















# 4D v15 R4 - アップグレード

-  デザインモード
-  ランゲージ
-  4D Write Pro
-  4D Mobile
-  SQL
-  最適化
-  4D Developer Edition 64-bit版(プレビュー)
-  コマンドリスト (文字順)

## デザインモード

-  リストボックス:エリプシス表示オプション
-  リストボックス:余分な空白行を隠すオプション
-  リストボックス:無効化された行と選択不可の行
-  データファイルパスをユーザー設定内に保存する
-  組み込みクライアントでの新しい接続プロシージャ
-  オブジェクトライブラリ内の新しい4D Write Proフォームオブジェクト

## 📄 リストボックス:エリプシス表示オプション

### 新しいエリプシスで省略オプション

4D v15 R4では新しい**エリプシスで省略オプション**が利用可能となり、これによってリストボックスのカラムの幅がその中身全体を表示するのに細すぎる場合でもその表示をコントロールできるようになりました。この新しいオプションはプロパティリストの"表示"テーマ内にあり、ピクチャー型とオブジェクト型を除いて全てのカラムの型に対して利用可能です:

Display	
Alpha Format	
Invisible	<input type="checkbox"/>
Wordwrap	<input type="checkbox"/>
Truncate with ellipsis	<input checked="" type="checkbox"/>

- このオプションが**チェックされている場合**(デフォルト)、リストボックスセルの中身がカラムの幅を越えた場合、その一部は切り落とされ、エリプシス記号が表示されます:

Cities	Popu...
Charlotte	792862
New York	8406...
Philadelp...	1553...
San Fran...	837442
San Jose	288054

**注:** エリプシス記号が置かれる位置は、OSによります。上記の例(Windows)では、テキストの右側に追加されます。OS X上では、エリプシスはテキストの中央部に追加されます。

- このオプションが**チェックされていない場合**、セルの中身がカラムの幅を越えた場合、越えた部分は表示されず、エリプシス記号もありません:

Cities	Popu...
Charlotte	792862
New York	3406000
Philadelphi	.553000
San Francis	837442
San Jose	288054

**エリプシスで省略オプション**はデフォルトでチェックされており、配列またはセレクション型のリストボックスに対して指定可能です。**リストボックス**(オブジェクト全体)に対して指定できますし、リストボックスの**カラム**あるいは**フッター**に対して個別に設定可能です。

### ワードラップとエリプシス

テキストまたは文字列型のカラムに対して適用する際、**エリプシスで省略オプション**は、**ワードラップオプション**が**チェックされていない**場合のみ利用可能です。ワードラップオプションがチェックされていた場合、セル内の表示範囲を超えた部分はワードラップ機能によって管理されるため、**エリプシスで省略オプション**は利用できません。

**注:** 統一性のため、**ワードラップオプション**はプロパティリスト内で"入力"テーマから"表示"テーマへと移動されました。

### ブール型の場合

**エリプシスで省略オプション**はブール型のカラムに対しても適用可能ですが、表示結果はフォーマットに応じて変化します:

- ポップアップ型ブールフォーマットの場合、ラベルは省略されエリプシス記号が表示されます。
- チェックボックス型のブールフォーマットの場合、ラベルは常に切り落とされます。

## リストボックス:余分な空白行を隠すオプション

4D v15 R4以降、リストボックスオブジェクトの下部に追加された余分な空白行の表示をコントロールする新しいリストボックスオプションが追加されました。4Dは空白のエリアを埋めるために、デフォルトでそういった余分な行を追加します:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

Extra blank rows

これらの余分な行は、プロパティリストの"背景と境界線"テーマ内にある、新しい**余分な空白行を隠す**オプションを選択することで取り除くことができます:

Background and Border	
Transparent	<input type="checkbox"/>
Background Color	
Alternate Background Color	
Row Background Color Array	
Border Line Style	System
Hide extra blank rows	<input checked="" type="checkbox"/>

このオプションをチェックした場合、リストボックスオブジェクトの下部は空白のままになります:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

この新しいオプションは配列型とセレクション型のリストボックスオブジェクトにおいて利用可能です。

## 📄 リストボックス:無効化された行と選択不可の行

4D v15 R4以降、配列型リストボックスにおいて、それぞれの行に対して新しい"無効化"と"選択可能"インターフェースプロパティをコントロールできるようになりました。

これらの新しい機能は**行コントロール配列**という新しいプロパティ配列を通して管理されます。この配列は、以前のバージョンにおいてリストボックスに関連づけられていた既存の"非表示行配列"プロパティの拡張版です。

### 行コントロール配列プロパティ

配列型のリストボックスは、"リストボックス"テーマ内に新しいプロパティ、**行コントロール配列**を持つようになりました

Objects	
Type	List Box
Object Name	List Box
Variable Name	List Box
Data Source	Arrays
List Box	
Number of Columns	6
Number of Locked Col...	0
Number of Static Colu...	0
Row Control Array	aLControlArr
Selection Mode	Multiple

**互換性に関する注意:** このプロパティは以前の4Dリリースにあった"非表示行配列"にとってかわるものです。互換性のため、ブール配列も行コントロール配列として引き続きサポートされます(以下を参照して下さい)。

ブール型配列、または倍長整数配列の名前を入力することができます:

- **ブール 配列:** これは以前のリリースとの互換性のために提供されています(上記の互換性に関する注意を参照して下さい)。この場合、配列は以前の4Dのバージョンと同じように使用されます。それぞれの配列の要素は、リストボックス内の対応した行の表示/非表示の状態をあらわします(True=非表示/False=表示をあらわします)。
- **倍長整数 配列(新機能):** この場合、リストボックスの複数のインターフェースプロパティをコントロールすることができます。それぞれの配列の要素は以下のように行をコントロールします:
  - 非表示または表示(デフォルトは表示)
  - 有効化あるいは無効化(デフォルトは有効化)
  - 選択可能あるいは選択不可(デフォルトは選択可能)

どちらの型でも、配列の要素の数はリストボックスの数と一致しなければなりません。それぞれの配列の要素は、対応したリストボックスの行を管理します。

**注:** **行コントロール配列**プロパティは**LISTBOX SET ARRAY**または**LISTBOX Get array**コマンドを使用して設定または取得することができます。**LISTBOX GET ARRAYS**コマンドを使用して配列を返すこともできます。より詳細な情報については、**LISTBOX Get array**、**LISTBOX SET ARRAY**、**LISTBOX GET ARRAYS**の章を参照して下さい。

### 行コントロール配列の管理

**注:** この段落では、倍長整数型の行コントロール配列についてのみ説明しています。ブール配列を使用した場合については、すでにリストボックスのドキュメントの中の詳細な説明があります(この章の"非表示行配列"の段落を参照して下さい)。

**行コントロール配列**のそれぞれの要素はそれにリストボックス内で対応する行のインターフェースの状態について定義します。"リストボックス"定数テーマ内の定数を使用する事によって3つのインターフェースプロパティが使用可能です:

定数	型	値	詳細
Listbox row is hidden	倍 長 整 数	1	対応する行は非表示(既存の非表示プロパティに相当)。デフォルト:表示
Listbox row is disabled	倍 長 整 数	2	対応する行は無効化(チェックボックスなどのテキストとコントロール類は灰色表示)。入力可能なテキスト入力エリアは入力不可になります。デフォルト:有効化
Listbox row is not selectable	倍 長 整 数	4	対応する行は選択不可(ハイライト不可)。入力可能なテキスト入力エリアは"シングルクリック編集"が有効化されていない限り入力不可。ただしチェックボックスやリストなどのコントロールは機能します。この設定はリストボックス選択モードが"なし"の場合には無視されます。デフォルト:選択可能

行の状態を変更するためには、その行に対応する配列要素に適切な定数を設定するだけです。例えば、10行目を選択可能にしたい場合、以下のように書く事ができます:

```
aLControlArr{10}:=Listbox row is not selectable
```

RowNum	Countries	Population	Landlocked
1	Luxembourg	0 502 202	<input checked="" type="checkbox"/>
2	Latvia	1 973 700	<input type="checkbox"/>
3	Kuwait	4 044 500	<input type="checkbox"/>
4	Croatia	4 284 889	<input type="checkbox"/>
5	Denmark	5 699 220	<input type="checkbox"/>
6	Nicaragua	6 071 045	<input type="checkbox"/>
7	Czech Republic	10 674 947	<input checked="" type="checkbox"/>
8	Serbia	7 306 677	<input checked="" type="checkbox"/>
9	Honduras	8 249 574	<input type="checkbox"/>
10	Austria	8 572 895	<input checked="" type="checkbox"/>
11	Hungary	10 005 000	<input checked="" type="checkbox"/>
12	Greece	10 815 197	<input type="checkbox"/>
13	Benin	10 879 829	<input type="checkbox"/>

複数のインターフェースプロパティを同時に設定する事もできます:

```
aLControlArr{8}:=Listbox row is not selectable+Listbox row is disabled
```

RowNum	Countries	Population	Landlocked
1	Luxembourg	0 502 202	<input checked="" type="checkbox"/>
2	Latvia	1 973 700	<input type="checkbox"/>
3	Kuwait	4 044 500	<input type="checkbox"/>
4	Croatia	4 284 889	<input type="checkbox"/>
5	Denmark	5 699 220	<input type="checkbox"/>
6	Nicaragua	6 071 045	<input type="checkbox"/>
7	Czech Republic	10 674 947	<input checked="" type="checkbox"/>
8	Serbia	7 306 677	<input checked="" type="checkbox"/>
9	Honduras	8 249 574	<input type="checkbox"/>
10	Austria	8 572 895	<input checked="" type="checkbox"/>
11	Hungary	10 005 000	<input checked="" type="checkbox"/>
12	Greece	10 815 197	<input type="checkbox"/>
13	Benin	10 879 829	<input type="checkbox"/>

要素に対してプロパティを設定すると、(再設定しない限り)その要素の他の値を上書きするという点に注意して下さい。例えば:

```
aLControlArr{6}:=Listbox row is disabled+Listbox row is not selectable //6行目を無効化し、かつ選択不可に設定
aLControlArr{6}:=Listbox row is disabled //6行目を無効化に設定するが、選択は可能になる
```

## データファイルパスをユーザー設定内に保存する

### 概要

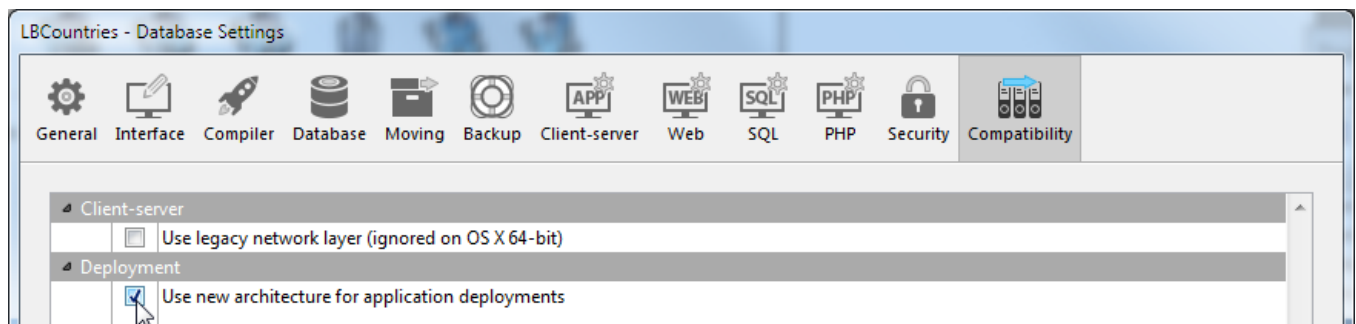
これまで4Dは、最後に開いたデータファイルのパスを自動的にストラクチャーファイルに保存してきました。この有益な機能により、4Dアプリケーションの起動が早くなるとともに、複数のデータファイルを切り替えて使用する事が可能でした。

以前のバージョンでは、この情報はストラクチャーファイル内に保存されてきました。これはほとんどの場合において適切である一方、組み込み(エンジン)アプリの配布コンテキストではトラブルの原因ともなり得ます。具体的には、デベロッパーが配布済みのアプリケーションにアップデートを送付した場合、ユーザーはアップデート後の初回起動時に必ず手動でデータファイルを再度選択する必要がありました。

この問題を避けるために、4D v15 R4以降、組み込みアプリにおいてデータファイルのパスはユーザー設定の中に保存できるようになりました。この新機能により、組み込みアプリをよりOSに準拠しやすくなり、より柔軟性を持たせられるようになりました。

### 機能の有効化(互換性オプション)

この章で説明がなされている機能は、データベース設定ダイアログボックスの"互換性"ページにある**アプリケーション配布に新しいアーキテクチャーを使用**オプションがチェックされているときのみ利用可能です:



- このオプションは、4D v15 R4以降で作成された4Dデータベースではデフォルトでチェックされています。"新アーキテクチャー"は有効化され、他には何も追加で設定する必要はありません。
- 互換性上の理由から、このオプションは以前のバージョンから変換されたデータベースではデフォルトでチェックが外されています。新機能を活用するためにはチェックする必要があります。

### 最後に開いたデータファイルパスの新しい保存場所

4D v15 R4で作成されたデータベース、あるいは変換されたデータベースで**アプリケーション配布には新アーキテクチャーを使用**互換性オプション(上記参照)がチェックされているものにおいては、4Dでビルドされたアプリケーションであればスタンドアロン、サーバー用に関わらず、最後に開いたデータファイルのパスをストラクチャーファイルには保存しなくなりました。その代わりに、そのパスはアプリケーションのユーザー設定フォルダに保存されます。

アプリケーションのユーザー設定フォルダの位置は、以下のコードで返されるパスに対応しています:

```
userPrefs:=Get 4D folder(Active 4D Folder)
```

データファイルのパスは、実際にはlastDataPath.xmlという名前の専用のファイル内に保存されます。

この新アーキテクチャーのおかげで、アプリケーションの新バージョンを配布するとき、アップデート後の初回起動時からローカルのユーザーデータファイル(最後に使用されたデータファイル)が自動的に開かれます。

この機構は通常、標準の配布に対して適しています。しかしながら特定の場合(例えば組み込みアプリを複製した場合など)では、データファイルとアプリケーションのリンク方法を変更したい場合もあるかもしれません。この点についての詳細な説明については、次の段落をご覧ください。

### データリンクモードの変更

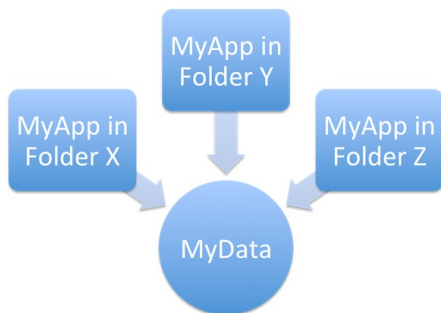
## 何故データリンクモードを変える必要があるのか？

v15 R4以降でも、4Dは以前のリリースのように、組み込みアプリにおいては最後に開いたデータファイルを自動的に使用します。しかしながら、"新アーキテクチャー"が有効化されると、データファイルのパスはユーザー設定フォルダー内に保存されます。その結果として、以前までは使用可能だったある特定の手段が使用できなくなります。**組み込みアプリを複製して異なるデータファイルを使用する**という方法です。

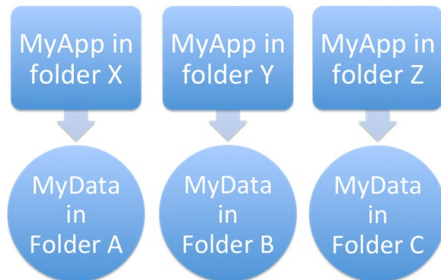
このトリックは、以前はデータファイルパスはアプリケーションパッケージ内に保存されており、ストラクチャーファイルに対して相対的であったために使用可能でした。そのため、複製されたバージョンは独立して動く事が可能でした("サンドボックス"として使用可能)。しかしながら、アプリケーションパッケージを複製することは異なるアプリケーションを作成するわけではありません。新しいデフォルトデータファイルパスの保存機能の元では、複製されたアプリにおいてもアプリケーションのユーザー設定フォルダを共有することになるので、結果的に同じデータファイルを常に使用する事になります。データファイルの名前を変更した場合でも、アプリケーションが最後に使用したファイルが開かれるので、結果は同じです。

新アーキテクチャーが有効化されたとき特有のシナリオをサポートするため、データファイルパスを**アプリケーションパス**とリンクする事ができます。この場合、データファイルは特定のパスを使用してリンクされるので、単に最後に開かれたファイルとは限りません。データリンクモードはビルドアプリケーションプロセス時に選択可能です。

データがアプリケーション名とリンクされている場合の複製のイメージ:



データがアプリケーションパスとリンクされている場合の複製のイメージ:



## リンクモードの選択肢

使用可能なリンクモードは2つあります。どちらも一長一短ですので、ご自分の目的に適したモードを選択して下さい。

### ● アプリケーション名でリンク(デフォルト)

このモードでは、4Dアプリケーションはストラクチャーファイルに対応したデータファイルの、一番最近開いたものを自動的に開きます。この柔軟で直感的なモードでは、ディスク内でアプリケーションパッケージを自由に移動させる事ができます。アプリケーションを複製したい場合を除き、組み込みアプリではこちらの使用が推奨されます。

### ● アプリケーションパスでリンク

このモードでは、4Dはアプリケーションの`lastDataPath.xml`ファイルを解析し、"executablePath"属性を持つデータファイルのうち、アプリケーションの完全パスに合致するものを開こうとします。そのようなエントリが見つかった場合には、それに対応する("dataFilePath"属性で定義されている)データファイルが開かれます。そうでない場合、最後に開かれたデータファイルが開かれます(デフォルトモード)。

このモードでは、データファイルへのリンクを壊す事なく組み込みアプリを複製する事ができます(上記参照)。

しかしながら、このオプション使用時には、アプリケーションパッケージがディスク上で移動してしまうと、アプリケーションパスは"executablePath"属性に合致しないため、ユーザーはデータファイルの場所を聞かれる事になります(ユーザーがデータファイルを選択後は、`lastDataPath.xml`ファイルは新しいデータファイルに従って更新されます)。

## データリンクモードの変更方法

データリンクモードは"アプリケーションをビルド"ダイアログボックス内か、XML BuildAppキーを通して選択可能です。

- "アプリケーションをビルド"ダイアログボックス内において、**アプリケーション**あるいは**クライアント/サーバ**タブ上でデータリンクモードを選択できるようになりました:



---

---

**注:** これらのオプションは、**アプリケーション配布時に新アーキテクチャーを使用**互換性オプションが選択されている場合にのみ使用可能です(上記参照)。

- または、新しい"LastDataPathLookup"XML BuildAppキーを使用する事もできます。これはスタンドアロン、クライアント-サーバー、両方のアプリケーションにおいて使用可能です:

*Preferences4D/BuildApp/RuntimeVL/LastDataPathLookup:* スタンドアロンアプリケーションの挙動を指定

*Preferences4D/BuildApp/CS/LastDataPathLookup:* サーバーアプリケーションの挙動を指定

これらのキーには、以下の値のどれか一つを渡す事ができます:

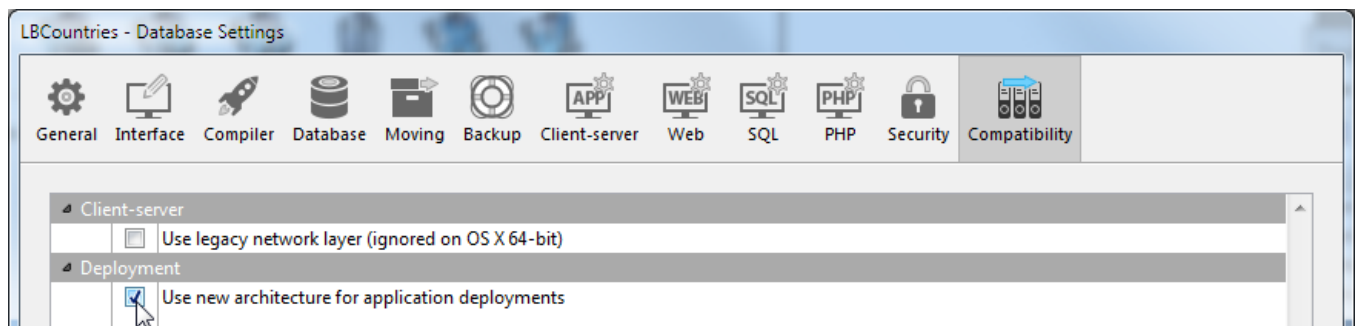
キー値	データファイルパス検索モード	データファイルパスの保管
InDbStruct	ストラクチャーファイル内のパスを使用	ストラクチャーファイル内部(以前と同じ挙動)
ByAppName	<i>LastDataPath.xml</i> に基づいた一番最近のパスを使用	ユーザー設定内
ByAppPath	<i>executablePath</i> がアプリケーションの.exeファイルパスに合致するエントリ	ユーザー設定内

## 組み込みクライアントでの新しい接続プロシージャ

4D v15 R4以降、サーバーへの接続を管理するクライアント側のプロシージャは変更されました。これにより接続エラーの際の適応性が高まり、デベロッパーがよりコントロールをしやすくなりました。

### 機能の有効化(互換性オプション)

この章で説明がなされている機能は、データベース設定ダイアログボックスの"互換性"ページにある**アプリケーション配布に新しいアーキテクチャーを使用**オプションがチェックされているときのみ利用可能です：



- このオプションは、4D v15 R4以降で作成された4Dデータベースではデフォルトでチェックされています。"新アーキテクチャー"は有効化され、他には何も追加で設定する必要はありません。
- 互換性上の理由から、このオプションは以前のバージョンから変換されたデータベースではデフォルトでチェックが外されています。新機能を活用するためにはチェックする必要があります。

### 新しい接続シナリオ

4D v15 R4で作成されたデータベース、または変換されたデータベースで**アプリケーション配布に新しいアーキテクチャーを使用**互換性オプションがチェックされているもの(上記参照)については、専用サーバーが使用不可能な場合に柔軟に対応できるように、組み込みクライアントアプリの接続プロシージャが変更されました。4Dクライアントアプリのスタートアップシナリオは、以下のようになりました：

- クライアントアプリは検索サービスを使用してサーバーに接続しようと試みます(同じsubnet内に公開されたサーバー名に基づいて検索します)。  
OR  
クライアントアプリ内のEnginedServer.4DLinkファイルに有効な接続情報が保存されていた場合、クライアントアプリは指定されたサーバーアドレスへ接続を試みます。
- これが失敗した場合、クライアントアプリケーションは、アプリケーションのユーザー設定フォルダ(lastServer.xmlファイル、詳細は最後の段落を参照)に保存されている情報を使用してサーバーへの接続を試みます。
- これが失敗した場合、クライアントアプリケーションは接続エラーダイアログボックスを表示します。
  - ユーザーが**選択...**ボタンをクリックした場合、標準の"サーバー接続"ダイアログボックスが表示されます(ビルドの段階で4Dデベロッパが許可していた場合に限りです。詳細は後述)。
  - ユーザーが**終了**ボタンをクリックした場合、クライアントアプリケーションは終了します。
- 接続が成功した場合、クライアントアプリケーションは将来の使用のために、その接続情報をアプリケーションのユーザー設定フォルダに保存します。

### 最後に使用したサーバーパスを保存する

4D v15 R4で作成されたデータベース、あるいは変換されたデータベースで**アプリケーション配布に新しいアーキテクチャーを使用**互換性オプションがチェックされているもの(上記参照)については、最後に使用され検証されたサーバーパスが、アプリケーションのユーザー設定フォルダ内のlastServer.xml という名前のファイルに自動的に保存されます。このフォルダは以下の場所に保存されています：

```
userPrefs:=-Get 4D folder(Active 4D Folder)
```

この新規メカニズムは、最初に指定したサーバーが何らかの理由(例えばメンテナンスモードなど)で一時的に使用できなかった場合を想定しています。最初にこういった状態に遭遇したとき、サーバー選択ダイアログボックスを表示し(ただし許可されていた場合に限り、以下参照)ユーザーが他のサーバーを手動で選択できるようにし、その接続が成功した場合にはそのパスが保存されます。それ以降に接続ができなかった場合には、lastServer.xml のパス情報を通して自動的に管理されます。

#### 注:

- クライアントアプリケーションが検索サービスから常に接続できない場合(例えばネットワーク設定に不具合がある場合など)でも、デベロッパーがビルド時にBuildApp.xml ファイル内のIPAddressキーを使用してホスト名を提供する事が推奨されます。この新しい機構はあくまで一時的な接続不可状態の場合を想定しています。
- スタートアップ時にAlt/Option キーを押しながら起動してサーバー接続ダイアログボックスを表示するのは、引き続き全ての場合においてサポートされます。

## エラー時のサーバー選択ダイアログボックス使用の可・不可

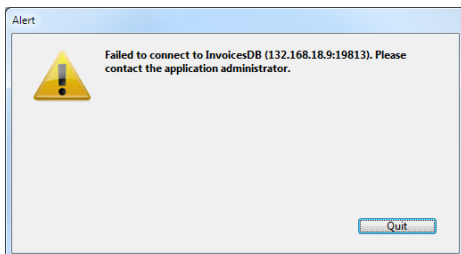
4D v15 R4以降、組み込みクライアントマシンにおいてサーバーに接続が出来なかった場合、標準のサーバー選択ダイアログボックスを自動的に表示しないようにすることが可能になりました。

この場合、設定は、**アプリケーション配布には新アーキテクチャーを使用**互換性オプションと、アプリケーションがビルドされたマシン上の新しいServerSelectionAllowedXMLキーを使用してコントロールすることもできます(次の章を参照)。以下の三つの選択肢があります:

- **エラーメッセージを表示し、サーバー選択ダイアログボックスを表示させない**

4D v15 R4以降で作成されたデータベースのデフォルトの挙動です。アプリケーションは終了する以外の選択肢がありません。この機能は以下の設定で使用する事が可能です:

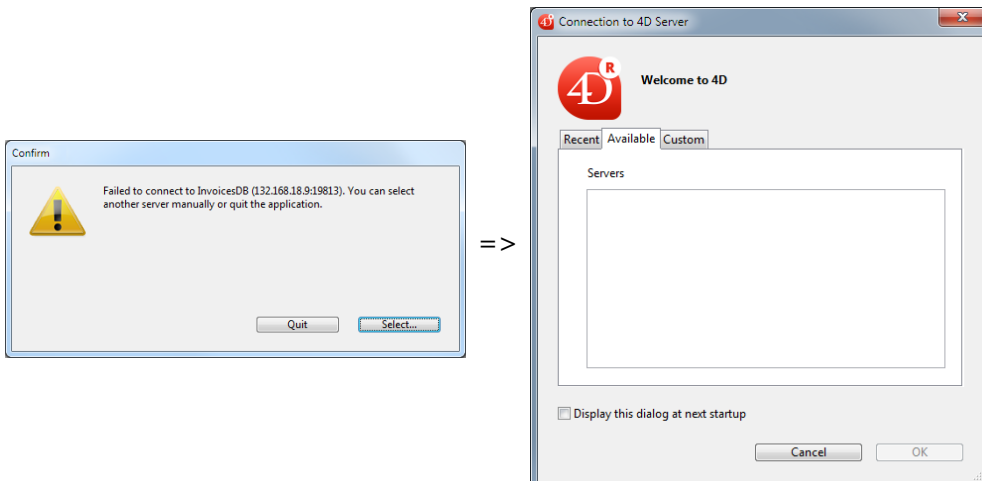
- **アプリケーション配布には新アーキテクチャーを使用**オプション: チェック
- ServerSelectionAllowed XMLキー: False値、またはキーを省略



- **エラーメッセージを表示し、サーバー選択ダイアログボックスへのアクセスを可能にする**

ユーザーは**選択...**ボタンをクリックする事によってサーバー選択ウィンドウへとアクセスできます。この機能は以下の設定で使用する事が可能です:

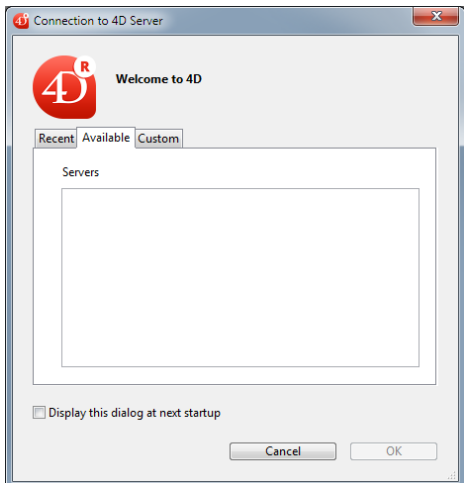
- **アプリケーション配布には新アーキテクチャーを使用**オプション: チェック
- ServerSelectionAllowed XMLキー: True値



- **サーバー選択ダイアログボックスを直接表示する**

変換されたデータベースでのデフォルトの挙動です。これは以前のバージョンの4Dでの振る舞いと同じです。この機能は以下の設定で使用する事が可能です:

- **アプリケーション配布には新アーキテクチャーを使用**オプション: チェック
- `ServerSelectionAllowed` XMLキー: 無視されます



### 新しいXMLキー: `ServerSelectionAllowed`

配布されたアプリケーションでは**新アーキテクチャーを使用**互換性オプションがチェックされている場合、アプリケーションビルダーで使用可能な新しい"`ServerSelectionAllowed`" XMLビルドアプリケーションキーを使用して、組み込みクライアントアプリを設定することができます:


```
/Preferences4D/BuildApp/CS/ServerSelectionAllowed
```

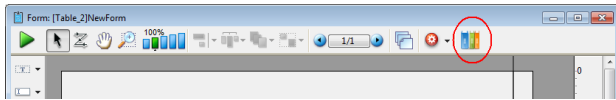
このキーが定義される時:

- **true**に設定された場合、アプリケーションはエラーダイアログボックスを表示し、そこからサーバー選択ウィンドウへとアクセスすることができます。
- **false**に設定された場合、エラーダイアログボックスからサーバー選択ウィンドウへは行けません。

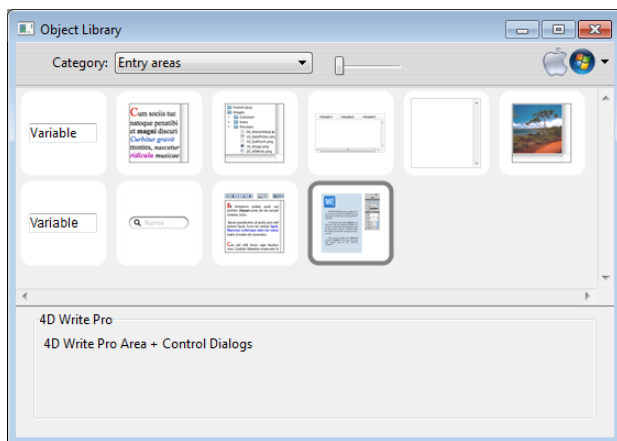
**リマインダ:** 上記の設定に関わらず、**Alt/Option** キーを押しながら起動してサーバー選択ダイアログボックスを表示する機能は、引き続き全ての場合においてサポートされます。

## ■ オブジェクトライブラリ内の新しい4D Write Proフォームオブジェクト

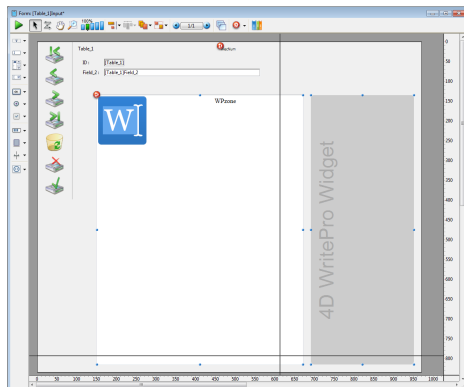
4D v15 R4では、フォームエディターの設定済みオブジェクトライブラリ内に新しい4D Write Proフォームオブジェクトが含まれました。オブジェクトライブラリにアクセスするためには、フォームエディターのツールバー内の  アイコンをクリックして下さい:



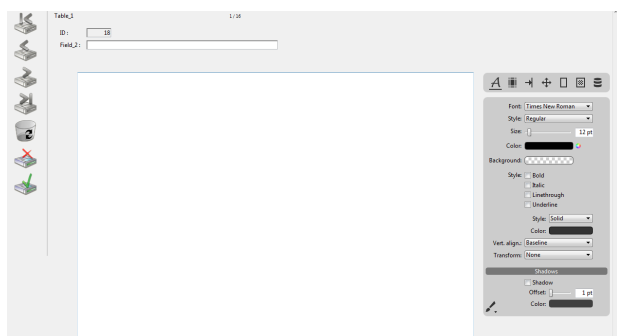
4D Write Proフォームオブジェクトは入力エリアテーマ内にあります:



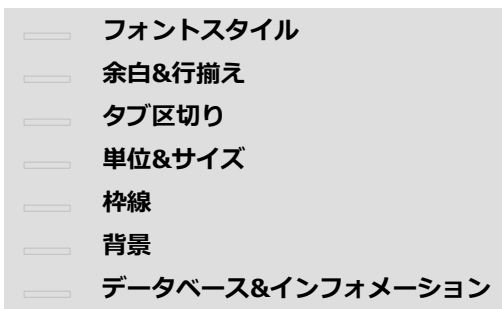
オブジェクトをフォームにドロップすると、設定済みの4D Write Proエリアと、に関連づけされた、エリアを管理するためのコントロールパネルを備えた4D Write Proウィジェットサブフォームを挿入します:




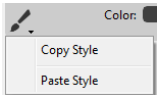
ユーザーモードでは、このパネルにはデフォルトで7つのアイコンが含まれて表示されています:



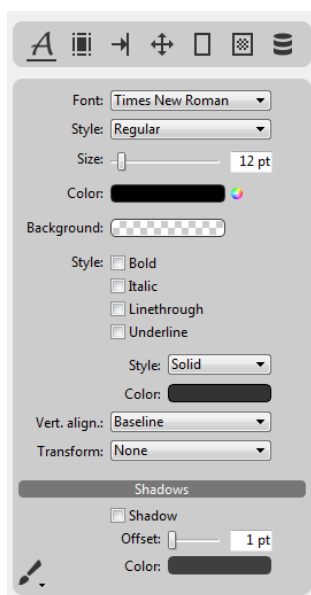
ツールバー内のそれぞれのアイコンを選択する事によって、異なるコントロールパネルが表示されます(以下詳細):



これに加え、**コピー/ペーストボタン**  があり(ほとんどのパネルの左下隅)、パネルの中身に応じた動的なコンテキストメニューが表示されます。例えば**フォントスタイル**パネルでは、スタイル付きテキストのセレクションをコピーした後は、メニューには自動的に"スタイルを貼り付け"アイテムが含まれるようになります:

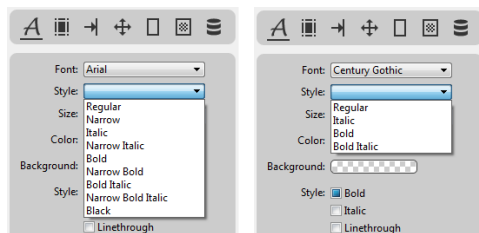


## フォントスタイル



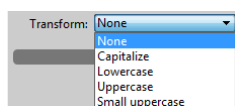
このパネルは4D Write Proエリアのテキストの標準のフォントスタイルとプロパティを管理します。


**スタイル**メニュー内にある項目は選択したフォントに応じて変化します:



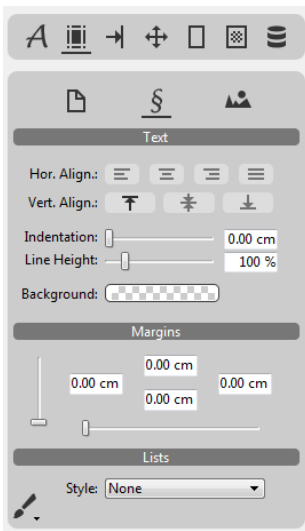
**注:** ドキュメントに設定された単位に関わらず、フォントサイズは常にポイント単位です。

**水平行並び**メニューはテキストを上付き文字あるいは下付き文字にし、**変換**メニューは英字の大文字/小文字の種類を切り替えます:



**コピー/ペーストボタン**  は選択されたテキストに適用されたスタイルをコピーします。このボタンのメニューはテキストの選択範囲をコピーすると自動的に"スタイルを貼り付け"項目を追加するので、その後そのスタイルを好きなところへと適用する事ができます。このメカニズムは、選択したテキストに単一のスタイルが全体に対して適用されている場合にのみ動作するという点に注意して下さい。

## 余白と行揃え

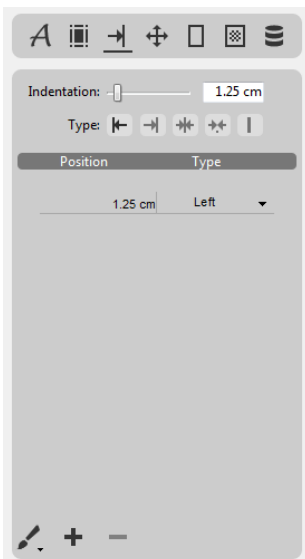


このパネルは4D Write Proエリアの標準のテキストの行揃えプロパティを管理し、余白を設定します。

ドキュメント全体に適用される一般設定に加え、テキストの行揃えと余白は、それぞれの段落やテキスト内のピクチャーに対しても独自に設定可能です。指定したいエリアに対してパネル上部にあるアイコンを使用してこれらの設定を個別に設定することができます(ドキュメント全体には ボタンを、個別の段落に対しては ボタンを、そしてテキスト内の画像に対しては のボタンを使用してください)。

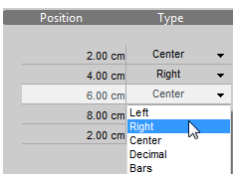
**コピー/貼り付けボタン** を使用すると、選択したテキストのテキスト設定や余白設定などをコピー・貼り付けすることができます。

## タブ区切り




このパネルは4D Write Proエリア内での段落のタブ区切りを管理します。**インデント**に値が(スライドバーを使用して、あるいはエリアに値を直接入力して)設定されていた場合、その値はその後追加されるタブの間隔としてデフォルトで使用されます。その下の**タイプ**ボタンをのいずれかを選択した場合、そのタイプは段落内の全ての既存のタブ区切りに対して適用されます。

新しい**距離**の値をセルに入力するか、または新しい**タイプ**をドロップダウンメニューから選択することによって個別のタブ区切りを手動で変更する事ができます:

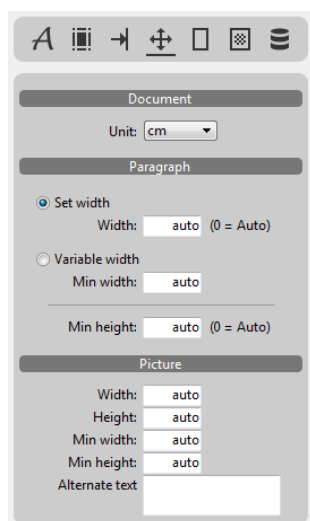


**注:** パネル上部のコントロールを使用して**インデント**または**タイプ**を変更すると、タブの一覧で設定した個別のタブ設定は全て上書きされます。

**+** ボタンをクリックする事で、段落に新しいタブ区切りを追加することができます。タブを一覧から選択して、**-** ボタンをクリックする事でタブを削除する事ができます。

タブ区切りはカレントの段落、あるいは選択した段落に対して適用されます。**コピー/貼り付けボタン**  を使用してタブ区切り設定をコピーし貼り付ける事もできます。

## 単位とサイズ



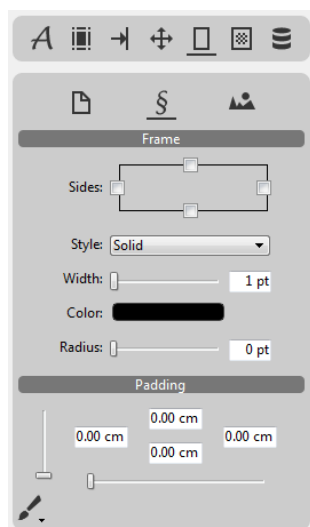
このパネルは4D Write Proドキュメントで使用される標準の単位を設定し、その段落と含まれるピクチャーに対して適用されるサイズを指定します。

単位はドキュメント全体に対して設定されます。**注:** ドキュメントに対して設定された単位とは関係なく、フォントサイズ(**フォントスタイル**パネルを参照)と線幅、および枠線の半径(**枠線**パネルを参照)は常にポイント単位で設定されます。




段落には固定または可変の幅を設定することができ、ピクチャーにも固定のサイズまたは最小の幅/高さを設定する事ができます。サイズが"自動"に設定されている場合、サイズはその要素の中身に基づいて決められます。

**代替テキスト**エリアでは、ピクチャーが表示できないときに使用される代替メッセージを指定する事ができます。

## 枠線




このパネルでは4D Write Proエリアの枠線とパッドを管理します。

枠線はドキュメント全体に対して、あるいは個別の段落に対して、またはテキスト内のピクチャーに対して設定することができます。パネル上部のアイコンに対して、指定したいエリアでのこれらの対象を設定します。つまり、ドキュメントに対しては  ボタンを、個別の段落に対しては  ボタンを、テキスト内のピクチャーに対しては  ボタンを使用します。

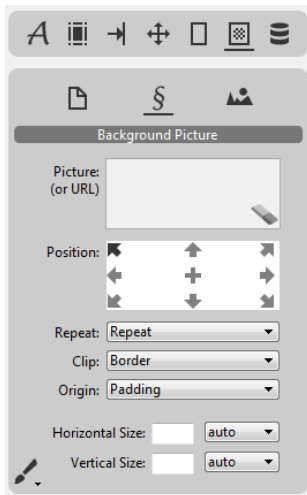
**注:** 枠線スタイルのうち、二重、くぼみ、リッジ、インセットについては、デフォルトの幅(1pt)でははっきりとは見えません。


**半径**を設定すると枠線の角を丸くします。**注:** この設定はくぼみ、リッジ、インセットに対しては設定できません。



**コピー/貼り付けボタン**  はある段落(またはピクチャー)から他の箇所へ、枠線とパッドをコピーし貼り付けます。


## 背景





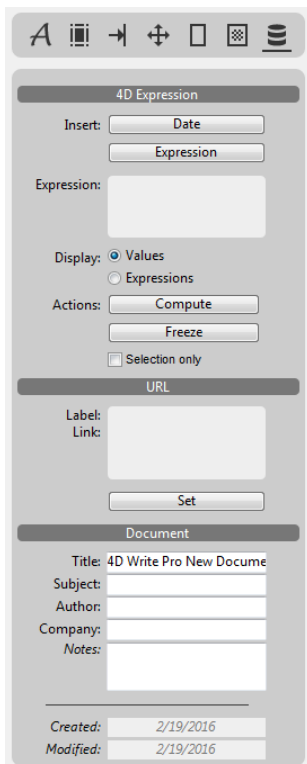
このパネルは、4D Write Proエリアの背景ピクチャーを管理します。**ピクチャー**エリアに、ピクチャーやURL等をドラッグ&ドロップすることができます。また、ピクチャーエリアを右クリックする事でドロップダウンリストが開かれ、デフォルトで使用可能な複数の背景パターンが表示されます。使用されているそれぞれのピクチャーあるいはパターンに対して位置やサイズを設定しカスタムを設定を定義することができます。  アイコンを使用して既存のピクチャーやパターンを取り除くこともできます。

背景ピクチャーはドキュメントレベル(  )に対して、あるいは個別の段落(  )に対して設定する事ができます。

**コピー/貼り付け**ボタン  は背景ピクチャーとともにある段落(あるいはピクチャー)の設定をコピーし他へと貼り付けます。

## データベースと情報

---



このパネルは4D Write Proエリアに4D式とURLを挿入・管理するとともにドキュメントについて有用な情報を入力するエリアを内包します。

以下のコントロールが利用可能です:

- **日付**: カレントの位置にカレントの日付を挿入します。
- **式(ボタン)**: フォーミュラエディターを開き、新しい4D式を作成、または既存の4D式を読み込み、カレントの位置にそれを挿入します。
- **式(エリア)**: 選択された4D式の参照を表示します。
- **値/式の表示**: 4D式を参照として表示するか、そのカレントの値を表示するかを切り替えます。

- **再計算:** ドキュメント内の4D式の値を更新します。
- **評価:** カレントの4D式を標準テキストへと変換します(これは取り消しできません)。
- **選択部分のみ:** このチェックボックスをチェックする事で**再計算**と**評価**のボタンのアクションの対象を4D Write Proエリア内で選択されたテキストに限定します。このボックスがチェックされていない場合、これらのアクションはドキュメント全体にたいして適用されます。
- **ラベル/リンク:** 選択されたURLのリンクアドレスとラベルを表示します。
- **リンク設定:** URL(リンクアドレスとラベル)を入力あるいは編集するダイアログボックスを表示します。

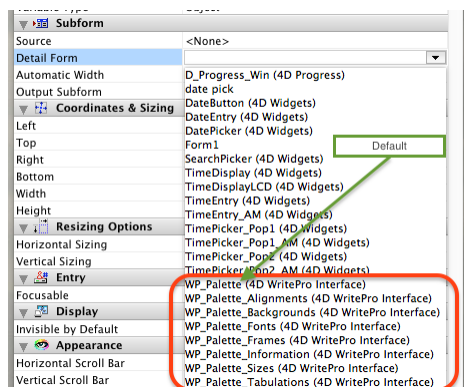
**注:** このコントロールパネルにおいて**4D式**と**URL**のセクションが有効になるためには、4D Write Proエリアにフォーカスが入っている必要があります。

## ドキュメント情報

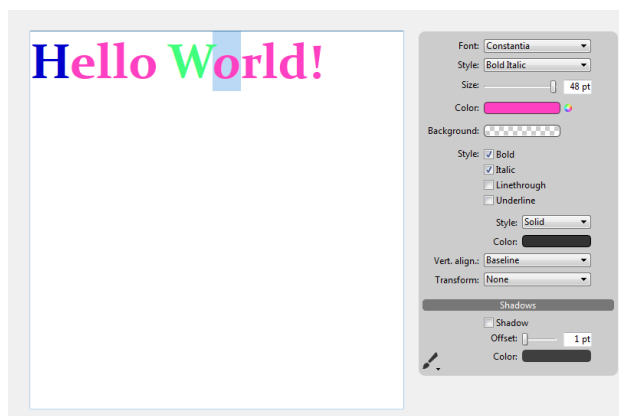
このセクションに入力された情報はドキュメントとともに保存されますが、ここ意外では表示されません。デフォルトでは"タイトル"は"新規4D Write Proドキュメント"となっています。また**ドキュメント情報**セクション内の"作成日"と"変更日"エリアは変更することはできません。

## ウィジェットコントロールパネルを個別に表示する

デフォルトで、4D Write Proフォームオブジェクトには、コントロールパネルのフルセットを内包する4D Write Proウィジェットサブフォームが含まれています。デフォルトのコントロールパネルは"WP\_Palette"詳細フォームです。しかしながら、フォームエディターのプロパティリスト内の適切な詳細フォームを選択する事で、これらのコントロールパネルを個別に使用することができます:



例えば、"WP\_Palette\_Fields"を詳細フォームとして選択した場合、フォーム内に以下のように表示されます:



## ウィジェットコントロールパネルインターフェースのカスタマイズ

コントロールパネルのインターフェースを変える事によって、異なるスキンとフォントを使用する事ができます。そのためには4D Write Proエリアに関連づけられたオブジェクトメソッドを変更するだけです。

変更する4D Write Proエリアのオブジェクト変数が、以下の必須の属性を含んでいるという事を必ず確認して下さい(これらの属性はデフォルトのオブジェクトメソッドには自動的に含まれています):

- **selection:** **WP GET ATTRIBUTES**や**WP SET ATTRIBUTES**などの4D Write Proコマンドで使用されます。
- **areaName:** ST コマンドで使用されます。

これに加え、任意の第三属性("skin")を追加する事でコントロールパネルの見た目をカスタマイズする事が可能になります。

この任意の属性の値はオブジェクトであり(以下の例の中では\$WP\_skinという名前)、以下の(任意の)属性を格納する事が出来ます:

- skinName: 取りうる値は"black"、"dark"、"grey"、"light"、"white"または"night"
- backgroundColor: 例 0x00A0A0A0
- separatorColor: 例 0x00D04060
- fontColor: 例 0x002080C0
- separatorFontColor: 例 0x00803000
- font: フォントファミリー。例 "Times"
- fontSize: 例 12
- scrollbar\*: True または False
  - \*scrollbar 属性は4D Write Proフォームオブジェクトが垂直方向にリサイズ可能な場合(縦方向サイズ変更プロパティが拡張に設定されている場合)にのみ使用可能・有効化可能となります。

4D Write Pro属性についてのより詳細な情報については、**4D Write Pro属性**を参照して下さい。

以下は、ウィジェットコントロールパネルのインターフェースをカスタマイズするコードの一例です:

```
C_OBJECT($WP_skin)












OB SET($WP_skin;"skinName";$skinName) // 取りうる値は"black"、"dark"、"grey"、"light"、"white"、"night"
OB SET($WP_skin;"backgroundColor";0x00A0A0A0) //コントロールパネルの背景カラーを設定します
OB SET($WP_skin;"separatorColor";0x00D04060) //セパレーターエリアの背景色を設定します
OB SET($WP_skin;"fontColor";0x002080C0) //コントロールパネルテキストのフォントカラーを設定します
OB SET($WP_skin;"separatorFontColor";0x00803000) //セパレーターエリアテキストのフォントカラーを設定します

OB SET($WP_skin;"font";"Times") // 使用されるフォントを設定します
OB SET($WP_skin;"fontSize";13) // 使用されるフォントのサイズを設定します

OB SET($WP_skin;"scrollbar";True)

// その後このカスタムのスキンを4D Write Proオブジェクトへと関連づけます
OB SET($WP_object;"skin";$WP_skin)
```

## ランゲージ

-  4D変換タグ
-  Active transaction
-  DISTINCT VALUES
-  Error formulaシステム変数
-  FONT LIST
-  FONT STYLE LIST
-  Get database parameter と SET DATABASE PARAMETER
-  GRAPH
-  INTEGRATE MIRROR LOG FILE
-  LAUNCH EXTERNAL PROCESS
-  LISTBOX Get array、LISTBOX SET ARRAY、LISTBOX GET ARRAYS
-  LOG FILE TO JSON
-  METHOD GET CODE
-  OB GetとOB SET
-  OBJECT Get horizontal alignmentとOBJECT SET HORIZONTAL ALIGNMENT
-  RESUME TRANSACTION
-  SUSPEND TRANSACTION
-  WEB Get session process count
-  トランザクションの一時停止と再開
-  括弧の代わりに4DVARコメントを使用オプションの削除

## 4D変換タグ

4D変換タグは4D v15 R4において改善されました:

- XML-準拠の新しいシンタックスをサポートする事により、有効なXMLテンプレートの使用が可能になりました。
- 新しく導入された"4DCODE"タグにより、4Dコードをブロックとして挿入する事が可能になりました。
- "."(ピリオド)を小数浮動点として使用する事により、コードをリージョン設定の影響を受けないようにしました。

**注:** 廃止予定であった"括弧の代わりに4DVARコメントを使用する"互換性オプションは、サポートされなくなりました(**括弧の代わりに4DVARコメントを使用オプションの削除**を参照して下さい)。

### 4DTEXT、4DHTML、4DEVALでの新しい\$シンタックス

いくつかの既存の4D変換タグは、ドル記号(\$)を使用した新しいシンタックスを用いて表現する事ができるようになりました:  
`<!--#4dtag expression-->`という表記の代わりに`$4dtag (expression)`という表記を使用可能になりました。

#### 必要条件

\$シンタックスは、特定のタグの、特定の評価コンテキストでのみ使用されます:

#### ● タグ

このシンタックスを受け付けるのは、処理された値を返すために使用されていたタグのみです:

- 4DTEXT
- 4DHTML
- 4DEVAL

#### ● 評価

\$シンタックスは評価が明示的である場合に限り、4Dによるタグの評価をトリガーします:

- **PROCESS 4D TAGS** コマンド
- URLを通してWebサーバーによって直接提供される静的な **.shtml** ページ
- 4DINCLUDEタグによって挿入されたページ

それ以外のタグ(4DIF、4DSCRIPTなど)と他の評価コンテキスト(**WEB SEND FILE**、**WEB SEND BLOB**と**WEB SEND TEXT**コマンド)においては、\$シンタックスはサポートされず、通常のシンタックスを使用する必要があります。

#### 使用法

例えば、以下のような書き方の代わりに:

```
<!--#4DEVAL (UserName) -->
```

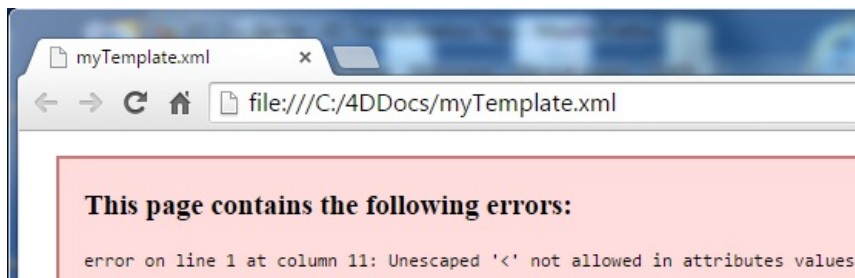
このような書き方が可能になりました:

```
$4DEVAL (UserName)
```

この新しいシンタックスの主な利点は、**XML-準拠のテンプレートを書く事ができる**という点にあります。一部の4Dデベロッパーは標準のXMLパーサーツールを使用してXML-ベースのテンプレートを作成あるいは解析する必要があります。"<"という文字はXML属性値としては無効なため、ドキュメントシンタックスを破ることなく4Dタグの"<!-- -->"シンタックスを使うという事はできませんでした。その一方で、"<"という文字をエスケープすると4Dがタグを正しく解釈しないという問題もありました。

例えば、以下のコードは属性の値の最初の"<"の時点でXML解析エラーを引き起こします:

```
<line x1="<!--#4DEVAL $x-->" y1="<!--#4DEVAL $graphY1-->" />
```



\$シンタックスを用いる事で、以下のコードはパーサーで評価されます:

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

\$4dtag と <!--#4dtag --> は厳密には全く同一のものではないという点に注意して下さい。<!--#4dtag -->とは異なり、\$4dtagは4Dタグを繰り返し解釈することはしません。\$タグは常に一度しか評価されず、その結果は標準テキストとみなされます。

**注:** 再帰的処理についてのより詳細な情報については、[再起的処理](#)の段落を参照して下さい。

この差異は悪意あるコードのインジェクション(注入)を防ぐためです。Language Referenceマニュアルに説明されているように、ユーザーのテキストを管理するためには、タグの不要な再帰的解釈を防ぐために、4DTEXTタグではなく4DHTMLタグの使用が強く推奨されています。4DTEXTでは、"<"などの特殊文字はエスケープされるため、<!--#4dtag expression -->シンタックスを使用する4Dタグはその特定の意味を失ってしまいます。しかしながら、4DTEXTは\$記号はエスケープしないため、悪意あるコードのインジェクションを防ぐため、\$4dtag (expression)シンタックスを使用する際の再帰的処理のサポートを止める事になりました。

以下の例は使用されるタグとシンタックスの処理による結果の違いを表しています:

```
// 例 1
myName:="<!--#4DHTML QUIT 4D-->" //悪意あるコードの注入
input:="My name is: <!--#4DHTML myName-->"
PROCESS 4D TAGS(input;output)
//4Dは終了してしまいます!
```

```
// 例 2
myName:="<!--#4DHTML QUIT 4D-->" //悪意あるコードの注入
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
//出力は "My name is: &lt;!--#4DHTML QUIT 4D--&gt;"
```

```
// 例 3
myName:="$4DEVAL(QUIT 4D)" //悪意あるコードの注入
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
//出力は"My name is: $4DEVAL(QUIT 4D)"
```

\$4dtagシンタックスは閉じる括弧と引用符のマッチングペアをサポートするという点に注意して下さい。例えば、以下の複雑な(非現実的な)文字列を評価する必要がある場合を考えます:

```
String(1) + "\"(hello)\""
```

この場合、以下のように書く事ができます:

```
input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"")"
PROCESS 4D TAGS(input;output)
-->1"(hello)"
```

## 4DCODE (新しいタグ)

## シンタックス: <!--#4DCODE 4Dcode-->

新しい4DCODEタグを使用するとテンプレート内に複数行の4Dコードのブロックを挿入する事ができます。

"<!--#4DCODE"シーケンスとそれに続くスペース、CRまたはLF文字が検知されたとき、4Dは次の"-->"シーケンスまでのコードの全ての行を解釈します。コードブロック自身はキャリッジリターン、ラインフィード、あるいはその両方を含む事ができ、それらは4Dによってシーケンシャルに解釈されます。

例えば、4DCODEタグを使用して、テンプレート内に以下のように書く事ができます:

```
<!--#4DCODE
//PARAMETERS 初期化

$graphType:=1
if (OB Is defined:C1231($graphParameters;"graphType"))
  $graphType:=OB GET:C1224($graphParameters;"graphType")
  if ($graphType=7)
    $nbSeries:=1
    if ($nbValues>8)
      DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}->;9;100000)
      $nbValues:=8
    end if
  end if
end if
end if
-->
```

4DCODEタグの機能は以下の通りです:

- **TRACE**コマンドはサポートされていて、4Dデバッガーを起動するので、お使いのテンプレートのコードをデバッグする事ができます。
- エラーが起きると標準のエラーダイアログボックスが表示され、ユーザーはコードの実行を中止するかでバグモードに入るかを選択する事ができます。
- <!--#4DCODE と -->の間のテキストはどれも複数行に分割され、どのような行末記号(cr、cf、またはcrlf)も受け付けます。
- テキストは**PROCESS 4D TAGS**コマンドを呼び出したデータベースのコンテキスト内でトークナイズされます。これは例えばプロジェクトメソッドの認識において重要です。  
**注:** "4DタグとURLの4DACTION経由で利用可"メソッドプロパティはここでは考慮されません(以下の**セキュリティに関する注意**も参照して下さい)。
- 4Dコードは常にEnglis-US言語を使用して書かなければなりません。そのため、4DCODEは4D言語の"リージョンシステム設定を使用"ユーザー設定を無視します(**コマンドと定数のためのランゲージ**を参照して下さい)。
- テキストが常にEnglisu-US言語を使用しているとしても、4Dのバージョン間でのコマンドや定数の改名に起因する問題を選けるため、コマンド名、定数名ではトークンシンタックス(:Cxxx)を使用する事が推奨されます。  
**注:** :Cxxxシンタックスについてのより詳細な情報については、**フォーミュラ内でのトークンの使用**の章を参照して下さい。

**セキュリティに関する注意:** 4DCODEタグがいかなる4Dランゲージコマンドやプロジェクトメソッドも呼び出す事もできるという点は、特にデータベースがHTTP経由で利用可能な場合にセキュリティ上の問題となり得ます。しかしながら、これらは独自のテンプレートファイル内からサーバー側のコードを実行するため、タグそのものがセキュリティ上の問題を表しているわけではありません。このコンテキストにおいては、どのようなWebサーバーにおいても、セキュリティは主にサーバーファイルへのリモートアクセスのレベルにおいて管理されています。

## "." を浮動小数点として使用

v15 R4以降、4Dタグ(4DTEXT、4DVAR、4DHTML、4DHTMLVAR、そして4DEVAL)を使用して数値表現を評価する際には、4Dは常にピリオド文字(.)を浮動小数点として使用します。リージョン設定は今後使用されません。

この新機能はコードのメンテナンスと、4D言語間・バージョン間での互換性の問題の解決を容易にします。

例えば、以下のコードはどのようなリージョン設定にも依存しません:

```
value:=10/4
input:="<!--#4DTEXT value-->"
PROCESS 4D TAGS(input;output)
// 例リージョン設定が", "(カンマ)を小数点として使用していた場合でも、常に2.5を出力します
```


**互換性に関する注意:** コードが、リージョン設定に即して4Dタグを使用した数値表現を評価する場合、**String**コマンドを使用してコードを適応させる必要があります:

- ピリオドを浮動小数点と使用した`value`を取得するためには: `<!--#4DTEXT value-->`
- リージョン設定に基づいた浮動小数点を使用した`value`を使用するためには: `<!--#4DTEXT String(value)-->`



## ⚙️ Active transaction

Active transaction -> 戻り値

引数	型	説明
戻り値	ブール	 カレントトランザクションが停止されている場合にはFalseを返します

### 説明

---

#### テーマ: トランザクション

**Active transaction** コマンドは、カレントプロセスがトランザクション中であり、かつそのトランザクションが停止されていない場合に **True** を返します。カレントトランザクションがない場合、あるいはカレントトランザクションが停止されている場合には **False** を返します。トランザクションは **SUSPEND TRANSACTION** コマンドを使用して一時停止する事ができます。

このコマンドはカレントプロセスがトランザクション中でない場合にも **False** を返すので、**In transaction** コマンドを使用してプロセスがトランザクション中であるかどうかをチェックする必要もあるかもしれません。

より詳細な情報については、[停止したトランザクションとプロセスのステータス](#)の章を参照して下さい。

### 例題

---

カレントトランザクションのステータスを調べたい場合を考えます:

```
If(In transaction)
  If(Not(Transaction active))
    ALERT("カレントトランザクションは停止されています")
  Else
    ALERT("カレントトランザクションはアクティブです")
  End if
Else
  ALERT("トランザクション中ではありません")
End if
```

`DISTINCT VALUES ( aField ; array{; countArray} )`

引数	型	詳細
<code>aField</code>	フィールド	-> データとして使用するインデックス可能なフィールド
<code>array</code>	配列	<- フィールドのデータを受信する配列
<code>countArray</code>	倍長整数配列あるいは実数配列	<- それぞれの値の数を受け取る配列

## 説明

**DISTINCT VALUES** コマンドは新しい任意の引数、`countArray`を受け取るようになりました。

この引数を渡した場合、`countArray`引数には、`aField`で指定したフィールド内のそれぞれの重複しない値について、カレントセクション内で検知されるオカレンス数が返されます。この引数は自動的に`array`引数内の要素の数に合わせたサイズになります。例えば、"A"、"B"、そして"A"という3つのレコードを含むセクションに対しては、`array`引数には{A;B}が含まれ、`countArray`には{2;1}が含まれます。

`countArray`引数には、倍長整数配列か実数配列のいずれかを渡す事ができます。

**注:** `countArray`引数は、テキストフィールドまたはキーワードインデックスと関連づけられているピクチャーフィールドはサポートしません(このコンテキストで使用した場合、空で返されます)。

## 例題

統計を計算するために、フィールド内の固有の値の個数で降順に並べ替えをしたい場合を考えます:

```
ARRAY TEXT($ _issue_type;0)
ARRAY LONGINT($ _issue_type_instance;0)
DISTINCT VALUES([Issue]iType;$ _issue_type;$ _issue_type_instances)
SORT ARRAY($ _issue_type_instances;$ _issue_type;<)
```

## 📄 Error formulaシステム変数

---

**ON ERR CALL** コマンドによって実装されたエラーキャッチメソッドにて新しいテキストシステム変数、**Error formula**が使用可能になりました。これは既存のエラー関連のシステム変数、*Error*、*Error method*、そして*Error line*に続くものです。

この新しい変数は、デベロッパーがエラーの元を、早く、シンプルに特定できるようにするものです。エラーが発生すると、**Error formula**システム変数には、エラーを引き起こした4Dコードが(標準テキストとして)代入されます。このフォーミュラのテキストはカレントの4Dプログラミング言語で表現されます。

**注:** エラーを引き起こしたソースコードが見つからない場合、**Error formula**変数には空の文字列が含まれます。これは、以下の様な場合に起き得ます:

- ソースコードが4Dアプリケーションビルダーによってコンパイルされたストラクチャーから削除されてしまっている
- ソースコードは残っているが、データベースは**範囲をチェック**オプションをつけずにコンパイルされてしまっている

### 例題

---

以下のエラーハンドリングメソッドはユーザー割り込みを無視して、エラーテキストを表示します:

```
//Show_only_errors プロジェクトメソッド
//ON ERR CALLに実装されている必要があります
If(Error#1006) //ユーザー割り込みではない
    ALERT("Error "+String(Error)+" が発生しました。発生したコード部分は: \""+Error formula+"\"")
End if
```

FONT LIST ( fonts {; listType | \*} )

引数	型	詳細
fonts	テキスト配列	<- フォント名の配列( <b>Windows上ではスケーラブルなフォントのみ</b> )
listType   *	倍長整数、演算子	-> OS X上で、返すフォントタイプのリストまたは*指定時には返すフォント名

## 説明

**FONT LIST** コマンドはWindows上において、*fonts* 引数にスケーラブルなフォントのみを返すようにアップデートされました。

Windows上では、スケーラブルでないフォント(例えばビットマップフォント)は、古いテクノロジーに基づいておりサイズ変化において限度があり問題となりうる事から、デザインインターフェースでの使用は推奨されていません。4D Write Pro エリアなどの4Dの最新機能ではこれらはサポートされていません。デベロッパーがインターフェースにおいて現代的なフォントのみを選択できるように、**FONT LIST** コマンドはスケーラブルでないフォントは返さないようになりました。"trueType"または"openType"スケーラブルフォントのみが返されます。例えば、"ASI\_Mono"、"MS Sans Serif"、"System"フォントは表示されなくなっています。

これに加えて、GDI名も表示されないようになり、DirectWriteフォントファミリー名前のみがサポートされるようになりました。例えば、"Arial Black"、"Segoe UI Black"フォントファミリーはリストには含まれず、"Arial"と"Segoe"のみが返されるようになります。

ビットマップフォントは、4Dフォームにおいて引き続きご利用いただけます(ただし4D Write Proエリアは除く)。このコマンドで返される一覧からは除外されているだけということです。しかしながら、将来のバージョンにおける4DとWindowsでの互換性の確保のためには、DirectWriteフォントファミリーを使用することが推奨されます。

**互換性に関する注意:** Windows上ではビットマップフォントは*fonts* 引数からフィルターされているため、以前のリリースと比較すると4D v15 R4以降のアプリケーションで返されるリストは異なります。このコマンドを使用してスケーラブルでないフォントを選択していた場合には、必ずコードを修正するようにして下さい。

このアップデートはOS Xには関係しません。OS Xでは、ビットマップフォント(QuickDraw)はOS X 10.4の時に廃止予定となっているからです。

FONT STYLE LIST ( fontFamily ; fontStyleList ; fontNameList )

引数	型	説明
fontFamily	文字	→ フォントファミリー名
fontStyleList	テキスト配列	← fontFamilyによってサポートされるフォントスタイルの一覧
fontNameList	テキスト配列	← fontFamilyによってサポートされているフォント名の完全な一覧

## 説明

新しい**FONT STYLE LIST**コマンドはfontFamily引数で定義されたフォントファミリーによってサポートされるフォントスタイルの一覧と、フォント名の完全な一覧を返します。このコマンドを使用すると(特に4D Write Proエリアのコンテキストにおいて)フォントとフォントスタイルを管理するインターフェースをデザインすることができます。

fontFamily引数には、サポートされているフォントスタイルとフォント名を知りたいフォントファミリーの名前を渡します。fontStyleList引数には、fontFamily引数で渡したフォントファミリーによってサポートされているフォントスタイルの一覧を受け取るテキスト配列を渡します。スタイルはそれらのローカライズされた名前で返される("Italic"要素はスペイン語では"Ítálico"と返されます)ので、例えばローカライズされた"スタイル"ポップアップメニューをビルドすることもできます。

fontNameList引数には、fontFamily引数で渡したフォントファミリーによってサポートされているフォント名の完全な一覧を受け取るテキスト配列を渡します。fontStyleList配列とは異なり、fontNameList配列が返す値はローカライズされていません。つまりフォント名はシステムの認識に基づきます。そのため、フォント名はシステム言語の影響を受けません。この配列の要素は文字列であり、**WP SET ATTRIBUTES** コマンドにてwk font 4D Write Pro属性で使用される事を想定しています。この機能を使用する事によって、4D Write Proドキュメントはフォント名を保存し、使用するシステム言語に関係なくどのマシンでもフォントの問題を起こす事なく開くことができます。

fontFamily引数に渡したフォントがマシン上に見つからない場合、空の配列が返されます。マシン上で使用可能なフォントファミリーの一覧を取得するためには、**FONT LIST**コマンドを使用して下さい(注: **FONT LIST**は4D v15 R4においてWindows版でアップデートされました)。

## 例題

"Verdana"フォントファミリー(使用可能であれば)のスタイルを選択したい場合を考えます:

```

ARRAY TEXT ($aTFonts;0)
ARRAY TEXT ($aTStyles;0)
ARRAY TEXT ($aTNames;0)
C_LONGINT ($numStyle)

FONT LIST ($aTFonts)
$numStyle:=Find in array ($aTFonts;"Verdana")
If ($numStyle#0)
    FONT STYLE LIST ($aTfont{$numStyle};$aTStyles;$aTNames)
End if

//一例として、返される配列は以下の様なものになります:
// $aTStyles{1}="Normal"
// $aTStyles{1}="Italic"
// $aTStyles{1}="Bold"
// $aTStyles{1}="Bold Italic"

// $aTNames{1}="Verdana"
// $aTNames{1}="Verdana Italic"
// $aTNames{1}="Verdana Bold"
// $aTNames{1}="Verdana Bold Italic"

```

## 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* 引数を使用できるようになりました:

定数	型	値
Number of formulas in cache	倍長整数	92

- **スコープ:** 4Dアプリケーション
- **異なるセッション間で値を保持:** No
- **詳細:** **EXECUTE FORMULA** コマンドによって使用される、フォーミュラのキャッシュに保存されるフォーミュラの最大数を設定あるいは取得します。この上限値は全てのプロセスに対して適用されますが、それぞれのプロセスは独自のフォーミュラキャッシュを持ちます。キャッシュされたフォーミュラは一度しかトークナイズされないため、フォーミュラをキャッシュすることによりコンパイルモードでの **EXECUTE FORMULA** コマンドの実行が速くなります(詳細な情報に関しては、**EXECUTE FORMULAのコンパイルモードでのキャッシュ**を参照して下さい)。キャッシュ値を変更したとき、たとえ新しいサイズが以前の値より大きくても、既存の中身はリセットされます。キャッシュ内のフォーミュラ数が上限に達したとき、新たに実行されたフォーミュラは、キャッシュ内の一番古いものを上書きします(FIFOモード)。このパラメータはコンパイルされたデータベースかコンパイルされたコンポーネントにおいてのみ適用されます。
- **取り得る値:** 正の倍長整数
- **デフォルト値:** 0 (キャッシュなし)

GRAPH ( graphPicture ; graphNumber | **graphSettings** ; xLabels {; yElements} {; yElements2 ; ... ; yElementsN} )

引数	型	詳細
graphPicture	ピクチャー変数	- > ピクチャー変数
graphNumber   <b>graphSettings</b>	倍長整数   <b>Object</b>	- > 倍長整数: グラフの型番号、 <b>Object: グラフ設定をプロパティ値として渡す</b>
xLabels	配列	- > x軸のラベル
yElements	配列	- > グラフに描画するデータ(最大8つまで可能)

## 説明

**注:** この新機能は4D 64-bit版でのみご利用いただけます(Windows用 4D Server 64-bit版は除く)。**4D Developer Edition 64-bit版(プレビュー版)**も参照して下さい。

**GRAPH**コマンドは第二引数(*graphSettings*)にObject型の値を受け付けるようになりました。

新しいシンタックスを使用すると、グラフタイプに加えてその詳細な設定(凡例、x軸の最小値など)を単一の呼び出しで定義できるようになります。以前のシンタックス(*graphNumber*引数で番号を第二引数として渡す)の場合には、これらの設定を定義するためには**GRAPH SETTINGS**コマンドをもう一度呼び出す必要がありました(\*)。これにより、ユーザーは生成されたグラフを通常のSVGピクチャーとして保存し、Firefox、Chrome、IEまたはSafariなどの標準のブラウザで表示できるようになりました(生成されたグラフはブラウザに実装されたSVGスタンダードにより準拠しています)。

これに加え、新しいシンタックスでは様々な特殊な設定にアクセスし、例えばバーの間の距離、マージン、バーのカラーなどをカスタマイズできるようになりました。

(\*) *graphSettings* objectシンタックスを使用した場合には**GRAPH SETTINGS**コマンドは呼び出してはいけません。

*graphSettings*引数には、グラフで定義したい様々なプロパティを含んだオブジェクトを渡します。ここでは"**Graph Parameters**"定数テーマ内にある以下の定数を使用する事ができます:

定数	型	値	コメント
Graph background color	文字列	graphBackgroundColor	<b>とりうる値:</b> SVG で使用可能な色名やRGB値 (テキスト)。例: "#7F8E00", "Pink", "#0a1414"
Graph background opacity	文字列	graphBackgroundOpacity	<b>とりうる値:</b> 0-100 の整数 <b>デフォルトの値:</b> 100
Graph background shadow color	文字列	graphBackgroundShadowColor	<b>とりうる値:</b> SVG で使用可能な色名やRGB値 (テキスト)。例: "#7F8E00", "Pink", "#0a1414"
Graph bottom margin	文字列	bottomMargin	<b>とりうる値:</b> 実数 <b>デフォルトの値:</b> 12
Graph colors	文字列	colors	<b>とりうる値:</b> テキスト配列。各グラフシリーズのカラー。 <b>デフォルト値:</b> Blue-green (#19BAC9), Yellow (#FFC338), Purple (#573E82), Green (#4FA839), Orange (#D95700), Blue (#1D9DF2), Yellow-green (#B5CF32), Red (#D43A26)
Graph column gap	文字列	columnGap	<b>とりうる値:</b> 倍長整数 <b>デフォルト値:</b> 12 棒の間の空白を設定します。
Graph column width max	文字列	columnWidthMax	<b>とりうる値:</b> 実数 <b>デフォルトの値:</b> 200
Graph column width min	文字列	columnWidthMin	<b>とりうる値:</b> 実数 <b>デフォルトの値:</b> 10
Graph default height	文字列	defaultHeight	<b>とりうる値:</b> 実数 <b>デフォルトの値:</b> 400。graphType=7 (円グラフ) の場合は 600。
Graph default width	文字列	defaultWidth	<b>とりうる値:</b> 実数 <b>デフォルトの値:</b> 600。graphType=7 (円グラフ) の場合は 800。
Graph display legend	文字列	displayLegend	<b>とりうる値:</b> ブール <b>デフォルトの値:</b> True
Graph document background color	文字列	documentBackgroundColor	<b>とりうる値:</b> SVG準拠のカラー表現(テキスト)、例えば"#7F8E00", "Pink", あるいは "#0a1414"など。 グラフとして保存されているSVGピクチャーが他で開かれた場合、ドキュメントの背景カラーはSVGレンダリングエンジンがSVG <i>tiny 1.2</i> 標準をサポートする場合に限り表示されます (IE、FirefoxではサポートされていますがChromeではサポートされていません)。
Graph document background opacity	文字列	documentBackgroundOpacity	<b>とりうる値:</b> 0-100の間の整数値(デフォルトの値: 100)。SVGとして保存されているグラフが他で開かれた場合、ドキュメントの背景の透明度は、SVGレンダリングエンジンが <i>SVG tiny 1.2</i> をサポートする場合に限り表示されます (IE、Firefox、ではサポートされますがChromeではサポートされません)。
Graph font color	文字列	fontColor	<b>とりうる値:</b> SVG で使用可能な色名やRGB値 (テキスト)。例: "#7F8E00", "Pink", "#0a1414"
Graph font size	文字列	fontSize	<b>とりうる値:</b> 倍長整数値 <b>デフォルトの値:</b> 12。graphType=7 (円グラフ)の場合、はGraph pie font sizeを参照して下さい。



Graph left margin	文字列	leftMargin	とりうる値: 実数 デフォルトの値: 12
Graph legend font color	文字列	legendFontColor	とりうる値: SVG で使用可能な色名やRGB値 (テキスト)。 例: "#7F8E00", "Pink", "#0a1414"
Graph legend icon gap	文字列	legendIconGap	とりうる値: 実数値 デフォルトの値: $\text{Graph legend icon height}/2$
Graph legend icon height	文字列	legendIconHeight	とりうる値: 実数 デフォルトの値: 20
Graph legend icon width	文字列	legendIconWidth	とりうる値: 実数 デフォルトの値: 20
Graph legend labels	文字列	legendLabels	とりうる値: テキスト配列。ない場合、4Dはテキストなしのアイコンを表示します。
Graph line width	文字列	lineWidth	とりうる値: 実数 デフォルトの値: 2
Graph pie font size	文字列	pieFontSize	とりうる値: 実数 デフォルトの値: 16
Graph pie shift	文字列	pieShift	とりうる値: 実数 デフォルトの値: 8
Graph plot height	文字列	plotHeight	とりうる値: 実数 デフォルトの値: 12
Graph plot radius	文字列	plotRadius	とりうる値: 実数 デフォルトの値: 12
Graph plot width	文字列	plotWidth	とりうる値: 実数 デフォルトの値: 12
Graph right margin	文字列	rightMargin	とりうる値: 実数 デフォルトの値: 2
Graph top margin	文字列	topMargin	とりうる値: 実数 デフォルトの値: 2
Graph type	文字列	graphType	とりうる値: 倍長整数 [1から8] 1 = 棒グラフ, 2 = 比率棒グラフ, 3 = 積み上げ棒グラフ, 4 = 線グラフ, 5 = 面グラフ, 6 = 点グラフ, 7 = 円グラフ, 8 = ピクチャーグラフ デフォルトの値: 1  nullの場合、グラフは描画されず、エラーメッセージが表示されます。範囲外の場合も、グラフは描画されず、エラーメッセージが表示されます。  ピクチャータイプのグラフ(値=8)の場合、使用されるピクチャーは以下のフォルダに保存されていなければなりません: 4D/Resources/GraphTemplates/Graph_8_Pictures/

ピクチャーの名前には特にパターンはありません。4Dはフォルダ内に含まれるファイルを並べ替えし、最初のファイルを最初のグラフに割り当てます。これらのファイルはSVGまたは画像タイプのファイルが使用可能です。

Graph  
xGrid  
文字列

**とりうる値:** ブール値  
**デフォルト値:** True  
4番か6番のプロポーショナルなタイプにのみ使用されます。

Graph  
xMax  
文字列

**とりうる値:** 数値、日付、時間(*xLabels* 引数と同じ型です)  
グラフ上ではxMaxより低い値のみが描画されます。xMaxは4、5、またはxPop=trueである6のグラフタイプに対してのみ、*xLabels*引数が数値、日付、時間のいずれかの型であった場合にのみ使用されます。これがない場合、あるいはxMin>xMaxであった場合、4Dは自動的にxMaxの値を計算します。

Graph  
xMin  
文字列

**とりうる値:** 数値、日付、時間(*xLabels* 引数と同じ型です)  
グラフ上ではxMinより高い値のみが描画されます。xMinは4、5、またはxPop=trueである6のグラフタイプに対してのみ、*xLabels*引数が数値、日付、時間のいずれかの型であった場合にのみ使用されます。これがない場合、あるいはxMin>xMaxであった場合、4Dは自動的にxMinの値を計算します。

Graph  
xProp  
文字列

**とりうる値:** ブール値  
**デフォルトの値:** False  
x軸が比例する場合にはTrue、標準のx軸の場合にはFalseを返します。xPropはグラフタイプ4,5,6に対してのみ使用されます。

Graph  
yGrid  
文字列

**とりうる値:** ブール  
**デフォルトの値:** True

Graph  
yMax  
文字列

**とりうる値:** 数値  
ない場合、4Dは自動的にyMaxの値を計算します。

Graph  
yMin  
文字列

**とりうる値:** 数値  
ない場合、4Dは自動的にyMinの値を計算します。

## 例題 1

*graphSettings*を使用するシンタックス: 以下の例では、時間の値に基づいた、シンプルな線グラフを描画する場合を考えます:

```
C_PICTURE (vGraph) //グラフ変数
ARRAY TIME (X;3) //x軸の配列を作成
X{1}:=?05:15:10? //X ラベル #1
X{2}:=?07:15:10? //X ラベル #2
X{3}:=?12:15:55? //X ラベル #3

ARRAY REAL (A;3) //y軸の配列を作成
A{1}:=30 //何かデータを挿入
A{2}:=22
A{3}:=50

ARRAY REAL (B;3) //y軸の配列をもう一つ作成
B{1}:=50 //何かデータを挿入
B{2}:=80
B{3}:=10

C_OBJECT (vSettings) //グラフ設定を初期化

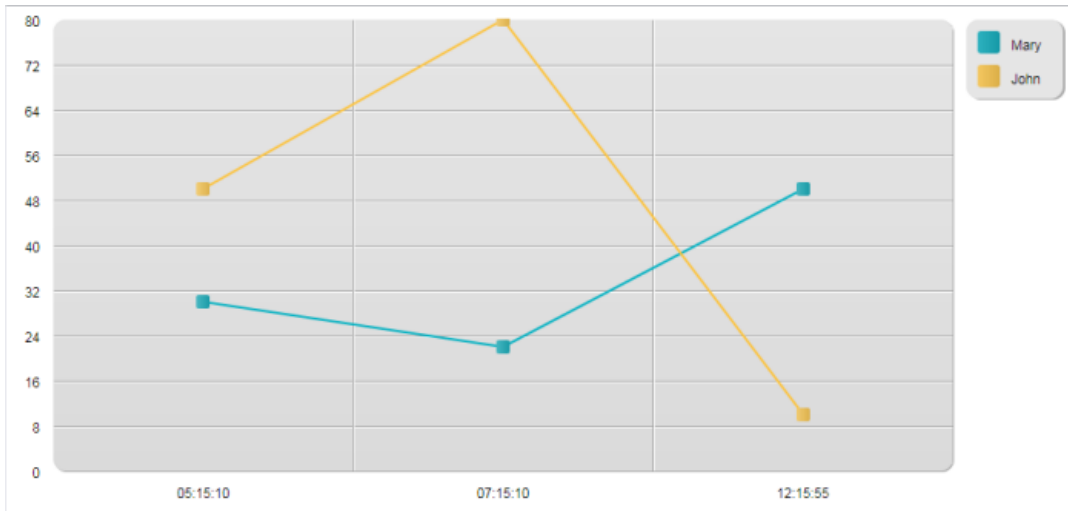
OB SET (vSettings;Graph_type;4) //線タイプ
```

```

ARRAY TEXT (aLabels;2) //グラフの凡例を設定
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY (vSettings;Graph_legend_labels;aLabels)

GRAPH (vGraph;vSettings;X;A;B) //グラフを描画

```



## 例題 2

同じ値を使いながら、カスタムを設定を追加することで異なるビューを得ることができます:

```

C_PICTURE (vGraph) //グラフ変数
ARRAY TIME (X;3) //X軸の配列を作成
X{1}:=?05:15:10? //X ラベル #1
X{2}:=?07:15:10? //X ラベル #2
X{3}:=?12:15:55? //X ラベル #3

ARRAY REAL (A;3) //Y軸の配列を作成
A{1}:=30 //何かデータを挿入
A{2}:=22
A{3}:=50

ARRAY REAL (B;3) //Create another array for the y-axis
B{1}:=50 //何かデータを挿入
B{2}:=80
B{3}:=10

C_OBJECT (vSettings) //グラフ設定を初期化

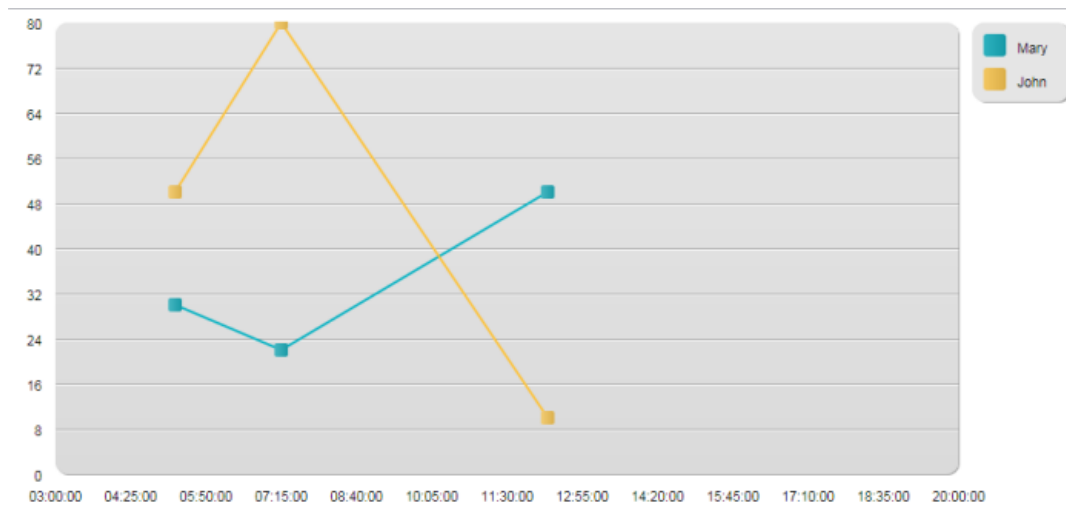
OB SET (vSettings;Graph_type;4) //線型

ARRAY TEXT (aLabels;2) //グラフの凡例を設定
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY (vSettings;Graph_legend_labels;aLabels)

//options
OB SET (vSettings;Graph_xProp;True) //プロポーショナルに設定
OB SET (vSettings;Graph_xGrid;False) //垂直グリッドを除去
OB SET (vSettings;Graph_xMin;?03:00:00?) //境界線を定義
OB SET (vSettings;Graph_xMax;?20:00:00?)

GRAPH (vGraph;vSettings;X;A;B) //グラフを描画

```



### 例題 3

この例では、いくつかの設定をカスタマイズする場合を考えます:

```

C_PICTURE (vGraph) //グラフ変数
ARRAY TEXT (X;5) //X軸の配列を作成
X{1}:="Monday" //X ラベル #1
X{2}:="Tuesday" //X ラベル #2
X{3}:="Wednesday" //X ラベル #3
X{4}:="Thursday" //X ラベル #4
X{5}:="Friday" //X ラベル #5

ARRAY LONGINT (A;5) //Y軸の配列を作成
A{1}:=30 //何かデータを挿入
A{2}:=22
A{3}:=50
A{4}:=45
A{5}:=55

ARRAY LONGINT (B;5) //Y軸の配列をもう一つ作成
B{1}:=50 //何かデータを挿入
B{2}:=80
B{3}:=10
B{4}:=5
B{5}:=72

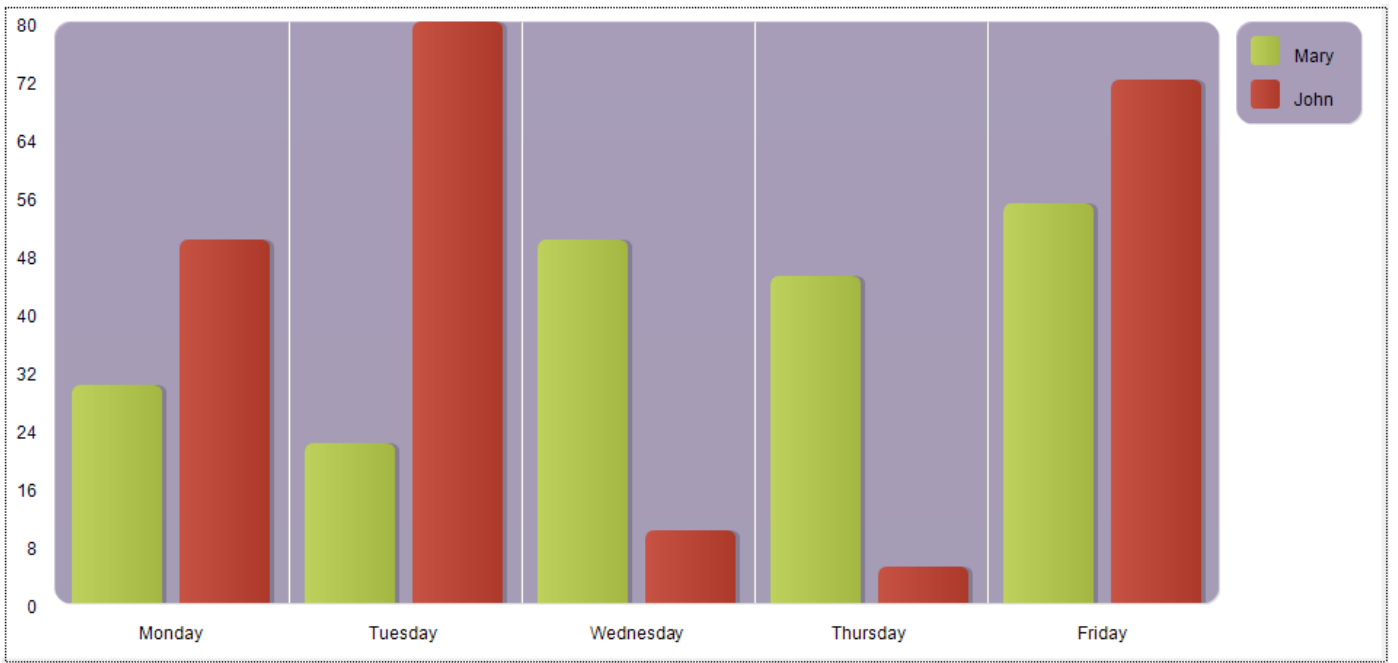
C_OBJECT (vSettings) //グラフ設定を初期化

OB SET (vSettings;Graph_type;1) //バー型

ARRAY TEXT (aLabels;2) //グラフの凡例を設定
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY (vSettings;Graph_legend_labels;aLabels)

//オプション
OB SET (vSettings;Graph_vGrid;False) //垂直グリッドを除去
OB SET (vSettings;Graph_background_color;"#573E82") //背景色を設定
OB SET (vSettings;Graph_background_opacity;40)
ARRAY TEXT ($aTcols;2) //グラフの色を設定
$aTcols{1}:="#B5CF32"
$aTcols{2}:="#D43A26"
OB SET ARRAY (vSettings;Graph_colors;$aTcols)
GRAPH (vGraph;vSettings;X;A;B) //グラフを描画

```



## INTEGRATE MIRROR LOG FILE

INTEGRATE MIRROR LOG FILE ( pathName; operationNum {; mode {; errObject} } )

引数	型	詳細
pathName	テキスト	-> 統合したいログファイルの名前またはパス名
operationNum	実数変数	-> 最後の統合オペレーションの番号、またはファイル全体を統合するには-2 <- 最後に統合された新しいオペレーションの番号
mode	倍長整数	-> 0=strict mode(デフォルト)、1=auto-repair mode
errObject	オブジェクト変数	<- 統合に失敗したオペレーション

### 説明

**事前注意:** このコマンドは4D Serverでのみ動作し、**Execute on server**コマンドを使用して、あるいはストアドプロシージャ内でのみ実行可能です。

**INTEGRATE MIRROR LOG FILE**コマンドは、新しい柔軟な"自動修復"統合モードを提供するようになりました。これは致命的ではないエラーは統合オペレーションを妨げないモードです。その一方で、*operationNum*引数は必須となりました。

#### 新シンタックス(第2引数が必須に)

4D v15 R4以降、*operationNum*引数は必須となりました。省略された場合、エラーが生成されるようになりました。ログファイル内の全てのオペレーションを統合する場合には-2を渡し、それ以外の場合には最後に統合されたオペレーションの番号を渡します。

**互換性に関する警告:** この引数を必須とする新しいシンタックスは、*operationNum*引数がない既存のコードに対しては互換性を破る事になります(エラーが生成されることになります)。この場合、以前の状態を回復するためには*operationNum*引数変数には-2を渡してください。

#### 新しい自動修復モード

新しい*mode*引数を使用して統合モードを指定することができ、新しい*errObject*引数を使用してエラープロパティを記録することができます。この引数はどちらも使用は任意です。

*mode*引数には、起動したい統合モードを渡します。"バックアップと保存"テーマ内にある、以下の新しい定数のうち一つを使用する事ができます:

定数	型	値	詳細
Strict mode	倍長整数	0	厳格な統合モードを使用します(デフォルト)。このモードは大体的な場合において推奨されています。
Auto repair mode	倍長整数	1	自動修復アクションを持った柔軟なモードを使用し、(エラーがあれば) <i>errObject</i> 引数に記録します。

- **Strict mode:** このモードでは以前の4Dリリースと同様に、統合の最中にエラーが発生した場合には中止をし、エラーを追跡するにはMSCを使用する必要があります。このモードはデフォルトで使用されており、ほとんどの場合において推奨されます。

- **Auto repair mode:** このモードでは、致命的ではないエラーが発生した場合には、それを飛ばして統合を続行します。*errObject*引数を渡した場合、そこにエラーが記録され、あとから分析することができます。

致命的ではないエラーというのは、以下のものが該当します:

- ログがレコードの追加をリクエストしたが、そのレコードは既にデータの中に存在する。  
修復アクション:4Dはレコードを更新します。
- ログはレコードの更新をリクエストしたが、そのレコードはまだ存在していない。  
修復アクション:4Dはレコードを追加します。
- ログはレコードの削除をリクエストしたが、そのレコードは既に存在していない。

修復アクション:4Dは何もしません。

自動修復モード時にこれらのエラーのどれか一つが発生した場合、関係するレコードは自動的に"修復"され、関連したオペレーションが`errObject`引数内に記録されます。実行終了後、`errObject`引数は修復されたレコードを全て表示します。ここには"operations"という名前がついた単一のオブジェクト配列が含まれ、内部は以下のようにビルトされています:

```
{ "operations":
  [
    {
      "operationType":24,
      "operationName":"Create record",
      "operationNumber":2,
      "contextID":48,
      "timeStamp":"2015-07-10T07:53:02.413Z",
      "dataLen":24,
      "recordNumber":0,
      "tableID":"F4CXXXXX",
      "tableName":"Customers",
      "fields": {
        "1": 9,
        "2": "test value",
        "3": "2003-03-03T00:00:00.000Z",
        "4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
        "8": "BlobID: 2"
      }
    },
    {...}
  ]
}
```

**警告:** 自動修復モードは4Dの内部データ統合チェック機能をバイパスするため、特定の場においてのみ使用されるべきです。例えば中間ログファイルが紛失または破損し、そこから可能な限りたくさんオペレーションを復元したい場合等に使用することができます。いずれにせよ、このモードを使用する場合にはデータの統合に特別に注意を払う必要があります。

実際に表示されるプロパティの一覧は、オペレーションのタイプによります(例:レコード作成、レコード削除、レコード編集、Blob作成、等)。主なプロパティは以下の通りです:

- `operationType`: オペレーションの内部コード
- `operationName`: オペレーションの種類、例えば"create record"や"modify record"など
- `operationNumber`: ログファイル内でのオペレーションの内部番号
- `contextID`: 実行コンテキストのID。コンテキストの詳細は`extraData`のセクションで説明されています。
- `timeStamp`: ログファイル内のオペレーションのタイムスタンプ
- `dataLen`: データの内部サイズ
- `recordNumber`: 内部でのレコード番号
- `tableID`: テーブルの内部ID
- `tableName`: テーブル名
- `fields`: フィールドの一覧を含むオブジェクトとその値。テーブル内の全てのフィールドは記録されています。  
Blobまたはピクチャーの値の場合、その保存場所に応じて異なる種類の情報が提供されます:
  - Blobまたはピクチャーがデータファイルの中に保存されていた場合、このプロパティは"BlobID:"+内部Blob番号になります。例: "BlobID:1"
  - Blobまたはピクチャーがデータファイルの外に保存されていた場合、このプロパティは"BlobPath:" + データのパスになります。例: "BlobPath: Table 1/Field 6/Data\_EE12D091535F9748BCE62EDE972A4BA2.jpg"
- `extraData`: ユーザーコンテキストデータ。ここにはユーザー名とID、タスク名とID、ホストマシン名、クライアントのバージョンが含まれます。
- `sequenceNumber`: 自動インクリメントシーケンスのカレントの番号
- `primaryKey`: 主キーの値

## 例題

ミラーサーバー上のログファイルを自動修復モードで統合したい場合を考えます:

```
//サーバー上で実行すること
```

```
C_OBJECT($err)
```

```
C_LONGINT($num) // -2 を渡すと全てのオペレーションを統合します。
```

```
INTEGRATE MIRROR LOG FILE ("c:\\mirror\\logNew.journal";$num;Auto_repair_mode;$err)
```



## LAUNCH EXTERNAL PROCESS

```
LAUNCH EXTERNAL PROCESS ( fileName {; inputStream {; outputStream {; errorStream {}} } {; pid})
```

引数	型	説明
fileName	文字列	-> ファイルパスと起動するファイルの引数
inputStream	文字列、BLOB	-> 入力ストリーム(stdin)
outputStream	文字列、BLOB	<- 出力ストリーム(stdout)
errorStream	文字列、BLOB	<- エラーストリーム(stderr)
pid	文字列	<- 外部プロセスのユニーク識別子

### 説明

4D v15 R4以降、**LAUNCH EXTERNAL PROCESS**コマンドは作成されたプロセスに対し、OSレベルのプロセス固有識別子 (PID)を返すようになりました。この情報によって、作成した外部プロセスとその後やりとりすること(例えば停止する等)がやりやすくなります。

PIDは新しい任意の倍長整数変数、*pid*に返されます。この変数を渡した場合、`_4D_OPTION_BLOCKING_EXTERNAL_PROCESS`オプションの状態に関わらず、この変数はコマンドをローンチするのに作成されたプロセスのシステムレベルIDを返します。

プロセスローンチが失敗した場合、*pid*引数は返されません。

### 例題

以下のコード例は、Windows上のプロセスの一覧をリクエストします:

```
C_LONGINT ($pid)
C_TEXT ($stdin; $stdout; $stderr)

LAUNCH EXTERNAL PROCESS ("tasklist"; $pid) //PIDのみを取得
LAUNCH EXTERNAL PROCESS ("tasklist"; $stdin; $stdout; $stderr; $pid) //全ての情報を取得
```

## LISTBOX Get array、LISTBOX SET ARRAY、LISTBOX GET ARRAYS

LISTBOX Get array ( { \* ; } object ; arrType ) -> Function result

LISTBOX SET ARRAY ( { \* ; } object ; arrType ; arrPtr )

LISTBOX GET ARRAYS ( { \* ; } object ; arrColNames ; arrHeaderNames ; arrColVars ; arrHeaderVars ; arrColsVisible ; arrStyles { ; arrFooterNames ; arrFooterVars } )

### 説明

---

これらのコマンドは、4D v15 R4以降の配列型リストボックスに関連づけ可能な新しい"行管理配列"をサポートします。この配列は行の三つのプロパティ(表示状態、有効化/無効化、そして選択可/不可)を管理します。詳細な説明は[リストボックス:無効化された行と選択不可の行](#)の章にあります。

- **LISTBOX Get array**と**LISTBOX SET ARRAY** コマンドはどちらも*arrType*引数において新しい定数を受け付けます。これによって配列を取得または設定できるようになります:

定数	型	値
Listbox control array	倍長整数	3

- **LISTBOX GET ARRAYS**コマンドは*arrStyles*ポインター配列引数を通して配列をサポートします。第四要素はリストボックスの"非表示状態配列"(ブール型、以前のバージョンの4D)または新しい"行管理配列"(倍長整数配列)を指定します。

METHOD GET CODE ( path ; code *{;option}* {; \*} )

引数	型	詳細
path	テキスト、テキスト配列	-> 一つ以上のメソッドパスを含んだテキストまたはテキスト配列
code	テキスト、テキスト配列	<- 指定されたメソッドのコード
option	倍長整数	-> <b>0</b> または省略時 = シンプルな書き出し(トークンを含めない), <b>1</b> = トークンとともに書き出し
*	演算子	-> 指定時 = コンポーネントから実行時にコマンドをホストデータベースへと適用(この引数はこのコンテキスト外では無視されます)

## 説明

テーマ: デザインオブジェクトアクセス

METHOD GET CODEコマンドは4D v15 R4において変更されました:

- メソッドコードはインデント付きのテキストとして返されます。
- トークン管理を有効にする新しい任意の引数を受け付けるようになりました。

### インデント付きテキスト

コマンドによって返されたコードのテキストは、メソッドエディターのように、タブ文字を使用してプログラム構造を反映したインデントがされています。この機能はコードの可読性を向上させます。

### 新しい"option"引数

新しいoption引数を使用すると、メソッドのトークナイズされたランゲージ要素の書き出しモードを選択することができます:

- option引数に0を渡すか省略した場合、メソッドコードはトークンを使用せずに書き出されます。つまり、メソッドエディターで表示されているのと同じ状態で書き出されます。
- option引数に1またはCode with tokens定数を渡した場合、メソッドはトークンを使用して書き出されます。つまり、引数に書き出された中身にはトークナイズされた要素の後に内部参照が続きます。例えば、"String(a)"という式は"String:C10(a)"と書き出され、ここでの"C10"はStringコマンドの内部参照を表します。

トークナイズされるランゲージ要素には、以下のものが含まれます:

- 4Dコマンドと定数
- テーブル名とフィールド名
- 4Dプラグインコマンド

4Dトークンシンタックスについてのより詳細な情報については、[フォーミュラ内でのトークンの使用](#)の章を参照して下さい。トークンを使用して書き出されたコードは、今後起こりうるどのようなランゲージ要素の改名の影響も受けません。トークンを使用する事で、METHOD SET CODEコマンドを使用した場合でもコピー/ペーストを使用した場合でも、テキストで提供されたコードは4Dによって常に正常に解釈されます。

## 例題

以下の例題は、新しいoption引数の効果について説明しています。

以下の"simple\_init"メソッドを書き出したい場合を考えます:

```
Case of
```

```
  : (Form event=On Load)  
    ALL RECORDS ([Customer])
```

```
End case
```

以下のコードを実行した場合:

```
C_TEXT($code)  
C_TEXT($contents)  
$code:=METHOD Get path(Path project method;"simple_init")  
METHOD GET CODE($code;$contents;0) //トークンなし  
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

書き出されるドキュメントには以下の結果が含まれます(インデントがなされている点に注目して下さい):

```
  //%attributes = {"lang":"en"} comment added and reserved by 4D.  
Case of  
  : (Form event=On Load)  
    ALL RECORDS ([Customer])  
End case
```

以下のコードを実行すると:

```
C_TEXT($code)  
C_TEXT($contents)  
$code:=METHOD Get path(Path project method;"simple_init")  
METHOD GET CODE($code;$contents;Code with tokens) //トークンを使用  
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

書き出されたドキュメントには以下の結果が含まれます:

```
  //%attributes = {"lang":"en"} comment added and reserved by 4D.  
Case of  
  : (Form event:C388=On Load:K2:1)  
    ALL RECORDS:C47 ([Customer:1])  
End case
```

## OB GetとOB SET

OB Get ( *objRange* | *wpDoc* ; attribute { ; type } ) -> 戻り値

OB SET ( *objRange* | *wpDoc* ; attribute ; value { ; attribute2 ; value2 ; ... ; attributeN ; valueN } )

### 説明

---

どちらのコマンドも**WP GET ATTRIBUTES**と**WP SET ATTRIBUTES**同様に、4D Write Proオブジェクトでの属性定義をサポートします。

しかしながら、それには限度があります。これらのコマンドを使用した場合、ピクチャー変数またはフィールドを属性値として直接管理する事はできません。

### 例題

---

4D Write Proエリアを含むフォームメソッド内に、以下のように書く事ができます:

```
If (Form event=On Validate)
  OB SET ([MyDocuments]My4DWP;"myatt_Last edition by";Current user)
  OB SET ([MyDocuments]My4DWP;"myatt_Category";"Memo")
End if
```

ドキュメントのカスタム属性を読み出す事もできます:

```
vAttrib:=OB Get ([MyDocuments]My4DWP;"myatt_Last edition by")
```

## OBJECT Get horizontal alignmentとOBJECT SET HORIZONTAL ALIGNMENT

OBJECT Get horizontal alignment ( { \* ; } object ) -> 戻り値  
OBJECT SET HORIZONTAL ALIGNMENT ( { \* ; } object ; alignment )

### 説明

これらのコマンドは4D Write Proオブジェクトをサポートします。行揃えを両端揃えに設定するには、4D Write Proオブジェクトの`alignment`引数に対して`wk_justify`定数を渡します。この定数は"4D Write Pro"テーマ内にあります:

定数	型	値	詳細
<code>wk_justify</code>	倍長整数	5	4D Write Proエリアでのみ使用可能(この定数は"4D Write Pro"テーマに追加されています)

これは段落属性であるため、4D Write Proドキュメント内で選択された段落に対してのみ適用されます。

### 例題

4D Write Proエリアの行揃えを両端揃えにしたい場合を考えます:

```
OBJECT SET HORIZONTAL ALIGNMENT (*; "my4DWPArea"; wk_justify)
```

### RESUME TRANSACTION

このコマンドは引数を必要としません

#### 説明

---

**テーマ:** トランザクション

**RESUME TRANSACTION** コマンドは、カレントプロセスの同レベルにて **SUSPEND TRANSACTION** を使用して停止していたトランザクションを再開させます。このコマンド後に実行されたオペレーションはトランザクションコントロール下で実行されます(ただし複数の停止されていたトランザクションがネストされていた場合を除きます)。

より詳細な情報については、[停止したトランザクションとプロセスのステータス](#)の章を参照して下さい。

### SUSPEND TRANSACTION

このコマンドは引数を必要としません

#### 説明

---

**テーマ:** トランザクション

**SUSPEND TRANSACTION** コマンドはカレントプロセス内のカレントトランザクションを一時停止させます。その後、トランザクションのコンテキストは手つかずで残したまま、データをデータベースの他の部分にて(例えばデータをトランザクションに含まれる事なく)操作することができます。その間更新あるいは追加されたレコードは、トランザクションが**RESUME TRANSACTION** コマンドを使用して再開されるまではロックされています。

より詳細な情報については、[トランザクションの一時停止と再開](#)のセクションを参照して下さい。



## WEB Get session process count

WEB Get session process count ( sessionID ) -> 戻り値

引数	型		説明
sessionID	テキスト	→	セッションUUID
戻り値	倍長整数	↩	セッションに関連づけられたプロセス数

### 説明

新しい**WEB Get session process count**コマンドは`sessionID`引数にUUIDを渡したセッションに関連づけられた実行中のプロセスの数を返します。

このコマンドは4D v15 R4以降実装された新しい**4D Mobileセッションをプログラミングで管理**機能のコンテキストに追加されたものです。これは主に4D Mobileセッションで実行されたプロセス数をカウントするためにデザインされました。

- 4D Mobileセッションに関しては、実際のプロセス数を返します。4D Mobileセッションでは複数のプロセスが実行可能です。
- 通常のWebセッションに関しては、このコマンドは常に1を返します(1 Webセッション=1プロセス)

### 例題

情報をカレントの4D Mobileセッション上で配列形式で保存したい場合を考えます:

```
C_TEXT($sessionID)
C_LONGINT($count)
C_DATE($expDate)
C_TIME($expTime)

$sessionID:=WEB Get Current Session ID
$count:=WEB Get session process count($sessionID)
WEB GET SESSION EXPIRATION($sessionID;$expDate;$expTime)

APPEND TO ARRAY($aTimestamp;String(Current date)+" "+String(Current time))
APPEND TO ARRAY($aSessionUID;$sessionID)
APPEND TO ARRAY($aNbProcesses;$count)
APPEND TO ARRAY($aExpirationDate;$expDate)
APPEND TO ARRAY($aExpirationTime;$expTime)
```

## トランザクションの一時停止と再開

4D v15 R4では、4Dコードの中でトランザクションを停止・再開する新しい機能が追加されました。トランザクションが*suspended*(停止中)のときにはトランザクション自身とは独立したオペレーションを実行する事ができ、その後トランザクションを*resume*(再開)した際にそれを通常通り評価またはキャンセルすることができます。トランザクションが停止中のときには、具体的には以下の様な事をする事ができます:

- トランザクション外でレコードを作成または編集できます(例えば請求書番号のカウンターをインクリメントするなど)。
- 他のトランザクションを開始、停止、または閉じることができます。

この機能は、トランザクションの管理下で実行する必要のない特定のオペレーションをトランザクション内で実行する必要がある場合に有用です。例えば、顧客が注文をし(トランザクションを開かれ)、そのついでに住所を更新したい場合を考えます。ここで顧客が考えを変えて注文をキャンセルしたとします。トランザクションはキャンセルとなりますが、このままでは住所の更新も戻されてしまいます。こうした場合に、トランザクションを一時停止するのが有用と言えるでしょう。

### トランザクションの停止・再開をするには？

トランザクションの停止と再開は、"トランザクション"テーマに追加された三つの新しいコマンドを使用して管理されます:

- **SUSPEND TRANSACTION:** カレントトランザクションを停止させます。更新・追加されたレコードはどれもロックされたままになります。
- **RESUME TRANSACTION:** 停止されたトランザクションを再開させます。
- **Transaction active:** トランザクションが停止中の場合、あるいはカレントトランザクションがない場合にはFalseを返し、トランザクションが開始・再開されている場合にはTrueを返します。

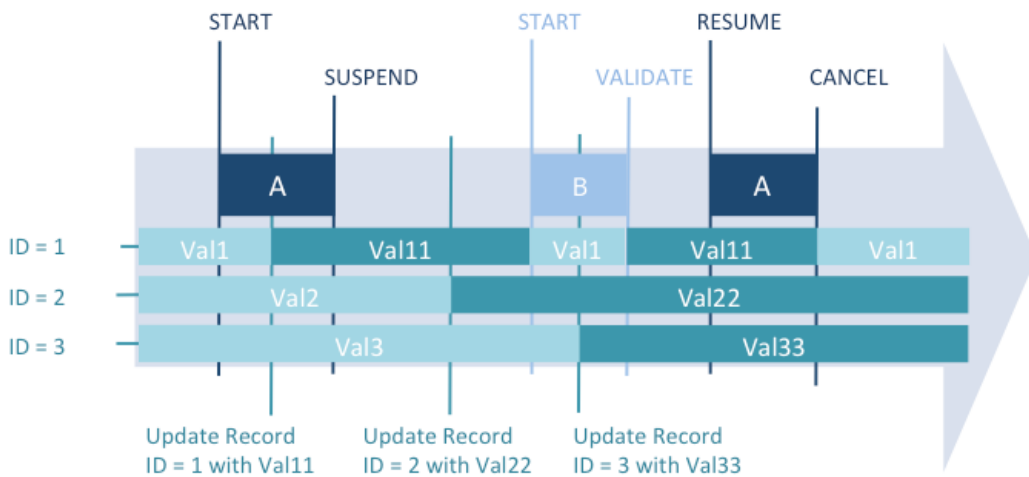
これらのコマンドはこのマニュアルの"ランゲージ"の章にそれぞれ詳細な説明があります。

### 停止されたトランザクションの仕組み

トランザクションの停止中においては、以下の原理が採用されます:

- トランザクション中に追加または編集されたレコードにはアクセスすることができ、トランザクション中に削除されたレコードはどれも見る事ができません。
- トランザクション外でレコードを作成、保存、編集することができます。
- 新規のトランザクションを開始することは可能ですが、そのトランザクション内では、停止中のトランザクションによって追加または編集されたレコードまたはレコードの値を見れません。この場合、トランザクションはそれぞれ完全に独立しており(別プロセスのトランザクションに似ています)、停止中のトランザクションはあとで再開またはキャンセル可能なことから、そこで追加あるいは編集されたレコードはどれも新規トランザクションからは自動的に非表示となります。新規トランザクションをコミットあるいはキャンセルすると、再びこれらのレコードを見る事ができるようになります。
- 停止されたトランザクション内で編集、削除、または追加されたレコードは、どれも他のプロセスからはロックされたままとなります。トランザクション外から、あるいは新規トランザクション内からこれらのレコードを編集あるいは削除しようとした場合、エラーが生成されます。

これらの原理をまとめると、以下の図のようになります:



トランザクションAで編集された値 (レコードID1にVal11が入る) はAの“停止中”に開始された新しいトランザクションBでは利用できません。“停止中”に編集された値 (レコードID2にVal22が入り、レコードID3にはVal33が入る) はトランザクションAがキャンセルされても、保存されたままです。

エラーを管理するために、特定の機能が追加されました:

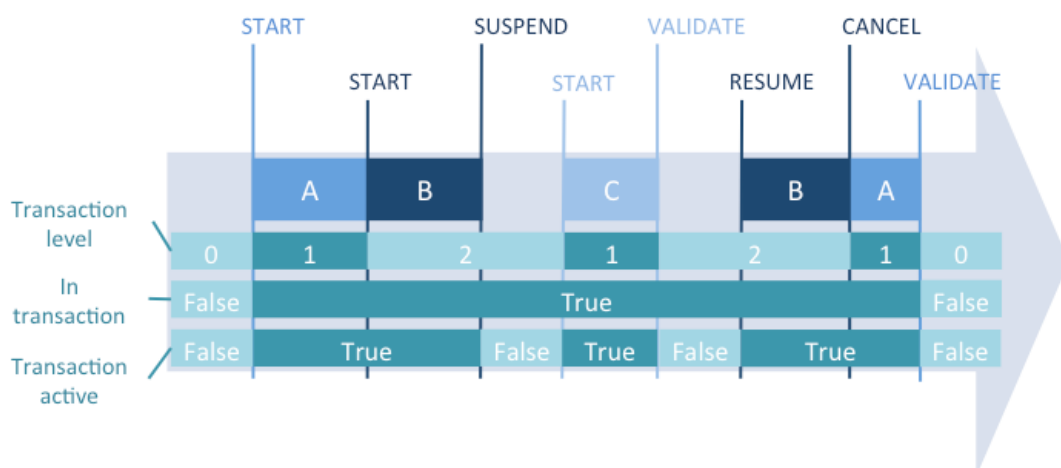
- それぞれのテーブルのカレントレコードは、トランザクション中に編集された場合には一時的にロックされ、トランザクションが再開されると自動的にロックが解除されます。この機構はトランザクションの一部で不要な保存を避けるために重要です。
- 例えばstart transaction / suspend transaction / start transaction / resume transactionのように無効なシーケンスを実行した場合には、エラーが生成されます。この機構は停止されたトランザクションを再開する際に、間に挟んだトランザクションをコミットまたはキャンセルするのを忘れるのを防ぎます。

## 停止したトランザクションとプロセスのステータス

既存の **In transaction** コマンドはトランザクションが開始されていれば、停止中であっても**True**を返します。カレントトランザクションが停止中であるかどうかを調べるためには、新しい**Transaction active**コマンドを使用する必要があります。こちらはこういった場合には**False**を返します。

しかし、開始されているトランザクションが存在しない場合には、どちらのコマンドも **False** を返します。この場合には既存の **Transaction level** コマンドを使用する必要があるかもしれません。こちらはこの場合に 0 (開始されているトランザクションはない) を返します。

以下の図は、様々なトランザクションのコンテキストと、トランザクションコマンドによって返される値の対応をまとめたものです:



## 例題

ここでは停止されたトランザクションの必要性について考えます。請求書データベースにおいて、トランザクション中に新しい請求書番号が必要になったとします。この番号は計算され、[Settings]テーブルに保存されます。マルチユーザー環境において、並行のアクセスは保護されなければなりません。メインのトランザクションとは無関係なデータにも関わらず、トランザクションの影響で [Setting] テーブルが他のユーザーによってロックされてしまう可能性があります。この場合、テーブルにアクセスする際にトランザクションを停止しておくことができます。

```

//請求書を作成する標準のメソッド
START TRANSACTION
...
CREATE RECORD([Invoices])
[Invoices]InvoiceID:=GetInvoiceNum //利用可能な番号を取得するメソッドを呼び出し
...
SAVE RECORD([Invoices])
VALIDATE TRANSACTION

```

**GetInvoiceNum**メソッドは実行前にトランザクションを一時停止させます。このコードは、トランザクション外からメソッドが呼び出された場合でも動作するという点に注意して下さい:

```

//GetInvoiceNum プロジェクトメソッド
//GetInvoiceNum -> 次に利用可能な請求書番号
C_LONGINT($0)
SUSPEND TRANSACTION
ALL RECORDS([Settings])
If(Locked([Settings])) //マルチユーザーアクセス
    While(Locked([Settings]))
        MESSAGE("Waiting for locked Settings record")
        DELAY PROCESS(Current process;30)
        LOAD RECORD([Settings])
    End while
End if
[Settings]InvoiceNum:=[Settings]InvoiceNum+1
$0:=[Settings]InvoiceNum
SAVE RECORD([Settings])
UNLOAD RECORD([Settings])
RESUME TRANSACTION

```

## 括弧の代わりに4DVARコメントを使用オプションの削除



---

4D v15 R4以降、**括弧の代わりに4DVARコメントを使用**互換性オプションがデータベース設定ダイアログボックスの"互換性"ページから削除されています。変換されたデータベースでは今後使用できなくなります。

この設定は括弧を使用した4Dタグ専用のシンタックス(例えば[myVar]など)を有効化するために使用されていました。以前のシンタックスはもう4Dデータベースでは解釈されていません。変換されたデータベースがこのオプションを使用していた場合、テンプレートをそれぞれ更新する必要があります。4D v15 R4以降でサポートされるシンタックスは以下のものだけです:

- 標準のHTML表記(<!--4DVAR myVar-->)
- **4DTEXT、4DHTML、4DEVALでの新しい\$シンタックス**

## 4D Write Pro

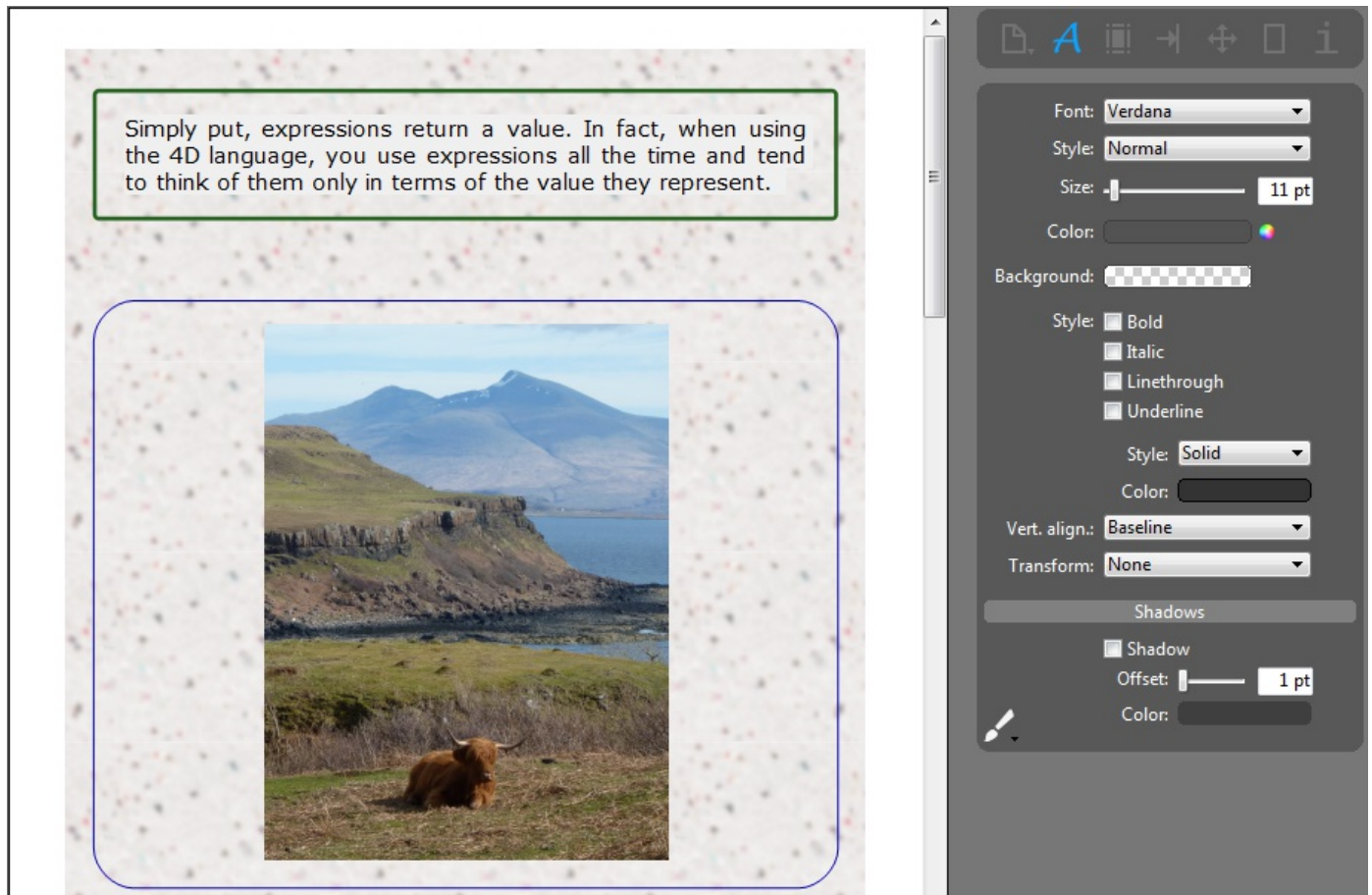
-  プログラムによるテキストと画像の属性の変更
-  新しい4D Write Proエリアフォームオブジェクト

## ■ プログラムによるテキストと画像の属性の変更

4D v15 R4では、4D Write Proでドキュメント内でテキストと画像の属性を変更するためのコマンドのフルセットを提供しています。これに加え、複数の4Dコマンドもこれに合わせてアップデートされています。

これらの新機能のおかげで、4Dデベロッパーは4D Write Proドキュメントに対して(ボタン、メニュー、チェックボックス等を使用した)独自のユーザーインターフェースをデザインすることができます。コマンドは、ユーザーセレクションあるいはカスタムの値を用いて、ドキュメント全体あるいは特定の一部分(レンジ)に対して適用する事ができます。利用可能なプロパティはドキュメント単位、マージン、パディング、背景、段落設定、カラー、フォント、フォントスタイル、そして画像プロパティ等を含みます。

例えば、**新しい4D Write Proエリアフォームオブジェクト**はこれらの新機能をふんだんに使用し、洗練された4D Write Proインターフェースを提供しています:



**注:** 4D Write Proエリアがオブジェクトフィールドに保存されている場合、カスタムの属性を定義する事もできます(**カスタムの属性を使用**の段落を参照して下さい)。

## 新しい4D Write Proコマンド

以下の新しい4D Write Proコマンドは、属性管理機能のために追加されました。これらのコマンドは**4D Write Proリファレンスマニュアル**に詳細な説明があります。

### セレクションレンジコマンド

ドキュメントのセレクションを管理するための複数の新しいコマンドが追加されました。選択されたテキストには(非表示の)フォーマットタグが含まれ得るため、4D Write Proはレンジという概念を用います。レンジとは4D Writeドキュメントの一部を表すオブジェクトです。

- **WP Get range**( *wpArea ; startRange ; endRange* ) -> *rangeObj*: 引数として渡した境界に対応する新しいレンジを返します。

- **WP Get selection**( {\*;} wpArea ) -> rangeObj: カレントのユーザーの選択範囲に対応した新しいレンジを返します。
- **WP Get pictures**(rangeObj) -> rangeObj: ピクチャーのみを含んだ新しいレンジを返します。
- **WP Get paragraphs**(rangeObj) -> rangeObj: 段落のみを含んだ新しいレンジを返します。
- **WP SELECT**( {\*;} wpArea {; rangeObj}{; startRange ; endRange} ): レンジに対応したテキストを選択します。

## 属性管理コマンド

以下の新しいコマンドはドキュメントのあらゆる属性を取得または設定することができます:

- **WP SET ATTRIBUTES**( rangeObj | wpDoc ; attribName ; attribValue {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN} ): ドキュメントあるいはレンジ内の一つ以上の属性/値のペアを設定します。
- **WP GET ATTRIBUTES**( rangeObj | wpDoc ; attribName ; attribValue {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN} ): ドキュメントあるいはレンジ内のカレントの/値を取得します。
- **WP RESET ATTRIBUTES**( rangeObj ; attribName {; attribName2 ; ... ; attribNameN } ): ドキュメントあるいはレンジ内の属性値をリセットします。

属性は**4D Write Pro属性**の章の中に詳細な説明があります。

## フォント管理コマンド

新コマンドを使用して、レンジにおけるスタイルのサポートについての情報を取得することができます:

- **WP Is font style supported**( rangeObj ; wpFontStyle ) -> true あるいは false: レンジがあるスタイルに対してサポートしているかどうかを調べます(インターフェースのデザインに有効です)。

## アップデートされた4Dコマンド

新しい4D Write Proの実装をサポートするために、いくつかのコマンドが4D v15 R4でアップデートされました。

### OBJECT SET HORIZONTAL ALIGNMENT

このコマンドは4D Write Proオブジェクトをサポートします。4D Write Proオブジェクト用の新しい定数がalignment引数で使用できるようになり、これによって両端並びを設定できるようになりました:

定数	型	値	コメント
wk justify	倍長整数	5	4D Write Proエリアに対してのみ利用可能("4D Write Pro"テーマに追加された定数)

### OB SET

このコマンドは**WP SET ATTRIBUTES**同様、4D Write Proオブジェクトの属性定義をサポートします。以下のシンタックスがサポートされます:

**OB SET ( rangeObj | wpDoc; attribName ; attribValue {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN} )**

しかしながら、制約が一つあります。このコマンドでは、ピクチャー変数あるいはフィールドを直接属性値として渡す事はできません。

### OB Get

このコマンドは**WP GET ATTRIBUTES**同様、4D Write Proオブジェクトの属性定義をサポートします。以下のシンタックスがサポートされます:

**OB Get ( rangeObj | wpDoc; attribName ) -> Function result**

このコマンドでは、OB SETと同じ制約が存在します。ピクチャー変数あるいはフィールドを属性値として使用する事はできません。

### 4D Write Pro属性の文字列化

4D v15 R4では、4D Write Proオブジェクトを**JSON Stringify**コマンドを使用してJSONに変換した場合、出力文字列に残るのは"title"属性のみになります。

カスタム属性があった場合、それらも文字列化されます(**4D Write Proドキュメントを4Dオブジェクトフィールドに保存する**の章の"カスタム属性を使用"を参照して下さい)。

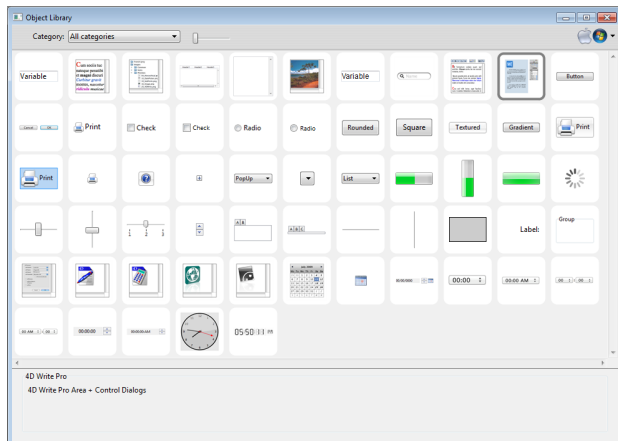
## QUERY BY ATTRIBUTE




**4D Write Proドキュメントを4Dオブジェクトフィールドに保存する**の章で定義されているように、**QUERY BY ATTRIBUTE**コマンドはドキュメントがオブジェクトフィールドに保存されている場合4D Write Pro属性(内部属性またはカスタム属性)をサポートします。

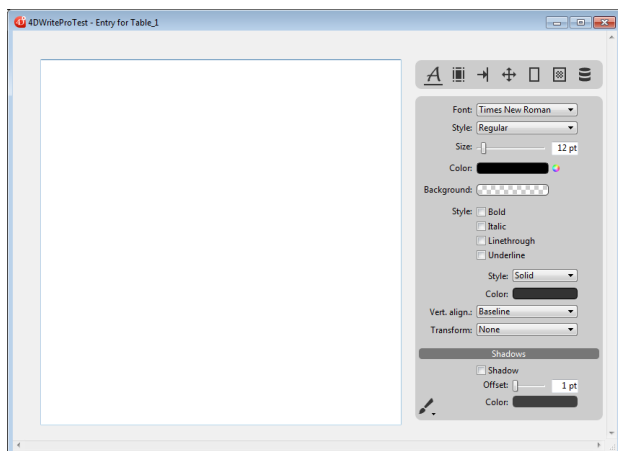
## 新しい4D Write Proエリアフォームオブジェクト

4D v15 R4以降、新しい4D Write Proエリアフォームオブジェクトが4Dのオブジェクトライブラリに追加されました:




(フォーム絵でいx他のオブジェクトバーで選択できる)標準の4D Write Proエリアと違い、事前に設定がされているこのオブジェクト内には、4D Write Proエリアの他にそれに関連づけられた4D Write Proウィジェットサブフォームがあり、その中には直感的なコントロールパネルが格納されています。また複数のパネル間を案内するツールバーもあり、これによりユーザーはエリアの中身を直接設定することができます。


新しい4D Write Proエリアフォームオブジェクト  をフォームにドロップすると、フォーム実行時に、何の追加のプログラミングもすることなく、4D Write Proエリアとそれに関連したコントロールパネルセットが既に使用可能な状態で準備されています:



この新しいフォームオブジェクトの使用についての詳細な情報に関しては、**オブジェクトライブラリ内の新しい4D Write Proフォームオブジェクト**の章を参照して下さい。

# 4D Mobile

 4D Mobileセッションをプログラミングで管理

 Test\_MOVE

## 4D Mobileセッションをプログラミングで管理

### 概要

---

4D v15 R4以降、4D Mobile WebセッションIDを4D Server側から取得できるようになりました。この新機能により、デベロッパーはセッションに関する情報をローカルに取得あるいは設定できるようになります(以下の例を参照して下さい)。

4D Mobileセッションは通常の4D Webセッションコマンドを通して管理できるようになりました。既存のコマンドは4D Mobileセッションをサポートするようにアップデートされました。これに加え、新しい**WEB Get session process count**コマンドが追加され、また分かりやすさのために、データベースメソッドも改名されました。

### 何の変化もなく引き続きサポートされるコマンド

---

以下の既存のWebセッション管理コマンドは、シンタックスに何の変化もないままで4D Mobileセッションがサポートされます。

#### WEB Get Current Session ID -> sessionID

**WEB Get Current Session ID**コマンドはカレントの4D Mobileセッションに関連づけられたUUIDを返すようになりました。

#### WEB CLOSE SESSION(sessionID)

**WEB CLOSE SESSION**コマンドは、*sessionID*引数に渡したIDの4D Mobileセッションを閉じます。4D Mobileセッションでは複数のプロセスを管理できるので、このコマンドはセッションに関連した全てのWebプロセスの実行の終了をリクエストします。

#### WEB GET SESSION EXPIRATION ( sessionID ; expDate ; expTime )

**WEB GET SESSION EXPIRATION**コマンドは、4D Mobileセッションのcookieに関連した失効情報を返します。

4D Mobileセッションに関連したプロセスには全て同じcookieが使用されます。

### On Web Close Processデータベースメソッド(旧On Web Session Suspend)

---

4D v15 R4以降、既存の**On Web Session Suspend**データベースメソッドは**On Web Close Process**データベースメソッドと名前が変更されました。4D Mobileセッションに関連しているプロセスは必ずしも一つだけではない(一つのセッションで複数のプロセスを実行可能)ため、このデータベースメソッドは一貫性のために改名されました。詳細な情報については、以下の**4D Mobileセッション vs Webセッション**の段落を参照して下さい。

**On Web Close Process**データベースメソッドはWebプロセスが閉じられようとしているたびに4Dから呼び出されます。これは4D Mobileセッションプロセスを完全にサポートします。このコンテキストにおいては各Webプロセスが閉じられるたびに呼び出され、それにより4D Mobileセッションプロセスにおいて生成されたどんなデータ(変数、セレクション、等)も保存する事ができます。

**注:** 通常のWebセッションに対しては、**On Web Close Process**データベースメソッドは何も変わりません。Webセッション、具体的にはWebセッション固有のプロセスが閉じられた時に呼び出されます。

### 新しいWEB Get session process countコマンド

---

新しい**WEB Get session process count**コマンドによって、特定のセッションに関連する存在しているプロセスの数を調べる事ができます。

- 通常のWebセッションに対しては、このコマンドは常に1を返します(Webセッション1つにつき1プロセス)
- 4D Mobileセッションに対しては、このコマンドは関連した全てのWebプロセスを返します。このコマンドはこのコンテキストにおいては、例えば4D Mobileセッションの全てのプロセスに対してループを実行する様な場合において有効です。詳細な情報については**WEB Get session process count**コマンドの詳細を参照して下さい。

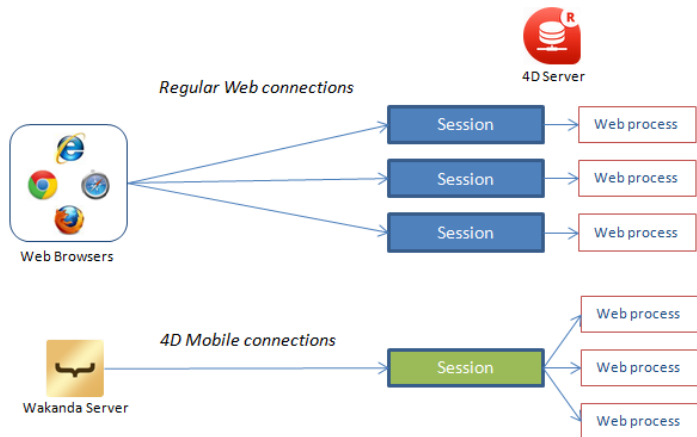
### 4D Mobileセッションを管理する意味と方法

---

## 4D Mobileセッション vs Webセッション

4D MobileセッションとWebセッションは二つの異なる種類のセッションです。一部の概念(とコマンド)に共通するものがありますが、持っているプロパティが異なります。主な違いはセッション、プロセス、そしてプロセスコンテキストの関係にあります:

- Webセッションは一つのWebプロセスと一体になります。自動セッション管理機能によって、セッションのプロセスコンテキスト(セレクション、変数インスタンス、等)は再利用する事ができます。
- 4D Mobileセッションは複数のWebプロセスと関連づけることができます。それぞれのプロセスコンテキストはプロセスメソッド実行後に自動的にリセットされます。



結果として、セッションに関連した4D Mobile Webプロセス間の情報を保持するためには、4D Server側にて追加のプログラミングが必要になります。

### 例題

単一の4D Mobileセッションの複数のプロセス間での情報を共有あるいは再利用したい場合、4D MobileセッションのUUIDを使用してセッション関連情報を指定する事ができます。例えば、レコードをクエリしたあと、命名セレクションを4D Serverに保持しておくことで、同じセッション内でのその後のRESTリクエストがそのセレクションに直接アクセスできるようにしたい場合を考えます。クエリ宣言後に、以下のように書く事ができます:

```
//セッションUUIDを含んだインタープロセスセレクションを作成
COPY NAMED SELECTION ([Emp]; "<>EmpSel"+WEB Get Current Session ID)


//あとでそのセッションからのセレクションを再利用することが可能
USE NAMED SELECTION ([Emp]; "<>EmpSel"+WEB Get Current Session ID)
```

## TEST

---

This is a move test

# SQL

 外部結合のサポートの拡張

## 外部結合のサポートの拡張

### 詳細

---

4D v15 R4以降、ビルトインのSQLサーバーはSQL外部結合のサポートを、三つ以上のテーブルを含むクエリへと拡張します。

同じSELECT宣言内で複数の結合とテーブルを組み合わせるクエリがサポートされるようになりました。この際、以下の条件に従う必要があります：

- それぞれの明示的な外部結合ON節は、それ以上でも以下でもなく、ちょうど二つのテーブルを参照します。
- 一つのテーブルはJOIN節のすぐ左から、もう一つのテーブルはすぐ右から指定する必要があります。

例えば、以下のクエリを実行すると成功します：

```
SELECT * FROM T1
LEFT JOIN
(T2 LEFT JOIN T3 ON T2.ID=T3.ID)
ON T1.ID=T3.ID
```

上記の条件に当てはまらないクエリは拒否されます。サポートされるシンタックスの詳細な説明については、*4D SQL Reference v15 R4*を参照して下さい。

### 制約

---

現在の4D v15 R4での外部結合の実装は、いくつかの制約がつかます：

- ON条件では、ANDとNOTブール操作のみがサポートされます(ORはサポートされていません)。また、IS NULL述部とCOALESCEファンクションもサポートされていません。
- JOIN節の左または右のどちらかが明示的外部結合だった場合、4Dメソッドの呼び出しはサポートされていません。
- 明示的外部結合宣言でのSQLビューはサポートされていません。
- 外部結合を使用するサブクエリはサポートされていません。

4D SQL Reference v15 R4マニュアルには更新され、これらの制約が全て記載されています。

### 最適化

---

現行の実装では、WHERE節が"field=constant"型のフィルターを含む場合に最適化が可能です。例えば：



```
SELECT * FROM T1
LEFT JOIN T2
ON T1.ID=T2.ID
LEFT JOIN T3
ON T1.ID=T3.ID
WHERE T1.ID=123
```

パフォーマンス向上のためには、WHEREフィルターを外部結合節へと複製します。以下のクエリは結果は同じになりますが、前述のクエリより格段に実行が早くなります：

```
SELECT * FROM T1
LEFT JOIN T2
ON T1.ID=T2.ID AND T1.ID=123 -- WHEREフィルター"T1.ID=123"がここで複製されます
LEFT JOIN T3
ON T1.ID=T3.ID AND T1.ID=123 -- WHEREフィルター"T1.ID=123"がここで複製されます
WHERE T1.ID=123
```



## 最適化

-  データベースの再インデックスがより高速に
-  EXECUTE FORMULAのコンパイルモードでのキャッシュ

## データベースの再インデックスがより高速に

---

4D v15 R4において、データベース全体の再インデックスのアルゴリズムが大幅に最適化されました。プロセス全体が劇的に早くなり、最大で2倍の早さを達成しました。

**注:** 全体再インデックスは、例えばデータベース修復後や、4dindxファイルが削除された後では必須です。

インデックスの際にはそれぞれのレコードのそれぞれのテーブルがメモリにロードされる必要があるため、最適化はディスクの交換を最小化するのを目的としています。このオペレーションはそれぞれのテーブルでシーケンシャルに実行されるので、レコードの読み込み・解放オペレーションを減らします。

理想的なシナリオでいえば、キャッシュがデータファイル全体とインデックス全体を格納するだけの大きさがあることです。この場合、この新アルゴリズムによるスピードの改善はありません。しかしながら、利用可能なサーバーメモリは通常そこまで大きくありません。キャッシュが、最大のテーブルとそのインデックスを格納できるだけの大きさがあれば、この新アルゴリズムでは速度は2倍に早くなります。

## EXECUTE FORMULAのコンパイルモードでのキャッシュ

4D v15 R4以降、コンパイルモードで**EXECUTE FORMULA**コマンドによって実行されたそれぞれのフォーミュラは、メモリー内の新しい専用のキャッシュに保存することができます。フォーミュラはトークナイズされた形式でキャッシュされます。フォーミュラは一度キャッシュされると、それ以降の実行はトークナイズのステップをバイパスするため、大幅に最適化されます。

キャッシュサイズはデフォルトではゼロ(キャッシュなし)です。**SET DATABASE PARAMETER**コマンドを使用して作成あるいは調整する必要があります(**Get database parameter** と **SET DATABASE PARAMETER**を参照して下さい)。

```
SET DATABASE PARAMETER (Number of formulas in cache;0) //フォーミュラのキャッシュはなし  
SET DATABASE PARAMETER (Number of formulas in cache;3) //各プロセスにおいて最大3フォーミュラをキャッシュ可能
```

**EXECUTE FORMULA**コマンドはコンパイルされたデータベースあるいはコンポーネントから呼び出されたときだけキャッシュを使用します。

内部テストによって、フォーミュラがキャッシュに読み込まれている際には、キャッシュされていない時に比べてフォーミュラの実行は格段に早くなっている事が明らかになっています。

## 4D Developer Edition 64-bit版(プレビュー)


---


4D v15 R2以降、4DはWindowsとOS X用の4D Developer Edition 64-bit版を提供しています。


これらが最終バージョンとしてご利用いただけるようになると、これによって4Dスタンドアロンアプリケーション(と4Dリモートアプリケーション)は64-bit版OSの利点を最大限活用する事ができます。64-bitテクノロジーの主な利点はより多くのRAMメモリーを割り当てることができることです。

これらのバージョンは近い将来登場する64-bit用製品を先取りする機会となることでしょう。またこれらは、独自のコードを適応させ、即座にフィードバックを一足早く得られるため、サードパーティプラグインの開発者にとっても有用でしょう。しかしながら、これらの製品はまだ開発途中であるため、プレビュー版も不安定なときが有り得るという点に注意して下さい。

- 4D Developer Edition 64-bit版のR2プレビューの情報については、[4D v15 R2 - Upgrade \(PDF\)](#)のマニュアルを参照して下さい。
- 4D Developer Edition 64-bit版のR3プレビューの情報については、[4D v15 R3 - Upgrade \(PDF\)](#)のマニュアルを参照して下さい。
- 4D Developer Edition 64-bit版のR4プレビューのアップデートされた情報については、以下の章を参照して下さい。

 システム設定

 機能概要

 再設計されたプロパティリスト

 新しいラベルエディター

 新しいクイックレポートエディター

### システム要項

---

64-bit版4D Developerを動かすには、以下のスペックが必要です:

	Windows	OS X
OS	Windows 7 またはそれ以上(64-bit版)	OS X 10.10 (Yosemite) またはそれ以上
RAM	8 GB	8 GB

お使いのバージョンの4DがどのOSと対応しているのかを探すためには、[4D Web サイトにある対応早見表](#)を参照して下さい。

### アーキテクチャー

---

64-bit アーキテクチャーを想定した 4Dアプリケーションは、この環境専用のバージョンとなります。言い換えるとそれらは 32-bit版OSでは実行できません。

インタープリタモードでは、64-bit版・32-bit版のどちらのアプリケーションでも同じ4Dデータベースを開くことができます(サーバー・ローカルを問いません)。どちらのアプリケーションを使おうと、開発の段階では一切違いはありません(ただし以下の制限を除く)。

コンパイルモードでは、データベースは適切なプロセッサ向けにコンパイルされている必要があります。つまり、64-bit版アプリケーションで開くためには64-bitの、32-bit版アプリケーションで開くためには32-bitのプロセッサ向けにコンパイルしなければなりません。32-bit用にコンパイルされた、インタープリタコードを含んでいないデータベースを 64-bitの 4Dアプリケーションで開くことはできず、その逆もまたしかりです。データベースはどちらか特定のアーキテクチャーだけにコンパイルすることもできますし、両方にコンパイルすることもできます。コンパイルについての詳細は、次の章を参照してください。

以下の一覧は様々な4D実行環境と、そのデータベースのコードとの互換性をまとめたものです:

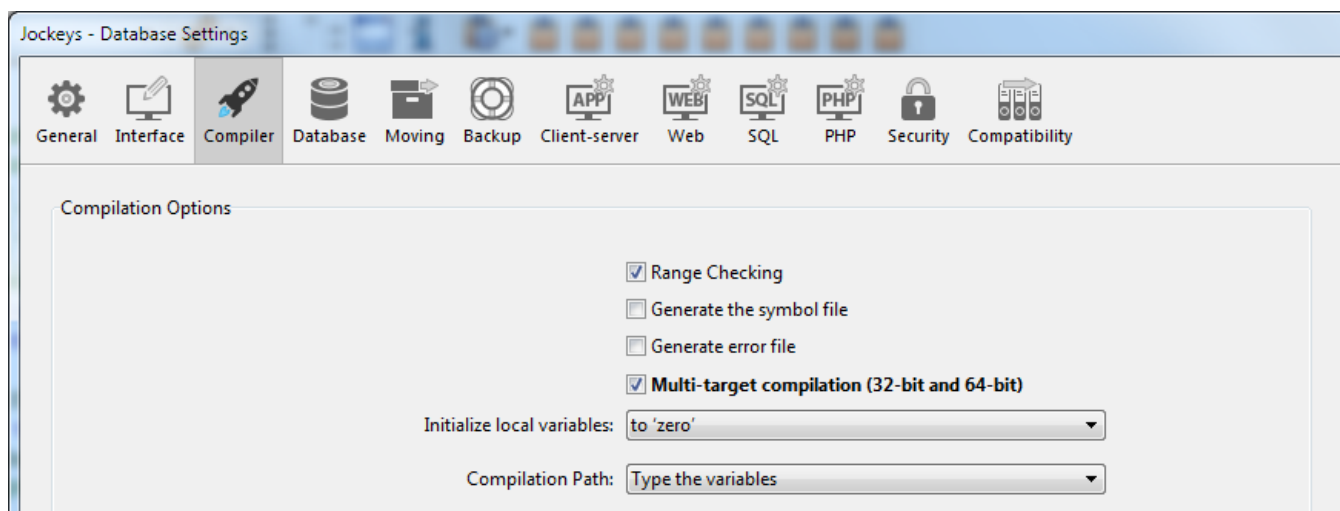
	利用可能コード	32-bit 4D	64-bit 4D
<b>32-bit 4D Server</b>	インタープリタ	OK	OK(*)
	32-bit コンパイル済みのみ	OK	-
	32-bit と 64-bit コンパイル済み	OK	OK(*)
<b>64-bit 4D Server</b>	インタープリタ	OK	OK(*)
	64-bit コンパイル済みのみ	-	OK(*)
	64-bit と 32-bit コンパイル済み	OK	OK(*)
<b>Local database</b>	インタープリタ	OK	OK
	32-bit コンパイル済みのみ	OK	-
	64-bit コンパイル済みのみ	-	OK
	32-bit と 64-bit コンパイル済み	OK	OK

(\*) 64-bit版 4D は旧式ネットワークレイヤーを利用できないため、64-bit版 4D から 32-bit版 4D Server (Windows・OS X とともに) および Windows用 64-bit版 4D Serverに接続する場合には、サーバー側において**ServerNet** ネットワークレイヤーが有効化されていることを確認する必要があります。なお、Mac OS用 64-bit版 4D Server はそもそも旧式ネットワークレイヤーをサポートしていないため、この確認は不要です。より詳細な情報については、[新しい ServerNet ネットワークレイヤー\(互換性\)](#)の章を参照してください。

### マルチターゲットコンパイル

---

4D アプリケーションは 32-bit用と 64-bit用、両方にコンパイルすることもできます。このためには、データベース設定の"コンパイル" ページにある、**マルチターゲットコンパイル(32-bitと64-bit)** オプションを使用する必要があります:



このオプションがチェックされていない場合(デフォルト)、コンパイルすると4D Developerのカレントのアーキテクチャー(32-bit または 64-bit)用にのみコードが一つだけコンパイルされます。このオプションがチェックされている場合には、コンパイラーは .4DC および .4DBファイル内に 64-bit版と32-bit版のコードを両方含めます。これにより、ファイルは32-bit版・64-bit版両方の4Dアプリケーションで実行できるようになります。

この章ではWindowsとOS X用の64-bit版4D Developer Editionの実装と使用に関する特性について記載しています。このバージョンの開発は継続的なステップ・バイ・ステップマナーで実行されています。以前は利用不可だった機能も順次有効化され、新しい特定の機能もそれぞれのバージョンに追加されています。

## 更新された機能

多くの4D機能とダイアログが、64-bitアーキテクチャーをサポートするために更新、あるいは書き換えられました。ほとんどの変化は気づかれなように行われており、32-bit版のリリースと全く同じように動作します。しかしながら、一部のエディターにおいては変更が行われ、32-bit版とは違っています。これらは64-bit版4D Developer Editionにて提供されています。

機能	対象となる4Dのバージョン	補足
クイックレポートエディター	OS X & Win	完全に書き換えられています。カレントのプレビューでは、埋め込みエリアとクロステーブルレポートはご利用いただけません。プレビューについては <a href="#">新しいクイックレポートエディター(プレビュー版)</a> の章で説明があります。
ラベルエディター	OS X & Win	完全に書き換えられています。 <a href="#">新しいラベルウィザード(プレビュー版)</a> の章を参照して下さい。
フォームエディタープロパティリスト	OS X & Win	新規デザイン

## 無効化された機能

最初の64-bit版4D Developer Editionでは、一部の機能は無効化されています：

機能/テクノロジー	対象となる4Dのバージョン	補足
シリアルポート通信	OS X	無効化されているコマンドは、 <b>GET SERIAL PORT MAPPING</b> 、 <b>RECEIVE BUFFER</b> 、 <b>RECEIVE PACKET</b> 、 <b>RECEIVE RECORD</b> 、 <b>RECEIVE VARIABLE</b> 、 <b>SEND PACKET</b> 、 <b>SEND RECORD</b> 、 <b>SEND VARIABLE</b> 、 <b>SET CHANNEL</b> 、 <b>SET TIMEOUT</b> 、それに加え、シリアルポートのコンテキストにおいてのみ <b>USE CHARACTER SET</b> も無効化されています。
読み込み/書き出しダイアログボックス	OS X	無効化されています。それに加え、 <b>IMPORT DATA</b> と <b>EXPORT DATA</b> も無効化されています。
クイックレポートの埋め込みエリア	OS X & Win	無効化
クイックレポートのクロステーブル	OS X & Win	無効化
Webエリア内の統合Web Kit	OS X & Win	無効化
チャート	OS X & Win	無効化(GRAPHコマンドはXSLTに依存するもので、これはサポートされていません)
印刷	Win	無効化

## サポートされない機能

以下の機能またはテクノロジーは、64-bit版4D Developer Editionではサポートされません:



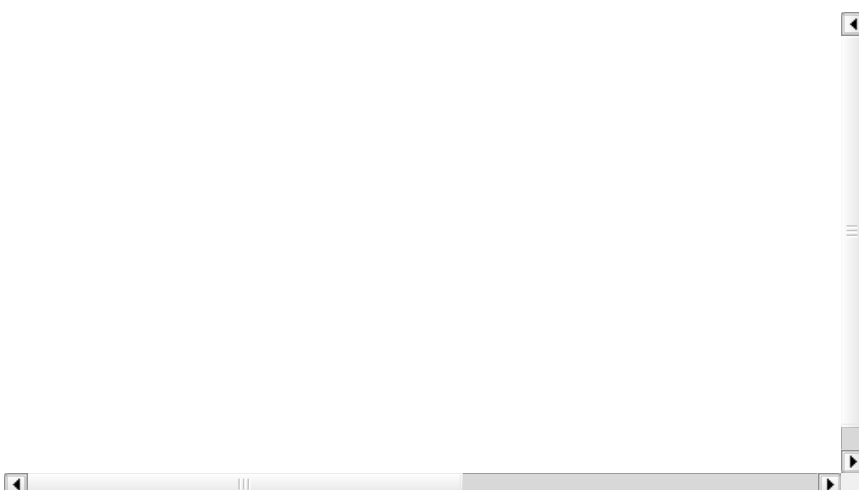
機能/テクノロジー	対象となる4Dバージョン	コメント
XSLT と Xalan	OS X & Win	<code>_o_XSLT APPLY TRANSFORMATION</code> 、 <code>_o_XSLT SET PARAMETER</code> 、そして <code>_o_XSLT GET ERROR</code> は何もしません。代わりに <code>PROCESS 4D TAGS</code> コマンドを使用するか、PHP <code>libxslt</code> モジュールを使用して下さい。
PICT フォーマット	OS X & Win	'サポートされていない画像フォーマットです'ピクチャ+画像の拡張子が代わりに表示されます。PICTフォーマットは4D全体において廃止予定となっています。 <a href="#">AP Is Picture Deprecated</a> も参照して下さい。
cicn アイコン	OS X & Win	<code>GET ICON RESOURCE</code> はサポートされていません。エラーを返します。
データベース .RSR ファイル	OS X & Win	データベース .RSR ファイルは自動的に開かれませんが、 <code>Open resource file</code> を使用する必要があります。
書き込み可能リソースファイル	OS X & Win	<code>_o_Create resource file</code> はサポートされていません。リソースファイルは読み込みのみモードでしか開く事ができません。
<code>_o_Font number</code>	OS X & Win	このコマンドはサポートされていません。これはエラーを返します。
旧式ネットワークレイヤー	OS X & Win	<code>ServerNet</code> のみがサポートされています。
ASCII 互換モード	OS X & Win	Unicode モードのみがサポートされています。
4D Write と 4D View プラグイン	Win	旧式プラグインはWindows用4Dの64-bit版では互換性がありません。代わりに4D Write Pro と4D View Pro(開発中)を使用して下さい。
OLE ツール	Win	サポートされていません

## OS Xでのアニメーションフォームオブジェクト

OS Xで実行される64-bit版の4DアプリケーションはOSネイティブのアニメーションの恩恵により、4Dフォームのユーザーエクスペリエンスを向上させる事ができます。具体的には以下のような点で向上しています:

- フィールド、入力可能変数、ラジオボタン、チェックボックスはフォーカスを受けるとアニメーションを表示します。
- ラジオボタンとチェックボックスはマウストラッキング時にアニメーションをトリガーします。
- スクロールバーには特殊なアニメーション効果があります(ただしYosemite版のみ)
- リストフォームでTABを押すと、カラム間をスムーズに移動することができます。

以下の動画はこれらのアニメーションの概要をまとめたものです:



**注:** Appleはアプリケーション内でどのようにアニメーションを使用したらいいかという[ガイドライン](#)を発行しています。

## 再設計されたプロパティリスト

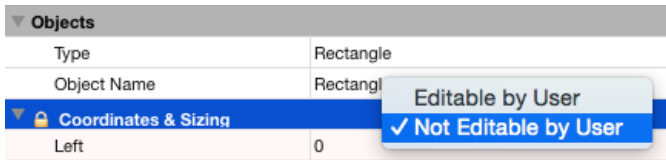
フォームエディターのプロパティリストは4D Developer Editionの64-bit版にて新設計となりました。  
これに加え、いくつかの機能も修正されています：

- ユーザーフォームエディター内にてプロパティのロック/アンロックを設定・閲覧する新機能
- カラーの管理と、カラーパレットが改善されました。

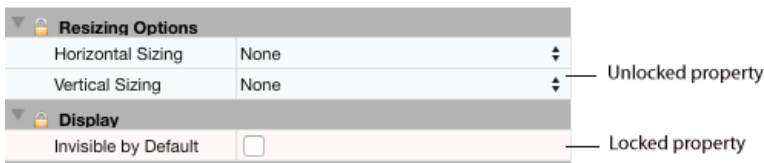
## ユーザーフォームプロパティ

フォームが"ユーザー更新可"に設定されているとき、プロパティリストにおいて二つの新しい機能が使用できるようになりました：

- デベロッパは新しいコンテキストメニューを使用して編集可能なプロパティを選択できるようになりました：



- 南京錠のアイコンに加えて、プロパティの背景色もロック・アンロックの状態を知らせるようになりました：
  - 薄い赤:ロック
  - 薄い青:アンロック



## カラープロパティ

プロパティリストの全てのカラープロパティにおいて、新しいデフォルトのカラーパレットが使用されるようになりました：

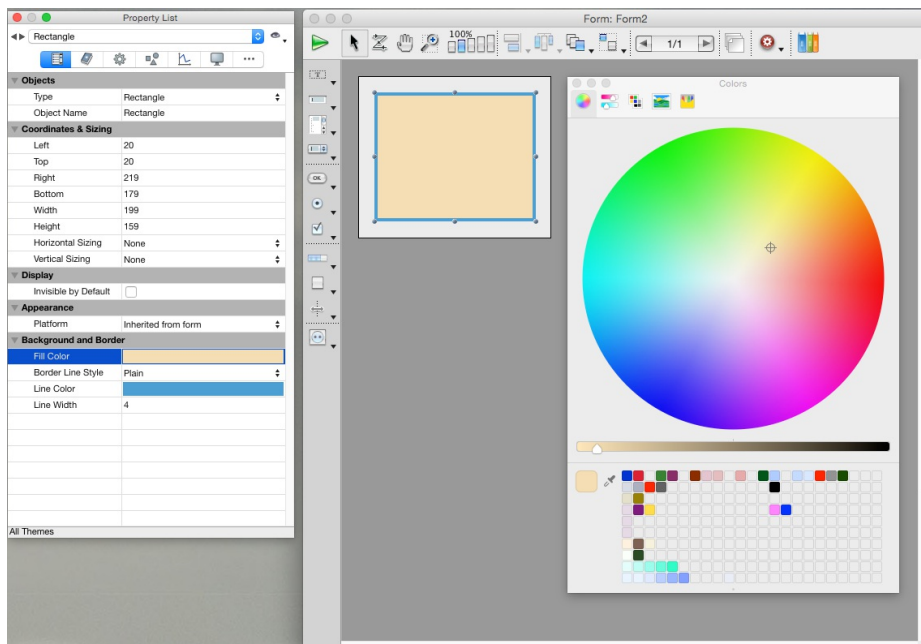
- 背景色
- 線カラー
- フォントカラー
- 交互背景色



## システム準拠のフローティングシステムカラーパレット(OS X)

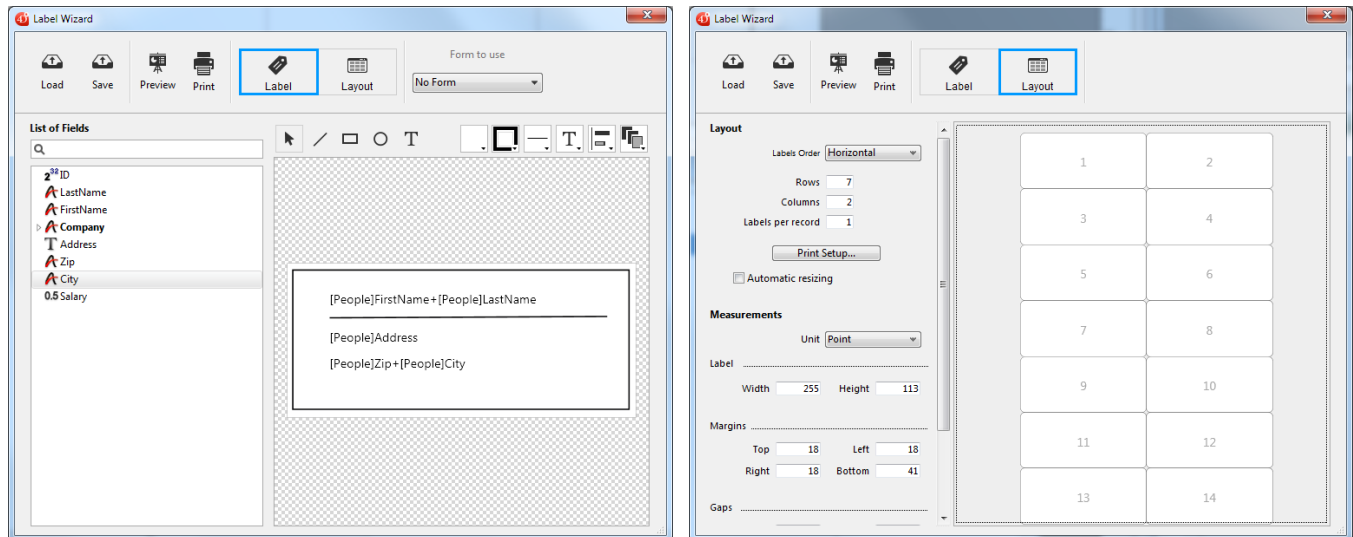
OS Xにおいて、ユーザーがカラーリストメニューのパーソナライズ...を選択したとき、フローティングのシステムカラーパレットが表示され、これはユーザーが閉じるボックスをクリックするまで開かれたままです。このカラーピッカーウィンドウはOS X-準拠であり、どのようなカラープロパティを選択してもこれを使用してカラーを変える事が可能です。

以下の例では、プロパティリストで"塗りカラー"プロパティが選択されていて、システムパレットで選択されたカラーは"塗りカラー"プロパティに適応されます:



## 新しいラベルエディター

64-bit版の4D Developer Editionには、将来のバージョンの4Dで現行のラベルエディターに取って代わる、新しい4Dラベルウィザードのプレビュー版が含まれます。この新しいウィザードでは新しいribbonベースのインターフェースとコンテキストメニューが特徴です：



4Dラベルウィザードは古いウィザードの機能の大部分をサポートする一方、新しい機能も提供します。

**注:** 4Dラベルウィザードの開発は引き続きステップバイステップ方式で行われます。

## 互換性

### ファイルフォーマット

新しいウィザードによって保存される 4Dラベルのファイル拡張子は ".4lb" です。このフォーマットは内部的にはXMLで書かれているので、オープンであるという点に注意してください。

以前のウィザードで作成された古い4Dラベルファイル (".4lb" 拡張子) はサポートされます。新しいウィザードはフォーマットを変えることなく、以前のラベルを読み込み、編集、保存することができます。

**注:** 以前のラベルファイルでは、ピクチャーは PICT フォーマットで保存されていました。このフォーマットは 4D ではもうサポートされないため (後述参照)、新しいエディターではこれらのピクチャーは赤いバツ印の付いたエリアで置き換えられます。

### サポートされない機能

以下の機能は、新しいラベルウィザードではサポートされません：

機能	補足
フォーマットメニュー	代わりに <b>String</b> コマンドを使用してください。例： <pre>String([Emp]Birth;internal date long)</pre>
アウトラインスタイル	4D でサポートされなくなりました。
パターン	4D でサポートされなくなりました。
Quickdraw PICT	4Dで廃止予定の機能です / 4D Developer Edition 64-bitではサポートされていません。
デフォルトロックボタン	今後は使用できません

### 無効化された機能

以下の機能は現在新しいラベルウィザードでは無効化されています：

## 機能

標準のコードポップアップメニュー

Windowsでの印刷

## 補足

4D Developer Edition 64-bit版(プレビュー)では無効化されています

## 新機能

### 使用可能なフォームとメソッドの定義

新しいラベルウィザードでは、ダイアログボックスで選択可能なフォームとメソッドを制限する機能が新しく追加されました。次のメニューが対象となります:

- "ラベル" ページの **使用するフォーム** ポップアップメニュー、または
- "レイアウト" ページの **適用するメソッド** ポップアップメニュー

選択可能なフォームとメソッドを指定するには、プロジェクトフォルダーに JSON ファイルを追加します。

**注:** 4D v15 R2 プレビューでは "使用するフォーム" ポップアップメニューの項目は特定のプロパティに基づいてフィルターされていました。このフィルターは取り除かれ、デフォルトですべてのテーブルフォームおよびプロジェクトメソッドが選択可能です。データベースがテーブルフォームを持たない場合には、このメニューは表示されません。

ラベルデザインに選択可能なフォームまたはメソッドを定義する手順は下のとおりです:

1. **label.json** という名前の JSON ファイルを作成し、データベースフォルダー内の **Resources** フォルダーにおきます。
2. このファイル内に、ラベルウィザードのメニュー内にて選択可能にしたいフォームまたはメソッド名を追加します。

**label.json** ファイルの中身は、以下のようになっています:

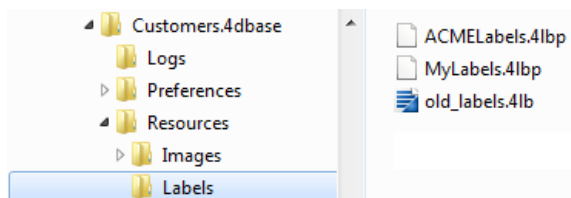
```
[ { "tableId":2, "forms":[], "methods":["myMethod1", "myMethod2"]}, { "tableId":1, "forms":["Sample Label 1", "Sample Label 2"], "methods":[] } ]
```

**labels.json** ファイルを設定しない場合には、メニューの選択項目は制限されません。

### ラベルファイルのプリロード

新しいラベルウィザードではラベルファイルをアプリケーション内に保存することができるため、ユーザーは直接 **読み込み** ボタンを使ってラベルデザインを選択し開くことができます。

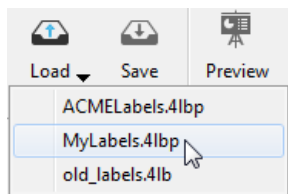
このためには、データベースの **Resources** フォルダー内に **Labels** フォルダーを作成し、そこにラベルファイルをコピーするだけです:



ラベルウィザードが起動するとこのフォルダーが検知され、中に有効なラベルファイルがあった場合には、**読み込み** ボタンにポップアップアイコンが追加されます:



これで、この一つのポップアップメニューからラベルデザインを選択することが可能になります。新しい ".4lbp" ファイルと旧式の ".4lb" ファイルは両方サポートされます:



### ズーム

ウィンドウの右下のリサイズカーソルをドラッグすることでラベルデザイン ("ラベル" ページ) をズームすることが可能です:



## ドラッグ&ドロップ

"ラベル" ページではドラッグ&ドロップがサポートされています:

- デスクトップからピクチャーファイルをラベルデザインエリアにドロップすることができます。
- デスクトップからラベルファイル (".4lbp" ファイルのみ) をラベルデザインエリアにドロップすることができます。

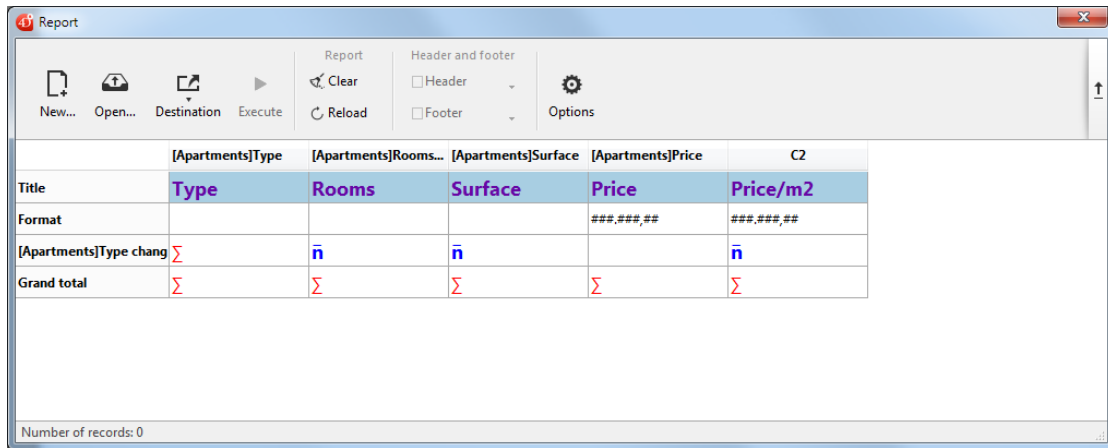
## 改善された操作性

一般的な操作性が改善されました:

- 静的なテキストはラベルデザインエリアに直接入力されます。
- グラフィックオブジェクトはコンテキストメニューを通して管理されます。
- 印刷プレビュー: 最初のページはラベルサイズを区切るすかし入りで印刷されます。

## 新しいクイックレポートエディター

4Dの64-bit版には新しく作成された4D クイックレポートエディターが含まれます。インターフェースを見て、少し時間をかけてこの新しいエディターの現在の機能を体験してみましょう。現代的なアーキテクチャーに基づき、このエディターは将来の4Dのバージョンにおいてカレントの古いエディターを全て置換します。直感的なribbonベースのインターフェースを使う事により、洗練された新しいレポートをデザイン・実行する事ができる一報で、以前の4Dのリリースにおいて作成されたものを取り扱う事もできます：

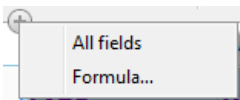


新しいクイックレポートエディターの主な特徴は以下の通りです：

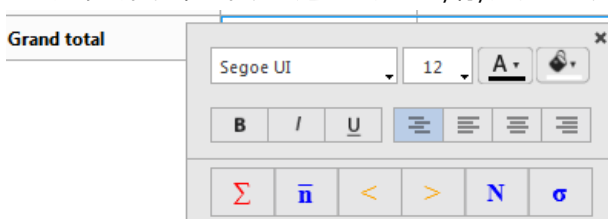
- 互換性：
  - 以前のリリースで作成されたレポートファイルとも互換性があります。フォーマットを変える事無く、古いレポートを開いたり、編集したり、保存したりする事ができます。
  - 既存のレポートの機能を全てサポートします。これにはランゲージコマンドも含まれます(ただしこのプレビューにおいて一時的に無効化されている機能を除きます。詳細は を参照して下さい)。
- 整頓されたインターフェース：
  - メニューバーの排除: 全ての機能はribbonベースのインターフェースを通して、またはポップアップメニューを通して使用する事ができます。
  - "ウィザードモード"の排除: 標準のインターフェースを使用し、自然なステップを通してレポートをデザイン・編集していきます。4Dコード生成ページはまだ提供されていません。

## 主な機能

- カラム (あるいはフォーミュラカラム)、または全てのフィールドの追加をコンテキストメニューから行えます：



- 総計行を定義することができます。
- コンテキストウィンドウを通して、セル/行/カラムのグラフィック属性を設定することができます：



## 一時的に無効化されている機能


この4D クイックレポートプレビューにおいては、いくつかの機能はご利用いただけません。これらは今後のリリースに置いて順次追加されて行く予定です：



- テンプレート
- クロステーブルレポート
- プレビュー
- レポートのフォームへの埋め込み(以前はプラグインエリアを使用)

## 4D v15 R4 - アップグレード - コマンドリスト (文字順)



 Active transaction