









4D v14 R4 - アップグレード


4Dの継続的なデリバリープログラムの最新バージョンとなる、4D v14 R4へようこそ。このマニュアルでは、この新しいリリースの中に含まれる全ての新機能や実装について説明しています。

-  ランゲージ
-  デザインモード
-  4D Server
-  最適化
-  4D Internet Commands
-  4D View
-  コマンドリスト (文字順)


ランゲージ

 4D変換タグ


 PROCESS 4D TAGS

 XSLT commands deprecated

 EXPORT STRUCTURE

 Find in sorted array

 IMPORT STRUCTURE

 OBJECT Get corner radius

 OBJECT SET CORNER RADIUS

4D変換タグ

4D v14 R4では、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: 4D v14 R4では、**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)。

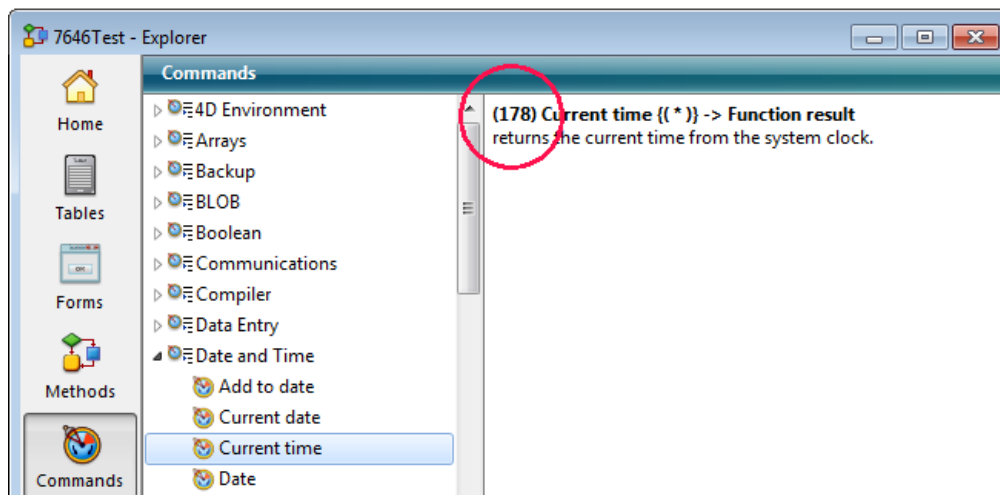
例えば、以下の様なコードを実行することができます:

```
$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。その際使用する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"
```

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

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くは変わりありません。

注: 4D v14 R4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"
```

XSLT commands deprecated

4D v14 R4 以降、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 v14 R4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、4D v14 R4においてその機能が著しく拡張されました。

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

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箇所。
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

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ー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ー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
```

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

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するとき、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)
```

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

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(**角の丸い四角向けの新しいプロパティ**、**角の半径**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

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。

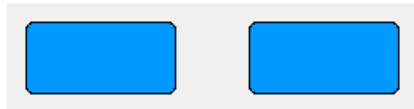
注: 現在の4D R リリースでは、このコマンドはスタティックなオブジェクトである角の丸い四角にのみ適用されるため、常に * 引数を渡し、オブジェクト名シンタックスを使用する必要があります。

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よってフォームレベルで設定することも可能です(**角の丸い四角向けの新しいプロパティ、角の半径**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)
```



デザインモード

-  デバッガとランタイムエクスプローラー
-  重複不可のフィールドではインデックスが必須に
-  角の丸い四角向けの新しいプロパティ、角の半径

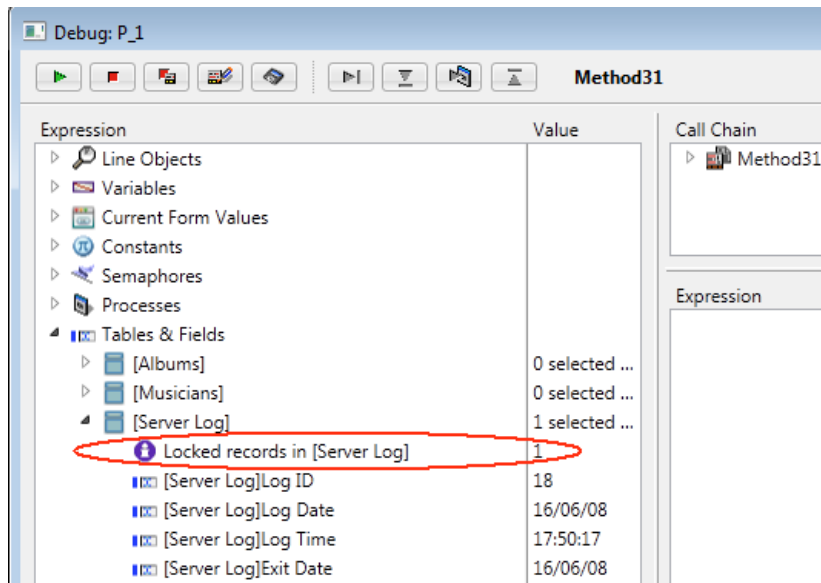
デバッガとランタイムエクスプローラー

4D v14 R3では、デベロッパがアプリケーションをデバッグするのを手助けするためのいくつかの新機能が追加されています。これらの機能はデバッガとランタイムエクスプローラーウィンドウズと、両方で使用可能です。

ロックされたレコード

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:



Expression	Value	Call Chain
Line Objects		
Variables		
Current Form Values		
Constants		
Semaphores		
Processes		
Tables & Fields		
[Albums]	0 selected ...	
[Musicians]	0 selected ...	
[Server Log]	1 selected ...	
Locked records in [Server Log]	1	
[Server Log]Log ID	18	
[Server Log]Log Date	16/06/08	
[Server Log]Log Time	17:50:17	
[Server Log]Exit Date	16/06/08	

カレントフォーム値

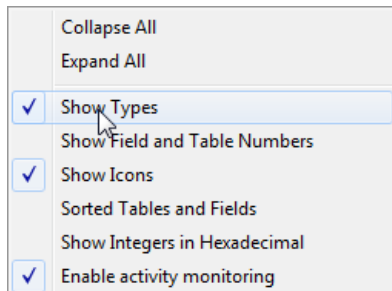
新しいカレントフォーム値 リストがデバッガとランタイムエクスプローラーに表示されるようになりました。このリストには、カレントフォームに含まれる動的なオブジェクトそれぞれの名前と、それに関連付けられた変数の値が含まれています:

Expression	Value
Line Objects	
Variables	
Current Form Values	
bCancel	0
bDelete	0
bValidate	0
FirstName	"Patrick"
ID	2
LastName	"Smith"
List Box	3 elements
List Box	Listbox sub objects
Column1	3 elements
Column1	0
Column1	""
Column1	"Henry"
Column1	"Marc"
Column1	"Lesly"
Footer1	""
Header1	0
List Box1	0 elements
List Box1	Listbox sub objects
TestButton	1
Variable	""
Variable1	""

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、"\$form.4B9.42"という形式になっています:

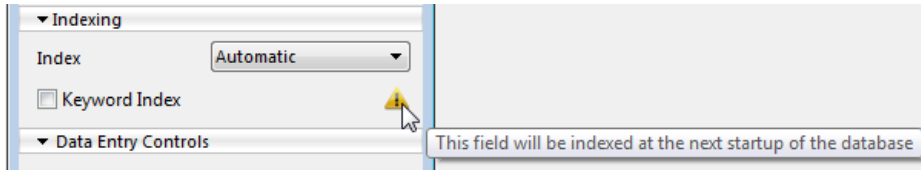
Variable2 -> \$form.4B9.42 : Text	""
vRecNum -> vRecNum : Text	"2 of 2"

重複不可のフィールドではインデックスが必須に

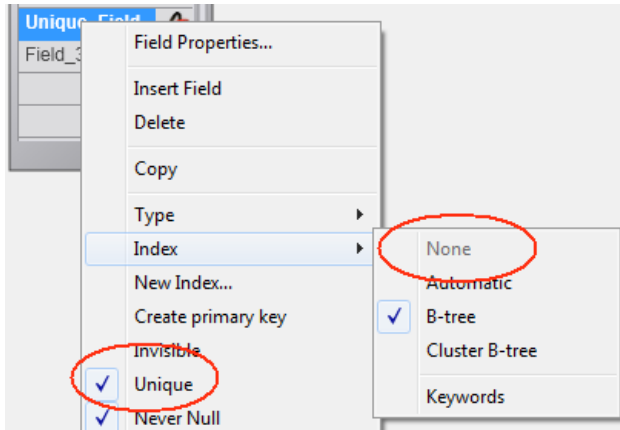
4Dでは、**重複不可**属性を持つフィールドはインデックスされる必要があります。4D v14 R4以降、ストラクチャーエディターにおいて、インデックスを持たない**重複不可**属性のフィールドを定義することはできなくなりました。以前のリリースでは、メンテナンスのために、そのような設定を維持することは可能でした。

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。

■ 角の丸い四角向けの新しいプロパティ、角の半径

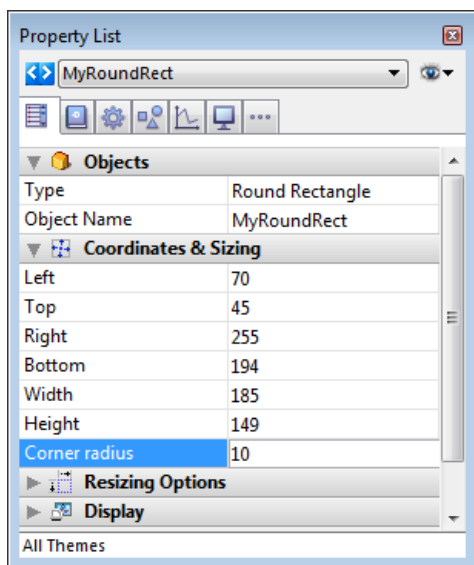
4D v14 R4 以降、角の丸い四角オブジェクトの角の半径を設定できるようになりました。このプロパティはそれぞれの丸い角の半径を定義します。

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)。

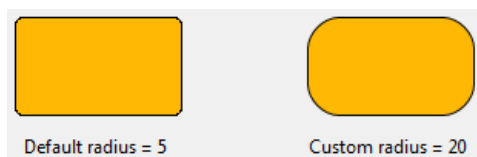
このプロパティはフォームエディターのプロパティリスト内の新しい**角の半径**オプション(以下参照)を通して、または二つの新しいコマンドを使用することによって設定可能です(**OBJECT SET CORNER RADIUS** と **OBJECT Get corner radius**を参照して下さい)。

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よって変化し(長方形の短辺の半分を超えることはできません)、動的に計算されます。

4D Server

 OS X用4D Serverの64ビット版について(プレビュー)

 ネットワーク互換性オプション(重要なお知らせ)

📄 OS X用4D Serverの64ビット版について(プレビュー)

4D v14 R3以降、4D ではOS X用の64-bit 版 4D Server の使用可能なプレビューバージョンを提供してきました。このバージョンについては、 [4D v14 R3 - Upgrade](#) (PDF) マニュアル内に説明があります。

このバージョンはステップバイステップ形式に継続的な開発行われており、以前は利用できなかった機能の一部がOS X用の4D Server 64-bit版において利用可能になっています。

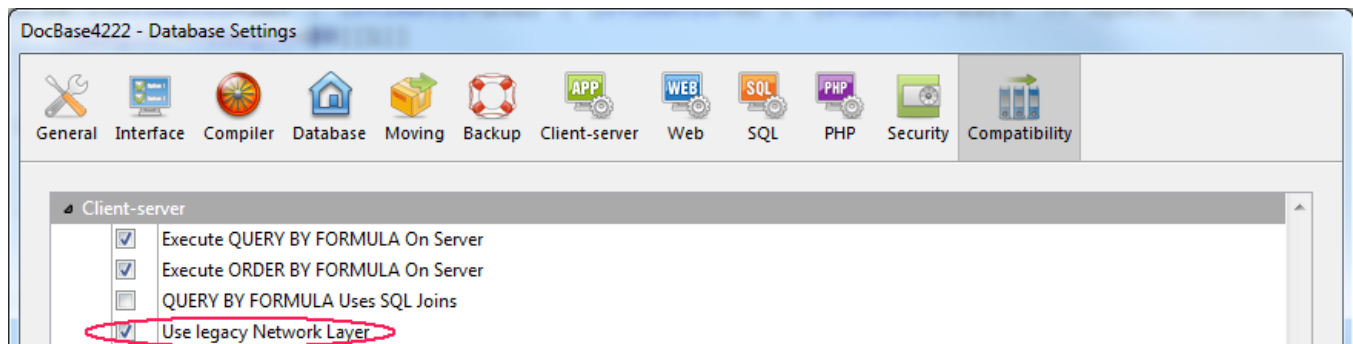
開発中の機能

この表では、最初のOS X用4D Server 64-bit版(v14 R3)でまだ利用できなかった機能の、現在の状況をまとめています。

機能/テクノロジー	現在の状況
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
サーバー上で生成されたグラフ	R4以降利用可能
HTTP クライアント(クライアント証明書の管理)	R4以降利用可能
4D Internet Commands (プラグイン)	R4以降利用可能 (OS X用4D Internet Commands 64-bit版 を参照して下さい)
4D Pack (プラグイン)	利用不可
4D ODBC Pro (プラグイン)	利用不可
4D For OCI (プラグイン)	利用不可


📄 ネットワーク互換性オプション(重要なお知らせ)


4D v14 R4 にはデータベース設定ダイアログボックス内の"互換性"ページ内に新しいオプションが付いてきます:



4Dでは、4D Serverのネットワークレイヤーの新しいバージョンに取り組んでおり、"旧式ネットワークレイヤーを賜与す売る"オプションは、使用するバージョンを選択するためにデザインされたものです。しかしながら、テストと検証が途中であるため、4D v14 R4においては4Dのエンジニアリング部門から指示がない限りはこのオプションのチェックを外さないよう
お願い申し上げます。

最適化

 SQL Select distinct 宣言

 ネットワークセキュリティレベルの増強

SQL Select distinct 宣言

4D v14 R4以降、SQL宣言を通して実行されたSELECT DISTINCT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、大抵の場合、4D v14 R4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
```

ネットワークセキュリティレベルの増強

4D v14 R4 では、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。

4D Internet Commands

 SMTP_Attachment

 OS X用4D Internet Commands 64-bit版

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

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	画像/jpeg
png	画像/png
gif	画像/gif
pdf	アプリケーション/pdf
doc	アプリケーション/msword
xls	アプリケーション/vnd.ms-excel
ppt	アプリケーション/vnd.ms-powerpoint
zip	アプリケーション/zip
gz	アプリケーション/gzip
json	アプリケーション/json
js	アプリケーション/javascript
ps	アプリケーション/postscript
xml	アプリケーション/xml
htm, html	テキスト/html
mp3	音声/mpeg
other	アプリケーション/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")
```

OS X用4D Internet Commands 64-bit版

v14 R4のリリースから、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(4D v14 R3にてリリース)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 FTP_Progress](#) コマンドはOS X用64-bit版では廃止予定となっています。

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 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。

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

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ことができます:

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:

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				
9				

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

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:

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				
9				

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 v14 R4 - アップグレード - コマンドリスト (文字順)

E F I O P

EXPORT STRUCTURE