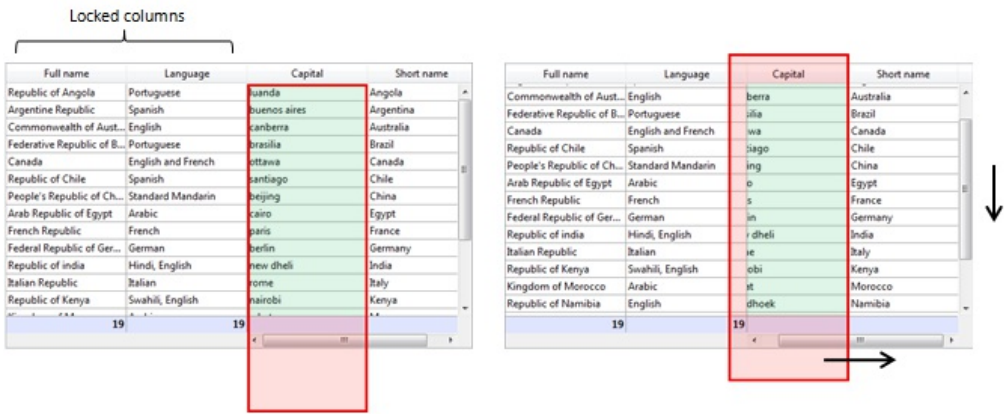
この新機能を使用して、例えばリストボックスのヘッダーにマウスオーバーしたときにそこに小さなアイコンを表示させ、ユーザーがそれをクリックするとコンテキストメニューが表示される、といったようなこともできるようになります。

互換性に関する注意: 既存のアプリケーションにおいてこのコマンドをリストボックスヘッダー、フッター、またはカラムに対して使用している場合、4D v14 R5以降のバージョンに変換した際には返される座標が異なります。リストボックス全体の座標を取得したい場合には、コードを修正してコマンドがサブオブ ジェクトに対してではなく、リストボックスオブジェクト自身に対して呼び出されているようにしなければなりません。

統一性のために、対象の オブジェクトがリストボックスサブオブジェクトであってもリストボックスオブジェクトであっても、使用される参照フレームは同じとなっています。原点はそ のオブジェクトを含むフォームの左上端となっています。リストボックスサブオブジェクトに対しては、返される値は理論値となり、クリッピングが起きない限 りリストボックスのスクロールの状態を考慮に入れます。結果として、サブオブジェクトはその座標において(一部または全部が)見えないこともあり、これら の座標がフォームの端より外側にあることもありえます(または、負の値を返す事も有り得ます)。サブオブジェクト(の一部または全部が)見えているかどう かを調べるためには、リストボックスの座標と比較する必要があります。その際、以下のルールが適用されます:

- 全てのサブオブジェクトは親のリストボックスの座標(**OBJECT GET COORDINATES** がリストボックスに対して返す値)に沿って切り取られます。
- ヘッダーとフッターのサブオブジェクトはカラムの中身の上に表示されます。カラムの座標とヘッダーまたはフッターの座標が交錯した場合、その重なった部分においてはカラムは表示されません。
- ロックされたカラムの要素はスクロール可能なカラムの要素の上に表示されます。スクロール可能なカラムの要素がロックされたカラムの座標と交錯した場合、スクロール可能なカラムの要素はこの重なった部分では表示されません。

具体例として、以下の画像において、*Capital* のカラムに注目して下さい(このカラムを囲っている赤い四角がこのカラムの座標を表しています):

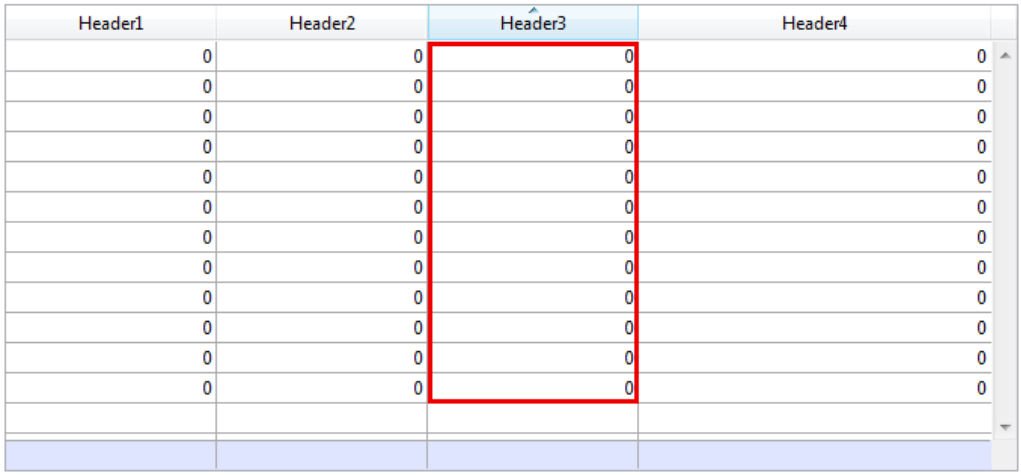


1枚目の画像にあるように、カラムはリストボックスより大きいので、カラムの座標はリストボックス(とフッター)の下限より下まで広がっています。2枚目の画像では、リストボックスはスクロールされており、その結果このカラムは*Language* カ

ラムとヘッダーエリアの"下"へと移動します。どちらの場合においても、実際に表示されている部分(緑のエリア)を計算するためには、赤いエリアを計算に入れる必要があります。

例題

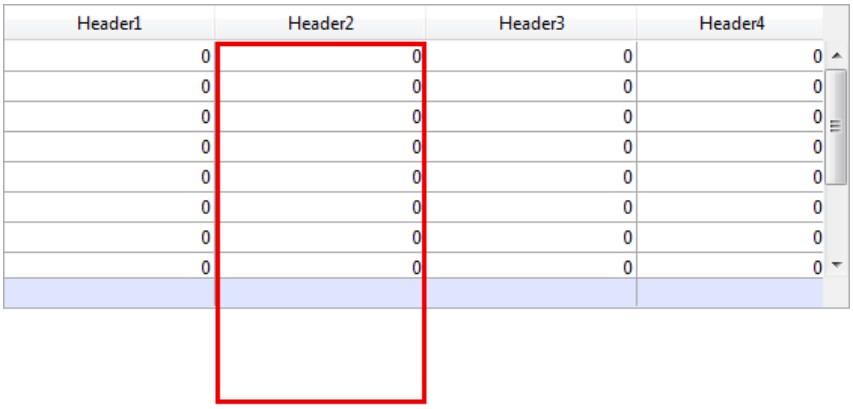
インターフェースのために、クリックされたエリアを赤い四角で囲いたい場合を考えます:



リストボックスのオブジェクトメソッドにおいて、以下の様に記述します:

```
OBJECT SET VISIBLE(*;"rectangleInfo";False) //赤い四角を初期化
$ptr:=OBJECT Get pointer(Object current)
OBJECT GET COORDINATES($ptr->;$x1;$y1;$x2;$y2)
OBJECT SET VISIBLE(*;"RedRect";True)
OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)

OBJECT GET COORDINATES(*;"LB1";$lbx1;$lby1;$lbx2;$lby2)
If($lby1>$y1)|($lby2<$y2) // クリックされたエリアがリストボックスの外側にある場合、
   OBJECT SET VISIBLE(*;"Alert";True) //警告を表示します
Else
   OBJECT SET VISIBLE(*;"Alert";False)
End if
```

メソッドは座標の理論値を返します。リストボックスがリサイズされた場合、どの部分が表示されているのかを調べるためにクリッピングを再計算が必要になる場合もあります:



On Scroll フォームイベント

On Picture Scrollの変更

以前の4Dのバージョンでは、ピクチャー変数またはフィールドのスクロールはOn Picture Scroll フォームイベント(値59)を通して管理することができました。

4D v15では、この種のイベントの管理がリストボックスオブジェクトまで拡張されました(以下参照)。その結果として、On Picture Scroll フォームイベントはOn Scroll と改名され、リストボックスおよびプロパティリストに対しても使用できるようになりました。ピクチャーに対するこのイベントの動作は変わりません。

互換性に関する注意:
以前のバージョンの4DでのOn Picture Scroll フォームイベントの実装と、新しいのOn Scroll 実装について二つの小さな変更点があります:

- On Picture Scroll はオブジェクトメソッドとフォームメソッドにおいて生成されました(フォームプロパティレベルでのチェック、あるいはチェックを外すことはできませんでした)。統一性のために、4D v15以降、On Scroll イベントはオブジェクトメソッドにおいてのみ生成されます。フォームメソッドでピクチャースクロールイベントを管理しているアプリケーションを変換した場合、そのコードを適切なオブジェクトメソッドへと移す必要があります。
- イベントスタックにおいては、On Picture Scroll を他のユーザーイベント(例えばOn Click等)の前に呼び出すことは可能です。On Scroll は常に他のユーザーイベントの後に生成されます。

On Scroll フォームイベント

4D v15**On Scroll** イベントは二つのスクロール可能なオブジェクトに対して使用可能です:

- "トランケート(中央合わせなし)"フォーマットのピクチャーフィールドまたは変数(既に以前の4Dのバージョンから On Picture Scroll という名前で使用可能)。
- リストボックス(4D v15から)

デフォルトでは、オブジェクトに対してのイベントはプロパティリスト内ではチェックされていません。

On Scroll フォームイベントはユーザーがフォームオブジェクトを、それが含まれるエリア内にてスクロールした瞬間に生成されます。イベントはスクロールがユーザーのアクション(スクロールバー、またはマウスホイールやキーボードの使用)の結果である場合においてのみ生成されます。オブジェクトが**OBJECT SET SCROLL POSITION** コマンドの実行の結果スクロールされた場合には生成されません。

このイベントはスクロールにアクションに関連する他の全てのイベント(On Clicked、On After Keystroke、等)のあとにトリガーされます。

このイベントはオブジェクトメソッドないにおいてのみ生成されます(フォームメソッド内では生成されません)。

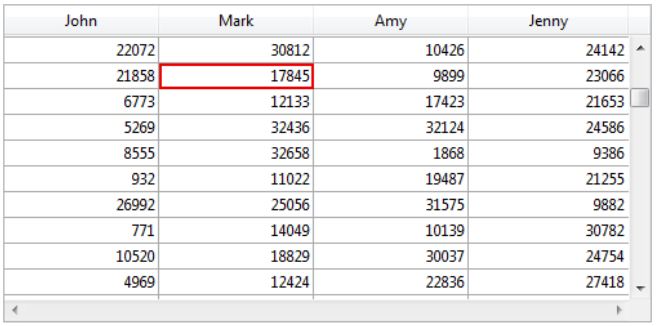
例題

リストボックスで選択されたセルの周りに赤い長方形を描画し、リストボックスがユーザーによって垂直方向にスクロールされた場合には、その長方形を一緒に移動させたい場合を考えます。その場合、リストボックスのオブジェクトメソッドに対して以下のように書きます:

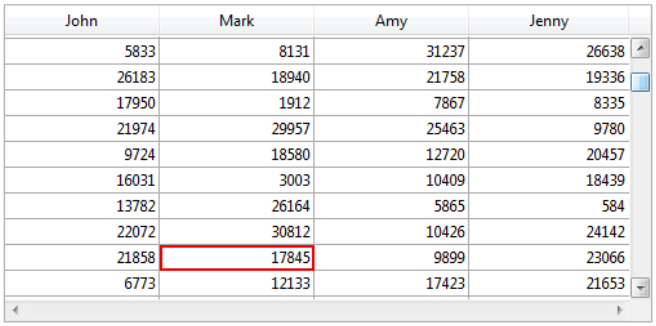
```
Case of

   :(Form event=On Clicked)
      LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
      LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
      OBJECT SET VISIBLE(*;"RedRect";True)  //赤い長方形を初期化
      OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)

   :(Form event=On Scroll)
      LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
      LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
      OBJECT GET COORDINATES(*;"LB1";$xlb1;$ylb1;$xlb2;$ylb2)
      $toAdd:=LISTBOX Get headers height(*;"LB1") //オーバーラップしないためにヘッダーの高さを取得
```

```
            If($ylb1+$toAdd<$y1) & ($ylb2>$y2) //リストボックス内にいるとき
        //単純かのため、ここではヘッダーのみを扱います
        //実際にはスクロールバーに加え、
        //水平方向のクリッピングも管理しなければなりません。
               OBJECT SET VISIBLE(*;"RedRect";True)
               OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)
            Else
               OBJECT SET VISIBLE(*;"RedRect";False)
            End if

    End case
```

結果として、赤い長方形はリストボックスのスクロールに沿って移動します:



John	Mark	Amy	Jenny
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653
5269	32436	32124	24586
8555	32658	1868	9386
932	11022	19487	21255
26992	25056	31575	9882
771	14049	10139	30782
10520	18829	30037	24754
4969	12424	22836	27418



John	Mark	Amy	Jenny
5833	8131	31237	26638
26183	18940	21758	19336
17950	1912	7867	8335
21974	29957	25463	9780
9724	18580	12720	20457
16031	3003	10409	18439
13782	26164	5865	584
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653

OPEN URL (OPEN WEB URL の新しい名前)

OPEN URL (*path {;appName} {; *}*)

引数	型		説明
path	文字列	->	開きたいドキュメントへのパスまたは URL
appName	文字列	->	使用するアプリケーション名
*	演算子	->	指定時にはURLは翻訳されます。未指定の場合にはURLは翻訳されません。

説明

*4D v14 R3*から導入

互換性に関する注意*: OPEN URL* は、***OPEN WEB URL*** コマンドの新しい名前です*("システム環境"テーマ)*。このコマンドは、機能が拡張されました。

OPEN URL コマンドは、新しい*appName* 引数を受け取るようになりました。これによりドキュメントまたはURLを開く際に使用するアプリケーションを指定できるようになりました。

この引数が指定された場合、コマンドは指定された名前を持つアプリケーションを検索するようにシステムに要求します。そして見つかった場合には、そのアプリケーションを起動して、指定されたURLまたはドキュメントを開きます。

Windowsでは、アプリケーションを認識させるためのメカニズムは、Windowsの「スタート」メニュー内の「ファイル名を指定して実行」と同じです。例えば:

- Internet Explorer を起動するには"iexplore" と入力します。
- Chrome を起動するには"chrome" と入力します(インストール済みの場合)
- MS Wordを起動するには"winword" と入力します(インストール済みの場合)

注: 認識されているアプリケーションの一覧は、次のキーのレジストリにあります:
HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥App Paths

OS Xでは、Finderデータベースのメカニズムを使用するので、全てのインストール済みのアプリケーションは自動的にインデックスされています。.appアプリケーションは全てそのバンドル名(.appの有無は問いません)で認識されます。例えば:

- "safari"
- "FireFox"
- "TextEdit"

等です。

appName で指定したアプリケーションが見つからなかった場合でも、エラーは返されません。コマンドはその引数の指定がなかったものとして実行されます(その場合、システムがそのドキュメントタイプまたはURLに対して最適なアプリケーションを選択します)。

例題

以下の様に、テキストファイルを異なるアプリケーションで開く事ができます:

```
OPEN URL("C:\\temp\\cookies.txt") //ファイルをNotepadで開く
OPEN URL("C:\\temp\\cookies.txt";"winword") //ファイルをMS Word で開く(インストールされていた場合)
OPEN URL("C:\\temp\\cookies.txt";"excel") //ファイルをMS Excel で開く(インストールされていた場合)
```

PROCESS 4D TAGS

PROCESS 4D TAGS (inputData ; outputData *{;param1;...;paramN}*)

引数	型		説明
inputData	テキスト、BLOB	->	処理をする*tags* を含んだデータ
outputData	テキスト、BLOB	<-	処理されたデータ
param1...N	テキスト、日付、時間、数値、ポインター	->	実行中のテンプレートへと渡す引数

説明

*4D v14 R4*から導入

テーマ: ツール

PROCESS 4D TAGS コマンドは、実行中のテンプレートへの引数の挿入をサポートするようになりました。

PROCESS 4D TAGS は*param* 引数内に、不定数の追加の数値を受け入れます。ちょうどプロジェクトメソッドのように、引数はどんなスカラー値でも可能です(テキスト、日付、時間、倍長整数、実数...)。配列ポインターを渡す事によって配列を使用することもできます。

4D タグで処理されるテンプレートのうち、4D メソッドのように、これらの引数は$1、$2... という引数を通して使用可能です(例題を参照して下さい)。
PROCESS 4D TAGS の実行コンテキストに対して、専用のローカル変数のセットが用意されるようになり、処理中の時間ずっと読み出しと設定が可能です。

互換性に関する注意: 以前のバージョンの4Dでは、呼び出しコンテキストで定義されたローカル変数は、インタープリターモードでの **PROCESS 4D TAGS** 実行コンテキストにおいてのみアクセス可能でした。今後のバージョンではこの制約にしばられることはありません。

注: 4D では新しく4DEval タグが追加され、また4DLOOP タグはポインターを受け取れるようになりました。詳細な情報に関しては、**4D 変換タグ** のセクションを参照して下さい。

例題

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
PROCESS 4D TAGS($input;$output;"elements = ";->$array)
 // $output = "elements = hello world"
```

QR Get text property と QR SET TEXT PROPERTY

QR Get text property (area ; colNum ; rowNum ; property) -> 戻り値

引数	型		詳細
area	倍長整数	->	エリアの参照
colNum	倍長整数	->	カラム番号
rowNum	倍長整数	->	行番号
property	倍長整数	->	プロパティ番号
Function result	倍長整数	文字列	<- 選択されたプロパティの値

QR SET TEXT PROPERTY (area ; colNum ; rowNum ; property ; value)

引数	型		詳細
area	倍長整数	->	エリアの参照
colNum	倍長整数	->	カラム番号
rowNum	倍長整数	->	行番号
property	倍長整数	->	プロパティ番号
value	倍長整数	文字列	-> 選択されたプロパティの値

説明

*4D v14 R3*から導入

これらのコマンドは、テキストプロパティを定義する際に、フォント番号ではなくフォント名をサポートするように修正されました。フォント番号は4D v14シリーズ以降で廃止予定です。この変更は、プログラムからQuickDrawを除去するための4D全体の戦略の一環です。

property 引数は以下の様に挙動が変更されました:

- qr font 定数は _O_qr font. と改名されました。これは廃止予定の定数であり、今後は使用されるべきではありません (互換性は保持されますが、将来のリリースではサポートされなくなります)。

新しいqr font name 定数 (10) が"QR Text Properties" テーマに追加されました。今後フォントを指定する際には、この定数と文字列 の値 を使用して下さい。渡す事の出来るフォント名は **FONT LIST** コマンドを使用した際に返される名前です。

例題

以下のメソッドは、最初のカラムタイトルのフォント属性を定義します:

```
  // 以下のメソッドはTimesフォントを指定します:
QR SET TEXT PROPERTY(qr_area;1;-1;qr font name;"Times")
```

RESOLVE POINTER

RESOLVE POINTER (aPointer ; varName ; tableNum ; fieldNum)

引数	型		詳細
aPointer	Pointer	->	参照しているオブジェクトを受け取るためのポインター
varName	String	<-	参照している変数名または空の文字列
tableNum	Longint	<-	参照しているテーブル数または配列の要素数、または0または-1
fieldNum	Longint	<-	参照しているフィールド数、または0または-1

説明

テーマ: ランゲージ

4D v15以降、二次元配列へのポインターがサポートされるようになりました。そのため、例えば次の様にコードを書くことができます->a{1}{2}.

結果として、二次元配列の要素に対してポインターを渡すとき、*fieldNum* 引数は二次元目の要素番号を受け取ります。また、**RESOLVE POINTER** は*fieldNum* 引数対して、変数と一次元配列に関しては、0ではなくて-1を返すようになりました。

結果の一覧は以下の様になります:

参照しているオブジェクト	引数		
	varName	tableNum	fieldNum
…			
変数	変数名	-1	*-1*
配列の要素	配列名	要素番号	*-1*
二次元配列の要素	二次元配列の名前	行要素の番号	列要素の番号
…			

例題

以下に二次元配列の例があります:

```
ARRAY TEXT(atCities;100;50)
C_POINTER($city)
atCities{1}{2}:="Rome"
atCities{1}{5}:="Paris"
atCities{2}{6}:="New York"
 // ...他の値
$city:=->atCities{1}{5}
RESOLVE POINTER($city;$var;$rowNum;$colNum)
 //$var="atCities"
 //$rowNum="1"
 //$colNum="5"
```

TRANSFORM PICTURE

TRANSFORM PICTURE (picture ; operator {; param1 {; param2 {; param3 {; param4}}}})

新しい演算子

*4D v14 R2*から導入

TRANSFORM PICTURE コマンド("**ピクチャ**"テーマ)は*operator* 引数において新しいTransparency 定数を受け取るようになりました。これにより、カスタムの透過度を変換されたピクチャに適用することが出来るようになりました。

この機能は特に、廃止されたPICTフォーマットのピクチャから変換されたピクチャの透過度を操作するために設定されたものですが、どんな種類のピクチャに対しても使用することが出来ます。

新しい定数は"Picture Transformation" テーマに追加されています。これを渡すと、使用できる引数は*param1* のみになります:

定数(値)	param1	param2	param3	param4	値
Transparency (102)	RGB color	-	-	-	16進数

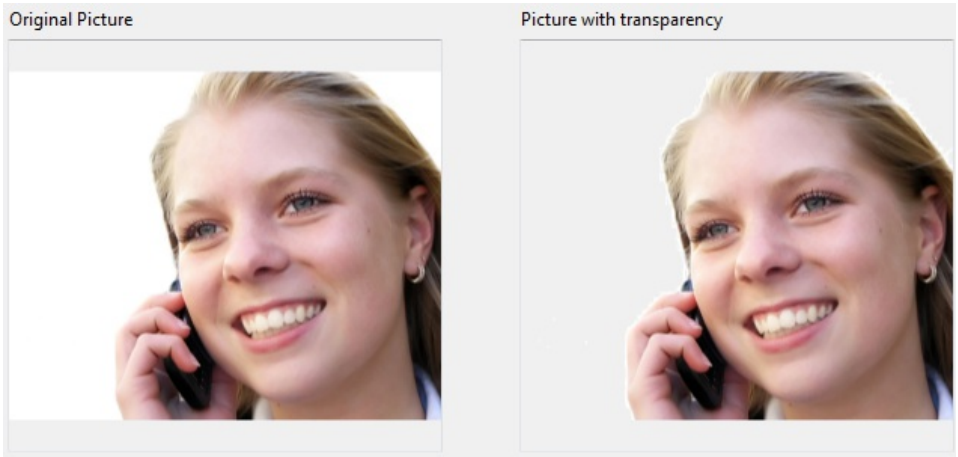
- Transparency: *param1* に渡された色に基づいて透過マスクがピクチャに適用されます。たとえば、0x00FFFFFF (白)を*param1*に渡した 場合、オリジナルのピクチャ内の白いピクセルは、変換されたピクチャの中では透明になります。この操作はビットマップピクチャにもヴェクター画像にも適用することが出来ます。デフォルトでは、*param1* 引数が省略されている場合は、白(0x00FFFFFF) がターゲットカラーとして設定されます。

例題

ピクチャの白い部分を透過にしたい場合を考えます。このためには、以下のコードを使用します:

```
TRANSFORM PICTURE(Pict1;Transparency;0x00FFFFFF) //0x00FFFFFF は白のカラーコード
```

結果は以下の様になります:



WA GET PREFERENCE と WA SET PREFERENCE

新しいデフォルトの挙動

*4D v14 R2*から導入

セキュリティ上の理由から、ファイルまたはURLをWebエリアにドロップすることによってエリアのコンテンツを変えることは、デフォルトでは不可となりました。ユーザー がファイルまたはURLをエリアにドロップしようとした場合には禁止アイコン  が表示されます。

以前のリリースでは、こういったアクションを禁止するためには特定のフィルターを、例えば **WA SET URL FILTERS**などを使用してインストールする必要がありました。

4Dでこういったドロップを許可するためには、新しいWebエリア設定、wa enable url drop を設定する必要があります。

の新しいセレクター

WA SET PREFERENCEコマンドの *selector* 引数において、WebエリアへのURLやファイルのドロップを許可するための新しい定数が追加されました:

定数	型	値	詳細
WA enable URL drop	倍長整数	101	WebエリアへのURLやファイルのドロップを有効化(デフォルト = False)

WebエリアへのURLドロップを有効化したい場合、URLをロードする前(On load フォームイベント中等が良いでしょう)にこの設定がなされている必要があります。

例題

'myarea' というWebエリア内でURLドロップを有効化したい場合:

```
WA SET PREFERENCE(*;"myarea";WA enable URL drop;True)
```

WEB GET OPTION と WEB SET OPTION

```
WEB GET OPTION ( selector ; value )
WEB SET OPTION ( selector ; value )
```

説明

*4D v14 R3*から導入

WEB GET OPTION と **WEB SET OPTION** コマンドにおいて、*selector* 引数に対して新しい値が使用できるようになりました:

定数	型	値
Web Session enable IP address validation	倍長整数	83

- **スコープ**: ローカル Web サーバー
- **2セッション間で設定を保持**: No
- **詳細**: セッション cookieに対して、IPアドレス認証を有効化または無効化します。セキュリティ上の理由から、4D Webサーバーはセッションcookieを格納しているリクエストのIPアドレスをチェックし、cookieを作成する際に使用されたIPアドレスと一致 しない場合にはそのリクエストを拒否します。一部の特定のアプリケーションにおいては、IPアドレスが合致しない場合でもセッションcookieを受け入 れるために認証を無効化したい場合があるかもしれません。例えば、モバイルデバイスがWifiから3G/4G回線へと切り替わる際には、IPは変わってし まいます。この場合、オプションに0を渡す事によって、IPアドレスが変わった場合でもそのクライアントがそのWebセッションを使用し続けられるように しなければなりません。この設定を使用することにより、アプリケーションのセキュリティレベルは下がるということに注意して下さい。
  この設定が定義されると、それは直ちに有効になります(HTTP サーバーを再起動する必要はありません)。
- **取り得る値**: 0 (無効化) または 1 (有効化)
- **デフォルト値**: 1 (IP アドレスはチェックされます)

説明

4D v15では、*selector*引数に対し、新しい定数が使用できるようになりました:Web debug log (84) です。

このセレクターは新しい4D WebサーバーHTTPリクエストログファイルの状態を取得・あるいは設定するものです。有効化された場合、"**HTTPDebugLog_*nn*.txt**"という名前のファイルがアプリケーションの"Logs"フォルダに保存されます(*nn*はファイル番号です)。これはWebサーバーに関連する事象をデバッグするのに有用です。このログはrawモードでそれぞれのリクエストとそれぞれのレスポンスを記録します。ヘッダーも含めたリクエスト全体は記録され、オプションでボディ部分もログを取ることができます。

Web debug logを*selector*に渡した場合、*value*引数に複数のオプションを記録したい情報に応じて取得、または設定することができます。以下の新しい定数は"Web Server"定数テーマ内に追加されたものです:

定数(値)	型	詳細
wdl disable (0)	倍長整数	Web HTTP デバッグログは無効化されます
wdl enable without body (1)	倍長整数	Web HTTP デバッグログはボディ部分を含めずに有効化されます(このときボディサイズは記録されます)
wdl enable with response body (3)	倍長整数	Web HTTP デバッグログは有効化され、ボディ部分はレスポンスに対してのみ有効化されます
wdl enable with request body (5)	倍長整数	Web HTTP デバッグログは有効化され、ボディ部分はリクエストに対してのみ有効化されます
wdl enable with all body parts (7)	倍長整数	Web HTTP デバッグログは有効化され、ボディ部分はレスポンスとリクエスト両方に対して有効化されます

注: HTTPリクエストログファイルは**WEB SET OPTION**コマンドを使用することによってのみ有効化/無効化できます。

例題 1

```
 // IPアドレス認証を無効化
WEB SET OPTION(Web session enable IP address validation;0)
... // 特定のコード
 // IPアドレス認証を有効化
WEB SET OPTION(Web session enable IP address validation;1)
```

例題 2

HTTP デバッグログをボディ部分なしで有効化する場合を考えます:

```
WEB SET OPTION(Web debug log;wdl enable without body)
```

ログエントリーは次のようになります:

```
# REQUEST
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089388
GET /4DWEBTEST HTTP/1.1
Connection: Close
Host: 127.0.0.1
User-Agent: 4D_HTTP_Client/0.0.0.0

# RESPONSE
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089389 (elapsed time: 1 ms)
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: close
Content-Length: 3555
Content-Type: text/plain; charset=UTF-8
Date: Tue, 20 Jan 2015 10:51:29 GMT
Expires: Tue, 20 Jan 2015 10:51:29 GMT
Pragma: no-cache
Server: 4D/14.6.0

[Body Size: 3555]
```

WEB SET HTTP HEADER

WEB SET HTTP HEADER (header|fieldArray {; valueArray})

サーバーのヘッダーが変更可能に

*4D v14 R3*から導入

WEB SET HTTP HEADER コマンドを使うことによって4DからWebクライアントへとリプライを送る際のサーバーフィールドを設定することが可能になりました。以前のリリースで は、このヘッダーは4Dによって自動的に設定されており、変更は不可能でした。

XSLT コマンドは廃止予定に

---

*4D v14 R4*から導入

XSLT 処理コマンドは廃止予定となり、以下の様に接頭辞が付きます:

以前の名前	4D v15での新しい名前
XSLT APPLY TRANSFORMATION	_o_XSLT APPLY TRANSFORMATION
XSLT GET ERROR	_o_XSLT GET ERROR
XSLT SET PARAMETER	_o_XSLT SET PARAMETER

互換性のために、XSL 変換は4Dでまだサポートされますが、その使用は推奨されません。XSLT 処理は将来の4Dのリリースで除去されます。

Mac用4D Server 64-bit 版に関する注意: XSLT は、Mac用4D Server 64-bit版では使用できません。その結果、これらのコマンドををアプリケーション内で呼び出した場合には、エラー 33 "実装されていないコマンドまたは関数です。"が生成されます。

4Dでは、データベース内のXSLTテクノロジーを置き換えるためのソリューションを二つ用意しています:

- PHP *libxslt* モジュールの、相当するファンクションを使用する方法(v14.2以降の4Dにインストールされています)。4Dでは、v14.2以降、PHP XSLを4D XSLT コマンドの代理として使用する際のヘルプドキュメントを提供しています: Download XSLT with PHP technical document (PDF)
- **PROCESS 4D TAGS** コマンドを使う方法。このコマンドは、機能が著しく拡張されました。

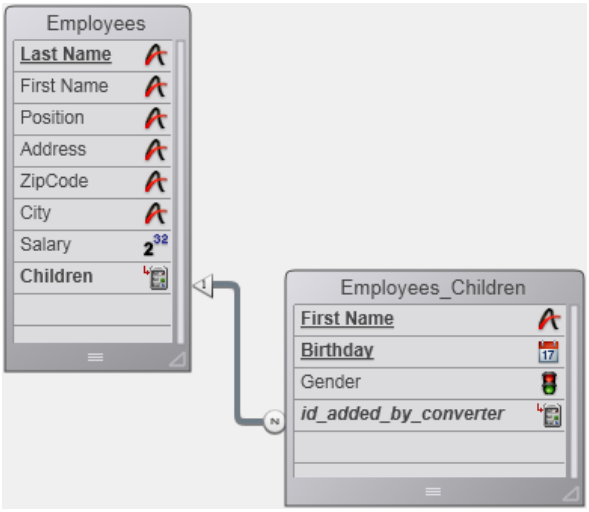
サブレコード変換

---

*4D v14 R3*から導入

v11以前のバージョンのデータベースを変換する際に4Dが自動的に追加していた特殊なフィールド "id_added_by_converter" の値を、ユーザーが割り当てる事ができるようになりました。以前のリリースでは、この値は4Dによってのみ割り当てることができ、変換されたサブテーブル に新しいレコードを追加する際には **_o_CREATE SUBRECORD** のような"廃止予定の"コマンドを使用する必要がありました。

この新 機能により、サブテーブルを使う古いデータベースをより簡単に変換する事ができるようになります。例えば、特殊な"サブテーブルリレーション"リンクをそ のままの状態で保持したり、リレートしているレコードを、標準のレコードのように追加・修正することができるようになります。全てのメソッドのアップデー トが完了すれば、コードに手を加えることなく特殊なリンクを通常なリンクと置き換えることができます。



これを使用すると、例えば以下の様に書くことができます:

```
CREATE RECORD([Employees])
[Employees]Last Name:="Jones"
CREATE RECORD([Employees_Children])
[Employees_Children]First Name:="Natacha"
[Employees_Children]Birthday:=!12/24/2013!
[Employees_Children]id_added_by_converter:=4 //旧リリースではタイプのミスマッチになる
SAVE RECORD([Employees_Children])
SAVE RECORD([Employees]
```

このコードは、特殊なリンクにも通常のリンクにも、どちらに対しても使用可能です。

ツールバーフォームウィンドウ

*4D v14 R5*から導入

4D v15 にはデベロッパがカスタムのツールバーをデザインし、管理するのを手助けするための様々な新機能が実装されています。ツールバーとは、自身の位置とサイズを管理する特定のプロパティを持ったウィンドウの事です。

以下のコマンドはツールバーの作成と管理をサポートします:

- **Open form window**: 新しい定数タイプをToolbar form window 受け付けます。
- **Tool bar height**: カスタムのツールバーの高さを返します。
- **HIDE TOOL BAR** と **SHOW TOOL BAR**: 以前は廃止予定でしたが、これらのコマンドはカスタムのツールバーを管理するために再度有効化されました。

Open form window

テーマ: ウィンドウ

Open form window ({aTable ;} formName {; *type* {; hPos {; vPos {; *}}}}) -> 戻り値

Open form window コマンドは、**Toolbar** タイプのフォームウィンドウを作成できるようになりました。

"Open Form Window" テーマに新しい定数が追加され、*type* 引数に対して使用できるようになりました:

定数	型	値
Toolbar form window	倍長整数	35

Toolbar form window 定数が渡された場合、ウィンドウはツールバーの位置、サイズ、グラフィカルプロパティで作成されます。具体的には以下の様な特性を持ちます:

- ウィンドウはメニューバー直下に常に表示されます。
- ウィンドウの横幅(水平方向の幅)は、デスクトップ上(OS X)または4Dのメインウィンドウ内(Windows)で使用可能な横幅を全て埋めるように調整されます。ウィンドウの縦方向(垂直方向の幅)は、他の全てのフォームウィンドウタイプ同様、フォームプロパティに基づいて決められます。
- ウィンドウには境界線はなく、手動で移動・リサイズはできません。表示されている間は*hPos*、*vPos* そして * 引数は無視されます。

二つの異なるツールバーウィンドウを同時に作成することはできません。ツールバー型のウィンドウが既に存在している状態で、**Open form window** コマンドを *type* 引数にToolbar form window 定数を渡して呼び出した場合、エラー -10613 ("ツールバー型のフォームウィンドウを二つ作成することはできません")が生成されます。

OS Xのフルスクリーンモードとツールバーフォームウィンドウに関する注意: アプリケーションがツールバーウィンドウと、フルスクリーンモードをサポート(Has full screen mode Mac オプション)する標準のウィンドウの両方を表示しているとき、インターフェースのルールに基づき、標準のウィンドウがフルスクリーンモードになった際にはツールバーが非表示にならなければなりません。ウィンドウがフルスクリーンモードになったかどうかを調べるためには、縦幅(垂直方向のサイズ)がスクリーンの高さと完全に一致するかどうかをテストします(以下参照)。

Tool bar height

テーマ: ウィンドウズ(このコマンドは"ユーザーインターフェース"から移動になりました)

Tool bar height -> 戻り値

引数	型		詳細
戻り値	倍長整数	<-	ツールバーの高さ(ピクセル単位)、またはツールバーが非表示の場合には0

このコマンドは、**Open form window** コマンドに Toolbar form window 型の定数を使用して作成されたツールバーに対しても使用できるようになりました。

このコマンドはカレントの表示されているツールバーの高さを、ピクセル単位で返します。ツールバーは4D デザインモードツールバーでも**Open form window** コマンドで作成されたツールバーでも、コンテキストに応じてどちらでも可能です(デザインモードのツールバーは**Open form window** コマンドで作成されたカスタムのツールバーが表示された際には自動的に非表示になります)。

ツールバーが非表示の場合、コマンドは0を返します。

HIDE TOOL BAR と SHOW TOOL BAR

テーマ: ウィンドウ(このコマンドは"ユーザーインターフェース"から移動になりました)

SHOW TOOL BAR
このコマンドは引数を必要としません。

HIDE TOOL BAR
このコマンドは引数を必要としません。

これらのコマンドは4Dでの廃止予定が取り消されます。これらのコマンドはカレントプロセスにおいて **Open form window** を使用して作成されたカスタムのツールバーフォームウィンドウを管理するために使用されます。

- **SHOW TOOL BAR**: ツールバーウィンドウが(**Open form window** コマンドの Toolbar form window オプションを使用して)開かれている場合、このコマンドはそのウィンドウを表示状態にします。ツールバーウィンドウが既に表示状態になっている、または存在しない場合には、このコマンドは何もしません。
- **HIDE TOOL BAR**: ツールバーウィンドウが(**Open form window** コマンドの Toolbar form window オプションを使用して)開かれている場合、このコマンドはそのウィンドウを非表示にします。ツールバーウィンドウが既に非表示になっている、または存在しない場合には、このコマンドは何もしません。

例題

OS Xにおいて、カスタムのツールバーフォームウィンドウとHas full screen mode Mac オプションを持った標準のウィンドウを定義したとします。ツールバーウィンドウが表示されているときに標準のウィンドウがユーザーによって最大化された場合、最大化されたウィンドウにツールバーがかぶさるのは避けたいところです。

これを避けるために、標準ウィンドウの"On Resize"フォームイベントにおいてウィンドウがフルスクリーンモードに切り替わったことを検知し、**HIDE TOOL BAR** コマンドを呼び出す必要があります:

```
Case of
   :(Form event=On Resize)
      GET WINDOW RECT($left;$top;$right;$bottom)
      If(Screen height=($bottom-$top))
         HIDE TOOL BAR
      Else
         SHOW TOOL BAR
      End if
End case
```

Has too l bar button Mac 定数は廃止予定に

Has toolbar button Mac 定数は今後廃止予定になりました。これに対応するオプションはAppleによってOS X 10.6から廃止予定となってきました。

この定数は"Open Form Window" と "Open Window" の両方の定数テーマにおいて使用可能でしたが、_O_Has toolbar button Mac と改名されました。

リストボックスにてダイナミック変数でカラムを作成

*4D v14 R3*から導入

4D の新機能の一つとして、ランタイムで動的に新しいカラムを追加または挿入できるようになりました。この新機能によって、4D は必要な変数の定義(カラム、フッター、そしてヘッダー)を自動的に行います。

その結果、以下の三つの4Dコマンドが一部新しくなりました:

- LISTBOX INSERT COLUMN
- LISTBOX INSERT COLUMN FORMULA
- LISTBOX DUPLICATE COLUMN

新シンタックス(Nilポインター)

以下のコマンドは、ダイナミック変数の作成に対応するために、シンタックスが一部更新されています。

LISTBOX INSERT COLUMN ({* ;} object ; colPosition ; colName ; *colVariable* ; headerName ; *headerVar* {; footerName ; *footerVar*})
LISTBOX INSERT COLUMN FORMULA ({* ;} object ; colPosition ; colName ; formula ; dataType ; headerName ; *headerVar* {; footerName ; *footerVar*})
LISTBOX DUPLICATE COLUMN ({* ;} object ; colPosition; colName ; *colVariable* ; headerName ; *headerVar* {; footerName ; *footerVar*})

引数	型		詳細
…			
colVariable	配列、フィールド、変数、**Nil ポインター**	->	カラム配列名またはフィールドまたは変数
headerVar	整数変数 **または Nil ポインター**	->	カラムヘッダー変数
…			
footerVar	変数 **または Nil ポインター**	->	カラムフッター変数

これらのコマンドは、*colVariable* (適用可能時のみ)、*headerVar* と *footerVar* 引数において、Nil ポインター (**->[]**) を値として受け取るようになりました。この場合、4D は、コマンドが実行された時に必要な変数を動的に作成します(詳細な情報に関しては、4Dランゲージマニュアルの"ダイナミック変数"のセクションを参照して下さい)。

ヘッダーとフッター変数は常に特定の型で作成されるという点に注意して下さい(ヘッダーは倍長整数、フッターはテキスト)。しかしながら、カラム変数は、リストボックスがこの変数に対して異なる型の配列(テキスト配列、整数配列、等)を受け入れるため、作成時に型を指定することはできません。そのため、配列の型は手動で設定しなければなりません。例えば:

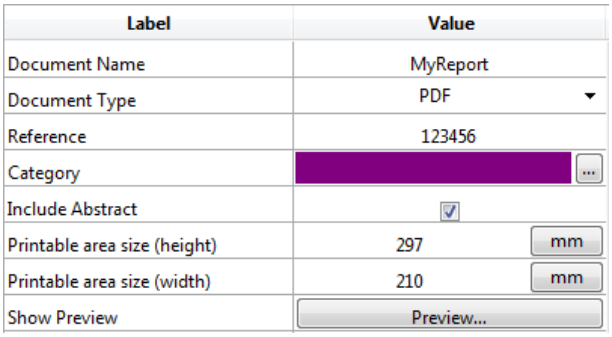
```
C_POINTER($NilPtr)
LISTBOX INSERT COLUMN(*;"MyListBox";1;"MyNewColumn";$NilPtr;"MyNewHeader";$NilPtr)
ColPtr:=OBJECT Get pointer(Object named;"MyNewColumn")
ARRAY TEXT(ColPtr->;0) // 配列内に要素を事前に割り振っておきたいときには、0を正の値で置き換えて下さい
```

配列内に新しい要素を挿入するために **LISTBOX INSERT ROWS** などのコマンドを使用する前にこのように型を指定しておくことは、重要な事です。その代わり、 **APPEND TO ARRAY** を使用して、配列の型を指定すると同時に要素を挿入することが可能です。

リストボックスのカラムでオブジェクト配列を使用 (4D View Pro)

4D v15以降、リストボックスのカラムはオブジェクト配列を扱えるようになりました。オブジェクト配列は異なる種類のデータを格納できるので、この強力な新機能を使用すれば、単一のカラム内の行ごとに異なる入力タイプを混ぜたり、様々なウィジェットを表示したりといった事ができるようになります。例えば、最初の行にテキスト入力を挿入し、二行目にチェックボックスを、そして産業目にドロップダウンを挿入する、と言ったことが可能になります。また、オブジェクト配列は、ボタンやカラーピッカーと言った新しいウィジェットへのアクセスも可能にします。

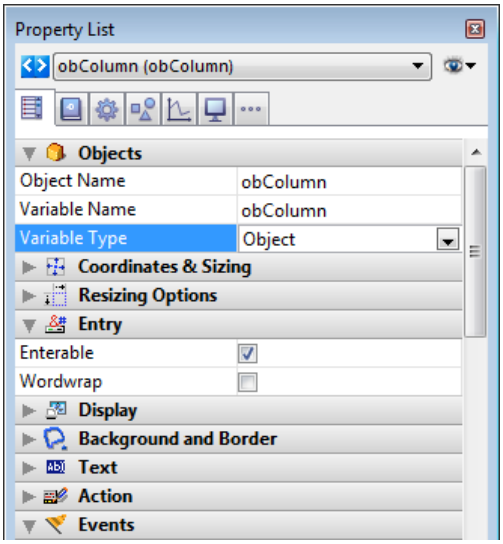
以下のリストボックスはオブジェクト配列を使用してデザインされました:



ライセンスについての注意: リストボックス内でのオブジェクト配列の使用は、来る"4D View Pro"ツールへの第一歩です(これは現行の4D View プラグインを置き換えるものです)。この機能を使用するためには、有効な4D Viewライセンスが必要になります。詳細な情報に関しては、4D Webサイトを参照して下さい。

オブジェクト配列カラムの設定

オブジェクト配列をリストボックスのカラムに割り当てるためには、プロパティリスト(の"変数名"フィールド)にオブジェクト配列名を設定するか、配列型のカラムのように**LISTBOX INSERT COLUMN** コマンドを使用します。プロパティリスト内では、カラムにおいて"変数タイプ"に**オブジェクト**が選択できるようになりました:



オブジェクトカラムに対しては、座標、サイズ、スタイルなどに関連した標準のプロパティは使用可能です。プロパティリストを使用して定義することもできますし、オブジェクト型のリストボックスカラムのそれぞれの行に対してスタイル、フォントカラー、背景色、表示状態をプログラムで定義することもできます。これらのタイプのカラムは非表示にすることも可能です。

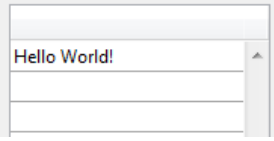
しかしながら、**データソース**テーマは、オブジェクト型のリストボックスカラムに対しては選択できません。実際、それぞれのカラムのそれぞれのセルの中身は、それに対応するオブジェクト配列の要素の属性に基づいています。それぞれの配列の要素は、以下をプロパティを定義できます:

- 値の型(必須):テキスト、カラー、イベント、他
- 値そのもの(任意):入力/出力に使用

- セルの中身の表示(任意):ボタン、リスト、他
- 追加の設定(任意):値の型によります

これらのプロパティを定義するためには、適切な属性をオブジェクト内に設定する必要があります(使用可能な属性は以下に一覧としてまとめてあります)。例えば、以下ような簡単なコードを使用して"Hello World!"とオブジェクトカラム内に書き込むことができます:

```
ARRAY OBJECT(obColumn;0) //カラム配列
C_OBJECT($ob) //第一要素
OB SET($ob;"valueType";"text") //値の型を定義(必須)
OB SET($ob;"value";"Hello World!") //値を定義
APPEND TO ARRAY(obColumn;$ob)
```



注: 表示フォーマットと入力フィルターはオブジェクトカラムに対しては設定できません。これらは値の型に応じて自動的に変わるからです。

valueTypeとデータ表示

リストボックスカラムにオブジェクト配列が割り当てられているとき、セルの表示、入力、編集の方法は、配列の要素の**valueType** 属性に基づきます。サポートされる**valueType** の値は以下の通りです:

- "text": テキスト値
- "real": <space>、<.>、<,>などのセパレータを含む数値
- "integer": 整数値
- "boolean": True/False 値
- "color": 背景色を定義
- "event": ラベル付きのボタンを表示

4D は"valueType"の値に応じたデフォルトのウィジェットを使用します(つまり、"text"と設定すればテキスト入力ウィジェットが表示され、"boolean"と設定すればチェックボックスが表示されます)。しかしオプションを使用することによって表示方法の選択が可能な場合もあります。以下の一覧はそれぞれの値の型に対してのデフォルトの表示方法と、他に選択可能な表示方の一覧を表しています:

valueType	デフォルトのウィジェット	他に選択可能なウィジェット
text	テキスト入力	ドロップダウンメニュー(指定リスト)またはコンボボックス(選択リスト)
real	管理されたテキスト入力(数字とセパレータ)	ドロップダウンメニュー(指定リスト)またはコンボボックス(選択リスト)
integer	管理されたテキスト入力(数字のみ)	ドロップダウンメニュー(指定リスト)またはコンボボックス(選択リスト)またはスリーステートチェックボックス
boolean	チェックボックス	ドロップダウンメニュー(指定リスト)
color	背景色	テキスト
event	ラベル付きのボタン	
		全てのウィジェットには、**単位切り替えボタン** または **省略ボタン** を追加でセルに付属させることができます

セルの表示とオプションは、オブジェクト内の特定の属性を使用することによって設定できます(以下を参照して下さい)。

表示フォーマットと入力フィルター

オブジェクト型のリストボックスのカラムにおいては、表示フォーマットと入力フィルターを設定することはできません。これらは値の型に応じて自動的に定義されます。どのように定義されるかについては、以下一覧にまとめてあります:

値の型	デフォルトのフォーマット	入力コントロール
text	オブジェクト内で定義されているものと同じ	制限なし
real	オブジェクト内で定義されているものと同じ(システムの小数点セパレータを使用)	"0-9" と "." と "-" min>=0 の場合、"0-9" と "."
integer	オブジェクト内で定義されているものと同じ	"0-9" と "-" min>=0 の場合、"0-9"
Boolean	チェックボックス	N/A
color	N/A	N/A
event	N/A	N/A

属性

オブジェクト配列のそれぞれの要素は、セルの中身とデータ表示を定義する一つまたは複数の属性を格納するオブジェクトです(上記の例を参照して下さい)。

唯一必須の属性は"valueType"であり、サポートされる値は"text"、"real"、"integer"、"boolean"、"color" そして "event"です。以下の表には、リストボックスオブジェクト配列において"valueType"の値に応じてサポートされる全ての属性がまとめてあります(他の属性は全て無視されます)。表示フォーマットに関しては、その更に下に詳細な説明と例があります。

	valueType	**text**	**real**	**integer**	**boolean**	**color**	**event**
属性	**詳細**						
value	セルの値(入力または出力)	x	x	x			
min	最小値		x	x			
max	最大値		x	x			
behavior	"スリーステート" の値			x			
requiredList	オブジェクト内で定義されたドロップダウンリスト	x	x	x			
choiceList	オブジェクト内で定義されたコンボボックス	x	x	x			
requiredListReference	4D リスト参照、"saveAs"の値による	x	x	x			
requiredListName	4D リスト名、"saveAs"の値による	x	x	x			
saveAs	"reference" または "value"	x	x	x			
choiceListReference	4D リスト参照、コンボボックスを表示	x	x	x			
choiceListName	4D リスト名、コンボボックスを表示	x	x	x			
unitList	X要素の配列	x	x	x			
unitReference	選択された要素のインデックス	x	x	x			
unitsListReference	単位の4D リスト参照	x	x	x			
unitsListName	単位の4D リスト名	x	x	x			
alternateButton	切り替えボタンを追加	x	x	x	x	x	

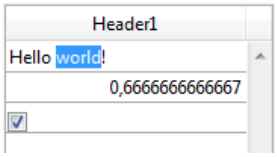
value

セルの値は"value"属性に保存されています。この属性は入力と出力に使用されます。またリストを使用する際のデフォルトの値を定義するのにも使用できます(以下を参照)。

例:

```
ARRAY OBJECT(obColumn;0) //カラム配列
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob1;"valueType";"text")
OB SET($ob1;"value";$entry) //ユーザーが新しい値を入力した場合、 編集された値は$entry に格納されます
C_OBJECT($ob2)
```

```
OB SET($ob2;"valueType";"real")
OB SET($ob2;"value";2/3)
C_OBJECT($ob3)
OB SET($ob3;"valueType";"boolean")
OB SET($ob3;"value";True)

APPEND TO ARRAY(obColumn;$ob1)
APPEND TO ARRAY(obColumn;$ob2)
APPEND TO ARRAY(obColumn;$ob3)
```



注: ヌル値はサポートされており、空のセルとして表示されます。

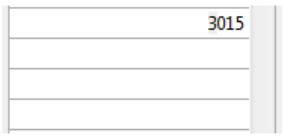
min と max

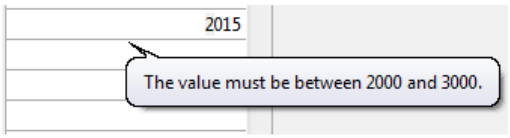
"valueType" が"real" または "integer" であるとき、オブジェクトは**min** と **max** 属性を受け取ります(値は適切な範囲である必要があり、また、valueTypeと同じ型である必要があります)。

これらの属性を使用すると入力値の範囲を管理する事ができます。セリが評価されたとき(フォーカスを失ったとき)、入力された値が**min**の値より低い場合、または**max**の値より大きい場合には、その値は拒否されます。この場合、入力をする前の値が保持され、tipに説明が表示されます。

例:

```
C_OBJECT($ob3)
$entry3:=2015
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";$entry3)
OB SET($ob3;"min";2000)
OB SET($ob3;"max";3000)
```





behavior

behavior 属性は値の通常の表示とは異なる他の表示方法を提供します。4D v15では、一つだけ他の表示方法が用意されています:

属性	使用可能な値	valueType	詳細
behavior	threeStates	integer	スリーステートチェックボックスを数値として表現します。2=セミチェック、1=チェック、0=チェックされていない、-1=非表示、-2=チェックなしが無効化、-3=チェックが無効化、-4=セミチェックが無効化

例:

```
C_OBJECT($ob3)
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";-3)
C_OBJECT($ob4)
OB SET($ob4;"valueType";"integer")
OB SET($ob4;"value";-3)
OB SET($ob4;"behavior";"threeStates")
```

requiredList と choiceList

"choiceList" または "requiredList"属性がオブジェクト内に存在しているとき、テキスト入力は以下の属性に応じて、ドロップダウンリストまたはコンボボックスで置き換えられます:

- 属性が"choiceList" の場合、セルはコンボボックスとして表示されます。これはつまり、ユーザーは値を選択、または入力できるということです。
- 属性が"requiredList" の場合、セルはドロップダウンリストとして表示されます。これはつまり、ユーザーはリストに提供されている値からどれか一つを選択するしかないという事です。

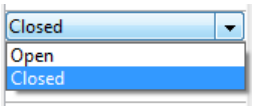
どちらの場合においても、"value"属性を使用してウィジェット内の値を事前に選択することができます。

注: ウィジェットの値は配列を通して定義されます。既存の4Dリストをウィジェットに割り当てたい場合、"requiredListReference"、"requiredListName"、"choiceListReference"属性を使用するか、"choiceListName" 属性を使用する必要があります。

例:

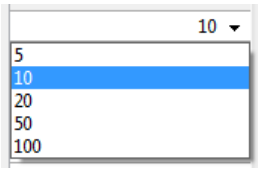
- 選択肢が二つ("Open"または"Closed")しかないドロップダウンリストを表示したい場合を考えます。"Closed"が選択された状態にしたいとします:

```
ARRAY TEXT($RequiredList;0)
APPEND TO ARRAY($RequiredList;"Open")
APPEND TO ARRAY($RequiredList;"Closed")
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"Closed")
OB SET ARRAY($ob;"requiredList";$RequiredList)
```



- 整数値であれば全て受け入れ可能な状態にしておいた上で、もっとも一般的な値を提示するためにコンボボックスを表示したい場合を考えます:

```
ARRAY LONGINT($ChoiceList;0)
APPEND TO ARRAY($ChoiceList;5)
APPEND TO ARRAY($ChoiceList;10)
APPEND TO ARRAY($ChoiceList;20)
APPEND TO ARRAY($ChoiceList;50)
APPEND TO ARRAY($ChoiceList;100)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";10) //10 をデフォルト値として使用
OB SET ARRAY($ob;"choiceList";$ChoiceList)
```



requiredListName と requiredListReference

"requiredListName" and "requiredListReference"属性を使用すると、リストボックスセル内において、デザインモード(ツールボックス内)において、またはプログラミングによって(**New list** コマンドを使用して)4Dで定義されたリストを使用することが出来るようになります。セルはドロップダウンリストとして表示されるようになります。これはつまり、ユーザーは、リスト内に提供された値のどれか一つのみを選択できるということを意味します。

"requiredListName" または "requiredListReference"は、リストの作成元に応じて使い分けます。リストがツールボックスで作成された場合、リスト名を渡します。そうでない場合、つまりリストがプログラミングによって定義された場合、リストの参照を渡します。どちらの場合においても、"value"属性を使用するとウィジェット内の値を事前に設定することができます。

注: これらの値を単純な配列を通して定義したい場合は、"requiredList" 属性を使用する必要があります。

この場合"saveAs" 属性は、選択された項目が"value"(値)として、または"reference”(参照)として保存されるかを定義します。

- "saveAs" = "reference" であった場合、項目は参照として保存されます。"valueType" はreal または integerである必

要があります。

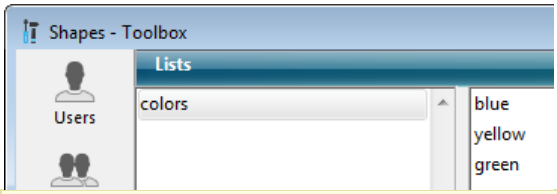
- "saveAs" = "value" であった場合、値が保存されます。この場合、"valueType" はリストの値と同じ型である必要があります(通常、"text" または "integer"です)。そうでない場合、4Dはリストの値をオブジェクトの"valuType"へと変換しようとします(以下の例を参照して下さい)。

"save as"オプションについてのより詳細な情報については、*Design Reference* マニュアルの**関連付け(値または参照番号)**の章を参照して下さい。

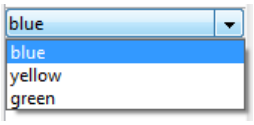
注: リストが実数値を表すテキストを含んでいる場合、小数点はローカル設定に関わらず、ピリオド(".")である必要があります。例:"17.6" "1234.456"

例:

- ツールボックスで定義された"colors"リスト("blue"、"yellow"、そして "green"の値を格納)に基づいたドロップダウンリストを表示し、値として保存し、デフォルトの表示は"blue"にしたい場合を考えます:

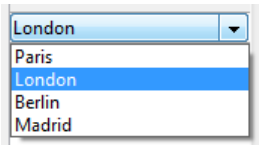



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"colors")
```



- プログラミングによって定位されたリストに基づいたドロップダウンリストを表示し、参照として保存したい場合を考えます:

```
<>List:=New list
APPEND TO LIST(<>List;"Paris";1)
APPEND TO LIST(<>List;"London";2)
APPEND TO LIST(<>List;"Berlin";3)
APPEND TO LIST(<>List;"Madrid";4)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"saveAs";"reference")
OB SET($ob;"value";2) //デフォルトでLondonを表示
OB SET($ob;"requiredListReference";<>List)
```



choiceListName と choiceListReference

"choiceListName" と "choiceListReference" 属性を使用すると、リストボックスセル内において、デザインモード(ツールボックス内)において、またはプログラミングによって(**New list** コマンドを使用して)4Dで定義されたリストを使用することが出来るようになります。セルはコンボボックスとして表示されるようになります。これはつまり、ユーザーは値を選択、または入力できるということを意味します。

"choiceListName" または "choiceListReference" は、リストの作成元に応じて使い分けます。リストがツールボックスで作成された場合、リスト名を渡しま す。そうでない場合、つまりリストがプログラミングによって定義された場合、リストの参照を渡します。どちらの場合においても、"value"属性を使用 するとウィジェット内の値を事前に設定することができます。

注: これらの値を単純な配列を通して定義したい場合は、"choiceList" 属性を使用する必要があります。

この場合、"saveAs" 属性は使用できません。選択された項目は自動的に"value"(値)として保存されるからです(詳細な情報についてはcf. **requiredListName と requiredListReference** を参照して下さい)。

注: リストが実数値を表すテキストを含んでいる場合、小数点はローカル設定に関わらず、ピリオド(".")である必要があります。例:"17.6" "1234.456"

例: ツールボックスで定義された"colors"リスト("blue"、"yellow"、そして "green"の値を格納)に基づいたドロップダウンリストを表示し、値として保存し、デフォルトの表示は"green"にしたい場合を考えます:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"blue")
OB SET($ob;"choiceListName";"colors")
```

unitsList、unitsListName、 unitsListReference と unitReference

特定の値を使用する事で、セルの値に関連した単位を追加することができます(例: "10 cm", "20 pixels"等)。単位リストを定義するためには、以下の属性のどれか一つを使用します:

- "unitsList": 利用可能な単位(例: "cm"、"inches"、"km"、"miles"、他)を定義するのに使用する *x* 要素を格納した配列。オブジェクト内で単位を定義するためにはこの属性を使用します。
- "unitsListReference": 利用可能な単位を含んだ4Dリストへの参照。**New list** コマンドで作成された4D リストで単位を定義するためにはこの属性を使用します。
- "unitsListName": 利用可能な単位を含んだデザインモードで作成された4Dリスト名。ツールボックスで作成された4Dリストで単位を定義するためにはこの属性を使用します。

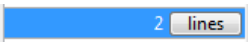
単位リストが定義された方法に関わらず、以下の属性を関連付けることができます:

- "unitReference": "unitList"、"unitsListReference" または "unitsListName" の値リスト内で選択された項目へのインデックス(1から *x*)を格納する単一の値。

カレントの単位は、ボタンとして表示されます。このボタンは、クリックするたびに"unitList"、"unitsListReference" または "unitsListName" の値を切り替えて行きます(例 "pixels" -> "rows" -> "cm" -> "pixels" -> 等)。

例: 数値の入力と、その後に可能性のある単位("rows" または "pixels")を二つ続けて表示したい場合を考えます。カレントの値は"2" + "lines"と、オブジェクト内で直接定義された値("unitsList" 属性)を使用するものとします:

```
ARRAY TEXT($_units;0)
APPEND TO ARRAY($_units;"lines")
APPEND TO ARRAY($_units;"pixels")
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";2) // 2 "units"
OB SET($ob;"unitReference";1) //"lines"
OB SET ARRAY($ob;"unitsList";$_units)
```



alternateButton

セルに省略ボタン[**...**] を追加したい場合、"alternateButton" 属性に**True** の値を入れてオブジェクトに渡すだけです。省略ボタンは自動的にセル内に表示されます。

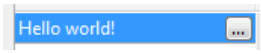
このボタンがユーザーによってクリックされた場合、On Alternate Click イベントが生成され、そのイベントを自由に管理することができます(詳細な情報に関しては"events management" の章を参照して下さい)。

注: On Alternate Click は、On Arrow Click イベントの新しい名前であり、4D v15においてその拡張されたスコープを強調するために改名されました。

例:

```
C_OBJECT($ob1)
$entry:="Hello world!"
```

```
OB SET($ob;"valueType";"text")
OB SET($ob;"alternateButton";True)
OB SET($ob;"value";$entry)
```



color valueType

"color" *valueType* を使用すると、色、または色を表すテキストを表示する事ができます。

- 値が数字の場合、色付けされた長方形がセル内に表示されます。例:

```
C_OBJECT($ob4)
OB SET($ob4;"valueType";"color")
OB SET($ob4;"value";0x00FF0000)
```

- 値がテキストの場合、そのテキストが表示されます(例: "value";"Automatic")。

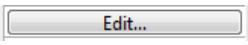
event valueType

"event" *valueType* を使用すると、クリックした際にOn Clicked イベントを生成する単純なボタンを表示します。データまたは値を渡す/返すことはできません。

オプションとして、"label"属性を渡す事ができます。

例:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"event")
OB SET($ob;"label";"Edit...")
```



イベント管理

オブジェクトリストボックス配列を使用している際には、複数のイベントを管理することができます:

- **On Data Change**: 以下の場所において、どんな値でも変更された場合には On Data Change イベントがトリガーされます:
  - テキスト入力ゾーン
  - ドロップダウンリスト
  - コンボボックスエリア
  - 単位ボタン(値x が値 x+1へとスイッチしたとき)
  - チェックボックス(チェック/チェックなしの状態がスイッチしたとき)
- **On Clicked**: ユーザーが、"event" *valueType* 属性を使用して実装されたボタンをクリックした場合、On Clicked イベントが生成されます。このイベントはプログラマーによって管理されます。
- **On Alternative Click**: ユーザーが省略ボタン("alternateButton" 属性)をクリックした場合、On Alternative Click イベントが生成されます。このイベントはプログラマーによって管理されます。

注: On Alternative Click は、4Dの以前のバージョンで使用されていた **On Arrow Click** イベントの新しい名前です。このイベントはスコープが拡張されたことにより、4D v15で改名されました。

実数値の、有効桁数の縮小

---

*4D v14 R3*から導入

実数の小数部分を表すのに使用される有効桁数は、以下の様に削減されました:

- 以前のバージョンでは、この桁数は15桁でした。

4D v15　以降のバージョンでは**13桁**になりました。

この変更が関係するのは実数をテキストに変換する際に限られ、内部での表現(保存)、または実数同士の計算には関係しません。つまり、実数の正確性に影響はないという事です。

この新しい原理は、テキストで表示した際に正確性の保証できない最後の2桁を含めない、というものです。この目的は、実数の演算が誤ったテキストとして結果が返ってくる状況を限定することです。例えば、この原理により、以下の様な場合に有効な結果を得ることができます:

演算	v15より前の4Dでの結果	v15以降の4Dでの結果
String(3216.36 - 3214.89)	"1.47000000000025"	"1.47"
String(0.321636-0.321489)	"0.000146999999999953"	"0.000147"

有 効桁数13桁を実数で使用することは、通常コンピューターアプリケーションにおいて十分な桁数であり、これらの数字の処理を向上させます。コンピューター 内部での実数の計算は、常に二進法と近似を使用して行われるものであり、10進法が使われているわけではないことに注意して下さい。有効桁数は、変換アル ゴリズムが切り上げ・切り下げや近似によって期待した数値を得られるようにします。同様に、実数の等式比較は、二つの数字をそれぞれ互いに減算し合った結 果を分析することによって行われています(4Dでは、この比較のデフォルトの正確性は **SET REAL COMPARISON LEVEL** コマンドを使うことによって変更することができます)。

もしあなたの4Dアプリケーションが実数の最初の15桁を使用する場合、データ フォーマットをテキストまたは倍長整数に変換する必要がでてくる可能性があります。上記で説明があるように、実数はその性質から正確性に限界があるため に、大量の有効桁数が必要となる値の保存や計算(例えばシリアル番号や天文学的数字など)には向いていません。

廃止予定のコマンドの改名

4D v15では分かりやすさのために、今までついていなかったものも含めて全ての廃止予定のコマンドに"_o_"の接頭辞をつけました。

廃止予定のコマンドは4Dリストには表示されなくなったため(**廃止予定のコマンドは非表示に** の章を参照して下さい)、それらを選択することはできなくなりました。廃止予定のコマンドは既存のコードの中でのみ改名されます。

以下は、4D v15で改名された廃止予定のコマンドの一覧です(ここにあるものは4Dリストには表示されません):

以前の名前	4D v15での新しい名前
ADD DATA SEGMENT	_o_ADD DATA SEGMENT
ADD SUBRECORD	_o_ADD SUBRECORD
ALL SUBRECORDS	_o_ALL SUBRECORDS
APPLY TO SUBSELECTION	_o_APPLY TO SUBSELECTION
ARRAY STRING	_o_ARRAY STRING
ARRAY TO STRING LIST	_o_ARRAY TO STRING LIST
Before subselection	_o_Before subselection
C_STRING	_o_C_STRING
Convert cas	_o_Convert case
Create resource file	_o_Create resource file
CREATE SUBRECORD	_o_CREATE SUBRECORD
DATA SEGMENT LIST	_o_DATA SEGMENT LIST
DELETE RESOURCE	_o_DELETE RESOURCE
DELETE SUBRECORD	_o_DELETE SUBRECORD
DISABLE BUTTON	_o_DISABLE BUTTON
ENABLE BUTTON	_o_ENABLE BUTTON
End subselection	_o_End subselection
FIRST SUBRECORD	_o_FIRST SUBRECORD
Font name	_o_Font name
Font number	_o_Font number
Get component resource ID	_o_Get component resource ID
Get platform interface	_o_Get platform interface
INVERT BACKGROUND	_o_INVERT BACKGROUND
ISO to Mac	_o_ISO to Mac
LAST SUBRECORD	_o_LAST SUBRECORD
Mac to ISO	_o_Mac to ISO
Mac to Win	_o_Mac to Win
MODIFY SUBRECORD	_o_MODIFY SUBRECORD
NEXT SUBRECORD	_o_NEXT SUBRECORD
ORDER SUBRECORDS BY	_o_ORDER SUBRECORDS BY
PICTURE TYPE LIST	_o_PICTURE TYPE LIST
PREVIOUS SUBRECORD	_o_PREVIOUS SUBRECORD
QT COMPRESS PICTURE	_o_QT COMPRESS PICTURE
QT COMPRESS PICTURE FILE	_o_QT COMPRESS PICTURE FILE
QT LOAD COMPRESS PICTURE FROM FILE	_o_QT LOAD COMPRESS PICTURE FROM FILE
QUERY SUBRECORDS	_o_QUERY SUBRECORDS
Records in subselection	_o_Records in subselection
REDRAW LIST	_o_REDRAW LIST
SAVE PICTURE TO FILE	_o_SAVE PICTURE TO FILE
SET PICTURE RESOURCE	_o_SET PICTURE RESOURCE
SET PLATFORM INTERFACE	_o_SET PLATFORM INTERFACE
SET RESOURCE	_o_SET RESOURCE
SET RESOURCE NAME	_o_SET RESOURCE NAME
SET RESOURCE PROPERTIES	_o_SET RESOURCE PROPERTIES
SET STRING RESOURCE	_o_SET STRING RESOURCE
SET TEXT RESOURCE	_o_SET TEXT RESOURCE
USE EXTERNAL DATABASE	_o_USE EXTERNAL DATABASE
USE INTERNAL DATABASE	_o_USE INTERNAL DATABASE

Win to Mac	_o_Win to Mac

XSLT コマンド

XSLT コマンドは4D v14 R4にて名前が変更になりました:

以前の名前	4D v15での新しい名前
XSLT APPLY TRANSFORMATION	_o_XSLT APPLY TRANSFORMATION
XSLT GET ERROR	_o_XSLT GET ERROR
XSLT SET PARAMETER	_o_XSLT SET PARAMETER

より詳細な情報に関しては、**XSLT コマンドは廃止予定に** の章を参照して下さい。

4D Pack コマンド

4D v14 R5以降、いくつかの4D Packコマンドも廃止予定となりました:

以前の名前	新しい名前
AP FCLOSE	**_o_AP FCLOSE**
AP fopen	**_o_AP fopen**
AP FPRINT	**_o_AP FPRINT**
AP fread	**_o_AP fread**
AP Save BMP 8 bits	**_o_AP Save BMP 8 bits**
AP Add table and fields	**_o_AP Add table and fields**
AP Create relation	**_o_AP Create relation**
AP Get file MD5 digest	**_o_AP Get file MD5 digest**
AP ShellExecute	**_o_AP ShellExecute**

より詳細な情報については、**廃止予定のコマンドの改名** の章を参照して下さい。

廃止予定ではなくなったコマンド

その一方で、ドキュメントにおいて廃止予定だと以前宣言されていた一部のコマンドのうち、上述の定義に当てはまらないものは、"再採用"されることになりました。

これらのコマンドは、以前のプログラミングモードに対応し、より効率的なコードで置き換えられるために現在では用法は限定的ですが、これらの維持に継続に対する疑問はないとされるものです:

Activated
Outside call
After
Before
Deactivated
In header
In footer
In break
Modified
Document type

Clickcount

Clickcount -> 戻り値

引数	型		説明
戻り値	倍長整数		連続したクリックの回数

説明

4D v14 R5 から導入

テーマ: フォームイベント

新しい**Clickcount** コマンドはマウスクリックイベントのコンテキストにおいて、ユーザーが同じマウスボタンを短時間に連続してクリックした回数を返します。通常、このコマンドはダブルクリックを表す2を返します。

このコマンドを使用することによって、リストボックスのヘッダー・フッターでのダブルクリックを検知したり、トリプルクリックなどのそれ以上の連続したクリックを扱えるようになります。

全てのマウスボタンのクリックは個別のクリックイベントを生成します。例えばユーザーがダブルクリックをした場合、最初のクリックに対しイベントが送信され、**Clickcount** は1を返します。二つ目のクリックに対してももう一つイベントが発生し、**Clickcount** は2を返します。

このコマンドはOn Clicked、On Header Click または On Footer Click フォームイベントのコンテキストにおいてのみ使用すべきものです。そのため、デザインモードを調べ、フォームプロパティや特定のオブジェクトのプロパティにおいて適切なイベントが正常に選択されているかどうかをチェックする必要があります。

On Clicked と On Double Clicked イベントが両方有効化されている場合、**Clickcount** コマンドは以下の組み合わせを返します:

- On Clicked イベントに1
- On Double Clicked イベントに2
- On Clicked イベントに2+n

例題 1

以下のコードストラクチャーをリストボックスヘッダー内に配置すると、シングルクリックとダブルクリックを管理することが出来るようになります:

```
Case of
   :(Form event=On Header Click)
      Case of
         :(Clickcount=1)
            ... //シングルクリックアクション
         :(Clickcount=2)
            ... //ダブルクリックアクション
      End case
End case
```

例題 2

ラベルは入力可能ではないですが、トリプルクリックをすると入力可能になります。ユーザーにラベルの編集を許可したい場合、以下の様なコードをオブジェクトメソッドに記載します:

```
If(Form event=On Clicked)
   Case of
      :(Clickcount=3)
         OBJECT SET ENTERABLE(*;"Label";True)
         EDIT ITEM(*;"Label")
   End case
End if
```

EXPORT STRUCTURE

EXPORT STRUCTURE (xmlStructure)

引数	型		説明
xmlStructure	テキスト変数	⇦	4D データベースストラクチャーを書き出したXML定義ファイル

説明

*4D v14 R4*から導入

テーマ: ストラクチャーアクセス

新しい **EXPORT STRUCTURE** コマンドは、*xmlStructure* 内にカレントの4Dデータベースストラクチャー定義をXMLフォーマットで書き出します。このコマンドは4D デザインモードインターフェースのメニュー内の、**書き出し -> ストラクチャー定義をXMLファイルに書き出し...** のメニュー項目と同じメカニズムを使用します(**ストラクチャー定義の書き出しと読み込み** を参照して下さい)。

xmlStructure 引数には、ストラクチャー定義を格納するテキスト変数を渡します。書き出された定義にはテーブル、フィールド、インデックス、そしてリレーションに加え、 それぞれの属性と、完全なストラクチャー定義を成すために必要な様々な特性が含まれます。非表示の要素は、適切な属性にタグ付けされて書き出しされます。 しかしながら削除された要素は、書き出されません。

4D ストラクチャー定義内部の"文法"はDTD(文書型宣言)ファイル(これはXMLファイルの評価にも使用されます)を通して作成されます。4Dによって使用される文書型宣言ファイルは、4Dアプリケーションの隣のDTDフォルダ内にグループ分けされます。**base_core.dtd** と **common.dtd** ファイルはストラクチャーの定義に使用されます。4D ストラクチャー定義についての詳細な情報については、これらのファイルの中身を閲覧したり、そこに含まれるコメントなどを参照して下さい。

例題

カレントデータベースのストラクチャーをテキストファイルに書き出す場合を考えます:

```
C_TEXT($vTStruc)
EXPORT STRUCTURE($vTStruc)
TEXT TO DOCUMENT("myStructure.xml";$vTStruc)
```

Find in sorted array

Find in sorted array (array ; value ; > or < {; posFirst {; posLast}}) -> 戻り値

引数	型		説明
array	配列	➡	検索する配列
value	式	➡	配列内で検索する値(配列と同じ型)
> or <	演算子	➡	配列が昇順に並んでいる場合には>、降順に並んでいる場合には<
posFirst	倍長整数	⬅	値が見つかった場合にはその最初の場所。そうでない場合には値が挿入されるべき個所。
posLast	倍長整数	⬅	値が見つかった場合にはその最後の場所。そうでない場合posFirstと同じ。
戻り値	ブール	↩	配列内に少なくとも一つ合致する値がある場合にはTrue、そうでなければFalse

説明

*4D v14 R4*から導入

テーマ: 配列

新しい**Find in sorted array** コマンドは*array* 引数で指定した配列内に、*value* 引数で指定した値と合致する要素が少なくとも一つある場合には**true**を返します。また、合致した要素の位置を返す事もできます(任意)。**Find in array** コマンドとは異なり、*array* 引数で指定した配列が既にソート済みで、値の順番(位置)についての情報がある場合にのみ有効です(これにより、必要であればその個所に値を挿入することも可能です)。

array 引数で指定した配列は既にソート済みであり、*> or <* 引数で指定された順番と合致している必要があります(">"の記号は配列が昇順であることを、"<"の記号は配列が降順であることを意味します)。**Find in sorted array** コマンドはソートされている利点を生かしてバイナリ―サーチアルゴリズムを使用します。これは大きな配列においてはより効率的なアルゴリズムです(詳細な情報に関しては、Wikipediaの二分探索のページを参照して下さい)。しかしながら配列が適切にソートされていない場合には、結果が不正確になることがあります。

以下のいずれかに該当する場合、このコマンドはソートの指示を無視し、標準的な**Find in array**と同じように振る舞います(シーケンシャル探索を行い、*value* 引数で指定した値が見つからない場合には*posFirst* と *posLast* 引数にそれぞれ-1が返されます):

- 配列の型がソートできない場合(例:ポインター配列)。
- 配列の型がブール型の場合(結果が正確ではありません)。
- 配列が文字列配列またはテキスト配列であり、データベースがUnicodeモード(互換性モード)でない場合。
- テキスト配列の中を、ワイルドカード('@')が文字列の先頭もしくは途中に来ている文字列を検索する場合(そのようなワイルドカードを使用してバイナリ―サーチしようとした場合、検索結果が配列内で不連続である場合があるため、使用できません)。

コマンドが**False** を返した場合、*posFirst* 引数に返された値を**INSERT IN ARRAY**コマンドに渡すことによって、配列のソートを乱すことなく*value* 引数の値を配列内に挿入することができます。この方法の方が、新しい項目を配列の最後に挿入し、その後でその項目を正しい場所に移動させるために**SORT ARRAY**コマンドを呼び出す方法より早いです。

posLast 引数に返された値と*posFirst* 引数に返された値を組み合わせることによって、*value* 引数と合致する配列内のそれぞれの要素に対して(**For...End for** ループを使用して)繰り返したり、同じ値が何個合致するかを探す事もできます(**Count in array**コマンドと同じですが、こちらの方が速いです)。

例題 1

配列のソートを崩さないまま、必要であれば値を挿入したいという場合を考えます:

```
C_LONGINT($pos)
If(Find in sorted array($array ;$value ;>;$pos)
   ALERT("Found at pos "+String($pos))
Else
   INSERT IN ARRAY($array ;$pos)
   $array{$pos}:=$value
End if
```

例題 2

"test"で始まる文字列の合致件数を探し、それらを全て連鎖させた文字列を作成したい場合を考えます:

```
C_LONGINT($posFirst ;$posLast)
C_TEXT($output)
If(Find in sorted array($array ;"test@";>;$posFirst ;$posLast))
   $output:="Found "+String($posLast-$posFirst+1)+" results :\n"
End if
For($i ;$posFirst ;$posLast)
   $output:=$output+$array{$i}+"\n"
End for
```

Get database measures

Get database measures {(options)} -> 戻り値

引数	型		説明
options	Object	➡	返された情報のオプション
戻り値	Object	⮌	データベース情報を格納したオブジェクト

説明

4D v14 R3 から導入、*4D v14 R5* で変更

Get database measures コマンドは、4Dデータベースエンジンイベントについての詳細な情報を取得します。返される情報には、ディスクやメモ リーキャッシュへの(もしくはからの)読み出し/書き込みアクセスに加え、データベースのインデックス、クエリ、並び替えの使用も含まれます。

Get database measures は全ての関連情報を内包する単一のオブジェクトを返します。*options* オブジェクト引数を使用して、その返される情報のオプションを指定する事ができます。

返されるオブジェクトの概要

返されたオブジェクトには、以下の基本構造を持つ、"DB"という名の単一のプロパティを格納しています:

```
{
    "DB": {
        "diskReadBytes": {…},
        "cacheReadBytes": {…},
        "cacheMissBytes": {…},
        "diskWriteBytes": {…},

        "diskReadCount": {…},
        "cacheReadCount": {…},
        "cacheMissCount": {…},
        "diskWriteCount": {…},

        "dataSegment1": {…},
        "indexSegment": {…},

        "tables": {…},
        "indexes": {…}

    }
}
```

こ のオブジェクトは8つの基本的な計測値("diskReadBytes", "cacheReadBytes", "cacheMissBytes", "diskWriteBytes", "diskReadCount", "cacheReadCount", "cacheMissCount", "diskWriteCount")と、追加のプロパティ("dataSegment1", "indexSegment", "tables", "index")から構成されています。また、異なる階層で異なるスコープの要素プロパティを格納してることもあります(詳細は以下を参照して下さい)。

注: プロパティは、中身を受け取った場合のみ、オブジェクトの中に存在します。中身がないプロパティはオブジェクトの中には含まれません。例えば、デー タベースが読み込み専用モードで開かれインデックスが使用されていなかった場合、返されたオブジェクトには、"diskWriteBytes", "diskWriteCount", "indexSegment", "indexes" が格納されていません。

要素プロパティ

要素プロパティは、データベースオブジェクトの様々な階層に存在します。同じ情報を異なるスコープから返します。要素プロパティの詳細は以下の通りです:

名前	返される情報
diskReadBytes	ディスクから読み出したバイト
cacheReadBytes	キャッシュから読み出したバイト
cacheMissBytes	キャッシュからの読み出しに失敗したバイト
diskWriteBytes	ディスクに書き込まれたバイト
diskReadCount	ディスクからの読み出しアクセス
cacheReadCount	キャッシュからの読み出しアクセス
cacheMissCount	キャッシュからの読み出しに失敗したアクセス
diskWriteCount	ディスクへの書き込みアクセス

8つの要素プロパティは全て同じオブジェクト構造を持ちます。例えば:

```
"diskReadBytes": {     "value": 33486473620,     "history": [         // 任意          {"value": 52564,"time": -1665},         {"value": 54202,"time": -1649},             …     ] }
```

- "**value**" (数字): この"value"プロパティにはバイトの量かアクセスの回数が格納されます。基本的には、この値は"history"オブジェクトの合計値となっています("history"オブジェクトが存在しない場合も同様です)。

- "**history**" (オブジェクト配列): "history" オブジェクト配列は、秒ごとにグループ分けされたイベント値の集合です。"history"プロパティは*options* 引数を通してリクエストされた場合にのみ表示されます(以下を参照して下さい)。"history" 配列は最大で200アイテムを格納する事ができます。配列内の各要素はそれ自体がオブジェクトであり、二つのプロパティを格納します:"value" と "time"です。
  - "value" (数字): 関連付けられた"time"プロパティで指定された時間内に扱われたバイトまたはアクセスの量。
  - "time" (数字): そのファンクションが呼び出されてから経過した秒数。上記の例の("time": -1649) は、1649秒前(厳密には1649秒前から1650秒前の間)を意味します。この1秒間に、54,202 バイトがディスク上から読み出されました。
    この例でのhistoryの配列には連続した値(-1650,-1651,-1652, 等)は格納されていません。これの前の値は -1665で、これはつまり1650秒前から1665秒前までの15秒間はディスクから何も読み出されていないことを意味します。
    注: デフォルトでは、配列には意味のある情報しか含まれません。
    配列の最大サイズが200なので、データベースが頻繁に使用されている(例えばディスク上から毎秒何かが読み出されている)場合、historyの長さの上限は200秒となります。反対に、3分に1度だけ読み込みが発生する以外には何もデータベース起こらない場合には、historyの長さは600分(3×200)となります。
    この例の場合は、以下の表のようにあらわされます:

4D internal history		Requested history: 30	
time	value	time	value
-2	4629	0	0
-4	7788	-1	0
-6	3718	-2	4629
-8	8814	-3	0
-10	3925	-4	7788
-12	775	-5	0
-14	6807	-6	3718
-16	3265	-7	0
-18	8086	-8	8814
-20	2539	-9	0
		-10	3925
		-11	0
		-12	775
		-13	0
		-14	6807
		-15	0
		-16	3265
		-17	0
		-18	8086
		-19	0
		-20	2539
		-21	-1
		-22	-1
		-23	-1
		-24	-1
		-25	-1
		-26	-1
		-27	-1
		-28	-1
		-29	-1
		-30	-1

dataSegment1 と indexSegment

"dataSegment1" と "indexSegment"プロパティには、最大で4つの要素プロパティが格納されます:

```
"dataSegment1": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
```

```
      "diskReadCount": {…},
      "diskWriteCount": {…}
      },
   "indexSegment": {
      "diskReadBytes": {…},
      "diskWriteBytes": {…},
      "diskReadCount": {…},
      "diskWriteCount": {…}
      }
```

これらのプロパティは要素プロパティと同じ情報を返しますが、それぞれのデータベースファイルに特化した情報を返します:

- "dataSegment1" はディスク上の.4DD データファイルの情報を返します。
- "indexSegment" はディスク上の.4dx インデックスファイルの情報を返します。

例えば、以下のオブジェクトが返ってきます:

```
{ "DB": { "diskReadBytes": {     "value": 718260     }, "diskReadCount": {     "value": 229     },
"dataSegment1": {     "diskReadBytes": {     "value": 679092     },     "diskReadCount": {     "value": 212
}     }, "indexSegment": {     "diskReadBytes": {     "value": 39168     },     "diskReadCount": {     "value":
17     } }
```

以下の様に返された値を計算する事で、どのように動作しているのか確認することができます:

diskReadBytes.value = dataSegment1.diskReadBytes.value + indexSegment.diskReadBytes.value
diskWriteBytes.value = dataSegment1.diskWriteBytes.value + indexSegment.diskWriteBytes.value
diskReadCount.value = dataSegment1.diskReadCount.value + indexSegment.diskReadCount.value
diskWriteCount.value = dataSegment1.diskWriteCount.value + indexSegment.diskWriteCount.value

tables

"tables" プロパティには、データベースが開かれて以来、読み込み・書き込みのいずれかでアクセスされたテーブルの数だけプロパティが格納されています。それぞれのプロパティ名は関連するテーブル名となっています。例えば:

```
"tables": {     "Employees": {…}     "Companies": {…}     }
```

それぞれのテーブルプロパティには、10個のプロパティが格納されています:

- 最初の8つのプロパティはアクセスのあったテーブルに関連した値を格納した要素プロパティ(上記参照)です。
- 残り二つのプロパティ"records" と "blobs"には、それぞれ同じように8つの要素プロパティがあり、特定のフィールドの型に関連する情報だけが格納されています:
  - "records" プロパティはテーブル全体のフィールド(文字列、日付、数字、等)のうち、テキスト、ピクチャー、Blobのフィールドを除いた情報が格納されています
  - "blobs" プロパティには、テーブルのうちテキスト、ピクチャー、Blobフィールドの情報が格納されています
- テーブルで実行されたクエリと並び替えによっては、"fields" と "queries"というプロパティが表示されている場合があります:
  - "fields" プロパティには、クエリと並び替えに使用されたフィールド数と同じ数だけ"field name" 属性(これはサブオブジェクトでもあります)が格納されています。
    それぞれのフィールド名オブジェクトには以下のものが格納されます:
    - そのフィールドを使用してクエリが実行されていた場合には"queryCount" オブジェクト(*options* 引数の値によって履歴の有無を指定する事ができます)。
    - そのフィールドを使用して並び替えが実行されていた場合には"sortCount" オブジェクト(*options* 引数の値によって履歴の有無を指定する事ができます)。

    この属性はインデックスの使用には基づいていません。クエリや並べ替えが考慮の対象となります。
    例: データベースが起動した瞬間から、複数のクエリと並べ替えが*CompID*、*Name* そして *FirstName* フィールドを使用して実行されてきました。返されたオブジェクトには以下のような"fileds"サブオブジェクトが格納されます(*options* 引数ではパスあり、履歴なしを指定しています):

```
{     "DB": {         "tables": {             "Employees": {                 "fields": {
                    "CompID": {                     "queryCount": {                         "Name": {
"value": 3                      }                      },                      "value": 1
                       "queryCount": {                          "value": 3
},                        "sortCount": {                  },                      "FirstName": {
                       }                                                       "value": 2
                       "sortCount": {
}                     } (...)
```

注:"fields" 属性はテーブル上でクエリまたは並び替えが実行された場合にのみ作成されます。そうでない場合はこの属性は存在しません。

- "queries" はテーブル上で実行されたそれぞれのクエリの詳細を提供するオブジェクトの配列です。配列のそれぞれの要素は3つの属性を格納します:
  - "queryStatement" (文字列): クエリ文字列(検索値ではなくフィールド名を含む)。例えば"(Companies.PK_ID != ?)"
  - "queryCount" (オブジェクト):
    - "value" (数値): 検索値に関係なく、クエリ宣言が実行された回数
    - "history" (オブジェクト配列) (*options*引数でリクエストされた場合に限る): "value" と "time" の標準の履歴のプロパティ
  - "duration" (オブジェクト) ("value" の値が>0の場合)
    - "value" (数値): ミリ秒数
    - "history" (オブジェクト配列) (*options*引数でリクエストされた場合に限る): "value" と "time" の標準の履歴のプロパティ

  例題: データベースが起動した瞬間から、単一のクエリがEmployees テーブル上で実行されてきました。(*options* 引数ではパスあり、履歴ありを指定しています):

```
{     "DB": {         "tables": {             "Employees": {                 "queries": [
                    {                           "queryStatement": "(Employees.Name == ?)",
                        "queryCount": {                             "value": 1,
                            "history": [                                {
                                    "value": 1,                                         "time": -2022
                                }                               ]                           },
                        "duration": {                           "value": 2,
                            "history": [                                {
                                    "value": 2,                                         "time": -2022
                                }                               ]                           }
                    }, (...)
```

注: "queries" 属性はテーブル上で少なくとも一つのクエリが実行された場合に作成されます。

indexes

これがもっとも複雑なオブジェクトです。一つ以上のインデックスを使用してアクセスされた全てのテーブルがプロパティとして保存され、そのプロパティ内には使用されたインデックス名もプロパティとして格納されています。キーワードインデックスは個別に表示され、後ろに"*(Keyword)*"が付属します。さらに、それぞれのインデックス名のプロパティオブジェクトには、そのインデッ クスに関連した8つの要素プロパティと、データベースが起動してからのインデックスの使用状況に応じて最大で4つまでのサブオブジェクトが格納されます(それぞれのサブオブジェクトは、データベースが起動して以降、それに対応するオペレーションが実行された場合にのみ存在します)。

例: データベースが起動したときから、[Employees]EmpLastName フィールドの複数のインデックスがアクセスされています。それに加え、[Companies] テーブル内で2レコードが作成され、16レコードが削除されました。このテーブルはインデックス付けがなされている"name" フィールドがあります。このテーブルはまた、このフィールドを使用してクエリと並び替えが行われました。返されるオブジェクトには、以下のようなデータが格納されます:

```
"indexes": {     "Employees": {         "EmpLastName": {                     "diskReadBytes": {…},
        "cacheReadBytes": {…},                  "cacheMissBytes": {…},                  "diskWriteBytes":
{…},                  "diskReadCount": {…},                  "cacheReadCount": {…},
"cacheMissCount": {…},                  "diskWriteCount": {…}             }         "EmpLastName (Keyword)":
{...},         "index3Name": {…},         "index4Name": {…},         …         }         "Companies": {
    "Name":             (...)                 "queryCount": {                     "value": 41                },
                "sortCount": {                      "value": 3                 },
"insertKeyCount": {                     "value": 2                 },                 "deleteKeyCount": {
                "value": 16                 }     table3Name: {…} }
```

options 引数

options 引数を使用すると、コマンドによって返される実際の情報をカスタマイズすることができます。*options* には、プロパティを3つまで格納することのできるオブジェクトを渡します。その3つとは、"withHistory", "historyLength", そして "path"です。

プロパティ	型	詳細
"withHistory"	ブール	"true" を渡した場合、historyは返されたオブジェクト内で呼び出されるファンクションによって返されることを表します。"false"を渡した場合、ファ ンクションによって返されるオブジェクトにはhistoryが格納されないことを意味します。
"historyLength"	数値	返される history 配列のサイズを秒数で定義します(*)
"path"	文字列 | 文字列配列	特定のプロパティへのフルパスまたは取得したい特定の全てのプロパティへのフルパス。文字列を渡した場合、対応する値のみが"DB"オブジェクト内に返されます(パスが有効な場合)。例: "DB.tables.Employees.records.diskWriteBytes"などです。文字列の配列を渡した場合、全ての対応する値が"DB"オブジェクトへと返されます(パスが有効な場合)。 例:["DB.tables.Employee.records.diskWriteBytes", "DB.tables.Employee.records.diskReadCount","DB.dataSegment1.diskReadBytes"]

(*) 上記にあるように、historyは連続した秒としてではなく、関連した値として保存されます。数秒間何も起こらなければ、何も保存はされず、内部 history配列には空欄が表示されます。"time"には、例えば -2, -4, -5, -10, -15, -30が格納され、それぞれに 200, 300, 250, 400, 500,150という値が附属してくる、といったような形です。"historyLength"が600未満(10分未満)の値に設定された場合、返され た配列にはtimeとして0, -1, -2, -3 … -599が返されますが、値が入っているのは-2, -4, -5, -10, -15, -30 のtimeのみです。他のものには全て0が値として入れられます。また上記にあるように、内部配列の唯一の上限はサイズ(200)だけであり、時間ではあ りません。これはつまり特定のプロパティが活発でなかった場合、一番古いtimeはとても大きい値になり得るという事です(例えば-3600＝1時間前、 等)。またデータベース開始直後には200未満の値しかない場合もあります。これらの場合には、内部history timeがリクエストされたものより若い、または関連する全ての値が返された配列内で既に設定されている場合、返される値は-1になります。
例: データベースはつい20秒前に開かれ、historyが60秒前をリクエストしたとします。返された値のうち0から-20は値が0に設定され、他のものは -1へと設定されます。"-1"の値が返された場合、これはリクエストされた時間が古すぎるか、または値が内部history配列内に残っていないことを 意味します(200アイテムという上限に達したため、古い値は削除されています)。

クライアント/サーバー、そしてコンポーネントについて

このコマンドはデータベースの使用についての情報を返します。つまり、関連した値を含む有効なオブジェクトが返されるのは、以下の様に呼び出された場合に限ります:

- 4D ローカルモード(コンポーネントから呼び出された場合、コマンドはホストデータベースについての情報を返します)。
- クライアント/サーバーモードにおけるサーバー側

コマンドがリモートの4Dから呼び出された場合、オブジェクトは空のまま返されます。
このコンテキストにおいて、サーバー側にあるデータベースの情報を取得したい場合には、もっとも簡単な手段は"Execute on server"オプションが有効化されているメソッドを作成してしまう事です。
この原理はコンポーネントに対しても同様です:もしコンポーネントがローカルの4Dにおいて呼び出された場合、返されるのはホストデータベースについての情報です。4D リモートの場合には、サーバーデータベースについての情報を返します。

例題 1

返されたオブジェクト内にhistoryのログを残したい場合:

```
C_OBJECT($param)
C_OBJECT($measures)
OB SET($param;"withHistory";True)
$measures:=Get database measures($param)
```

例題 2

キャッシュ内で読み込まれた全体のバイト数("cacheReadBytes")だけを知りたい場合:

```
C_OBJECT($oStats)
C_OBJECT($oParams)
OB SET($oParams;"path";"DB.cacheReadBytes")
$oStats:=Get database measures($oParams)
```

返されたオブジェクトには、以下の様な情報が含まれます:

```
{     "DB": {          "cacheReadBytes": {               "value": 9516637          }     } }
```

例題 3

直近の2分間に読み込みされたキャッシュのバイトサイズを取得したい場合:

```
C_OBJECT($oParams)
C_OBJECT($measures)
OB SET($oParams;"path";"DB.cacheReadBytes")
OB SET($oParams;"withHistory";True)
OB SET($oParams;"historyLength";2*60)
$measures:=Get database measures($oParams)
```

Get locked records info

Get locked records info (aTable) -> 戻り値

引数	型		説明
aTable	テーブル	➡	ロックされたレコードについて調べたいテーブルを指定
戻り値	Object	⮌	ロックされたテーブルについての詳細

説明

4D v14 R3 から導入

テーマ: レコードロック

新しい**Get locked records info** coコマンドは、 *aTable* で指定したテーブル内で現在ロックされているレコードについての情報を含んだ*object* を返します。

返されたオブジェクトは、オブジェクトの配列である"records"プロパティを格納しています:

```
{
    "records": [
        description object,
        (…)
    ]
}
```

それぞれの"description object" 配列の要素は、指定されたテーブル内でのロックされたレコードを検知し、以下のプロパティに情報を格納します:

プロパティ	型	詳細
contextID	UUID (文字列)	ロックをしているデータベースコンテキストのUUID
contextAttributes	オブジェクト	**LOCKED BY** コマンドをレコードに適用したときと同じ情報を含むオブジェクト。**LOCKED BY**との違いは、**Get locked records info** はシステムで定義されたユーザー名を返し、4Dユーザー名を返すのではないという点です(以下の説明を参照)
recordNumber	倍長整数	ロックされたレコードのレコード番号

注: コマンドの挙動をより分かりやすくするために、4D v14 R3 以降 **LOCKED ATTRIBUTES** コマンドは **LOCKED BY** と改名されました。

contextAttributes オブジェクトは以下のプロパティから構成されています:

- task_id: プロセスの参照番号
- user_name: OSによって定義されたユーザー名
- user4d_id: 4Dユーザー番号
- host_name: ホストマシンの名前
- task_name: プロセス名
- client_version: クライアントアプリケーションのバージョン

注: 以下のコードを使用することによって、 *user4d_id* の値から4Dユーザー名を取得することができます:

```
GET USER LIST($arrNames;$arrIDs)
$4DUserName:=Find in array($arrIDs;user4d_id)
```

注: このコマンドは 4D と 4D Server に対してのみ有効です。リモートの4D またはコンポーネントから呼び出された場合には常に無効なオブジェクトを返します。ただし、 "Execute on server" オプションが有効化されている場合には呼び出し可能です。この場合返されるオブジェクトには、4Dリモートからの呼び出しの場合にはサーバーの、コンポー ネントからの呼び出しの場合にはホストデータベースの情報が含まれます。

例題

以下のコードを実行します:

```
$vOlocked :=Get locked records info([Table])
```

[Table]のテーブル内にて二つのレコードがロックされていた場合には、以下の様なオブジェクトが$vOlockedに返されます:

```
{
    "records": [
        {
            "contextID": "A9BB84C0E57349E089FA44E04C0F2F25",
            "contextAttributes": {
                "task_id": 8,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 1
        },
        {
            "contextID": "8916338D1B8A4D86B857D92F593CCAC3",
            "contextAttributes": {
                "task_id": 9,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 2
        }
    ]
}
```

コードが4D Server上で実行され、リモートのクライアントマシンによってロックされている場合、以下の様なオブジェクトが$vOlocked 内に返されます:

```
{
    "records": [
        {
            "contextID": "B0EC087DC2FA704496C0EA15DC011D1C",
            "contextAttributes": {
                "task_id": 2,
                "user_name": "achim",
                "user4d_id": 1,
                "host_name": "achim-pcwin",
                "task_name": "P_RandomLock",
                "is_remote_context": true,
                "client_uid": "0696E66F6CD731468E6XXX581A87554A",
                "client_version": -268364752
            },
            "recordNumber": 1
        }
    ]
}
```

IMPORT STRUCTURE

IMPORT STRUCTURE (xmlStructure)

引数	型		説明
xmlStructure	テキスト	➡	4D データベースストラクチャーのXML定義ファイル

説明

4D v14 R4 から導入

テーマ: ストラクチャーアクセス

新しい**IMPORT STRUCTURE** コマンドは、カレントデータベース内に、 *xmlStructure* に渡した4D XMLストラクチャー定義を読み込みます。

xmlStructure 引数にはXMLフォーマットでの有効な4Dストラクチャー定義を渡す必要があります。以下の機能うちのどれか一つを使用して、有効なストラクチャー定義を得ることができます:

- 新しい **EXPORT STRUCTURE** コマンドを実行します。
- 4D デザインモードのインターフェースでのメニューから、**書き出し -> ストラクチャー定義をXMLファイルに書き出し...** を選択します(**ストラクチャー定義の書き出しと読み込み**を参照して下さい)
- 4D アプリケーションの"DTD"フォルダ内のPublic文書型宣言に基づいてカスタムのXMLファイルを作成、または編集します。

イ ンポートされたストラクチャー定義はカレントのストラクチャーに追加され、既存のテーブル(存在する場合)と一緒に標準の4D ストラクチャーエディター内に表示されます。読み込んだテーブルがもともとあったテーブルと同じ名前のとき、エラーが生成され、読み込みオペレーションは キャンセルされます。

ストラクチャーは空のデータベースにも読み込むことができ、その結果新しいデータベースを作成することになります。

ストラクチャーが読み込みのみモード、もしくはコンパイル済みの場合はエラーが生成されます。また、ストラクチャーは4Dリモートアプリケーションから呼び出すことはできません。

例題

保存されたストラクチャー定義を、カレントデータベースに読み込む場合を考えます:

```
$struc:=Document to text("c:\\4DStructures\\Employee.xml")
IMPORT STRUCTURE($struc)
```

LISTBOX GET CELL COORDINATES

LISTBOX GET CELL COORDINATES ({* ;} object ; column ; row ; left ; top ; right ; bottom)

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はフィールドまたは変数
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
column	倍長整数	➡	列番号
row	倍長整数	➡	行番号
left	倍長整数	⬅	オブジェクトの左座標
top	倍長整数	⬅	オブジェクトの上座標
right	倍長整数	⬅	オブジェクトの右座標
bottom	倍長整数	⬅	オブジェクトの下座標

説明

4D v14 R5 から導入

LISTBOX GET CELL COORDINATES コマンドは引数 * および*object*によって指定されたリストボックス内の、*column* と *row* 引数で指定したセルの*left*、*top*、*right* および *bottom*にそれぞれ左端、上端、右端、下端の座標を(ポイント単位で)返します。

任意の * 引数を指定した場合、*object*はオブジェクト名です(文字列)。任意の * 引数を省略すると、*object* はフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照(フィールドまたは変数のみ)を指定します。

[#title id="663　"/] コマンドとの統一性のため、原点はセルを含むフォームの左上端になります。また返される座標は理論値となります。この値は、クリッピングが起こるまではス クロールを考慮に入れます。結果として、そのセルは表示されていない(または一部しか表示されていない)こともあり、座標の位置はフォームの範囲を超えて いる(負の数値が返される)こともあります。セルが表示されているか(また、表示されているならどの部分が表示されているか)を調べるためには、返された 座標と、リストボックスの座標を比較する必要があります。その際、以下の点に注意する必要があります:
全てのセルは、親のリストボックスの座標(リストボックスに対しての**OBJECT GET COORDINATES** の値)に沿ってクリップされています。

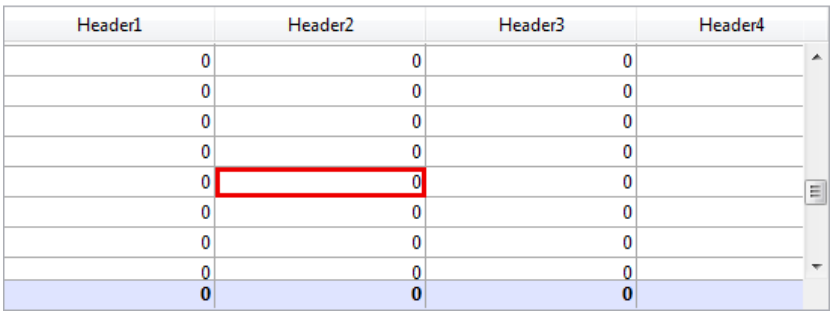
- ヘッダー・フッターのサブオブジェクトは、列のコンテンツの上に表示されています。セルの座標がヘッダーやフッターの線と交差する場合には、その部分のセルは表示されません。
- ロックされた列の要素は、スクロール可能な列の要素の上に表示されます。スクロール可能な列の要素がロックされた列の要素と交差するとき、スクロール可能な列の要素は表示されません。

より詳細な情報に関しては、**OBJECT GET COORDINATES** コマンドの詳細を参照して下さい。

例題

リストボックス内の選択されたセルの周りに赤い長方形を描画する場合を考えます:

```
OBJECT SET VISIBLE(*;"rectangleInfo";False) //赤い長方形を初期化
 //長方形はフォーム内のどこかに既に定義済み
LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
OBJECT SET VISIBLE(*;"RedRect";True)
OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)
```



METHOD GET ATTRIBUTES

METHOD GET ATTRIBUTES (path ; attributes {; *})

引数	型		説明
path	テキスト, テキスト配列	➡	メソッドのパス
attributes	Object, Object array	⬅	選択したメソッドの属性
*	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)

説明

新しい**METHOD GET ATTRIBUTES** コマンドは、*path* 引数で指定されたメソッドの全ての属性のカレント値を*attributes* 引数に返します。

このコマンドはプロジェクトメソッドに対してのみ使用できます。*path* 引数に無効なパスを渡した場合、エラーが生成されます。

path 引数にはメソッドパスを含んだテキストか、パスの配列を含んだテキスト配列を渡す事ができます。*attributes* 引数には、属性を適切に取得するために、同様の引数(文字列または配列)を渡す必要があります。

attributes 引数には、*path* 引数に渡した引数の種類に応じて、オブジェクトまたはオブジェクトの配列を渡します。全てのメソッドの属性はオブジェクトプロパティとして返され、その内部は"true"/"false" 値を持つブール型の属性ですが、必要に応じて追加の値が渡されます(例えば"scope":"table" 4D Mobile property など)。

このコマンドがコンポーネントから実行された場合、デフォルトではコンポーネントメソッドに対して適用されます。* 引数を渡した場合、ホストデータベースのメソッドへとアクセスします。

注: 既存の**METHOD Get attribute****METHOD Get attribute** コマンドは引き続きサポートはされますが、ブール型の値しか返せないため、4D Mobileプロパティのような拡張された属性に対しては使用はできません。

例題

sendMail プロジェクトメソッドの属性を取得したい場合を考えます。以下の用にコードを書くことができます:

```
C_OBJECT($att)
METHOD GET ATTRIBUTES("sendMail";$att)
```

実行後、$att には例えば以下のような値が含まれます:

```
{     "invisible":false,     "publishedWeb":false,     "publishedSoap":false,     "publishedWsdl":false,
"shared":false,     "publishedSql":false,     "executedOnServer":false,     "published4DMobile":{
"scope":"table",          "table":"Table_1"     } }
```

METHOD SET ATTRIBUTES

METHOD SET ATTRIBUTES (path ; attributes {; *})

引数	型		説明
path	テキスト, テキスト配列	➡	メソッドのパス
attributes	Object, Object array	➡	選択したメソッドで設定する属性
*	演算子	➡	If passed = command applies to host database when executed from a component (parameter ignored outside of this context)

説明

METHOD SET ATTRIBUTESコマンドは、*path*引数で指定したメソッドの、*attributes*引数で指定した値を設定することができます。

*path*引数にはメソッドパスを含んだテキストか、パスの配列を含んだテキスト配列を渡す事ができます。*attributes*引数には、属性を適切に設定するために、同様の引数(文字列または配列)を渡す必要があります。このコマンドはプロジェクトメソッドに対してのみ使用できます。*path*引数に無効なパスを渡した場合、エラーが生成されます。

*attributes*引数には、メソッドに対して設定した属性を全て含んだオブジェクトまたはオブジェクトの配列を渡します(渡した*path*引数の種類によります)。

メソッドの属性は**OB SET**または**OB SET ARRAY**コマンドを使用して設定する必要があります。ブール型の属性に対してTrueまたはFalseの値を、拡張された属性に対しては特定の値(例えば4D Mobileプロパティにおいて"scope":"table" など)を設定します。*attributes*引数に存在する属性のみがメソッド属性内で更新されます。

コマンドがコンポーネントから実行された場合、デフォルトではコンポーネントメソッドに対して適用されます。* 引数を渡した場合、コマンドはホストデータベースのメソッドへとアクセスします。

注: 既存の**METHOD SET ATTRIBUTE**コマンドは引き続きサポートされますが、このコマンドはブール型の値しか扱えないために、4D Mobileプロパティなどの拡張された属性に対しては使用することができません。

サポートされる属性は以下の通りです:

```
{     "invisible" : false, // 表示状態であればtrue     "publishedWeb" : false,  // 4D tags と URLを通して使用可能であればtrue     "publishedSoap": false,  // Webサービスとして提供されていればtrue     "publishedWsdl": false,  // WSDLで公開されていればtrue     "shared" : false,  // コンポーネントとホストデータベースで共有されていればtrue     "publishedSql" : false,  // SQLを通して利用可能であればtrue     "executedOnServer" : false, // サーバー側で実行されていればtrue     "published4DMobile" : { "scope": "table",  // "none" または "table" または "currentRecord" または "currentSelection"         "table": "aTableName"  // scope が "none" 以外の場合には表示される     } }
```

注: "published4DMobile" 属性については、"table"の値が存在しない、または"scope" が有効でない場合には、これらの属性は無視されます。

例題 1

属性を一つだけ設定したい場合を考えます:

```
C_OBJECT($attributes)
OB SET($attributes;"executedOnServer";True)
METHOD SET ATTRIBUTES("aMethod";$attributes) //"executedOnServer" 属性のみが変更されます
```

例題 2

メソッドを、4D Mobileを経由では使用不可にしたい場合を考えます("scope"属性には"none"値を渡す必要があります):

```
C_OBJECT($attributes)
C_OBJECT($fourDMobileAttribute)
OB SET($fourDMobileAttribute;"scope";"none")
```

```
OB SET($attributes;"published4DMobile";$fourDMobileAttribute)
METHOD SET ATTRIBUTES("aMethod";$attributes)
```

MOBILE Return selection

MOBILE Return selection (aTable) -> 戻り値

引数	型		説明
aTable	テーブル	➡	返したいカレントセレクションを含んだテーブル
戻り値	Object	⮌	Wakanda準拠のセレクション

説明

4D v14 R3 から導入

MOBILE Return selection コマンドは、*object* 内に、 *aTable* のカレントセレクションをWakandaに準拠したentity collectionへと変換したものを、JSONオブジェクトとして返します。

このコマンドは、4D Mobile接続(通常はRESTを経由した4DとWakanda間の接続)のコンテキストにおいて呼び出されることを想定しています。4D Mobile接続が確立され適切なアクセス権が設定されると、Wakandaは*$0* 引数に値を返す4Dプロジェクトメソッドを実行する事ができます。

MOBILE Return selection コマンドは、*aTable* で指定したテーブルのレコードのカレントセレクションを、JSONフォーマットの *entity collection* オブジェクト形式で*$0* 引数に返します。このオブジェクトはWakandaでレコード(または*entities*)のセレクションを内包するentity collectionsに準拠しています。

4D Mobileアクセスのためには、4Dデータベース内において、以下の特定の設定をしなければならないことに注意して下さい:

- Webサーバーが起動している必要があります。
- データベース設定内にて、"4D Mobile サービスを有効化"のオプションがチェックされているかどうかを確認して下さい。
- 有効なライセンスが必要になります。
- 公開したテーブルとフィールドがどちらも"4D Mobileサービスで公開"のオプションにチェックがされていなければなりません(デフォルトではチェックがされている)。
- 呼び出されるメソッドは、"4D Mobile からの利用を許可"のオプションにチェックがされている必要があります(デフォルトではチェックされていません)。

aTable には、有効なテーブルであればデータベース内のどんなテーブルでも渡す事ができ、メソッドプロパティにてテーブルと関連付けがなされているテーブルに限ら ないという点に注意して下さい。この引数はメソッドが呼び出し可能なオブジェクトをWakanda側で判断するためにのみ使用されます。

4D Mobileの設定についての詳細な情報に関しては、**4D Mobile**ドキュメントを参照して下さい。

例題

[Countries] テーブルのクエリに基づいたカレントセレクションを Wakanda のグリッドに表示させたい場合を考えます。

まず、以下の様な4Dメソッドを作成します:

```
   //FindCountries プロジェクトメソッド
   //FindCountries( 文字列 ) -> object

C_TEXT($1)
C_OBJECT($0)
QUERY([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Return selection([Countries])
```

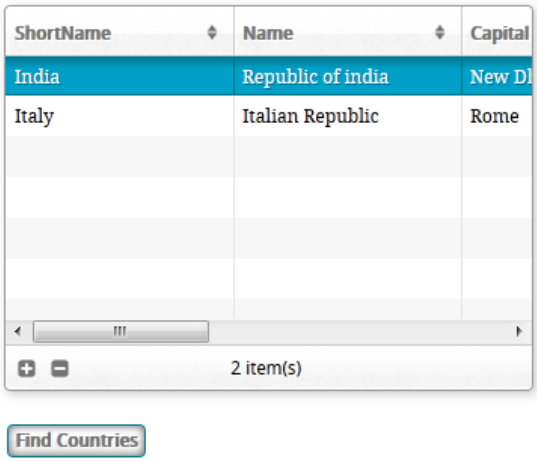
返されたセレクションは有効なコレクションとして、Wakanda 内で直接使用する事ができます。

4D Mobileを経由して4Dと接続しているWakanda Serverにおいて、4DのCountries tableと関連付けられたグリッドを持つページを作成したとします。デフォルトでは、ランタイムでは、4D テーブルからの全てエンティティが表示されています:

ボタンに記述されているコードは以下の通りです:

```
button1.click = function button1_click (event)     {          sources.countries.FindCountries("i", {   //4Dメソッドを呼び出し。"i" は$1として渡されます。                    onSuccess:function(coll){   //コールバックファンクション（非同期）。$0を引数として受け取ります。
                    sources.countries.setEntityCollection(coll.result); //カレントのエンティティコレクションを
                // coll.resultオブジェクト内のものと置き換えます                    }              });             };
```

その結果、グリッドが更新され以下の様になります:



OBJECT Get corner radius

OBJECT Get corner radius ({* ;} object) -> 戻り値

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はフィールドまたは変数
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
戻り値	倍長整数	⮌	丸い角のカレントの半径(ピクセル単位)

説明

4D v14 R4 から導入

テーマ: オブジェクト(フォーム)

新しい**OBJECT Get corner radius** コマンドは、*object* で指定した角の丸い四角オブジェクトの、丸い角のカレントの半径を返します。この値は、プロパティリストを使用してフォームレベルで設定されているか (**新しい角の半径プロパティ** を参照して下さい)、またはカレントプロセスにおいて新しい**OBJECT SET CORNER RADIUS** コマンドを使用して設定されています。

オプションの * 引数を指定した場合、*object* はオブジェクト名です (文字列) 。オプションの * 引数を省略すると、*object* はフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照(フィールドまたは変数のみ)を指定します。

注: 現行の4Dリリースでは、このコマンドは角の丸い四角(静的なオブジェクト)に対してのみ適用可能なため、必ず * 引数を渡してオブジェクト名シンタックスを使用する必要があります。

このコマンドはオブジェクトの角のピクセルの半径の値を返します。デフォルトでは、この値は5ピクセルとなっています。

例題

以下のコードをボタンのメソッドに追加します:

```
C_LONGINT($radius)
$radius:=OBJECT Get corner radius(*;"GreenRect") //カレントの値を取得
OBJECT SET CORNER RADIUS(*;"GreenRect";$radius+1) //半径を大きくする
 // 半径の最大値は自動的に管理されます。
 // 最大値に達した場合、このボタンは何の効力もありません
```

OBJECT SET CORNER RADIUS

OBJECT SET CORNER RADIUS ({* ;} object ; radius)

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)、またはフィールドまたは変数 (* 省略時)
radius	倍長整数	➡	丸い角の新しい半径(ピクセル単位)

説明

4D v14 R4 から導入

テーマ: オブジェクト(フォーム)

新しい **OBJECT SET CORNER RADIUS** コマンドは、*object* 引数で指定した角の丸い四角オブジェクトの角の半径を変更することができます。新しい半径はそのプロセスに対してのみ有効で、フォーム内には保存されません。

オプションの * 引数を渡すと、*object* 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、*object* は変数であり、文字列ではなく変数参照を渡します。

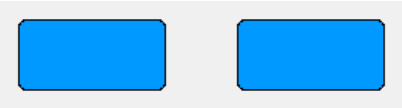
注: 現在の4D R リリースでは、このコマンドはスタティックなオブジェクトである角の丸い四角にのみ適用されるため、常に * 引数を渡し、オブジェクト名シンタックスを使用する必要があります。

radius 引数には、オブジェクトの角の、新しい半径をピクセル単位で渡します。デフォルトでは、この値は5ピクセルとなっています。

注: この値はプロパティリストを使用することにってフォームレベルで設定することも可能です(**新しい角の半径プロパティ** を参照して下さい)。

例題

フォーム内に、以下の様に"Rect1" と "Rect2" と名付けられた長方形が入っています:



以下のコードを実行することによってその角の半径を変えることができます:

```
OBJECT SET CORNER RADIUS(*;"Rect@";20)
```

OPEN DATABASE

OPEN DATABASE (filePath)

引数	型		説明
filePath	文字	➡	開くデータベースファイル(.4db, .4dc, .4dbase, または .4dlink)の名前またはフルパス

説明

4D v14 R3 から導入

テーマ: 4D環境

新しい**OPEN DATABASE** コマンドは、現在開いている4Dデータベースを閉じ、*filePath* で定義されたデータベースを、動的に開きます。このコマンドは自動的にテストをする目的や、コンパイル後にデータベースを自動的に開くのに有用です。

filePath 引数には、開きたいデータベースの名前または完全なアクセスパスを渡します。指定できるファイルの拡張子は以下のとおりです:

- .4db (インタープリタストラクチャーファイル)
- .4dc (コンパイルされたストラクチャーファイル)
- .4dbase (OS X パッケージ)
- .4dlink (ショートカットファイル)

ファイル名のみを渡す場合、現在開いているデータベースのストラクチャーファイルと同じ階層にそのファイルが置かれている必要があります。
ア クセスパスが有効なデータベースを設定していた場合、4Dは開いているデータベースを閉じ、指定されたデータベースを開きます。シングルユーザーモードに おいては、閉じられた方のデータベースの **On Exitデータベースメソッド** と、新たに開かれた方のデータベースの [#title id="142"/] が順番に開かれます。

注: このコマンドは指定されたデータベースを開く前にアプリケーションを強制的に閉じるため、 **On Startupデータベースメソッド** の中やこのデータベースメソッドから呼び出されるメソッド内で呼び出すことは推奨されません。

このコマンドは非同期的に実行されます。つまり、呼び出しの後、4Dは他のメソッドを実行し続けます。そして、アプリケーションは**ファイル**メニューの**4Dを終了**コマンドを選択したのと同じ挙動をします。ファイルを開くダイアログボックスはキャンセルされ、開いているプロセスは全て10秒間の猶予の後に終了します。

指定されたデータベースファイルが見つからないか無効である時、標準のファイルシステムエラーが返され、4Dは何もしません。

このコマンドは標準のデータベースからのみ実行する事ができます。組み込まれたアプリケーション(シングルユーザーまたはサーバーどちらでも)から呼び出された場合、エラー-10509"データベースが開けません"が返されます。

例題

```
OPEN DATABASE("C:\\databases\\Invoices\\Invoices.4db")
```

QUERY BY ATTRIBUTE

QUERY BY ATTRIBUTE (aTable {; conjOp}; objectField ; attributePath ; queryOp ; value {; *})

引数	型		説明
aTable	テーブル	➡	レコードのセレクションを返すテーブル、または省略時にはデフォルトテーブル
conjOp	演算子	➡	複数のクエリ(あれば)を接続するための結合演算子
objectField	フィールド	➡	属性をクエリするオブジェクトフィールド
attributePath	文字	➡	属性の名前またはパス
queryOp	演算子, 文字	➡	クエリ演算子(コンパレータ)
value	テキスト, Number, 日付, 時間	➡	比較する値
*	演算子	➡	クエリを続行するフラグ

説明

テーマ: クエリ

QUERY BY ATTRIBUTE は*objectField*、*attributePath*、*queryOp* そして *value* 引数を使用して定義されたクエリ文字列に合致するレコードを検索し、*aTable* に対しレコードのセレクションを返します。

注: オブジェクトフィールド(4D v15より新たに導入)についての詳細な情報に関しては、**オブジェクトフィールドデータ型** のセクションを参照して下さい。

QUERY BY ATTRIBUTE はカレントプロセスにおいて*aTable* で指定されたテーブルのカレントセレクションを変更し、新しいセレクションの第一レコードをカレントレコードとします。*aTable* 引数が省略されていた場合、コマンドはデフォルトのテーブルへと適用されます。デフォルトテーブルが設定されていない場合、エラーが発生します。

任意の*conjOp* 引数を使用すると、**QUERY BY ATTRIBUTE** の呼び出しを複数のクエリ定義と組み合わせることができます。使用可能な接続演算子は**QUERY** コマンドに対して使用できるものと同じです:

接続子	QUERY BY ATTRIBUTEで使用する記号
AND	&
OR	|
Except	#

conjOp 引数は、複数のクエリの最初の**QUERY BY ATTRIBUTE** の呼び出しには使用しません。また単一のクエリの場合にも使用しません。複数のクエリにおいて省略された場合、AND (&) 演算子がデフォルトで使用されます。

objectField 引数には、クエリしたい属性のオブジェクトフィールドを渡します。そのオブジェクトフィールドが*aTable* 引数で指定したテーブルに自動または手動でリレートした1テーブルに属していた場合、*objectField* には他のテーブルに属するフィールドを指定することもできます。

attributePath 引数には値を比較したい属性の名前またはパスを渡します。単一の属性、例えば"age"などを渡す事もできます。この場合、この名前を持つすべての属性はレコード内にて比較されます。また、"children.girls.age"などのパスを渡す事もできます。この場合、それに合致する属性のみがレコード内にて比較されます。属性"x"が配列であった場合、**QUERY BY ATTRIBUTE** コマンドは、少なくとも一つの要素が条件に合致する属性"x"を含むレコードを検索します。

配列の属性内を検索するためには、*attributePath* 引数内において属性"x"の名前に"[]"を付与することにより、**QUERY BY ATTRIBUTE** コマンドに対し、属性"x"が配列であるという事を指示する必要があります(例3を参照して下さい)。

注:

- 属性名は大文字と小文字が区別されるという点に注して下さい。つまり同じレコード内にて"MyAtt" と "myAtt" という、異なる属性名を持つことができるということです。
- 属性名は不要な空白を取り除くために短縮されます。例えば、" my first attribute .my second attribute " は、 "my first attribute.my second attribute"として解釈されます。

queryOp 引数は、*objectField* 引数と *value* 引数の間に適用される比較演算子です。以下の記号のどれか一つを渡す事ができます:

比較	QUERY BY ATTRIBUTEで使用する記号
同値である	=
同値でない	#
未満	<
を超える	>
以下	<=
以上	>=

注: 比較演算子を記号ではなく、テキスト表現で指定することも可能です。詳細な情報に関しては、**QUERY** コマンドの説明を参照して下さい。

value 引数は、*attributePath* 引数と比較するためのデータです。この値は*attributePath*引数と同じデータ型として評価されるものであればどんな表現も可能です。値は一度だけ、クエリの最初に評価されます。値はそれぞれのレコードに対して毎回評価されることはありません。文字列内に含まれる文字列をクエリする("を含む"クエリ)ためには、ワイルドカード記号(@)を*value* 引数に使用して検索したい文字列を隔離します(例:"@Smith@")。この場合、インデックスの利点を一部しか享受しないという点に注して下さい(データ保存のコンパクト化)。

属性によるクエリのストラクチャーは以下の様になります:

```
QUERY BY ATTRIBUTE([Table] ;[Table]ObjectField ;"attribute1.attribute2";=;value)
```

注: 全ての演算子(ただし"#"は除く)に対して、オブジェクトフィールドには属性が含まれている、というのが暗示的な前提条件になります。しかしながら、"#"演算子に対しては、未定義の属性も使用可能です(以下を参照して下さい)。

オブジェクトフィールドにおいては、"#"演算子を使用すると、フィールド内にて検索した値がどの属性にも存在しないレコードを検索します。このコンテキストにおいては、4Dは同じように以下の例を対応します:

- 属性の値が検索した値とは異なるフィールド
- 属性が存在しない(あるいはヌル値を含む)フィールド

例えば、以下のクエリは、Rexという名前ではない犬を飼っている人のレコードに加えて、犬を飼っていない人、あるいは名前のない犬を飼っている人のレコードも返します:

```
QUERY BY ATTRIBUTE([People];[People]Animals;"dog.name";#;"Rex")
```

その他の例:以下のクエリは、*[Table]ObjectField* 内で値が*value* ではない*attribute2* 属性を含むオブジェクトである*attribute1* 属性を含んでいるオブジェクトを含んでいる全てのレコードに加え、*attribute1* も *attribute2* も含まないオブジェクトフィールドのレコードを返します:

```
QUERY BY ATTRIBUTE([Table] ;[Table]ObjectField ;"attribute1.attribute2";#;value)
```

この原則は配列属性にも適用されます。例えば、:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";#;"paris")
```

上記のクエリは、Parisに住所を持っていないPeopleのレコードを返します。

属性が未定義であるレコードだけを特定して取得したい場合、空のオブジェクトを使用して下さい(例2を参照して下さい)。

複数のクエリの作成

属性によるクエリを複数組み合わせる際には、以下のルールが適用されます:

- 最初のクエリ文字列は接続子を含んではいけません。
- その後のそれぞれのクエリ文字列は接続子から始まります。省略した場合mAND(&)演算子がデフォルトで使用されます。
- 最初のクエリと、最後を除く他の全てのクエリは、 * 演算子を使用しなければなりません。
- **QUERY BY ATTRIBUTE** コマンドは**QUERY** コマンドと組み合わせて使用することができます(例を参照して下さい)。
- クエリを実行するためには、最後の**QUERY BY ATTRIBUTE** コマンドにおいて * 引数を指定してはいけません。その代り、**QUERY** コマンドをテーブルのみで(他の引数を必要とせず)実行する事ができます。

注: それぞれのテーブルは現在ビルトされたクエリを維持します。これはつまり、それぞれのテーブルに対して一つずつ、複数のクエリを同時に作成できるという事です。

どのように定義されたとしても、クエリには以下の制限がつきますNo matter which way a query has been defined:

- 実際のクエリオペレーションが実行にある程度の時間が必要になる場合、4Dは自動的に進捗バー(サーモメーター)を含めたメッセージを表示します。このメッセージは**MESSAGES ON** や **MESSAGES OFF** コマンドを使用することによってオン・オフを切り替えることができます。進捗バーが表示されているとき、ユーザーはストップボタンを押すことによってクエリを中断することができます。クエリが完了すると、OK変数が1に設定されます。それ以外の場合、例えがクエリが中断されたばあなどには、OK変数は0(ゼロ)に設定されます。
- インデックスされたオブジェクトフィールドが指定された場合、クエリは毎回可能な限り最適化されるので(インデックスされたフィールドから先に検索されます)、結果として可能な限り最小限の時間でクエリを終えることができます。

オブジェクト内の日付の値

日付はオブジェクト内において、データベース設定に沿った形で保存されています。デフォルトでは、タイムゾーンは考慮されます(**SET DATABASE PARAMETER** コマンドのJSON use local time を参照して下さい)。

```
!1973-05-22! -> "1973-05-21T23:00:00.000Z"
```

この設定はクエリにおいても影響するので、データベースを常に毎回同じ場所で使用し、データにアクセスる全てのマシンの設定が同じであれば何も心配する必要がありません。この場合、以下のクエリは、Birthday属性が!1973-05-22!("1973-05-21T23:00:00.00Z"として保存されている)に一致するレコードを正確に返します:

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Birthday";=;!1973-05-22!)
```

GMT設定を使用したくない場合、これらの設定を以下の様にして変更する事ができます:

```
SET DATABASE PARAMETER(JSON use local time;0)
```

ただし、この設定のスコープはプロセスのみであるという点に注意して下さい。設定をこのように変更した場合、1965年10月1日は"1965-10-01T00:00:00.000Z"として保存されますが、クエリを実行する前に同じ引数を設定する必要が出てきます:

```
SET DATABASE PARAMETER(JSON use local time;0)
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Birthday";=;!1976-11-27!)
```

例題 1

この例においては、"age"属性は文字列か整数の値であり、年齢が20歳から29歳の人を検索したい場合を考えます。最初の2行は属性を整数としてクエリし(>=20 かつ < 30)、最後の1行はフィールドを文字列としてクエリします("2" で始まるが、"2"ではない値)。

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"age";>=;20;*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"age";<;30;*)
QUERY BY ATTRIBUTE([Persons];|;[Persons]OB_Info;"age";=;"2@";*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"age";#;"2") //実行したいので、ここでは最後の * は無い
```

例題 2

QUERY BY ATTRIBUTE コマンドを使用すると、特定の属性が定義されている(あるいは定義されていない)レコードを探す事ができます。そのためには、空のオブジェクトを使用します。

```
 //オブジェクトフィールド内にてEメールが定義されているレコードを探す
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"e-mail";#;$undefined)
```

```
 //オブジェクトフィールド内にてZIPコード(郵便番号)が定義されていないレコードを探す
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"zip code";=;$undefined)
```

例題 3

配列の属性を含むフィールドを検索したい場合を考えます。以下の二つのレコードがあった時、:

```
{     "name":"martin",     "locations" : [ {                "kind":"office",                "city":"paris"
          } ] } , {     "name":"smith",     "locations" : [ {                "kind":"home",
"city":"lyon"             } , {                "kind":"office",                "city":"paris"             } ]
}
```

... **QUERY BY ATTRIBUTE** コマンドに対して以下の宣言をすると、locationが"paris"である人を探します:

```
 //配列の属性を".[]" シンタックスでフラグ付けする
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations.[].city";=;"paris")
 //"martin" と "smith" を返す
```

注: 同じ配列の属性に対し複数の条件を定義した場合、合致した条件は必ずしも同じ配列の要素に対して適用されるとは限りません。以下の例においては、”kind"が"home"である”locations"要素と、"city"が"paris"である"locations"要素を持っているために"smith"が返されますが、これら二つは同じ要素ではありません:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations.[].kind";=;"home";*)
QUERY BY ATTRIBUTE([People]; & ;[People]OB_Field;"locations.[].city";=;"paris")
 //"smith"を返す
```

WEB Is server running

WEB Is server running -> 戻り値

引数	型		説明
戻り値	ブール		Webサーバーが実行中が作動中であればTrue、それ以外はFalse

説明

4D v14 R2 から導入

新しい**WEB Is server running** コマンド("**Webサーバ**"テーマ)は、4Dにビルトインされているwebサーバーが動作中である場合には**True**を、Webサーバーがオフである場合には**False**を返します。

このコマンドは、それが実行されたWebサーバーの動作状況を返します:

実行されたコンテキスト	どこの状況を返すか
4D スタンドアローンアプリケーション	ローカルの4D Web サーバー
4D Server	4D Server Web サーバー
4D リモートモード(ローカルプロセス)	ローカルの4D Web サーバー
4D リモートモード(4D Server ストアドプロシージャー)	4D Server Web サーバー
4D リモートモード(他の4D のリモートストアドプロシージャー)	リモート4D Web サーバー

例題

Webサーバーが実行中かどうかをチェックしたい場合:

```
If(WEB Is server running)
   ... //実行する処理
End if
```

4D Server

- OS X用4D Serverの64ビット版について(プレビュー)
- リアルタイムモニタリングページ
- 新しいServerNetネットワークレイヤー

OS X用4D Serverの64ビット版について(プレビュー)

---

v15のリリースにおいて、OS X用4D Server 64-bit版のオペレーショナル・プレビュー版が提供されます(これまでにもv14のRリリースプログラムでは、既に複数のプレビュー版が提供されてきました)。

この製品により、お使いの4D Server アプリケーションは64-bitアップルマシンの性能を全て引き出して利用することが出来るようになります。

OS X用4D Server 64-bit版の機能と使用要件の詳細については、4D Serverリファレンスマニュアルの**OS X用4D Serverの64ビット版の使用(暫定版)** の章を参照して下さい。

実装に関する注意: OS X用4D Server 64-bit版のアプリケーション部分は既に最終段階に達していますが、統合されている*ServerNet* ネットワークレイヤー(プレビューリリース内にて提供中)は、まだ少し最適化を必要としている状態です。そのため、OS X用64-ビット版 4D Server は運用には推奨されません。

以下の表は、OS X用64-bit版4D Serverの最初のプレビュー版ではご利用いただけなかった機能の現在の状況をまとめたものです。これらの機能はリリースごとに順次提供されてきました。

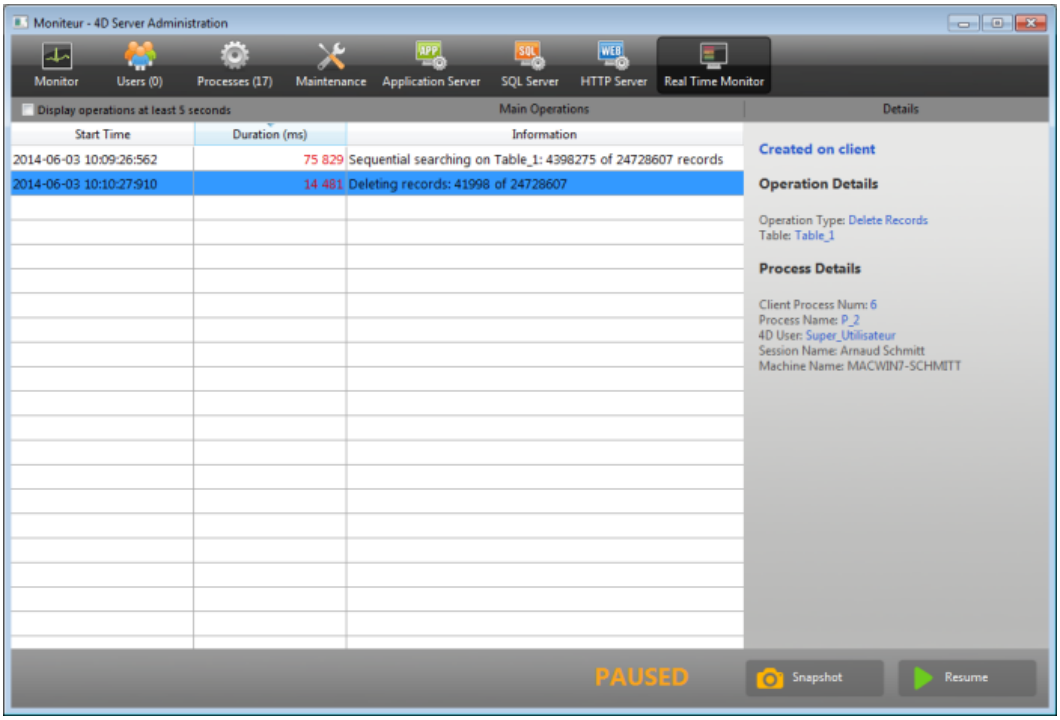
機能/テクノロジー	**現在の状況**
サーバー上で生成されたグラフ	v14 R4以降利用可能
HTTP クライアント(クライアント認証の管理)	v14 R4以降利用可能
4D Internet Commands (プラグイン)	v14 R4以降利用可能
4D Pack (プラグイン)	v14 R5以降利用可能
4D ODBC Pro (プラグイン)	**v15にて利用可能**
4D For OCI (プラグイン)	**v15にて利用可能** (**4D for OCI**を参照して下さい)
シリアルポート通信	利用不可
読み込み/書き出しダイアログボックス	利用不可
ラベルエディター	利用不可

リアルタイムモニタリングページ

*4D v14 R3*から導入

4D Server 管理ウィンドウのリアルタイムモニター(RTM)ページは、トラブルシューティングと最適化のために、より詳細な情報を提供できるよう以下の点が改良されました:

- 新機能(スナップショット、アドバンスドモード)が利用できるようになりました。
- それぞれのオペレーションに対してさらに情報が表示されるようになりました。

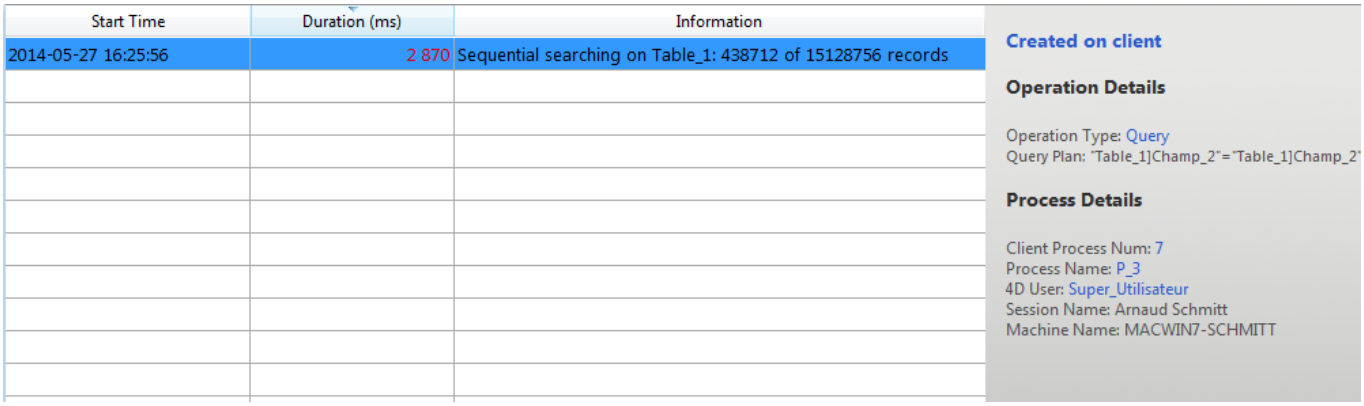


注: **GET ACTIVITY SNAPSHOT** コマンドも、さらなる情報を返すように拡張されました。

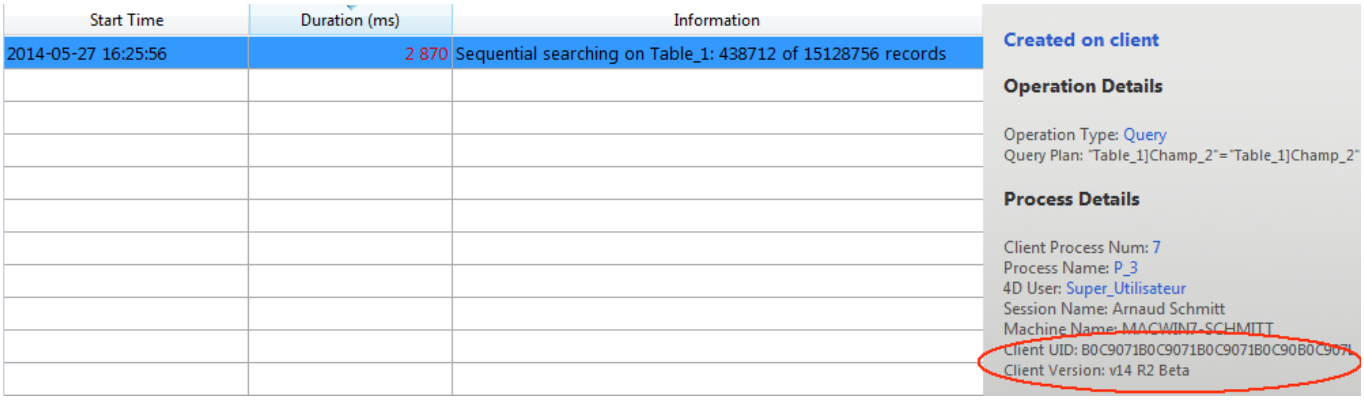
アドバンスドモード

RTMページではさらに情報を表示できるようになり、必要であれば表示されたそれぞれのオペレーションに対しても表示できます。特定のオペレーションに対してアドバンスドモードにアクセスするためには、**Shift** を押しながら見たいオペレーションを選択して下さい。表示可能な情報が全てフィルタリングなしで詳細パネルに表示されます(**GET ACTIVITY SNAPSHOT** コマンドで返されるのと同様です)。表示可能な情報は選択したオペレーションによって異なります。

以下は、標準モードで表示される情報の例です:

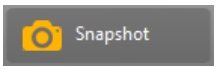


アドバンスドモード(オペレーションを**Shift**+クリック)では、追加の情報が表示されます:



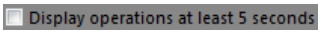
スナップショットボタン

新しいスナップショットボタンは、RTMパネルに表示されている全てのオペレーションをクリップボードへとコピーします。関連した詳細(プロセス情報とサブオペレーション)もそれに含まれます:



オペレーションを最低5秒間表示する

新しい**オペレーションを最低5秒間表示する**オプションにチェックを入れると、表示されたオペレーションは全て、そのページに最低5秒間は表示されるようになります(オペレーションの実行が終わったあとも表示され続けます):



このオプションが適用されたオペレーションは、オペレーションリストの中で灰色に表示されます。

この機能は、実行時間がとても短いオペレーションの情報を取得したい場合に有効です。

新しく追加される情報

ページ内でオペレーションを選択したときに、新しい情報が表示されるようになりました。具体的には以下の情報です:

- **作成された場所:** そのオペレーションの結果がクライアントのオペレーションによる(**クライアント上で作成**)ものなのか、またはストアドプロシージャーもしくはExecute on serverオプションを使用して明示的にサーバー側で開始されたのか(**サーバー上で作成**)を表示します。
- **クエリプラン**: クエリオペレーションに対しては常に表示されます。
- **その他の情報**: オペレーションタイプに応じて、テーブル、フィールド、プロセス、クライアントに関する情報が表示されます。これらの追加の情報は、オペレーションをShift+クリックで選択することで取得可能です。

進捗メッセージはオフに

RTMページに進行中のオペレーションに関する情報が全て集められるようになったため、サーバーマシン側には、デフォルトで進捗メッセージウィンドウは表示されないようになりました。

進捗ウィンドウを表示させたい場合には、サーバー側で **MESSAGES ON** コマンドを呼び出す必要があります。

新しいServerNetネットワークレイヤー

*4D v14 R5*から導入

4D v15 には*ServerNet* という名前の新しいネットワークレイヤーが追加されました。これは4D Serverとリモート4Dマシン(クライアント)間の通信を管理するためのものです。*ServerNet* は現代的で強固なAPIに基づいており、維持が簡単で、最新のネットワークテクノロジーを簡単に導入できる一方、高いレベルのパフォーマンスを発揮することができます。

実装についての注記: *ServerNet* ネットワークレイヤーは 4D v15の "プレビュー" リリースで提供されています。

現在の"旧式"のネットワークレイヤーは既存データベースとの互換性のために引き続きサポートされます。*ServerNet* は新規に作成されたデータベース内では自動的に採用されます。

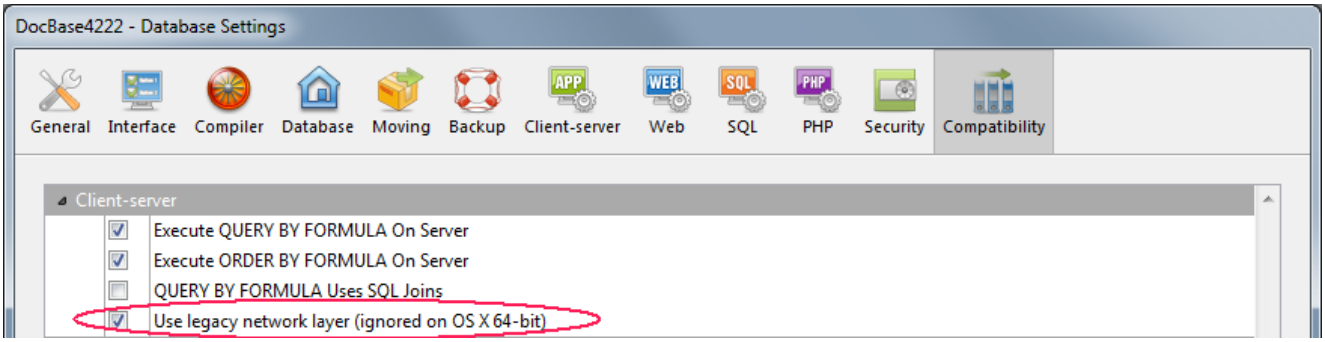
新しいオプションを使用して*ServerNet* を有効化/無効化することができます。

旧式ネットワークレイヤーの有効化または無効化

新しい互換性オプションによって、4D Serverにおける旧式ネットワークレイヤーをいつでも有効化または無効化することができます。以下のどちらかの方法を使用して下さい:

- データベース設定ダイアログボックス内で**旧式ネットワークレイヤーを使用**オプションをチェックする(以下参照)。
- SET DATABASE PARAMETER コマンドに対してUse legacy network layer 定数を使用する(これについては**Get database parameter と SET DATABASE PARAMETER** の章で説明があります)。

新しい互換性オプションは、互換性のページ内にあります:



注: このオプションはOS X用の4D Server 64-bit版では無視されます。このプラットフォームでは*ServerNet* のみがご利用になれます。

デフォルトでは、このオプションは以下の様に設定されています:

- 4D v15(または4D v14 R5)以降で作成されたデータベースにおいてはチェックがされていません(この場合*ServerNet* レイヤーが使用されます)。
- 変換された既存のデータベースにおいてはチェックがされています(この場合旧式のネットワークレイヤーが使用されます)。

このオプションは必要に応じてチェックをしたり外したりすることができます。例えば、クライアントアプリケーションの移行作業の途中などの場合に有効でしょう(以下を参照してください)。

設定を変更した際には、その設定を有効にするためにはアプリケーションの再起動が必要になるという点に注意して下さい。また、接続してたクライアントアプリケーションも、新しいレイヤー設定で接続するためには全て再起動する必要があります(*ServerNet* を使用するための必要最低限のクライアントのバージョンは4D v14 R4です。以下を参照して下さい)。

組み込み4Dクライアントを移行する

サーバーアプリケーションにおいて*ServerNet* レイヤーを有効化した場合、適合する4Dクライアントアプリケーションのみが接続することができます:

- 4D v15(または4D v14 R4)以降のバージョンのクライアントは何も変更しないまま接続することができます。
- それ以前のバージョン(v14.x とそれ以外のv14'R'リリース)のクライアントはサーバーに接続する前にアップグレード

をしなければなりません。

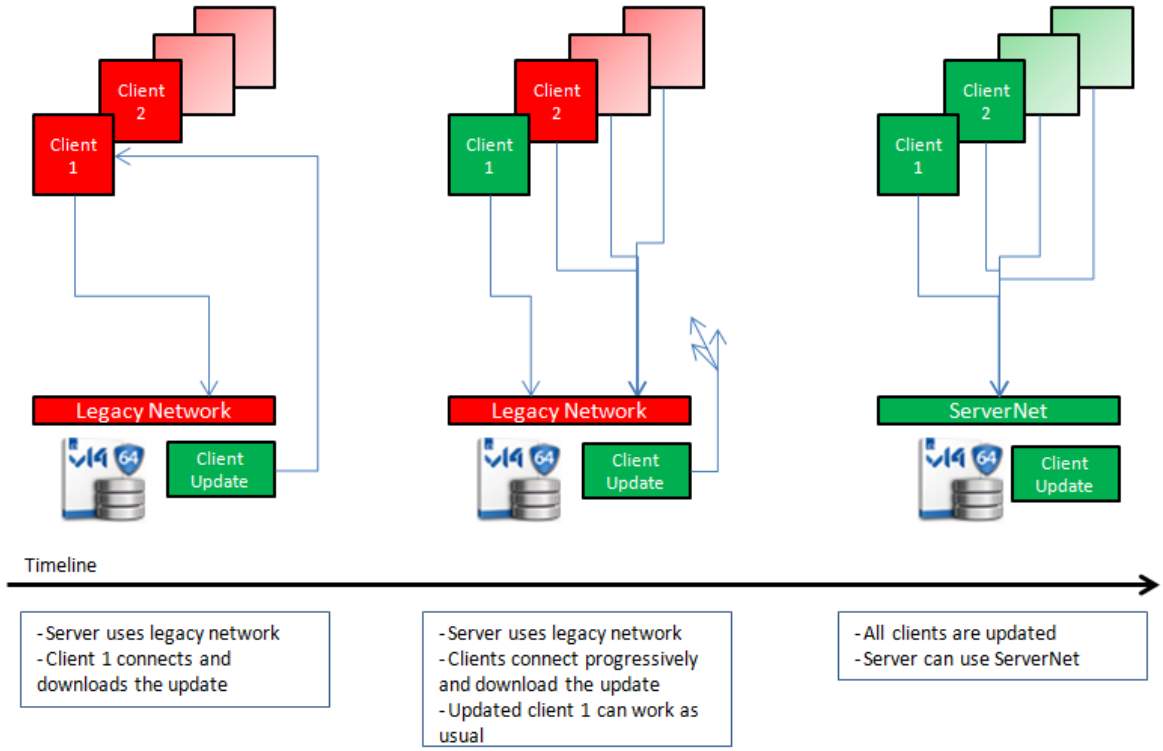
ご自分のアプリケーションがv14 R4以前のバージョンの組み込みクライアントで動いていて、4D Serverの自動機構を使用してアップデートされたクライアントアプリケーションをネットワーク越しに配付したい場合、移行戦略を練る必要があります。この戦略は以下の原則に則って練る必要があります:

- 互換性のないクライアントは旧式ネットワークレイヤーを使用する4D Server にしか接続することができません。
- アップデートされたクライアントはプロトコルを動的に適応させることができるので、サーバーが使用しているネットワークレイヤーに関わらず4D Server v15以降に接続することができます。

移行戦略は、以下の様な段階を踏む必要があります:

1. 4D v15以降を使用した、アップデートされたクライアントアプリケーションをビルドします。
2. 4D Server v15 を、"旧式ネットワークレイヤーを使用"ネットワーク引数を有効化して実行します。
   この設定により、全てのクライアントが接続することができます。
   注: OS X用4D Server v15 64-bit 版はこのオプションをサポートしていないことに注意して下さい。
3. 全てのクライアントが接続し、新しいバージョンをダウンロードし終わるまで一定時間待ちます。
   これには1日、1週間、あるいはそれ以上の時間がかかる可能性があります。この移行期間中、以前のバージョンのクライアントも、アップデートされたクライアントも、旧式のネットワークサーバーに接続することができます。
4. 全てのクライアントのアップデートが完了したら、旧式のネットワークレイヤーを無効化し、4D Serverを*ServerNet*へと切り替えることができます。

この戦略を図に表すと、以下の様になります:



クライアントのリクエストのログを取る

移行プロセスの間、"Diagnostic log recording"ファイルを有効化することが推奨されます。このファイルが有効化されると、4D Serverはそれぞれのクライアントのアップデートリクエストをこのファイルに記録するので、プロセスをモニターすることが出来るようになります。このログファイルはデフォルトでは有効化されていません。**SET DATABASE PARAMETER** コマンドを、 Diagnostic log recording 定数を1に設定して呼び出す必要があります。

それぞれのアップデートリクエストに対して、以下の情報が記録されます:

- クライアントのIPアドレス
- クライアントのバージョン
- "Update client" イベント

ログファイルをモニタリングすることは、サーバーを*ServerNet* ネットワークレイヤーに切り替えた後も、全てのクライアントが適切にアップデートされたかどうかを確認するために有用です。互換性のないクライアントが接続しようとした場合、サーバーは以下の情報を記録します:

- クライアントのIPアドレス
- クライアントのバージョン
- "Fail to connect" イベント

この場合、例えばクライアントを手動でアップデートするかどうか等を自分で判断することができます。

4D SQL Server

_USER_INDEXES システムテーブルの新しいKEYWORD フィールド

4D v14 R2 から導入

新しいKEYWORDというブール型のフィールドが_USER_INDEXESのシステムテーブルに追加されました。これにより、通常のインデックスとキーワードに基づいたインデックスを識別することが出来るようになりました。

_USER_INDEXES		**データベースのユーザーインデックスを記述します**
KEYWORD	BOOLEAN	インデックスがキーワードに基づいていた場合にはTrue、そうでない場合にはFalse

キーワードインデックスはクラスター、またはBTree型のものが使用できます。

ALTER DATABASE DISABLE TRIGGERS

ALTER DATABASE {ENABLE | DISABLE} {INDEXES | CONSTRAINTS | *TRIGGERS*}

説明

4D v14 R3 から導入

ALTER DATABASE コマンドは、カレントセッションにおいて、カレントデータベースの全てのテーブルのトリガーを無効化・有効化できるようになりました(つまりデータベースが再起動されるまでは全ユーザー・全プロセスに対して無効化・有効化ができます)。

トリガーをテーブルレベルで管理したい場合は、**ALTER TABLE** コマンドを使用して下さい(**ALTER TABLE DISABLE TRIGGERS** を参照して下さい)。

例題

インポートのためにSQLオプションを全て一時的に無効化する例です:

```
Begin SQL
   ALTER DATABASE DISABLE INDEXES;
   ALTER DATABASE DISABLE CONSTRAINTS;
   ALTER DATABASE DISABLE TRIGGERS;
End SQL
SQL EXECUTE SCRIPT("C:\\Exported_data\\Export.sql";SQL On error continue)
Begin SQL
   ALTER DATABASE ENABLE INDEXES;
   ALTER DATABASE ENABLE CONSTRAINTS;
   ALTER DATABASE ENABLE TRIGGERS;
End SQL
```

ALTER TABLE DISABLE TRIGGERS

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] |
*[{**ENABLE** | **DISABLE**} **TRIGGERS**]* |
SET SCHEMA sql_name}

説明

4D v14 R3 から導入

ALTER TABLE コマンドはカレントセッションにおいて*sql_name* で指定したテーブルのトリガーを無効化・有効化できるようになりました(つまり、データベースが再起動されるまでは全ユーザー・全プロセスに対して無効化・有効化ができます)。

データベースレベルでトリガーを包括的に管理したい場合は、**ALTER DATABASE** コマンドを使用してください(**ALTER DATABASE DISABLE TRIGGERS** を参照して下さい)。

ポートIDをプログラムで管理

4D v14 R3 から導入

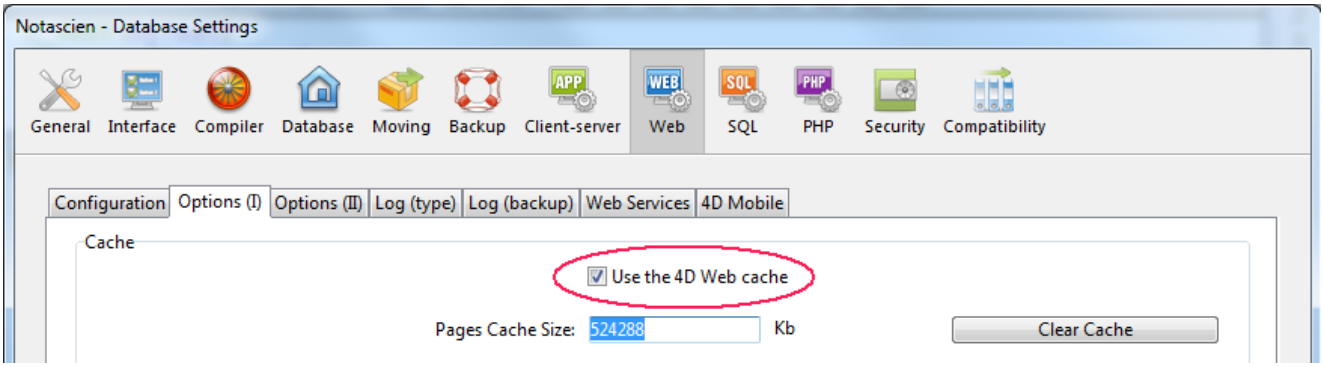
新しいデータベース引数を使用することにより、4D SQL Server のポートIDをプログラミングによって管理する事ができるようになりました。詳細な情報に関しては、 **Get database parameter と SET DATABASE PARAMETER**セクションを参照して下さい。

最適化

4D Webキャッシュはデフォルトで有効に

4D v14 R5 から導入

新規作成されたデータベースにおいて、スタティックリソースの4D Webキャッシュがデフォルトで有効化されます。データベース設定ダイアログボックス内において、**4D Webキャッシュを使用する**オプションは新規作成されたデータベースにおいては自動的にチェックされています:



以前のリリースから変換されたデータベースにおいては、このオプションの値は変更されずにそのままになっています。

SQL Select の最適化

SQL Select distinct 宣言

4D v14 R4 から導入

SQL宣言を通して実行されたSELECT DICTINCTオペレーションは、単一のテーブルと単一のカラムに対して適用されたときに最適化されるようになりました。

以下のコードなどは、大体の場合、4D v15以降の方が速く実行されるようになりました:

```
Begin SQL
   SELECT DISTINCT Names FROM Employees INTO :$Emp;
End SQL
```

SQL Select Group by and Order by statements

4D v14 R5 から導入

SQL SELECT宣言のGROUP BY と ORDER BY節が異なる設定において最適化されました:

- SELECT FROM GROUP BY または SELECT FROM ORDER BY が単一のテーブルに適用されたとき
- SELECT FROM GROUP BY または SELECT FROM ORDER BY がinner joinsを使用して複数のテーブルに適用されたとき

この最適化が関係するのは、単純な列の参照に関してのみです(式には適用されません)。

4D v15以降では、一般的に、以下の場合において実行速度が速くなります:

- **Order by の例**

```
Begin SQL
   DROP TABLE IF EXISTS T1;
   DROP TABLE IF EXISTS T2;
   CREATE TABLE T1 (C1 INT);
   CREATE TABLE T2 (C2 INT);
   INSERT INTO T1(C1) VALUES (1);
   INSERT INTO T1(C1) VALUES (2);
   INSERT INTO T1(C1) VALUES (3);
   INSERT INTO T2(C2) VALUES (2);
   INSERT INTO T2(C2) VALUES (3);
End SQL

ARRAY LONGINT($result;0)
ARRAY LONGINT($result1;0)
ARRAY LONGINT($result2;0)

 // T1 と T2 を組み合わせたシンプルなORDER BYの例
 // $result には [2, 3]が格納されます
Begin SQL
     SELECT C1
       FROM T1, T2
       WHERE C1=C2
       ORDER BY C1
       INTO :$result
End SQL

 // ORDER BY は複数の列が使用されているときにも使用できます。
 // $result1 と $result2 にはどちらも[2, 3]が格納されます。
Begin SQL
     SELECT C1, C2
       FROM T1, T2
       WHERE C1=C2
```

```
        ORDER BY C1, C2
        INTO :$result1, :$result2
End SQL

 // 最適化されるのは単純な列の参照のみです。
 // 以下の様に、(C1 + 1) のような式がセレクションに使用されている場合、
 // 実行速度は速くなりません。$result には[3, 4]が格納されます(ただし値は正確です)。
Begin SQL
     SELECT C1 + 1
        FROM T1, T2
        WHERE C1=C2
        ORDER BY C1
        INTO :$result
End SQL

 // インデックスを使用してセレクションの内部を参照することもできます。
 // 既に述べられている通り、実行速度が速くなるのはセレクションに入っているのが単純な列の参照のみのときです。
 // $result には[ 2, 3 ]が格納されます。
Begin SQL
     SELECT C1
        FROM T1, T2
        WHERE C1=C2
        ORDER BY 1
        INTO :$result
End SQL
```

- **Group by の例**

```
Begin SQL
   DROP TABLE IF EXISTS T1;
   DROP TABLE IF EXISTS T2;
   CREATE TABLE T1 (C1 INT, C3 INT);
   CREATE TABLE T2 (C2 INT);
   INSERT INTO T1(C1, C3) VALUES (3, 1);
   INSERT INTO T1(C1, C3) VALUES (1, 1);
   INSERT INTO T1(C1, C3) VALUES (2, 1);
   INSERT INTO T1(C1, C3) VALUES (3, 1);
   INSERT INTO T1(C1, C3) VALUES (2, 1);
   INSERT INTO T1(C1, C3) VALUES (3, 1);
   INSERT INTO T2(C2) VALUES (2);
   INSERT INTO T2(C2) VALUES (3);
End SQL

ARRAY LONGINT($result;0)
ARRAY LONGINT($result1;0)
ARRAY LONGINT($result2;0)

 //T1 と T2 を組み合わせたシンプルなGROUP BY の例
 //$result には[2, 3]が格納されます。必ずしもグループが昇順で返されるとは限らない、
 //という点に注意して下さい。つまり、$result の中身は[3, 2]の可能性もあるという事です。
Begin SQL
   SELECT C1
        FROM T1, T2
        WHERE C1=C2
        GROUP BY C1
        INTO :$result
End SQL

 // 以下は合計をリクエストするシンプルな例です。
 // C3はそれぞれのグループに対して常に1なので(C1の固有値)、
 // SUM(C3) が繰り返しの回数という事になります。
 // $result1 には[2, 3]が格納されます。
 // $result2 にも [2, 3]が格納されます。
Begin SQL
   SELECT C1, SUM(C3)
        FROM T1, T2
        WHERE C1=C2
```

```
        GROUP BY C1
        INTO :$result1, :$result2
End SQL
```

ネットワークセキュリティ

SSL v2 と v3 の無効化

4D v14 R4 から導入

安全なネットワーク通信のデフォルトの設定が、セキュリティレベルを上げるために進化しました。その潜在的な脆弱性から、**SSL v2 と SSL v3 プロトコルは、恒久的に利用不可になりました**(無効化がハードコーディングされています)。今後はデフォルトではTLS v1のみが受け付けられます。

この変更は4Dの全てのセキュアな通信に適用されます。以下のものが含まれます:

- HTTP サーバー接続
- SQL サーバー接続
- クライアント/サーバー接続
- HTTP クライアント接続

結果として、アプリケーションの必要要件の中で以下の接続に関係するものをアップグレードする必要がある可能性があります:

- TLSをサポートしていないブラウザまたはHTTPクライアント。具体的にはIE6またはそれ以前のIEは、セキュアプロトコル(ポート443)を使用して4D Web Server/Serviceへと接続することはできなくなりました。
- **HTTP Get** または **HTTP Request** コマンドは、TLSをサポートしていないサーバーへセキュアモードで接続することはできなくなりました。

証明書の強化

4D v14 R5 から導入

ネットワークセキュリティ機能は最適化され、4Dアプリケーションの保護を強化できるようにしました:

- 脆弱な暗号リストの組み合わせは削除されました。
- デフォルトの4D証明書キーの長さは、2048bitへと増加されました。
- 安全な通信のために、自分で設定した暗号化キーを使用できるようになりました。

これらの変更が関係するのは、以下のセキュアな接続です:

- クライアント/サーバー間
- SQL サーバー
- HTTP クライアント

4D Write Pro

4D v14 R5 から導入

4D Write Pro とは 4D Write plug-in の後継版です。

4D Write Pro は、4Dにとって大きな発展です。ですから、R-リリースの利点を利用してそれを徐々に配付しています。カレントバージョンの4D write Proでは、ドキュメントの互換性とHTMLフォーマットのEメールに焦点を当てています。今後の各ステップにおいて、使用できる機能とプログラミングのしやすさが改善していく予定です。

以下の点にご注意ください:

- 4D Write Pro は4D Writeと同じライセンスを使用しています。
- 4D Write Pro はプラグインではなくなり、4D自身に完全に統合されました。その結果、配布と管理がしやすくなりました。

4D Write Pro のドキュメントについて

新しいランゲージコマンドを含めた4D Write Pro の機能は、新しい**4D Write Proリファレンス** マニュアル内に全て記載されています。

このテクニカルプレビューについて

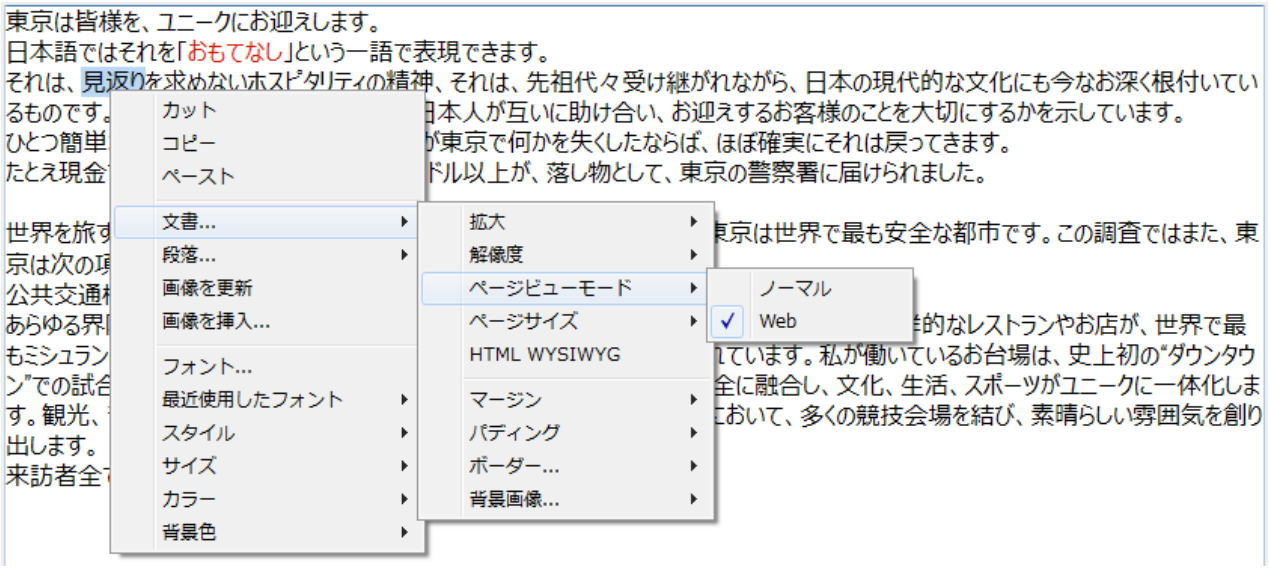
以下の一覧は、主な4D Write Proの機能の現在の状況をまとめたものです:

機能	現状	補足
フォーム内に4D Write Pro エリアを描画する	使用可能	
新規4D Write Pro ドキュメントを作成する	使用可能	**WP New**
4D Write ドキュメントを4D Write Pro オブジェクト内に読み込む	使用可能	**WP Import document** または **WP New**
4D Write Pro オブジェクトをドキュメントまたは変数へと書き出す	使用可能	**WP EXPORT DOCUMENT** または **WP EXPORT VARIABLE**
ドキュメントをコンテキストポップアップメニューを使用して編集	使用可能	
式(データベースフィールド、4Dメソッドの結果)を挿入する	使用可能	**ST INSERT EXPRESSION** コマンドを使用して下さい
ドキュメントの中身をプログラムを使用して編集する	一部利用可能	"ST" と "OBJECT" 4D コマンドのみ使用可能です
4D Write Pro ドキュメントを印刷する	利用不可	
ページ付けの管理	利用不可	
プログラムを使用してドキュメントのプロパティにアクセス	利用不可	
4D Write Proドキュメントをデータベースフィールド内に保存する	利用不可	

4D Write Pro インターフェース
4D Write Pro コマンドの現在の状況
4D Write ドキュメントの読み込み

4D Write Pro インターフェース

4D Write Pro エリアにおいて**コンテキストメニュー**プロパティがチェックされている場合(**4D Write Pro エリアの作成**を参照して下さい)、アプリケーションモードにおいてユーザーは包括的なコンテキストメニューを使用することができます:



このメニューでは現在利用可能な全ての4D Write Proの機能を提供します(次のリリースでは、全ての機能とプロパティがプログラミングを通じてもご利用になれます)。

現時点ではコンテキストメニューの構成も中身も最終決定はしていない段階のため、このドキュメントではここの機能について詳細に記述することはしていません。
様々なサブメニューの中を見ることによって、4D Write Proでどんなことが出来るのかということを、直接感じ取っていただくことをお勧めします。

注: カレントのバージョンにおいては、4D Write Proドキュメントはデフォルトで**Web**ページビューモードで表示されています。このモードにおいては、テキストは自動的に折り返され、水平スクロールバーは(設定されていても)無効化されます。水平スクロールバーを使い、テキストを("ページサイズ"プロパティで定義されている)固定長に設定したい場合には、**ノーマル**ページビューモードへと切り替える必要があります。

4D Write Pro コマンドの現在の状況

いくつかの新しいコマンドが4D Write Proエリアを管理するために"4D Write Pro"テーマに追加されました。これらのコマンドは4D Write Proリファレンスマニュアルの**4D Write Proランゲージ** の章内に詳細が書かれています。

以下の一覧は、カレントバージョンにおける4D Write Proコマンドの実装状況をまとめたものです:

コマンド名	現時点での状況(R5 テクニカルプレビュー時点)
WP EXPORT VARIABLE	現時点では書き出しのフォーマットは二つのみご利用いただけます。
WP EXPORT DOCUMENT	*format* 引数を渡さない場合、".htm" または ".html"ファイル拡張子を使う必要がありますが、ご利用いただける書き出しフォーマットは現在一つだけです。
WP Import document	4D Write Pro オブジェクト内で現在サポートされている4D Write 機能の詳細な一覧については、**4D Write ドキュメントの読み込み** の章を参照して下さい。
WP New	4D Write Pro オブジェクト内で現在サポートされている4D Write 機能の詳細な一覧については、**4D Write ドキュメントの読み込み** の章を参照して下さい。

注: 4D Write Pro エリアでは"オブジェクト(フォーム)" と "スタイル付テキスト" テーマコマンドのみがご利用いただけるという点にご注意ください(**オブジェクト(フォーム)テーマのコマンドの使用** と **スタイル付テキストテーマのコマンドの使用** の章を参照して下さい)。

4D Write ドキュメントの読み込み

新しい4D Write Proオブジェクトの主要な機能の一つとして、既存の4D Writeドキュメントの読み込みと変換機能が挙げられます。これによって4D Write プラグインに依存しているアプリケーションを移行させることができます。

互換性に関する注意: サポートされるのは、4DWite ドキュメントのうち最後の世代("4D Write v7")のドキュメントのみに限られます。

4D Write ドキュメントを読み込むためには

4D Write Proオブジェクトは4D Writeドキュメントを読み込むための方法を二つご用意しています:

- ディスク上にに保存されいている4D Writeに対しては、**WP Import document** コマンドを使用して下さい。
- BLOBフィールドに4D Writeエリアに対しては、**WP New** コマンドを使用して下さい。

より詳細な情報に関しては、それぞれのコマンドの詳細を参照して下さい。

4D Write のプロパティで復元されるもの

4D Write プラグインから4D Write Proへの移行を簡単にするために、4D Write の機能のうち、出来る限り多くのものが4D Write Proオブジェクトでサポートされようとしています。

以下の段落では、4D Write プラグインのプロパティのうち、 **WP Import document** または **WP New** コマンドを使用して4D Write Proエリアへと読み込んだときに復元されるプロパティについてまとめています。

しかしながら一部の機能において、バグとはみなされない小さな差異が生じることがあります。これらは例えば、4D Write Proで使用されるデフォルトの行頭の記号や、下線のタイプの小さな変化などによるものです。

ドキュメント情報

4D Write プラグイン	4D Write Pro
作成日時	利用可能
修正日時	利用可能
ロック	利用不可(読み込み専用オブジェクトプロパティを使用して下さい)
タイトル	利用可能
件名	利用可能(標準テキストのみ)
作者	利用可能
会社名	利用可能
注記	利用可能

ドキュメントの表示の引数

4D Writeプラグイン	4D Write Pro
ページモード	読み込まれません(コンテキストメニューの、ドキュメント/ページモードを使用して下さい)
ルーラー	利用不可
枠	利用不可
ヘッダー	利用不可
フッター	利用不可
最初のページのヘッダー	利用不可
最初のページのフッター	利用不可
画像	利用不可
縦スクロールバー	読み込まれません(オブジェクトプロパティの縦スクロールバーを使用して下さい)
横スクロールバー	読み込まれません(オブジェクトプロパティの横スクロールバーを使用して下さい)
非表示文字	利用不可
参照	利用不可(**ST SET OPTIONS**を参照して下さい)
カラム分割	利用不可
縦スプリッター	利用不可
横スプリッター	利用不可
Wysiwyg	利用不可
ズーム	読み込まれません(コンテキストメニューの、ドキュメント/ズームを使用して下さい)

ドキュメントの引数

4D Write プラグイン	4D Write Pro
単位	利用不可
言語	利用不可
カラム数	利用不可
行間隔	利用不可
ウィンドウ&オーファン	利用不可
デフォルトのタブ	利用可能
先頭のタブ	利用不可
URLカラー	利用不可
アクセス済みのURLカラー	利用不可

ドキュメントのページ付の引数

4D Writeプラグイン	4D Write Pro
ページ幅	利用不可
ページの高さ	利用不可
最初のページ番号	利用不可
最初のページのヘッダーとフッターを別にする	利用不可
左右のページでヘッダーとフッターが異なる	利用不可
ページバインディング	利用不可
反対側のページ	利用不可
ページマージン	利用可能(一時的な実装)
ヘッダーの上マージン	利用不可
ヘッダーの下マージン	利用不可
フッターの上マージン	利用不可
フッターの下マージン	利用不可
最初のページの上マージン	利用不可
最初のページの下マージン	利用不可
最初のページのヘッダーの上マージン	利用不可
最初のページのヘッダーの下マージン	利用不可
最初のページのフッターの上マージン	利用不可
最初のページのフッターの下マージン	利用不可
最初のページを右側にする	利用不可

ドキュメントの印刷引数

4D Write プラグイン	4D Write Pro
用紙の種類	利用不可
横向きに印刷	利用不可
幅	利用不可
高さ	利用不可
ユーザー設定マージン	利用不可
倍率	利用不可
X 解像度	利用不可
Y 解像度	利用不可

画像

注: 4D Write Pro はページ内での画像の絶対位置に関してはまだサポートしていません。このテクニカルプレビューにおいてはインライン画像のみがサポートされ、読み込まれます。

4D Write プラグイン	4D Write Pro
X (左)	(& position :absolute) (ページ内の画像のみ)
Y (上)	(& position :absolute) (ページ内の画像のみ)
幅	ご利用可能
高さ	ご利用可能
ページ番号	利用不可
背景画像	利用不可
最初のページには含めない	利用不可
ビューポートモード(縮小して画像を入れる)	ご利用可能
式である	利用不可
サイズを保持	利用不可

文字プロパティ

4D Write プラグイン	4D Write Pro (span プロパティ)
イタリック	利用可能
ボールド	利用可能
取り消し線	利用可能
下線	利用可能
シャドウ	利用可能
指数(上付きまたは下付き)	利用可能
大文字(英大文字または小型英大文字)	利用可能
フォントファミリー	利用可能
フォントサイズ	利用可能
テキストカラー	利用可能
テキスト背景カラー	利用可能
下線カラー	利用可能利用可能
取り消し線カラー	利用可能
シャドウカラー	利用可能
ユーザープロパティ	利用不可
スペルチェック(シンタックス&グラマー onまたはoff)	利用不可
表示	利用不可
スタイルシート	読み込まれません(スタイルは読み込まれますが、スタイルシートは利用不可です)

段落プロパティ

4D Write プラグイン	4D Write Pro
均等位置	利用可能
Interline	利用可能
箇条書き	利用可能
左マージン	利用可能
右マージン	利用可能
テキストインデント	利用可能
境界線スタイル	利用可能
境界線カラー	利用可能
境界線の背景カラー	利用可能
左境界線	利用可能
右境界線	利用可能
上境界線&上の内側の境界線	利用可能
下の境界線&下の内側の境界線	利用可能
境界線間隔	利用可能
スタイルシート	利用可能
タブ	利用可能

ハイパーリンク

4D Write プラグイン	4D Write Pro
URL リンク	利用可能
4D メソッドリンク	利用不可
ドキュメントを開くリンク	利用不可

4D 式

4D Write プラグイン	4D Write Pro
4D 式	利用可能
日付と時間	利用可能
HTML 式	利用不可
RTF 式	利用不可

テキストデータ

4D Write プラグイン	4D Write Pro
メインテキストデータ	利用可能
ヘッダーテキストデータ	利用不可
フッターテキストデータ	利用不可

4D Internet Commands

OS X用4D Serverの64ビット版での4D Internet Commandsについて

4D v14 R4 から導入

4DはOS X用の64-bit版の4D Internet Commandsの提供を開始します。このバージョンによって4D Internet Commands プラグインをOS X用の4D Server 64-bit 版にて使用できるようになります。より詳細な情報に関しては、**OS X用4D Serverの64ビット版について(プレビュー)**のセクションを参照して下さい。

OS X用の64 bit 版 4D Internet Commandsプラグインのインストールと使用法は、既存のバージョンとほぼ同じですが、以下の様な制限があります。

システム要件

- 4D IC 64-bit 版はOS X用4D Server 64-bit版のv14 R3 以降のバージョンと使用する必要があります。
- 4D Server と同様、4D IC の 64-bit版はOS Xのバージョン10.9 (Mavericks) 以上が必要となります。

より長いファイル名もサポート

OS X 用 4D IC の64-bit版では、以前のOS Xのバージョンのように62文字まで限定されるという事はなくなりました。ファイル名は現行のOSの上限に合わせて1024文字まで入力可能です。

この制限の撤廃は、FTP、IMAP、MSG、POP、SMTP、そして IT コマンドなど、ファイル名を管理する全てのコマンドに関係します。

サポートされていないインターフェイス機能

OS Xのユーザーインターフェースの詳細により、以下の機能はOS X用4D Server 64-bit版ではサポートされていません。

進捗バー

OS X用4D IC 64-bit版は、時間のかかるオペレーションを実行中でも進捗バーを表示しなくなりました。**FTP_Progress** コマンドはOS X用64-bit版では廃止予定となています。

この環境において呼び出された場合、コマンドはエラー-2201(未実装の機能です)を返します。

ファイル選択ダイアログ

ファイル選択ダイアログボックスはOS X 用 4D IC 64-bit 版では表示させることができません。このバージョンでは、*localPath* または *hostPath* 引数が必須であり、有効なファイルパスを含んでいる必要があります。そうでない場合にはエラー -2201(未実装の機能です)が返されます。このシンタックスは、FTP、IMAP、MSG、POP、SMTP、そして IT コマンドなど、ファイル名を管理する全てのコマンドに関係します。

ユーザーにコンピューターからファイルを選んでもらいたい場合には、 **Select document** などの4Dにビルトインされているコマンドの使用が推奨されます。

SMTP_QuickSend

SMTP_QuickSend (hostName ; msgFrom ; msgTo ; subject ; message {; sessionParam}{; port}{; userName ; password}) -> Function result

引数	型		詳細
hostName	文字列	->	ホスト名またはIPアドレス
msgFrom	テキスト	->	メールアドレスまたはアドレスリスト
msgTo	テキスト	->	メールアドレスまたはアドレスリスト
subject	テキスト	->	メッセージの件名(デフォルトではUTF-8)
message	テキスト	->	メッセージ(デフォルトではUTF-8)
sessionParam	倍長整数	->	0 または 省略時 = SSLは使用しないがSSLへの変更は許可、1 = SSLを使用、2 = SSLを決して使用しない(SSLへの変更も許可しない)、4 = HTMLテキストをSSLを使用せずに送信する、5 = HTML テキストをSSLを用いて送信する、*8 = Mime HTML をSSL/TLS*を使用せずに送信する、*9 = Mime HTML をSSL/TLS*を使用して送信する
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

説明

4D v14 R5 から導入

SMTP_QuickSend コマンドはMime HTML フォーマットのメッセージを送信することができるようになりました(SSL/TLS プロトコルの有無は選択できます)。Mime HTML (ファイル拡張子は.mht または .mhtml)とはHTMLコードやほかの外部リ

ソース(画像など)を単一のドキュメントへとマージすることのできるWebページアーカイブフォーマット のことです。これは複数のブラウザと、それに加えてMSワードなどでもサポートされているフォーマットです。このフォーマットは4D Write Pro エリアでサポートされていることから、4D Write Proエリアを保存したり、そのリソースも含めてメールで送信したりすることが簡単にできるようになりました。

sessionParam 引数に8を渡すと、Mime HTMLフォーマットのメッセージを標準モードで送信します。
sessionParam 引数に9を渡すと、Mime HTMLフォーマットのメッセージをSSL/TLSモードで送信します。

これらの値は通常の組み合わせと対応しますが、*sessionParam* 引数は実際はビットフィールドであり、bitwise 演算子を使用するとどんなカスタムの組み合わせも使用できます:

ビット数	bitが1の際に使用されるフォーマット
0	SSLまたはデフォルトの仕様を使用、クリアに接続、可能であればSSLへとアップグレードする
1	決してアップグレードをしない、アップグレードが可能な場合でも非-暗号化モードにとどまる。SSL(bit-0)が選択されている際にはビットは無視されます。
2	メッセージ本文はHTMLで、ヘッダーもそれに応じて設定されます。
3	MHTML メッセージを使用し、bit-2 (HTML) は無視されます。ユーザーは"To"、"From"、"Date"、そして"Subject"を除き、他の全ての設定を自分で設定する必要があります。

注: このコマンドは"非-Unicode"モードで実行されている変換されたデータベースをサポートしません。

例題

ディスク上に保存した .mht ドキュメントを、Eメールで送信する場合を考えます。この場合、以下の様な記述で送信できます:

```
$Message:=Document to text("c:\\documents\\invitation.mht")
$Host:="smtp.gmail.com"
$ToAddress:="john@4d.com"
$FromAddress:="harry@gmail.com"
$Subject:="Let's party"
$Param:=9 //SSLを賜与空いて MHTML で送信
$Port:=465 //gmail のSSLポート
$User:="harry@gmail.com"
$Password:="xyz&@!&@"
$Error:=SMTP_QuickSend($Host;$FromAddress;$ToAddress;$Subject;$Message;$Param;$Port;$User;$Password)
```

SMTP_Attachment

SMTP_Attachment (smtp_ID ; fileName ; encodeType ; deleteOption {; attachmentID *{; contentType}*}) -> 戻り値

引数	型		詳細
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(DataFork のみ送信) ±1 = BinHex ±2 = Base64; (DataFork のみ送信) ±3 = AppleSingle ±4 = AppleDouble ±5 = AppleSingle と Base64 ±6 = AppleDouble と Base64 ±7 = UUEncode
deleteOption	整数	->	0 = 既存のリストに追加、 1 = ファイル名で全ての添付を置き換え、 2 = この添付のみ削除
attachmentID	テキスト	->	添付のID(HTML メッセージのみ)
contentType	テキスト	->	設定するコンテンツタイプの値
戻り値	整数	<-	エラーコード

説明

4D v14 R4 から導入

テーマ: ICメール送信

SMTP_Attachment コマンドは、新しい*contentType* 引数を受け取るようになり、これによって添付ファイルのコンテンツタイプを設定することが出来るようになりました。

デフォルトでは、*contentType* parameter 引数が省略されている、または空の文字列を含む場合、4DICは自動的に添付ファイルのコンテンツタイプを、その拡張子に応じて設定します。その際、以下のルールが適用されます:

拡張子	コンテンツタイプ
jpg, jpeg	image/jpeg
png	image/png
gif	image/gif
pdf	application/pdf
doc	application/msword
xls	application/vnd.ms-excel
ppt	application/vnd.ms-powerpoint
zip	application/zip
gz	application/gzip
json	application/json
js	application/javascript
ps	application/postscript
xml	アプリapplication/xml
htm, html	text/html
mp3	audio/mpeg
other	application/octet-stream

contentType には、ファイルのコンテンツタイプ(MIMEタイプ)を定義する文字列を渡す事ができます(例えば、"video/mpeg"などです)。このコンテンツタイプの値は添付ファイルの拡張子に関係なくその添付ファイルに適用されます。

注: この引数を使用しない場合、*attachmentID* に空の文字列("")を渡して下さい。

例題

設定ファイルをXMLファイルとして宣言したい場合を考えます:

```
$path:=Get 4D folder(Database folder)+"Settings.mySettings"
$err:=SMTP_Attachment($smtp_id;$path;2;0;"myID123";"application/xml")
```

4D View

PV GET BORDER ROW RANGES
PV GET BORDER COLUMN RANGES

PV GET BORDER ROW RANGES

PV GET BORDER ROW RANGES (area ; left ; top ; right ; bottom ; borderTypes ; borderColors)

引数	型		説明
area	倍長整数	➡	4D Viewエリア
left	倍長整数配列	⬅	範囲の左端のセルのカラム番号の配列
top	倍長整数配列	⬅	範囲の上端のセルの行番号の配列
right	倍長整数配列	⬅	範囲の右端のセルのカラム番号の配列
bottom	倍長整数配列	⬅	範囲の下端のセルの行番号の配列
borderTypes	倍長整数配列	⬅	境界線タイプの配列
borderColors	倍長整数配列	⬅	境界線カラーの配列

説明

4D v14 R4 から導入

テーマ: PV Borders

新しい**PV GET BORDER ROW RANGES** コマンドは同じ上側の枠線スタイルを共有するセルの範囲のリストを返します(枠線スタイルは、タイプとカラーの二つの要素からなります)。

それぞれの範囲は、*left*、*top*、*right* と *bottom* という、同期した配列を通して返されます。それぞれの要素は、範囲の左端、上端、右端、そして下端のセル番号を表します。

対応する枠線スタイルは*borderTypes* と *borderColors* 配列内に返されます:

- *borderTypes* の値は、 **PV Border style** テーマの定数と比較する事ができます:

定数	型	値
pv border style 1	倍長整数	1
pv border style 111	倍長整数	7
pv border style 112	倍長整数	9
pv border style 2	倍長整数	2
pv border style 211	倍長整数	8
pv border style 212	倍長整数	10
pv border style 222	倍長整数	11
pv border style 232	倍長整数	12
pv border style 3	倍長整数	3
pv border style 4	倍長整数	4
pv border style 5	倍長整数	5
pv border style 6	倍長整数	6
pv border style half	倍長整数	14
pv border style none	倍長整数	0
pv border style quarter	倍長整数	13

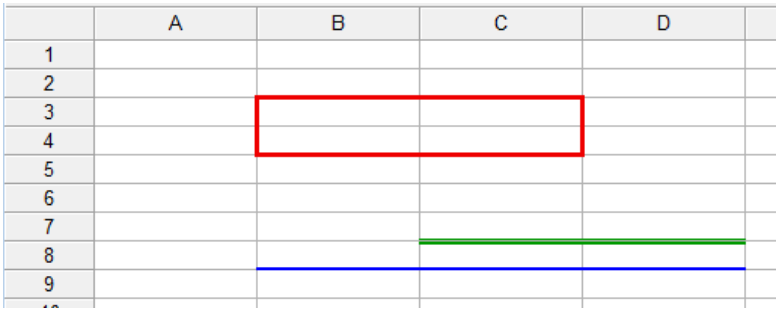
- *borderColors* の値は、BGR-型の倍長整数です。詳細な情報に関しては、 **PV Color to index** *PV Color to index* コマンドを参照して下さい。

このコマンドは、新しい **PV GET BORDER COLUMN RANGES** コマンドと組み合わせることで、エリアの枠線の完全な定義を取得することができます。これは、例えば4D ViewエリアをMSエクセルフォーマットに書き出したいときなどに有効です。

注: コマンドによって返される範囲のリストは、その範囲がどのように定義されたかによって異なってきます。例えば、範囲(4A;4E)を選択している状態で水 平な線を描画した場合、コマンドはその範囲(4A;4E)に対応した単一の値のみを返します。それに対し、行4に対して水平な線を描画し、それを列A～E に対してループさせた場合、コマンドはそれぞれの描画に対応して5つの値を返します。描画の結果は見た目上では全く同じですが、内部で保存されている情報 は異なるという事です。

例題

エリア内に、以下の様に枠線が配置されていたとします:



このとき、以下のコードを実行すると:

```
PV GET BORDER ROW
RANGES(myArea;LeftArray;TopArray;RightArray;BottomArray;BorderTypeArray;BorderColorArray)
```

四つの範囲が検出され、その結果配列内には以下の様な値が格納されて返されます:

LeftArray	TopArray	RightArray	BottomArray	BorderTypeArray	BorderColorArray
2	3	3	3	3	15597568
2	5	3	5	3	15597568
3	8	4	8	9	39168
2	9	4	9	2	255

PV GET BORDER COLUMN RANGES

PV GET BORDER COLUMN RANGES (area ; left ; top ; right ; bottom ; borderTypes ; borderColors)

引数	型		説明
area	倍長整数	➡	4D Viewエリア
left	倍長整数配列	⬅	範囲の左端のセルのカラム番号の配列
top	倍長整数配列	⬅	範囲の上端のセルの行番号の配列
right	倍長整数配列	⬅	範囲の右端のセルのカラム番号の配列
bottom	倍長整数配列	⬅	範囲の下端のセルの行番号の配列
borderTypes	倍長整数配列	⬅	境界線タイプの配列
borderColors	倍長整数配列	⬅	境界線カラーの配列

説明

4D v14 R4 から導入

テーマ: PV Borders

新しい**PV GET BORDER COLUMN RANGES** コマンドは、同じ左側の枠線スタイルを共有するセルの範囲のリストを返します(枠線スタイルは、タイプとカラーの二つの要素からなります)。

それぞれの範囲は、 *left*、*top*、*right* と *bottom* という、同期した配列を通して返されます。それぞれの要素は、範囲の左側、上側、右側、そして下側のセル番号を表します。

対応する枠線スタイルは*borderTypes* と *borderColors* 配列内に返されます:

- *borderTypes* の値は、**PV Border style** テーマの定数と比較する事ができます:

定数	型	値
pv border style 1	倍長整数	1
pv border style 111	倍長整数	7
pv border style 112	倍長整数	9
pv border style 2	倍長整数	2
pv border style 211	倍長整数	8
pv border style 212	倍長整数	10
pv border style 222	倍長整数	11
pv border style 232	倍長整数	12
pv border style 3	倍長整数	3
pv border style 4	倍長整数	4
pv border style 5	倍長整数	5
pv border style 6	倍長整数	6
pv border style half	倍長整数	14
pv border style none	倍長整数	0
pv border style quarter	倍長整数	13

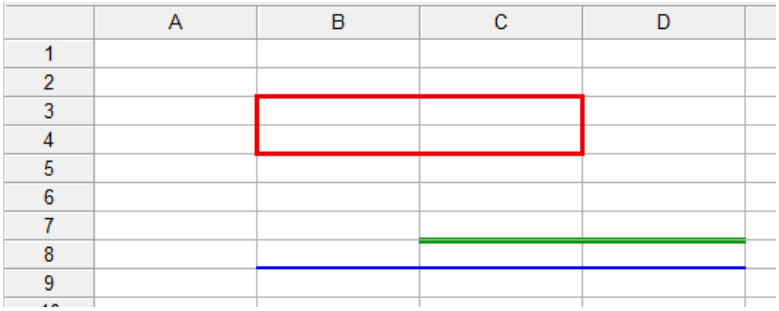
- *borderColors* の値は、BGR-型の倍長整数です。詳細な情報に関しては、 **PV Color to index** *PV Color to index* コマンドを参照して下さい。

このコマンドは、新しい **PV GET BORDER ROW RANGES** コマンドと組み合わせることで、エリアの枠線の完全な定義を取得することができます。これは、例えば4D ViewエリアをMSエクセルフォーマットに書き出したいときなどに有効です。

注: コマンドによって返される範囲のリストは、その範囲がどのように定義されたかによって異なってきます。例えば、範囲(1B;5B)を選択している状態で垂 直な線を描画した場合、コマンドはその範囲(1B;5B)に対応した単一の値のみを返します。それに対し、列Bに対して垂直な線を描画し、それを行1～5 に対してループさせた場合、コマンドはそれぞれの描画に対応して5つの値を返します。描画の結果は見た目上では全く同じですが、内部で保存されている情報 は異なるという事です。

例題

エリア内に、以下の様に枠線が配置されていたとします:



このとき、以下のコードを実行すると:

```
 PV GET BORDER COLUMN
RANGES(myArea;LeftArray;TopArray;RightArray;BottomArray;BorderTypeArray;BorderColorArray)
```

二つの範囲が検出され、その結果配列内には以下の様な値が格納されて返されます:

LeftArray	*TopArray*	*RightArray*	*BottomArray*	*BorderTypeArray*	*BorderColorArray*
2	3	2	4	3	15597568
4	3	4	4	3	15597568

4D SVG

Direct2Dが有効化されているWindowsでのフィルター

Direct2Dが有効化されているWindowsでのフィルター

4D v14 R5 から導入

SVG_Filter_Blend、**SVG_Filter_Blur** そして **SVG_Filter_Offset** コマンドはDirect2Dが有効化されているWindowsのグラフィックソフトウェアのコンテキストにおいてもサポートされるようになりました(4D v14のデフォルトコンテキスト。詳細は**SET DATABASE PARAMETER**コマンドの Direct2D Software オプションを参照して下さい)。

以前の4Dのリリースでは、これらのコマンドはWindows上ではDirect2Dが無効化されている必要がありました。

4D Pack

OS X用64-bit版の4D Pack
廃止予定のコマンドの改名

OS X用64-bit版の4D Pack

4D v14 R5 から導入

4DではOS X用64-bit版の4D Packを提供しています。このバージョンによって、OS X用の4D Serverの64-bit版で、4D Packのプラグインを使用できるようになります(**OS X用4D Serverの64ビット版について(プレビュー)** を参照して下さい)。

OS X用64-bit版の4D Packのインストールと使用は既存のバージョンに近いですが、以下のようないくつかの制約がつきます。

サポートされないコマンド

既存のバージョンで利用できた4D Pack コマンドは、以下のものを除き、全てOS X用の64-bit版で引き続きサポートされます:

- **_o_AP Save BMP 8 bits**
  このコマンドは廃止予定のテクノロジーに基づいており、既に以前の4D Packのリリースで廃止予定となっていました。このコマンドはOS X用64-bit版の4D Packではサポートされません。

廃止予定のコマンドの改名

4D v14 R5 から導入

いくつかの4D Pack コマンドは長い間廃止予定となってきました。これらのコマンドは今後の新しいプロジェクト内においては使用しないことが強く推奨されます。廃止予定である という事を明確にするために、これらのコマンドには"_o_"の接頭辞がつけられました:

テーマ	以前の名前	新しい名前	状態
4D_Pack: ANSI streams	AP FCLOSE	**_o_AP FCLOSE**	廃止予定
	AP fopen	**_o_AP fopen**	廃止予定
	AP FPRINT	**_o_AP FPRINT**	廃止予定
	AP fread	**_o_AP fread**	廃止予定
4D Pack: Picture files	AP Save BMP 8 bits	**_o_AP Save BMP 8 bits**	廃止予定、OS X用64-bit版ではサポートされません
4D Pack: Utilities	AP Add table and fields	**_o_AP Add table and fields**	廃止予定、ドキュメントから削除
	AP Create relation	**_o_AP Create relation**	廃止予定、ドキュメントから削除
	AP Get file MD5 digest	**_o_AP Get file MD5 digest**	廃止予定、**Generate digest** に置き換え
	AP ShellExecute	**_o_AP ShellExecute**	廃止予定、**LAUNCH EXTERNAL PROCESS** に置き換え

カンパニーディレクトリ(LDAP)

4D v15では、LDAPプロトコルを使用して、お使いの4Dアプリケーションを既存のカンパニーディレクトリに接続するための新機能があります。新しい"LDAP"テーマのコマンドには、MS Active Directoryなどのカンパニーディレクトリにログインし、クエリするためのツールが用意されています。

これらの新コマンドにより、以下のようなことが可能になります:

- Windows セッションログインとパスワード(MS Active directoryの場合)を使用することにより4Dアプリケーションへのアクセス権を与え、その結果としてエンドユーザーはたった一つのパスワードだけ覚えていれば良いということになります。
- 企業内ディレクトリをクエリすることにより、ユーザー情報(姓名、メールアドレス、電話番号、オフィスの建物、所属グループ等)を取得できるようになります。

注: LDAPまたはLightweight Directory Access Protocolは、分散型情報サービスへとアクセスし、維持するためのオープンなスタンダードです。より詳細な情報に関しては、Wikipediaページ または OpenLDAP Software のメインページを参照して下さい。

4Dでは、LDAP接続は**LDAP LOGIN**を使用して開かれます。その後カレントの4Dプロセスと結びつけられ、閉じる際には**LDAP LOGOUT**を使用するか、プロセスが実行を終了する必要があります。

用語集

以下の一覧は、LDAP環境で使用される主な略語の一覧です:

略語	定義
LDAP	ライトウェイトディレクトリアクセスプロトコル(Lightweight Directory Access Protocol)
AD	アクティブディレクトリ(Active Directory)。AD とはMicrosoftによって実装されたディレクトリサービスデータベースで、LDAPはそれとの対話のためのプロトコルの一つです。
CN	一般名(Common Name)、例えば "John Doe" など。
DN	識別された名前(Distinguished Name)、例えば "cn=John Doe,ou=users,dc=example,dc=com"など。
SAM-Account-Name	セキュリティアカウントマネージャー(Security Account Manager)。ADへのログオン名、例えば "jdoe"
OU	組織単位(Organizational unit)。サーバーツリーのグループ。
DC	ドメインコンポーネント(Domain components)。サーバーツリーのルートと最初の枝。
uid	ユーザーID(User identifier)

- LDAP LOGIN
- LDAP LOGOUT
- LDAP Search
- LDAP SEARCH ALL

LDAP LOGIN

LDAP LOGIN (url ; login ; password {; digest})

引数	型		説明
url	文字	➡	接続するLDAPサーバーのURL
login	文字	➡	ログインエントリー
password	文字	➡	ログインエントリーのパスワード
digest	倍長整数	➡	0 = パスワードをダイジェストMD5で送信(デフォルト)、1 = 暗号化なしでパスワードを送信

説明

LDAP LOGIN コマンドは*url* 引数で指定したLDAPサーバーに対し、*login* 引数と *password* 引数に渡された識別子をもって読み込み専用の接続を開きます。サーバーに受け入れられた場合、**LDAP LOGOUT** コマンドが実行されるまで(あるいはプロセスが閉じられるまで)、カレントプロセスにおいてその後に実行される全てのLDAP検索にはこの接続が使用されます。

url 引数には、スキームとポート(デフォルトでは389)を含め、接続するLDAPサーバーへの完全なURLを渡します。この引数はrfc2255に準拠している必要があります。
url 引数に対し、"ldaps"で始まる、特定のポート番号(例: "ldaps://svr.ldap.acme.com:1389" 等)を使用した場合、TLS経由の安全な接続を開くことができます。LDAPサーバーは、(少なくともMicrosoft Active Directoryに対する)SSL証明書を持っている必要があります。パスワードが通常のテキストとして送信される場合にはTLS接続の使用が強く推奨されます(以下を参照して下さい)。

注: *url* 引数に対して、空の文字列を渡した場合、コマンドはドメイン上で使用可能なデフォルトのLDAPサーバーへと接続しようとします(この機能は試験目的用のみのもので、パフォーマンス上の理由から製品で使用されるべきではありません)。

login 引数には、LDAPサーバー上のユーザーアカウントを渡し、*password* 引数にはパスワードを渡します。デフォルトで、*login* 引数にはLDAPサーバーの設定に応じて、以下の文字列のどれかを渡すことができます:

- 識別名(DN)。例えば、"CN=John Smith,OU=users,DC=example,DC=com"
- ユーザー名(CN)。例えば、"CN=John Smith"
- メールアドレス。例えば、"johnsmith@4d.fr"
- SAM-アカウント名。例えば、"jsmith"

login 引数で受け入れ可能な値は、*digest* 引数で定義された送信モードと関係しているという点に注意して下さい。例えば、MS Active Directoryのデフォルトの設定においては、以下のようになっています:

- 送信モードがLDAP password MD5 であるとき、ログインに受け入れ可能な値はSAM-アカウント名だけです。
- 送信モードがLDAP password plain text (通常のテキスト)であるとき、*login* 引数には、DN、CN、メールアドレスのどれかを渡すことができます。SAM-アカウント名も使用可能ですが、その後にドメイン名が着いていなければなりません(例： "dc-acme.com/jsmith")

digest 引数を使用すると、パスワードがネットワークでどのように送信されるかを変更することができます。"LDAP"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(値)	型	詳細
LDAP password MD5 (0)	倍長整数	(デフォルト)パスワードをMD5で暗号化して送信します。
LDAP password plain text (1)	倍長整数	パスワードは暗号化されずに送信されます(TLS 接続が推奨されます)

デフォルトでは、*password* 引数はMD5ダイジェストで送信されます。必要であればLDAP password plain text を渡して下さい(例えば、LDAPサーバーとは異なるログイン型の値を使用したい場合等)。製品環境に置いては、*url* 引数に対しTLS接続を使用することが推奨されます。

注: 空のパスワードでの認証をすると、匿名のバインディングモードになります(LDAPサーバーから認証された場合)。しかしながら、このモードにおいては、この種のバインドでは許可されていないオペレーションを実行しようとした場合にエラーが発声する可能性があります。

ログイン引数が有効であった場合、LDAPサーバーへの接続が4Dプロセスにおいて開かれます。その結果LDAPコマンドを使用して情報の検索・取得ができるようになります。

LDAPサーバーへの接続が必要なくなった倍には、必ず忘れずに**LDAP LOGOUT** コマンドを呼び出して下さい。

例題 1

LDAPサーバーにログインして、検索をしたい場合を考えます:

```
ARRAY TEXT($_tabAttributes;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes;"phoneNumber")
LDAP LOGIN("ldap://srv.dc.acme.com:389";"John Smith";"qrnSurBret2elburg")
$vfound:=LDAP Search("OU=UO_Users,DC=ACME,DC=com";cn=John Doe";LDAP all levels;$_tabAttributes)
LDAP LOGOUT //ログアウトをお忘れなく
```

例題 2

以下の例は、アプリケーションへの接続を試みます:

```
ON ERR CALL("ErrHdlr") //エラーをハンドル
errOccured:=False
errMsg:=""
If(ppBindMode=1) //パスワードモードがデフォルトの場合
   LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP password MD5)
Else
   LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP password plain text)
End if

Case of
   :(Not(errOccured))
      ALERT(" You are now connected to your LDAP server.  ")

   :(errOccured)
      ALERT("Error in your parameters")
End case

LDAP LOGOUT
ON ERR CALL("")
```

LDAP LOGOUT

LDAP LOGOUT
このコマンドは引数を必要としません

説明

LDAP LOGOUT コマンドカレントプロセスにおいて、LDAPサーバーとの接続を(開いていた場合)閉じます。接続がなかった場合、1003エラーが返されて、ログインしていないことが警告されます。

LDAP Search

LDAP Search (dnRootEntry ; filter {; scope {; attributes {; attributesAsArray}}}) -> 戻り値

引数	型		説明
dnRootEntry	文字	➡	検索を開始するルートエントリーの識別名
filter	文字	➡	LDAP検索フィルター
scope	文字	➡	検索のスコープ: "base" (デフォルト)、"one"、または"sub"
attributes	文字配列	➡	取得する属性
attributesAsArray	ブール配列	➡	True = 属性を強制的に配列として返す; false = 属性を強制的に単純な変数として返す;
戻り値	Object	↩	キー/値 属性

説明

LDAP Search コマンドはターゲットとなるLDAPサーバー内にて、定義された条件に合致する最初のオカレンスを検索します。このコマンドは**LDAP LOGIN** によって開かれたLDAPサーバーへの接続の中で実行される必要があります(それ以外の場合にはエラー1003が返されます)。

dnRootEntry 引数には、LDAPサーバールートエントリーの識別名を渡します。検索はここのエントリーから開始されます。

filter 引数には、実行するLDAP検索フィルターを渡します。フィルター文字列はrfc2225 に準拠している必要があります。空の文字列("")を渡すことに寄って検索にフィルターをかけないこともできます。"*"は部分列の検索をサポートします。

scope 引数には、以下の"LDAP"テーマのどれか一つを渡します:

定数	型	値	詳細
LDAP root only	テキスト	"base"	*dnRootEntry* によって定義されたルートエントリーレベルのみを検索します(省略された場合のデフォルト)
LDAP root and next	テキスト	"one"	*dnRootEntry* によって定義されたルートエントリーレベルと、その１階層下のレベルにあるエントリーを検索します。
LDAP all levels	テキスト	"sub"	*dnRootEntry* によって定義されたルートエントリーレベルと、その下にある階層全てのエントリーを検索します。

attributes 引数には適合したエントリーから取得する全てのLDAP属性を一覧を格納するためのテキスト配列を渡します。デフォルトではこの引数が省略された場合、全ての属性が取得されます。

注: LDAP 属性名は大文字/小文字を区別するという点に注意して下さい。LDAP 属性についてのより詳細な譲歩運関しては、MS Active directory用の全ての属性をまとめてあるこちらのページ を参照して下さい。

デフォルトではコマンドは、複数の結果が見つかった場合には属性を配列として返し、単一の結果が見つかった場合には単純な変数として返します。*attributesAsArray* 引数を使用すると、定義したそれぞれの属性に対し、返される属性のフォーマットを指定することができます:

- 要素に**true**を渡した場合、対応する要素の*attributes* 引数は配列として返されます。単一のエントリーが見つかった場合、コマンドは単一の要素を含む配列を返します。
- 要素に**false** を渡した場合、対応する要素の*attributes* 引数は単純な変数として返されます。複数のエントリーが見つかった場合、コマンドは最初の要素のみを返します。

例題 1

カンパニーディレクトリ内から、"smith"というユーザーの電話番号を取得したい場合を考えます:

```
ARRAY TEXT($_tabAttributes;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes;"phoneNumber")
LDAP LOGIN($url;$dn;$pwd)
$filter:="cn=*smith*"
$vfound:=LDAP Search($dnSearchRootEntry;$filter;LDAP all levels;$_tabAttributes)
LDAP LOGOUT
```

例題 2

"memberOf"属性で見つかった全てのエントリーを格納した配列を取得したい場合を考えます:

```
C_OBJECT($entry)
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;True)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
$entry:=LDAP Search($dnSearchRootEntry;"cn=adrien*";LDAP all
levels;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

ARRAY TEXT($_arrMemberOf;0)
OB GET ARRAY($entry;"memberOf";$_arrMemberOf)
 // $_arrMemberOf 内には、全てのエントリーグループを格納した配列が入っています
```

LDAP SEARCH ALL

LDAP SEARCH ALL (dnRootEntry ; arrResult ; filter {; scope {; attributes {; attributesAsArray}}})

引数	型		説明
dnRootEntry	文字	➡	検索を開始するためのルートエントリーの識別名
arrResult	Object array	⬅	検索の結果
filter	文字	➡	LDAP検索フィルター
scope	文字	➡	検索のスコープ: "base" (デフォルト)、"one"、または"sub"
attributes	文字配列	➡	取得する属性
attributesAsArray	ブール配列	➡	True = 属性を配列として返す; false = 属性を単純な変数として返す

説明

LDAP SEARCH ALL コマンドは、ターゲットとなるLDAPサーバー内のオカレンスのうち、定義された条件に合致するものを全て検索します。このコマンドは**LDAP LOGIN**によって開かれたLDAPサーバーへの接続の中で実行される必要があります(それ以外の場合にはエラー1003が返されます)。

LDAPサーバーには通常、検索のために受け付けられるエントリー数には上限設定されているという点に注意して下さい。例えば、Microsoft Active director は、デフォルトではエントリー数を1000に制限しています。

dnRootEntry 引数には、LDAPサーバールートエントリーの識別名を渡します。検索はここのエントリーから開始されます。

tabResult 引数には、全ての合致したエントリーを受け入れるためのオブジェクト配列を渡します。この配列内には、それぞれの要素に、合致したエントリーの属性/値のペアが格納されています。*attributes* 引数を使用することによって返される属性を定義することもできます。

filter 引数には、実行するLDAP検索フィルターを渡します。フィルター文字列はrfc2225 に準拠している必要があります。空の文字列("")を渡すことに寄って検索にフィルターをかけないこともできます。"*"は部分列の検索をサポートします。

scope 引数には、以下の"LDAP"テーマのどれか一つを渡します:

定数	型	値	詳細
LDAP root only	テキスト	"base"	*dnRootEntry* によって定義されたルートエントリーレベルのみを検索します(省略された場合のデフォルト)
LDAP root and next	テキスト	"one"	*dnRootEntry* によって定義されたルートエントリーレベルと、その１階層下のレベルにあるエントリーを検索します。
LDAP all levels	テキスト	"sub"	*dnRootEntry* によって定義されたルートエントリーレベルと、その下にある階層全てのエントリーを検索します。

attributes 引数には適合したエントリーから取得する全てのLDAP属性を一覧を格納するためのテキスト配列を渡します。デフォルトではこの引数が省略された場合、全ての属性が取得されます。

注: LDAP 属性名は大文字/小文字を区別するという点に注意して下さい。LDAP 属性についてのより詳細な譲歩運関しては、MS Active directory用の全ての属性をまとめてあるこちらのページ を参照して下さい。

デフォルトではコマンドは、複数の結果が見つかった場合には属性を配列として返し、単一の結果が見つかった場合には単純な変数として返します。*attributesAsArray* 引数を使用すると、定義したそれぞれの属性に対し、返される属性のフォーマットを指定することができます:

- 要素に**true**を渡した場合、対応する要素の*attributes* 引数は配列として返されます。単一のエントリーが見つかった場合、コマンドは単一の要素を含む配列を返します。
- 要素に**false** を渡した場合、対応する要素の*attributes* 引数は単純な変数として返されます。複数のエントリーが見つかった場合、コマンドは最初の要素のみを返します。

例題 1

カンパニーディレクトリから、"smith"という名前を持つ全てのユーザーの電話番号を取得したい場合を考えます:

```
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
```

```
APPEND TO ARRAY($_tabAttributes;"telephoneNumber")
APPEND TO ARRAY($_tabAttributes_asArray;False)
ARRAY OBJECT($_entry;0)

LDAP LOGIN($url;$myLogin;$pwd)
$filter:="cn=*smith*"
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all levels;$_tabAttributes)
LDAP LOGOUT


 //$_entry には、例えば以下のような値が返されます
 // $_entry{1} = {"cn":"John Smith","telephoneNumber":"01 40 87 00 00"}
 // $_entry{2} = {"cn":"Adele Smith","telephoneNumber":"01 40 87 00 01"}
 // $_entry{3} = {"cn":"Adrian Smith","telephoneNumber":"01 23 45 67 89"}
 // ...
```

例題 2

以下の例は*attributesAsArray* 引数の使用方について説明しています:

```
ARRAY OBJECT($_entry;0)
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;True)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all
levels;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

ARRAY TEXT($_arrMemberOf;0)
OB GET ARRAY($_entry{1};"memberOf";$_arrMemberOf)
 // $_arrMemberOf 内には、エントリーのグループを全て格納した配列が返されます。
```

```
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;False)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all
levels;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

$memberOf:=OB Get($_entry{1};"memberOf")
 // $memberOf には、エントリーの最初のグループのみを格納した変数が返されます。
```

4D v15 - アップグレードリファレンス(標準版) - コマンドリスト (文字順)