

■ 4D v15 - アップグレードリファレンス(Rリリース版)







4D v15へようこそ。4Dの継続的なデリバリー(Rリリース)プログラムの事を御存じであれば、このバージョンが4D v14 R5に続く、新しいステップの一つであるという事がお分かり頂けるかと思います。事実、4D v15は"4D v14 R6"であり、R5と比較すると、完全にテストされているものの少量の新機能が追加されているにすぎません。

4D v15に対しては、二種類の"アップグレードマニュアル"がリリースされています:

- この"Rリリースエディション"マニュアルでは、4D v14 R5と4D v15との間に追加された新機能・実装についてのみ説明しています。Rリリースプログラムのメンバーで、4D v14 R5以降に追加された新機能についてだけ知りたい方は、こちらのマニュアルをお読みください。以前のRリリースのアップグレードマニュアルは、以下のリンクからダウンロードできます(PDF):
 - [4D v14 R2 アップグレード](#)
 - [4D v14 R3 アップグレード](#)
 - [4D v14 R4 アップグレード](#)
 - [4D v14 R5 アップグレード](#)
- "4D v15 - アップグレードリファレンス(標準版)"は4D v14.xと4D v15リリースの間に追加された全ての新機能について説明しています。言い換えると、これは4D v14リリース全体で追加された全ての機能のまとめであるという事もできます(4D v14 R5の後に追加された機能も含めます)。Rリリースプログラムのメンバーでない方、またはRリリースで紹介された新機能などを見落としてしまった方などは こちらのマニュアルを読むことをお勧めします。

- 📄 デザインモード
- 📄 ランゲージ
- 📄 4D Server
- 👤 カンパニーディレクトリ(LDAP)
- 🔑 コマンドリスト (文字順)

デザインモード

-  組み込みアプリ内のデフォルトのデータファイル
-  オブジェクトフィールドデータ型
-  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
-  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

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:

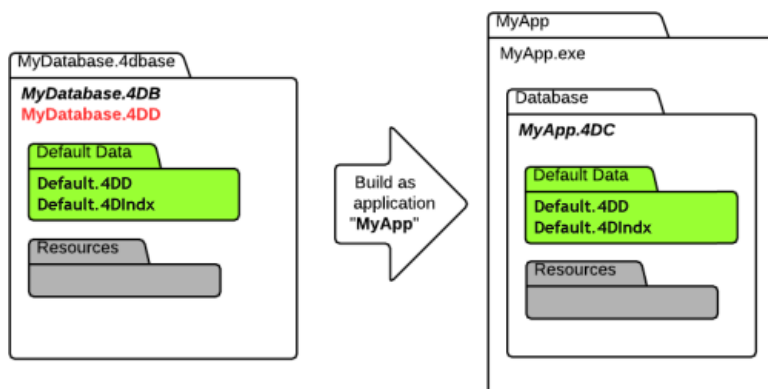
- デベロッパはデフォルトのデータファイルを、データベースパッケージ(4dbase)内のデフォルトのフォルダ内に保存する必要があります:
 - デフォルトのデータフォルダ名は"Default Data"である必要があります。
 - デフォルトのデータファイル名は"Default.4DD" である必要があります、デフォルトのデータフォルダ内に保存されている必要があります。

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ています。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。

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)。

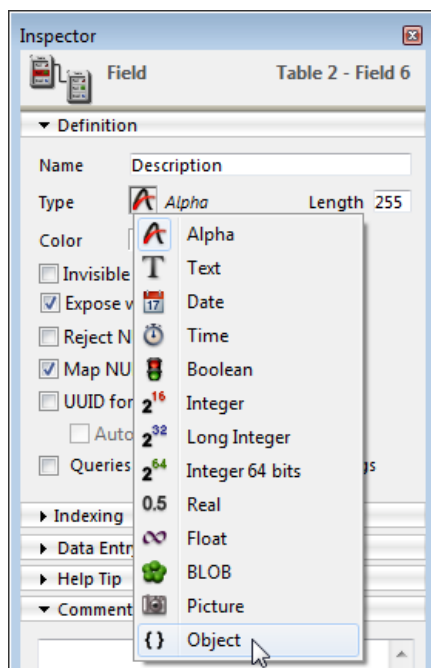
何故オブジェクトフィールドを使用するのか？

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])
```

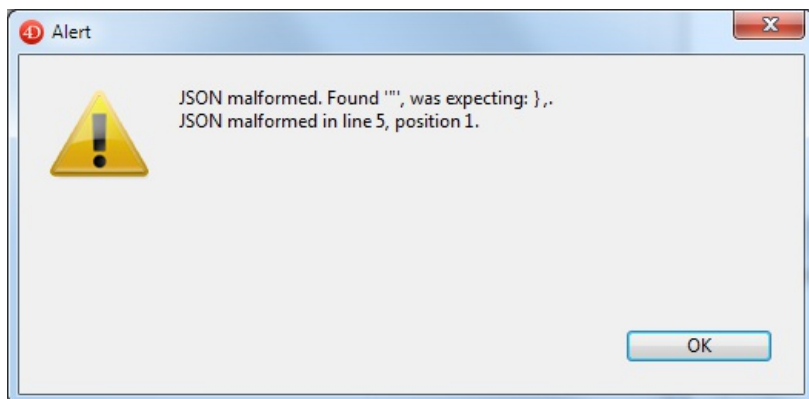
[Rect]Descフィールドがフォーム内に含まれているとき、以下の内容が表示されます:



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。

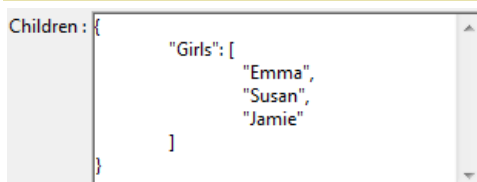
標準のランゲージオブジェクトの様に、オブジェクトフィールド値は**オブジェクト(ランゲージ)**テーマのコマンドを使用して管理することができます。例えば：

```
// 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。こちらの方がより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、フォーミュラ内においてのみ、オブジェクトフィールドをサポートします。

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以下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

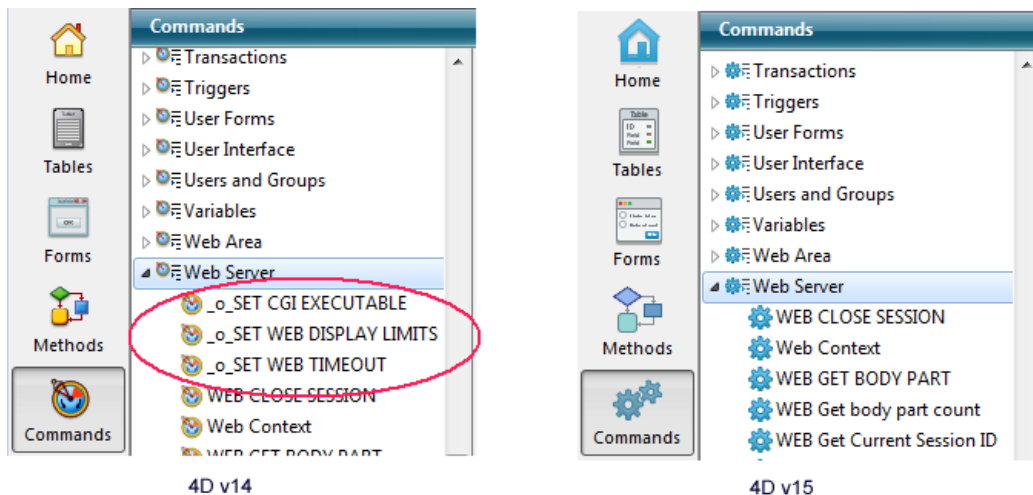
SCAN INDEX

❏ 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のリストとコンプリート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：



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ヘルパーリスト(コマンドの一覧と、テーマごとのコマンドの一覧)、それから入力補完ポップアップ
- フォーミュラエディターヘルパーリスト

もちろん、既存のコードの中で、廃止予定のコマンドが勝手に削除されていることはありません(ただし先頭に"_o_"がつけられています)し、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：

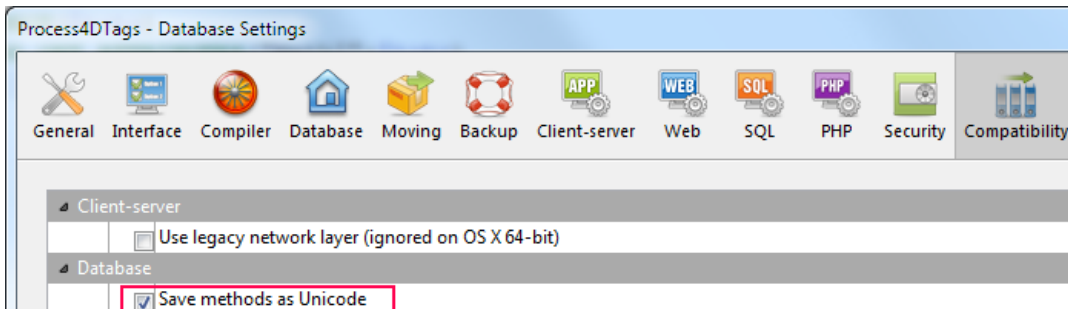
```
4 | _o_SET CGI EXECUTABLE |
```

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](#)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、(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。

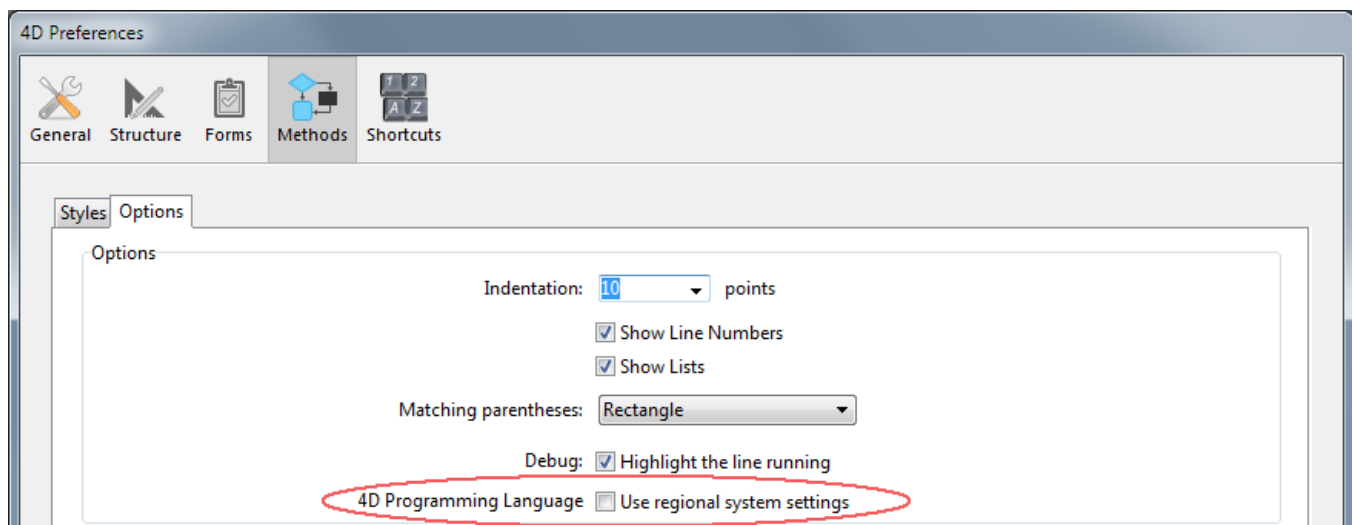
この新しいデフォルト設定により、4Dデベロッパには主に二つの利点があります：

- 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。

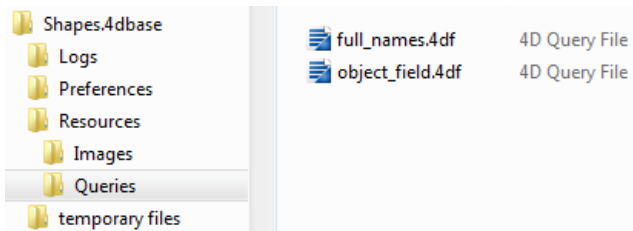
クエリエディター

定義済みのクエリ

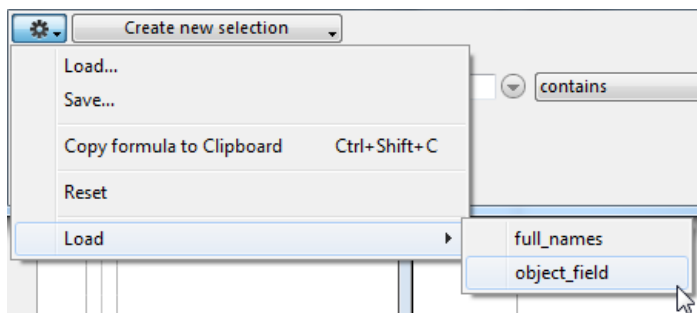
4D v15のクエリエディターでは**定義済みクエリ**をサポートします。ちょうど標準の保存されたクエリのように、定義済みクエリには全ての条件を含む完全なクエリ定義が格納されており、いつでもクエリエディターに読み込むことができます。定義済みのクエリは配付されたアプリケーションにも埋め込むことができ、クエリエディターの新しいサブメニュー内に表示されます。

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。

ランゲージ

-  Get database parameter と SET DATABASE PARAMETER
-  GET FIELD PROPERTIES
-  Get pointer
-  METHOD GET CODE と METHOD SET CODE
-  On Scroll フォームイベント
-  RESOLVE POINTER
-  WEB GET OPTION と WEB SET OPTION
-  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)
-  廃止予定のコマンドの改名
-  新しいLDAPコマンドテーマ
-  METHOD GET ATTRIBUTES
-  METHOD SET ATTRIBUTES
-  QUERY BY ATTRIBUTE

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ServerNet 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。

GET FIELD PROPERTIES

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

説明

テーマ: ストラクチャーアクセス

4D v15以降、フィールドには新しいオブジェクト型を選択することができるようになりました。詳細な情報については、[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。

結果として、このコマンドは*fieldType* 引数に対し"Field and Variable Types" 定数テーマに追加された新しい値を返すようになりました:

定数	型	値
Is object	倍長整数	38

Get pointer

Get pointer (varName) -> 戻り値

説明

テーマ: ランゲージ

Get pointer コマンドは4D v15にて新しいポインター型をサポートするために書き換えられました([メソッドでのUnicode](#)の章を参照して下さい)。以下の変更点に注意する必要があります:

- 二次元配列へのポインター(式の使用を含む)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{2}{6}
```

- 変数に無効な名前を指定した場合、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、これは使用可能でした。例:

```
$pt:=Get pointer("123") //4D < v15では使用可能
// 4D v15 以降のバージョンでは使用不可
```

- 余分な空白が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"
// 4D < v15では使用不可
```

METHOD GET CODE と 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    "publishedWsd1": 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。

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 v15On 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";$col;$row)
  LISTBOX GET CELL COORDINATES (*;"LB1";$col;$row;$x1;$y1;$x2;$y2)
  OBJECT SET VISIBLE (*;"RedRect";True) &NBSP; // 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";$col;$row)
  LISTBOX GET CELL COORDINATES (*;"LB1";$col;$row;$x1;$y1;$x2;$y2)
  OBJECT GET COORDINATES (*;"LB1";$x1b1;$y1b1;$x1b2;$y1b2)
```

```

$toAdd:=LISTBOX Get headers height(*;"LB1") //オーバーラップしないためにヘッダーの高さを取得
If($y1b1+$toAdd<$y1) & ($y1b2>$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

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"
```

WEB GET OPTION と WEB SET OPTION

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

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

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]
```

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。

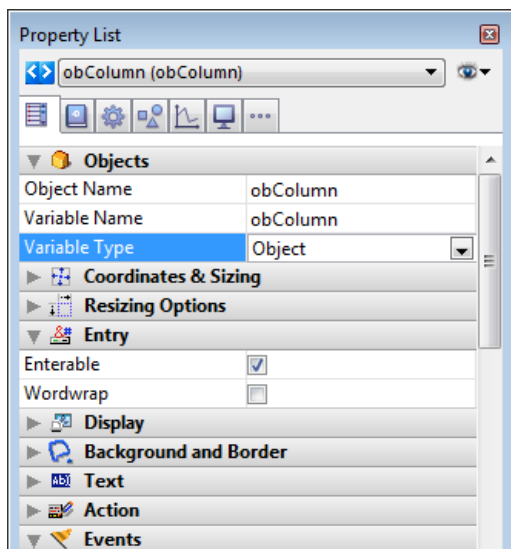
以下のリストボックスはオブジェクト配列を使用してデザインされました:

Label	Value
Document Name	MyReport
Document Type	PDF
Reference	123456
Category	<input type="text" value=""/>
Include Abstract	<input checked="" type="checkbox"/>
Printable area size (height)	297 <input type="text" value="mm"/>
Printable area size (width)	210 <input type="text" value="mm"/>
Show Preview	<input type="button" value="Preview..."/>

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**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

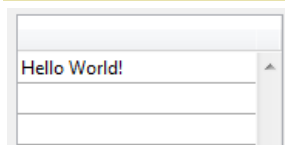
- 値そのもの(任意):入力/出力に使用
- セルの中身の表示(任意):ボタン、リスト、他
- 追加の設定(任意):値の型によります

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、以下の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。

属性	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

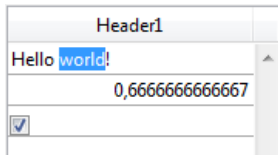
```

```

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す、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:

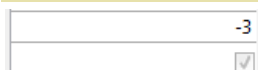
属性	使用可能な値	valueType	詳細
behavior	threeStates	integer	スリーステートチェックボックスを数値として表現します。2=セミチェック、1=チェック、0=チェックされていない、-1=非表示、-2=チェックなしが無効化、-3=チェックが無効化、-4=セミチェックが無効化

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、テキスト入力に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。

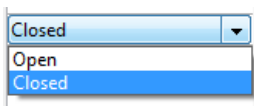
どちらの場合においても、"value"属性を使用してウィジェット内の値を事前に選択することができます。

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:

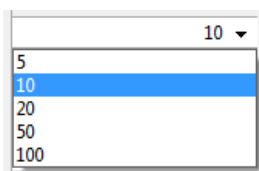
- 選択肢が二つ("Open"または"Closed")しかないドロップダウンリストを表示したい場合を考えます。"Closed"が選択された状態にしたいとします:

```
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。

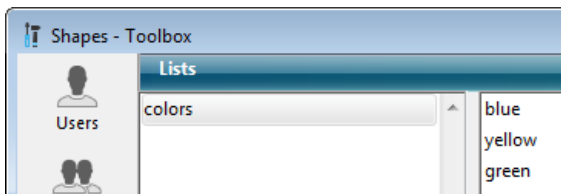
- "saveAs" = "reference" であった場合、項目は参照として保存されます。"valueType" はreal または integerである必要があります。
- "saveAs" = "value" であった場合、値が保存されます。この場合、"valueType" はリストの値と同じ型である必要があります(通常、"text" または "integer"です)。そうでない場合、4Dはリストの値をオブジェクトの"valuType"へと変換しようとします(以下の例を参照して下さい)。

"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。

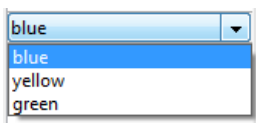
注: リストが実数値を表すテキストを含んでいる場合、小数点はローカル設定に関わらず、ピリオド(".")である必要があります。例:"17.6" "1234.456"

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:

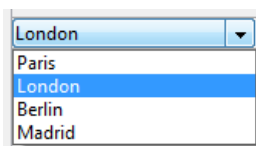


```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"colors")
```



- プログラミングによって定義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](#))。

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、以下の属性に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

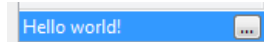
セルに省略ボタン[...] を追加したい場合、"alternateButton" 属性に**True** の値を入れてオブジェクトに渡すだけです。省略ボタンは自動的にセル内に表示されます。

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($obj1)
$entry:="Hello world!"
OB SET($obj;"valueType";"text")
OB SET($obj;"alternateButton";True)
OB SET($obj;"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ことができます。

- 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($obj4)
OB SET($obj4;"valueType";"color")
OB SET($obj4;"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")。

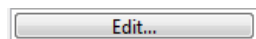
event valueType

"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"属性を渡すことができます。

例:

```
C_OBJECT($obj)
OB SET($obj;"valueType";"event")
OB SET($obj;"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。

廃止予定のコマンドの改名

廃止予定のコマンドとは、使用の中止が強く推奨されるようになり、将来のバージョンの4Dにおいては維持されない予定のものであります。

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が既存のコードの中で使用されていた場合、それらは単純に改名されます。

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_INTEGER	_o_C_INTEGER
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
During	_o_During
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

廃止予定ではなくなったコマンド

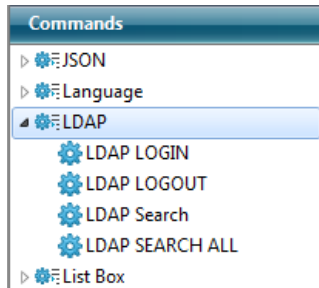
その一方で、ドキュメントにおいて廃止予定だと以前宣言されていた一部のコマンドのうち、上述の定義に当てはまらないものは、"再採用"されることになりました。

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**

新しいLDAP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LDAP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。

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ことができます。*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** 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, "publishedWsd1":false,
"shared":false, "publishedSql":false, "executedOnServer":false, "published4DMobile":{
"scope":"table", "table":"Table_1" } }
```

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	→ 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

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ことができます。*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
  "publishedWsd1": 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"
    "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)
```

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ようになります:

```
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付きまます 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、例えば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、データにアクセスす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、この設定の範囲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
```

4D Server

 OS X用4D Serverの64ビット版について(プレビュー)

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-bit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

🔧 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け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 (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
password	文字	→ ログインエントリのパスワード
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。

ログイン引数が無効であった場合、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際には、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 (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	→ 検索の範囲: "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 = 属性を強制的に単なる変数として返す;
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ブジェクトを検索します。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"	<i>dnRootEntry</i> 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"	<i>dnRootEntry</i> 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"	<i>dnRootEntry</i> 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に関しては、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なる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* 引数は単なる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 (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"	<i>dnRootEntry</i> 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"	<i>dnRootEntry</i> 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"	<i>dnRootEntry</i> 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` 引数の使い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(Rリリース版) - コマンドリスト (文字順)

- ⚙ LDAP LOGOUT
- ⚙ LDAP Search
- ⚙ LDAP SEARCH ALL