4D v11 SQL Release 3 (11.3)
ADDENDUM

4D v11 SQL Release 3 にようこそ。このドキュメントでは、以下に示す新しいバージョンの新機能や変更点について説明しています :

- 4D SQL エンジンおよびサーバの性能に関連するいくつかの新機能、特に**スキーマ**と (ODBC driver なしの) SQL による 4D データベース**直接接続**が追加されました。
- 4D 2004 から 4D v11 への**データベース変換**手順が変更され、同じストラクチャを複数回変換することが可能になりました。
- プログラムによる**エラー管理**が簡潔になりました。4D エンジンが生成するすべてのエラーは汎用的なコマンド **GET LAST ERROR STACK** でとらえることができます。
- **SOAP Web サービスを経由した 4D データベース**間のデータ交換が最適化されました。
- 新しく **XML DOM コマンド**が追加されました。

注 : 4D v11 SQL r3 では新しく **SVG component** が提供され、4D に統合された SVG 描画エンジンの利用が容易になりました。このコンポーネントについては別のドキュメントで説明しています。

SQL エンジン

スキーマ

4D v11 SQL に統合された SQL エンジンではスキーマのコンセプトが実装されています。この実装により、インタフェースの変更および新しい SQL コマンドが加えられました。スキーマは SQL コマンドを使用して作成、変更、そして削除されます。またインスペクタパレットの新しいオプションを使用して、テーブルをスキーマに割り当てることができます。

概要

スキーマはデータベーステーブルを格納した仮想オブジェクトです。SQL 中で、スキーマの目的は、データベースオブジェクトの異なるセットごとに特定のアクセス権を割り当てることにあります。

スキーマはデータベースを個々のエンティティに分割します。この分割したスキーマを合体させるとデータベース全体となります。つまり、あるテーブルは常に唯一つのスキーマに属します。

データベースが 4D v11 SQL r3 以降で作成あるいは変換されると、データベースのすべてのテーブルをグループ化するデフォルトのスキーマが作成されます。このスキーマは "DEFAULT_SCHEMA" と名付けられます。このスキーマは削除したり名称変更したりできません。

以前のバージョンの 4D では、SQL 経由のアクセス権はデータベース全体に対して設定されていました。今バージョンからは、スキーマごとに設定されます。スキーマごとに以下のいずれかのアクセスタイプを割り当てられます :

- 読み込みのみ (データ)
- 読み / 書き (データ)
- フル (データとストラクチャ)

注 : - 以前のデータベースをバージョン 11.3 以降に変換する際、(アプリケーション環境設定の SQL ページで定義される) 全体的なアクセス権がデフォルトスキーマに転送されます。
- 以前のバージョンと同様、アクセスコントロールは外部からの接続に対してのみ適用されます。**Begin SQL/End SQL タグ、SQL EXECUTE、QUERY BY SQL** 等により 4D 内部で実行される SQL コードは今バージョンでもフルアクセスを持っています。

データベースの Designer と Administrator のみがスキーマを作成、変更、削除できます。
4D のアクセス管理システムが有効になっていない場合 (つまり Designer パスワードが設定されていない場合)、すべてのユーザが制限なしにスキーマの作成や変更を行えます。

スキーマの作成

スキーマは以下の SQL コマンドを使用するプログラムでのみ作成できます :

CREATE SCHEMA Schema_Name

新しいスキーマを作成する際、デフォルトで割り当てられるアクセス権は以下のとおりです :

- 読み込みのみ (データ): Everybody
- 読み / 書き (データ): Everybody
- フル (データとストラクチャ): Nobody

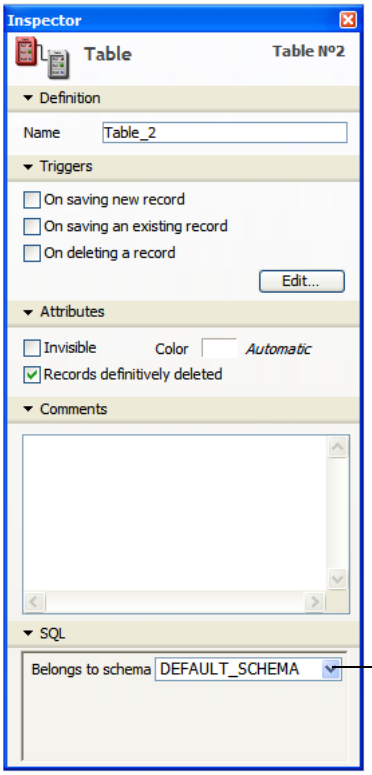
▶ "Accounting_Rights" という名前のスキーマを作成する例 :

```
CREATE SCHEMA Accounting_Rights
```

スキーマにテーブルを割り当てる

それぞれのテーブルはただ一つのスキーマに属します。インスペクタまたはプログラムを使用してテーブルにスキーマを割り当てます。

- **ストラクチャでの割り当て**
  ストラクチャウィンドウのテーブルインスペクタパレット中の SQL エリアで、データベースに定義されたスキーマが表示されるポップアップメニューにより、テーブルにスキーマを割り当てることができます :



- **プログラムによる割り当て**
  SQL ランゲージでテーブルを作成する際、スキーマにテーブルを割り当てることができます。

  ▶ テーブルを作成し、 それを MySchema1 スキーマに割り当てる :

  CREATE TABLE MySchema1.MyTable

  MYSCHEMA1 スキーマが存在しない場合、エラーが返されテーブルはデフォルトスキーマに割り当てられます。

  ▶ テーブルを作成し、 それをデフォルトスキーマに割り当てる :

  CREATE TABLE MyTable

- **割り当ての変更**
  スキーマの割り当てを変更するために、SQL の **ALTER TABLE** コマンドを使用できます :

  ALTER TABLE Table_Name SET SCHEMA Schema_Name

  ▶ "MyTable" テーブルを "MySchema2" スキーマに転送する :

  ALTER TABLE MyTable SET SCHEMA MySchema2

注 : システムテーブル (_USER_TABLES、_USER_COLUMNS、_USER_INDEXES、_USER_CONSTRAINTS、_USER_IND_COLUMNS、そして _USER_CONS_COLUMN) は特別なスキーマ **SYSTEM_SCHEMA** に割り当てられます。このスキーマをユーザが変更したり削除したりすることはできません。このスキーマはテーブルインスペクタのリストには表示されません。すべてのユーザに対し読み込みのみです。

スキーマの名称変更

SQL の **ALTER SCHEMA** コマンドを使用してスキーマの名称を変更できます :

ALTER SCHEMA former_name RENAME TO new_name

▶ "MyFirstSchema" スキーマの名称を "MyLastSchema" に変更する :

ALTER SCHEMA MyFirstSchema RENAME TO MyLastSchema

アクセス権の変更

スキーマに関連付けられているアクセス権を SQL の **GRANT** コマンドで変更できます :

GRANT [READ | READ_WRITE | ALL] ON Schema_Name TO 4D_User_Group

4D_User_Group には、スキーマにアクセス権を割り当てる 4D ユーザグループを指定します。

注 : 4D ではグループ名にスペースやアクセント付きキャラクタを許可していますが、標準 SQL では許可されていません。この場合、グループ名を [と] 文字の間に置きます。例えば : **GRANT READ ON [my schema] TO [the admins!]**

READ、**READ-WRITE**、および **ALL** キーワードは、環境設定の SQL ページで定義したアクセスタイプに対応します :

- **READ** 読み込みのみアクセスを確立します (データ)
- **READ_WRITE** は読み / 書きアクセスを確立します (データ)
- **ALL** はフルアクセスモードを確立します (データとストラクチャ)。

▶ "Power_Users" グループに、 MySchema1 スキーマのデータへの読み / 書きアクセスを許可したい場合 :

```
GRANT READ_WRITE ON MySchema1 TO POWER_USERS
```

アクセス権の削除

SQL の **REVOKE** コマンドを使用して、スキーマに割り当てられた特定のアクセス権を削除できます :

```
REVOKE [READ | READ_WRITE | ALL] ON Schema_Name
```

このコマンドを実行すると、Nobody 疑似グループが指定されたアクセス権に割り当てられます。

▶ MySchema1 スキーマへのすべての読み / 書きアクセスを削除する場合 :

```
REVOKE READ_WRITE ON MySchema1
```

スキーマの削除

デフォルトスキーマを除き、スキーマを削除できます。スキーマを削除すると、それに割り当てられていたすべてのテーブルはデフォルトスキーマに転送されます。転送されたテーブルはデフォルトスキーマのアクセス権を継承します。

スキーマの削除は SQL の **DROP SCHEMA** コマンドを使用して行います :

```
DROP SCHEMA Schema_name
```

▶ (Table1 と Table2 が割り当てられた) MyFirstSchema スキーマを削除する :

```
DROP SCHEMA MyFirstSchema
```

この処理の後、Table1 と Table2 はデフォルトスキーマに再割り当てされます。

存在しない、あるいは既に削除されたスキーマを削除しようとすると、エラーが生成されます。

参照整合性

4D はアクセス権とは独立して、参照整合性の原則を確保します。例えば 2 つのテーブル Table1 と Table2 があり、N 対 1 タイプのリレーションで接続されているとします (Table2 -> Table1)。Table1 はスキーマ S1 に、Table2 はスキーマ S2 に属します。

スキーマ S1 へのアクセス権を持っているが、S2 へのアクセス権を持たないユーザは Table1 のレコードを削除することができます。この場合、参照整合性の原則を守るため、Table1 から削除されるレコードにリレートするすべての Table2 のレコードも削除されます。

システムテーブル

4D への SQL スキーマの実装により **_USER_SCHEMAS** システムテーブルが追加されました :

_USER_SCHEMAS		データベーススキーマ定義
SCHEMA_ID	INT32	スキーマ番号
SCHEMA_NAME	VARCHAR	スキーマ名
READ_GROUP_ID	INT32	読み込みのみアクセスをもつグループ番号
READ_GROUP_NAME	VARCHAR	読み込みのみアクセスをもつグループ名
READ_WRITE_GROUP_ID	INT32	読み / 書きアクセスをもつグループ番号
READ_WRITE_GROUP_NAME	VARCHAR	読み / 書きアクセスをもつグループ名
ALL_GROUP_ID	INT32	フルアクセスをもつグループ番号
ALL_GROUP_NAME	VARCHAR	フルアクセスをもつグループ名

さらに **_USER_TABLES** システムテーブルに **SCHEMA_ID** カラムが追加され、テーブルが属するスキーマが格納されます。

スキーマの読み込みと書き出し

4D ではシステムテーブルを使用して、データベースに定義されたスキーマを書き出して読み込むことができます。これはストラクチャを更新する際に特に便利です : 新しいストラクチャにスキーマ定義を読み込むだけで即座に運用可能となります。

これを行うには新しいシステムテーブル **_USER_SCHEMAS** を使用します。以下のようなメソッドを使用して、スキーマに関連するデータを暗号化した BLOB に格納できます :

```
Begin SQL
  SELECT * from [_USER_SCHEMAS] INTO Array1,Array2,Array3...;
End SQL
VARIABLE TO BLOB(Array1;SchemaBlob;*)
VARIABLE TO BLOB(Array2;SchemaBlob;*)
VARIABLE TO BLOB(Array3;SchemaBlob;*)
...
```

そして BLOB をテキストファイルやフィールドに保存します。

データベースへのスキーマ読み込みは、以下のコードで示す方法で実行できます :

```
  ` 配列変数を初期化
BLOB TO VARIABLE(SchemaBlob;Array1;Offset)
BLOB TO VARIABLE(SchemaBlob;Array2;Offset)
BLOB TO VARIABLE(SchemaBlob;Array3;Offset)
...
  ` 現在の _USER_SCHEMAS システムテーブルを
  ` BLOB の内容で置き換える
$Execute:="INSERT into [_USER_SCHEMAS] ID, SchemaName, ...VALUES
("...)
Begin SQL
  EXECUTE IMMEDIATE $Execute;
End SQL
```

注 : 当然ながら、読み込みを行うユーザは適切なアクセス権を持っていなければなりません (Designer または Administrator でなければなりません)。

INSERT

INSERT コマンドに 2 つの新しい機能が追加されました :

- ファイルコンテンツの挿入
- 複数ロー挿入

ファイルコンテンツの挿入 (INFILE)

INSERT コマンドは外部ファイルのコンテンツを使用して、新規レコードのデータを設定できるようになりました。これを行うために、コマンドはオプションの **INFILE** キーワードを受け入れます :

```
INSERT INTO {sql_name | sql_string}
[(column_ref, ..., column_ref)]
{VALUES([INFILE]{arithmetic_expression |NULL}, ...,
[INFILE]{arithmetic_expression |NULL}) |subquery}
```

INFILE キーワードは VARCHAR タイプの式とともに使用しなければなりません。**INFILE** キーワードが渡されると、arithmetic_expression の値はファイルパス名として評価されます。ファイルが存在すれば、ファイルのコンテンツが対応するカラムに挿入されます。テキストまたは BLOB タイプのフィールドのみが **INFILE** からの値を受け取れます。ファイルのコンテンツは生データとして変換されずに転送されます。

クエリがリモートクライアントから実行される場合でも、検索するファイルは SQL エンジンをホストするコンピュータ上になければなりません。同様に、パス名は SQL エンジンの OS のシンタックスで記述しなければなりません。

複数ロー挿入

4D に統合された SQL エンジンは、複数ローの値の挿入を受け入れます。これによりコードを簡潔および最適にできます。複数ロー挿入のシンタックスを使用するコードの実行は、とくに大量のデータを挿入する際に最適化されます。

複数ロー挿入のシンタックスは以下のとおりです :

```
INSERT INTO {sql_name | sql_string}
[(column_ref, ..., column_ref)]
VALUES(arithmetic_expression, ..., arithmetic_expression), ...,
(arithmetic_expression, ..., arithmetic_expression);
```

このシンタックスにより、以前のバージョンでは必要だったローの繰り返しを避けることができます。例えば :

```
Begin SQL
  INSERT INTO MyTable (Field1,Field2,BoolField,DateField,TimeField,
        InfoField) VALUES (1,1,1,'11/01/01','11:01:01','First row');
  INSERT INTO MyTable (Field1,Field2,BoolField,DateField,TimeField,
        InfoField) VALUES (2,2,0,'12/01/02','12:02:02','2nd row'),
.....
  INSERT INTO MyTable (Field1,Field2,BoolField,DateField,TimeField,
        InfoField) VALUES (7,7,1,'17/01/07','17:07:07','7th row');
End SQL
```

これは以下のように記述できます :

```
Begin SQL
  INSERT INTO MyTable
    (Field1,Field2,BoolField,DateField,TimeField, InfoField)
    VALUES
      (1,1,1,'11/01/01','11:01:01','First row'),
      (2,2,0,'12/01/02','12:02:02','2nd row'),
```

```
        (3,3,1,'13/01/03','13:03:03','3rd row'),
        .........
        (7,7,1,'17/01/07','17:07:07','7th row');
End SQL
```

変数も使用できます。例えば :

```
vid1:=1
vidx1:=1
vbol1:=1
vdate1:= !11/01/01!
vtime1:=?11:01:01?
vtext1:="First row"
  ` 複数ロー挿入
Begin SQL
  INSERT INTO MyTable
     (Field1,Field2,BoolField,DateField,TimeField, InfoField)
     VALUES
        (:vid1, :vidx1, :vbol1, :vdate1, :vtime1, :vtext1),
        (2,2,0,'12/01/02','12:02:02','2nd row'),
        .........
        (7,7,1,'17/01/07','17:07:07','7th row');
End SQL
```

このシンタックスで配列を使用すると、特に便利です :

```
ARRAY TEXT(vArrId;0)
ARRAY TEXT(vArrIdx;0)
ARRAY TEXT(vArrText;0)
ARRAY BOOLEAN(vArrbool;0)
ARRAY DATE(vArrdate;0)
ARRAY LONGINT(vArrL;0)
...

Begin SQL
  INSERT INTO MyTable
     (Field1,Field2,BoolField,DateField,TimeField, InfoField)
     VALUES (:vArrId, :vArrIdx, :vArrbool, :vArrdate, :vArrL, :vArrText);
End SQL
```

注 : 同じ **INSERT** ステートメント中で変数と配列を組み合わせて使用することはできません。

SELECT

LIMIT および OFFSET 節で、SELECT コマンドは 4D 変数への参照を受け入れます。例えば、以下のように記述できます :
SELECT... OFFSET :var1 LIMIT :var2 ...

SQL サーバ

SQL を使用した 2 つの 4D 間の直接接続

4D v11 SQL r3 では SQL を使用して、4D アプリケーションが直接他の 4D データベースに接続し、データ交換を行うことができます。前のバージョンでは、この処理は ODBC プロトコルを経由しなければなりませんでした。直接接続の主たるメリットはデータ交換が速いことです。

概要

直接の SQL 接続には **SQL LOGIN**[1] コマンドを使用します。ODBC データソースやカレントデータベースに加え、このコマンドを使用して SQL クエリを行う接続先として直接 4D Server データベースを指定できます。4D Server は IP アドレスまたは公開名で指定できます。

影響を受ける SQL クエリは **SQL EXECUTE** コマンドで行われたものおよび **Begin SQL / End SQL** タグで囲まれたものです (* 引数が渡された場合)。セッションを終了するには **SQL LOGOUT** コマンドを使用します。

SQL LOGIN コマンドの詳細なシンタックスについてはページ 14、"SQL LOGIN" の段落を参照してください。

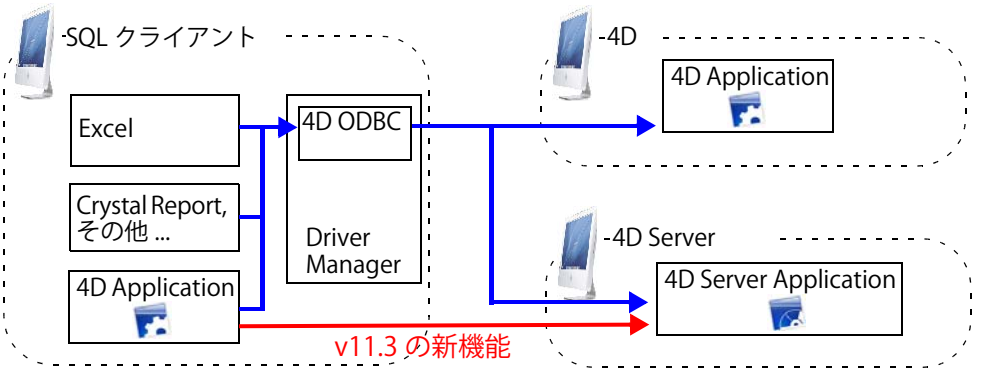
この変更に伴い、既存のコマンドが再構成されました (ページ 12 "ODBC コマンドの名称変更 " の段落参照)。

設定

4D Server v11 SQL アプリケーションだけが他の 4D アプリケーションからの SQL クエリに直接応答できます。

同様に、4D アプリケーションの "Professional" 製品ラインのみが、他の 4D アプリケーションに対して直接の接続を開くことができます。

4D SQL サーバへの可能な外部アクセスは以下のとおりです :



1. SQL LOGIN は以前の ODBC LOGIN コマンドが名称変更された名前です (ページ 12 "ODBC コマンドの名称変更 " の段落参照)。

動作について

直接接続の際、同期とデータ整合性の問題を排除するため、データ交換は自動で同期モードで行われます。

プロセスあたり 1 つの接続のみが許可されます。同時に複数の接続を行いたいときは、必要なだけプロセスを作成しなければなりません。

直接接続は TCP/IP プロトコルで行われます。接続先 4D Server の環境設定の SQL ページで **SSL を有効にする**オプションがチェックされていれば、通信は保護されます。

直接接続は SQL サーバが開始された 4D Server のみにより、許可されます。

クエリのエンコーディング

外部 SQL クエリで 4D が使用するエンコーディングを **SQL SET OPTION** コマンドで指定できるようになりました。

SQL SET OPTION

SQL SET OPTION (option; value)

引数	型		説明
option	倍長整数	→	設定するオプションの番号
value	倍長整数	→	オプションの新しい値

注 : **SQL SET OPTION** は **ODBC SET OPTION** コマンドの新しい名称です (ページ 12、"ODBC コマンドの名称変更 " の段落参照)。

SQL SET OPTION を使用して、(SQL パススルーで) 外部ソースに送信されるクエリに使用されるエンコーディングを指定できるようになりました。

注 : **SQL GET OPTION** コマンドを使用して現在の設定を取得できます。

これを行うために、新しい定数 **SQL Charset** (値 100) が "SQL" テーマに追加されました。option 引数にこの定数を渡す場合、value 引数には使用する文字セットの MIBEnum ID を渡さなければなりません。MIBEnum 番号は以下のアドレスで参照できます :
http://www.iana.org/assignments/character-sets.

例えば、Shift_JIS をエンコーディングに使用したい場合、以下のステートメントを実行します :

```
SQL SET OPTION(SQL Charset;17)
```

デフォルトで 4D は UTF-8 (値 106) エンコーディングを使用します。

SQL SET OPTION コマンドを使用してエンコーディングを変更したとき、その変更はカレントプロセスのカレント接続に影響します。

コマンドが正しく実行されると OK システム変数に 1 が設定されます。そうでない場合、例えば有効な外部接続がない状態で **SQL SET OPTION** コマンドを実行すると、0 が設定されます。

ODBC コマンドの名称変更

4D Server への直接接続の実装に伴い、既存のコマンドが再構成されました。" 外部データソース " テーマコマンドのスコープが拡張されました。コマンドの名称に新しい接頭辞が付けられ、いくつかのコマンドが廃止されました。

- 今まで "ODBC" 接頭辞が付けられていた " 外部データソース " テーマのすべてのコマンドは名称が変更されました。今バージョンより "SQL" が接頭辞となります。例えば **ODBC LOGIN** は **SQL LOGIN** となります。
  これらの通信コマンドは今バージョンより ODBC プロトコルを必要としない直接の SQL クエリでも使用できるようになったため、この変更が行われました。
- 明確にするため、"SQL" 接頭辞が付けられたコマンドはすべて "SQL" テーマに移動されました。" 外部データソース " テーマは削除されました。
- このテーマに関連付けられた定数も "SQL" 接頭辞が付けられ、" 外部データソース " 定数テーマも "SQL" に名称変更されました。
- **SQL LOGIN** コマンドは直接の SQL クエリおよびデータソースを必要とするクエリで使用できます (内部データベース、ODBC ソース、および 4D Server データベース)。また **SQL LOGOUT** コマンドは開かれた接続と閉じるために使用します。
  USE INTERNAL DATABASE と **USE EXTERNAL DATABASE** コマンドはもう必要ありません。これらのコマンドの利用は推奨されず、今後のバージョンではメンテナンスされません。

以上の変更を表にまとめます :

4D v11 SQL	**4D v11 SQL r3**
コマンド	
" 外部データソース " テーマ	削除 コマンドは "**SQL**" テーマに移行
ODBC CANCEL LOAD	**SQL CANCEL LOAD**
ODBC End selection	**SQL End selection**
ODBC EXECUTE	**SQL EXECUTE**
ODBC EXPORT	**SQL EXPORT**
ODBC GET LAST ERROR	**SQL GET LAST ERROR**
ODBC GET OPTION	**SQL GET OPTION**
ODBC IMPORT	**SQL IMPORT**
ODBC LOAD RECORD	**SQL LOAD RECORD**
ODBC LOGIN	**SQL LOGIN**
ODBC LOGOUT	**SQL LOGOUT**
ODBC SET OPTION	**SQL SET OPTION**
ODBC SET PARAMETER	**SQL SET PARAMETER**
"SQL" テーマ	
USE INTERNAL DATABASE	廃止。**SQL LOGOUT** を使用
USE EXTERNAL DATABASE	廃止。**SQL LOGIN** を使用
定数	
" 外部データソース " テーマ	"**SQL**" に名称変更
ODBC All Records	**SQL All Records**
ODBC Asynchronous	**SQL Asynchronous**
ODBC Connection Time Out	**SQL Connection Time Out**
ODBC Max Data Length	**SQL Max Data Length**
ODBC Max Rows	**SQL Max Rows**
ODBC Param In	**SQL Param In**
ODBC Param In Out	**SQL Param In Out**
ODBC Param Out	**SQL Param Out**
ODBC Query Time Out	**SQL Query Time Out**

SQL LOGIN

SQL LOGIN{(dataEntry; userName; password{;*})}

引数	型		説明
dataEntry	テキスト	→	外部データソース名 または 外部データソースの IP アドレス または ODBC データソース名 または "" で選択ダイアログ表示
userName	テキスト	→	ユーザ名
password	テキスト	→	パスワード
*	*	→	Begin SQL/End SQL に適用 省略時 : いいえ (内部データベース) 指定時 : はい

注 : **SQL LOGIN** は **ODBC LOGIN** コマンドの新しい名称です。

SQL LOGIN コマンドは dataEntry 引数に指定した SQL データソースとの接続を開くために使用します。このコマンドはアプリケーション内で以降実行される SQL コードのターゲットを指定します :

- **SQL EXECUTE**
- **Begin SQL** / **End SQL** タグの間に置かれたコード (* 引数が渡された場合、後述参照)。

SQL データソースは以下のいずれかです :

- 外部 4D Server データベース (4D v11 SQL r3 の新機能)
- 外部 ODBC ソース
- 内部 SQL エンジン

dataEntry には以下のいずれかの値を渡します :

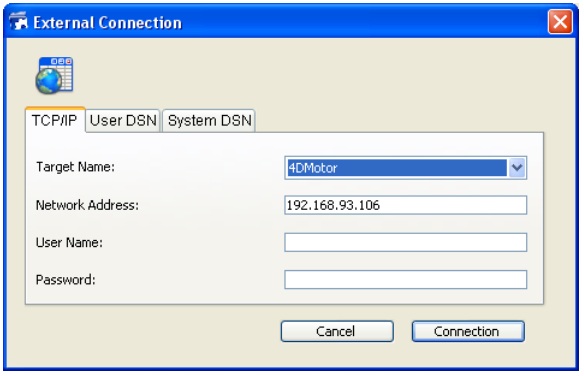
- **IP アドレス**
  シンタックス : IP:<IPAddress>{:<TCPPort>}
  この場合、コマンドは指定された IP アドレスのマシン上で動作する 4D Server との直接接続を開きます。" ターゲット " マシン上で SQL サーバが開始されていなければなりません。" ターゲット " データベースの SQL サーバ用に指定された公開ポートを TCP ポート番号として渡すこともできます。TCP ポート番号を渡さない場合、デフォルトポート (19812) が使用されます。SQL サーバの TCP ポート番号はアプリケーション環境設定の SQL/ 設定ページで変更できます。
  例題 1 と 2 を参照。

- **4D データベース公開名**
  シンタックス : 4D:<Publication_Name>
  この場合、コマンドはネットワーク上の公開名が指定した名前と一致する 4D Server データベースとの直接接続を開きます。データベースのネットワーク公開名はアプリケーション環境設定のクライアント - サーバ / 設定ページで設定できます。
  例題 4 を参照

注 : (4D データベースを公開する) ターゲットの 4D SQL サーバのポート番号と、接続を開こうとする 4D アプリケーションの SQL サーバポート番号は同じでなければなりません。

- 有効な **ODBC データソース名**
  シンタックス : ODBC:<My_DSN> または <My_DSN>
  この場合、dataEntry 引数には ODBC ドライバマネージャに設定したデータソース名を渡します。この方式は **ODBC LOGIN** コマンドの以前の動作に対応します。
  "ODBC:" 接頭辞を使用しないシンタックスは以前のバージョンとの互換性を保つために保持されています。しかしコードの可読性のために、"ODBC:" 接頭辞の使用をお勧めします。
  例題 4 を参照

- **空の文字列**
  シンタックス : ""
  この場合、コマンドは接続ダイアログを表示し、接続先データソースをマニュアルで指定できます :



このダイアログボックスには複数のページがあります。**TCP/IP** ページには以下の要素があります :

- **ターゲット名** : このメニューは 2 つのリストを使用して作成されます :
  - 直接接続を使用して最近開かれたデータベースのリスト。このリストの更新メカニズムは 4D アプリケーションのそれと同じです。ただし .4DLink ファイルが格納されるフォルダ名は "Favorites v11" ではなく "Favorites SQL v11" です。
  - SQL サーバ開始され、SQL 接続の TCP 公開ポートが接続元アプリケーションと同じ 4D Server アプリケーションのリスト。このリストは dataEntry 引数なしで **SQL LOGIN** コマンドが呼び出されるたびに動的に更新されます。
  "^" 文字がデータベース名の前に置かれている場合、接続は SSL を使用した保護モードで行われることを示しています。
- **ネットワークアドレス**: このエリアにはターゲット名メニューで選択したデータベースの IP アドレス (と TCP ポート番号) が表示されます。またこのエリアに IP アドレスを入力して**接続**ボタンをクリックし、対応する 4D Server データベースに接続することもできます。TCP ポートを指定する必要がある場合はアドレスの後にコロン (:) を置いて入力します。例えば : 192.168.93.105:19855。
- **ユーザ名**と**パスワード**: これらのエリアには接続識別情報を入力するために使用します。

ユーザ DSN と**システム DSN** ページには、マシンの ODBC アドミニストレータに定義されたユーザおよびシステム ODBC データソースが表示されます。これらのページではデータソースを選択し、外部 ODBC データソースへの接続を開くために識別情報を入力できます。

接続が確立されると、OK システム変数は 1 に設定されます。そうでない場合は 0 に設定され、エラーが生成されます。このエラーは **ON ERR CALL** コマンドでインストールされるエラー処理メソッドでとらえることができます。

- **SQL_INTERNAL** 定数
  シンタックス : SQL_INTERNAL
  この場合、コマンドは続く SQL クエリをデータベースの内部 SQL エンジンに転送します。

互換性のためにオプションの * 引数が追加されました。以前のバージョンの 4D では、**ODBC LOGIN** コマンドは **Begin SQL/End SQL** タグに囲まれた SQL コードに影響しませんでした (これを行うには **USE INTERNAL DATABASE** と **USE EXTERNAL DATABASE** コマンドを使用する必要がありました)。既存のデータベースの動作を変更しないようにするため、**SQL**

LOGIN を * 引数なしで呼び出した場合 **Begin SQL/End SQL** タグ内の SQL コードのターゲットは変更されません。
Begin SQL/End SQL タグの間に置いたコードが **SQL LOGIN** コマンドで指定したターゲットに適用されるようにするには * 引数を渡す必要があります。

注 : 直接接続の場合、userName と password に空の文字列を渡すと、接続はターゲットデータベースで 4D パスワードシステムが有効でない場合にのみ受け入れられます。パスワードシステムが有効の場合、接続は拒否されます。

カレントの接続を閉じてメモリを解放するには、**SQL LOGOUT** コマンドを実行します。その後すべての SQL クエリはデータベースの内部 SQL エンジンに送信されます。
カレントの接続を明示的に閉じずに **SQL LOGIN** を呼び出した場合、その接続は自動で閉じられます。

- 例 1: IP アドレス 192.168.45.34 のマシン上で、 デフォルト TCP ポートで待ち受けを行う 4D Server v11 アプリケーションとの直接接続を開きます。 **SQL EXECUTE** コマンドで実行される SQL クエリはこの接続に転送されます。 **Begin SQL/End SQL** タグに含まれるクエリは転送されません。

```
SQL LOGIN("IP:192.168.45.34";"John";"azerty")
```

- 例 2: IP アドレス 192.168.45.34 のマシン上で、 TCP ポート 20150 で待ち受けを行う 4D Server v11 アプリケーションとの直接接続を開きます。 **SQL EXECUTE** コマンドで実行される、 および **Begin SQL/End SQL** タグに含まれる SQL クエリはこの接続に転送されます。

```
SQL LOGIN("IP:192.168.45.34:20150";"John";"azerty";*)
```

- 例 3: ローカルネットワーク上で公開名 "Accounts_DB" で公開されている 4D Server v11 アプリケーションとの直接接続を開きます。 (環境設定の SQL/ 設定ページで設定する) SQL サーバ用の TCP ポートは両方のデータベースで同じでなければなりません (デフォルトで 19812)。 **SQL EXECUTE** コマンドで実行される SQL クエリはこの接続に転送されます。 **Begin SQL/End SQL** タグに含まれるクエリは転送されません。

```
SQL LOGIN ("4D:Accounts_DB";"John";"azerty")
```

- 例 4: "MyOracle" という名称の外部データソースに ODBC プロトコルで接続を開きます。 **SQL EXECUTE** コマンドで実行される、 および **Begin SQL/End SQL** タグに含まれる SQL クエリはこの接続に転送されます。

```
SQL LOGIN ("ODBC:MyOracle";"Scott";"Tiger";*)
```

▶ この例は **SQL LOGIN** コマンドで提供される、 可能な接続を示します :

```
ARRAY TEXT (30;aNames)
ARRAY LONGINT(aAges;0)
SQL LOGIN("ODBC:MyORACLE";"Mark";"azerty")
If (OK=1)
   ` 以降のクエリは外部の ORACLE データベースに送られます
  SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames; aAges)
   ` 以下のクエリはローカルの 4D データベースに送られます
  Begin SQL
    SELECT Name, Age
    FROM PERSONS
    INTO :aNames, :aAges;
  End SQL
   ` 次の SQL LOGIN で ORACLE データベースへの接続は閉じられ、
   ` 外部の MySQL データベースへの新しい接続が開かれます
  SQL LOGIN ("ODBC:MySQL";"Jean";"qwerty";*)
  If (OK=1)
     ` 以下のクエリは外部 MySQL データベースに転送されます
    SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames; aAges)
     ` 以下のクエリも外部 MySQL データベースに転送されます
    Begin SQL
      SELECT Name, Age
      FROM PERSONS
      INTO :aNames, :aAges;
    End SQL
    SQL LOGOUT
     ` 以下のクエリはローカルの 4D データベースに送信されます
    Begin SQL
      SELECT Name, Age
      FROM PERSONS
      INTO :aNames, :aAges;
    End SQL
  End if
End if
```

SQL 接続用の SSL ファイルの場所

4D SQL サーバで接続の処理に SSL プロトコルを使用する場合、以下を行う必要があります :

- アプリケーション環境設定の SQL/ 設定ページで **SSL を有効にする**オプションをチェックする。
- key.pem と cert.pem ファイルを以下の場所にコピーする : **MyDatabase/Preferences/SQL** ("MyDatabase" はデータベースフォルダ / パッケージを表します)。

On SQL Authentication データベースメソッド

On SQL Authentication データベースメソッドの動作が変更され、4D SQL サーバのセキュリティが強化されました。

今バージョンより、接続およびクエリが受け入れられるためには、$0 が **True** を返しかつ **CHANGE CURRENT USER** コマンドが呼び出されて正しく実行されなければなりません。

前のバージョンでは、$0 が **True** を返せば、**CHANGE CURRENT USER** コマンドの呼び出しに関係なく接続が受け入れられていました。このコマンドの呼び出しとその正しい実行は、クエリが受け入れられるための必須条件となりました。

Flash Player クエリサポートの追加

環境設定の SQL/ 設定ページに追加された新しい **Flash Player のリクエストを許可する** オプションは、4D SQL サーバで、Flash Player リクエストをサポートするメカニズムを有効にするために使用します。

このメカニズムは、データベースの Preferences フォルダ内で (Preferences/SQL/Flash/)、"socketpolicy.xml" という名前のファイルの存在に基づいています。このファイルは、Flash Player により必要とされ、ドメインをまたいだ接続や Flex (Web 2.0) アプリケーションによるソケット接続を可能にします。

4D の以前のバージョンでは、このファイルを手動で追加する必要がありました。これからは、**Flash Player のリクエストを許可する**オプションでこの機能を有効にします。このオプションをチェックすることにより、Flash Player リクエストが受け入れられ、必要に応じて、データベース用の汎用 "socketpolicy.xml" ファイルが作成されます。

このオプションを選択解除すると、"socketpolicy.xml" ファイルは、無効（名前が変更される）となります。SQL サーバが、後に受け取るすべての Flash Player クエリは拒否されます。

データベースを開いた時、このオプションがチェックされているか、されていないかは、データベースの Preferences フォルダに有効な "socketpolicy.xml" ファイルが存在するかしないかによって決定されます。

データベースを複数回変換する

バージョン 200x からバージョン 11 への変換メカニズムが変更され、(データファイルが 1 回しか変換されないときでも) 同じストラクチャファイルを複数回変換することが容易になりました。

このシナリオは、アプリケーションがバージョン 2004 で開発されていて、運用のためにそれを v11 に変換して運用を開始し、開発が進むたびにストラクチャファイルを v11 に変換するような時に起こります。このケースではストラクチャファイルのみの変換が必要です。

このようなケースでアプリケーションを正しく機能させるため、変換時に catalog.xml という名前のファイルが v11 のファイルと同階層に作成されます。このファイルには変換されたデータベースストラクチャ情報と、v11 で使用されている新しい内部的な識別情報が格納されています。同じデータファイルを、何度か変換されたそれぞれのストラクチャファイルで利用できるようにするには、このファイルが必要です。

WEDD 環境設定の削除

以前の 4D では、ストラクチャファイルとデータファイルを関連付けるために、任意の文字列を指定することができました (環境設定の WEDD " ストラクチャファイルとデータファイル間のリンク署名 " オプション)。

今バージョンからは、このメカニズムは使用されません。データファイルとストラクチャファイル間のリンクは常に使用され、内部的な番号を使用した自動的なものとなります。WEDD オプションは環境設定から取り除かれました。4D はデータベース作成あるいは変換時に、自動で内部的なリンク番号を割り当てます。

プログラムによるエラー管理

コード実行時に発生するエラーの管理が、4D v11 SQL r3 で最適化及び簡易化されました。

エラーをとらえる方法は変更されていません : **ON ERR CALL** コマンドでインストールされたメソッドがエラーを受け取り、そのメソッドでエラーを処理できます。同様にこのメソッド内で Error システム変数を使用して、現在のエラー番号を知ることができます。

他方 1 つのコマンド、**GET LAST ERROR STACK** を使用して、とらえたエラーの情報を取得できるようになりました。エラー処理メソッドの内部で、コンテキストに関わらず、このコマンドを使用してすべてのタイプのエラーを処理できます (後述参照)。

以前のバージョンでは、実行コンテキストにより複数のコマンド (**GET XML ERROR**, **GET XSLT ERROR**, **Get Web Service error info** 等) を使用しなければなりませんでした。

注 : **GET LAST ERROR STACK** は **GET LAST SQL ERROR** コマンドの新しい名称です。このコマンドのスコープは拡張され、また機能を明確にするために " 割り込み " テーマに移動されました。

他のエラー情報コマンドは互換性の目的で保持されていますが、その実用性は制限されます :

- **Get Web Service error info**, Web サービス (クライアント) テーマ
- **ODBC GET LAST ERROR** (**SQL GET LAST ERROR** に名称変更), SQL テーマ (ページ 12 "ODBC コマンドの名称変更 " の段落参照)
- **GET XML ERROR**, XML ユーティリティ テーマ
- **GET XSLT ERROR**, XML ユーティリティ テーマ

GET LAST ERROR STACK

GET LAST ERROR STACK(codesArray; intCompArray; textArray)

引数	型		説明
codesArray	数値配列	←	エラー番号
intCompArray	文字配列	←	内部コンポーネントコード
textArray	文字配列	←	エラーテキスト

注 : **GET LAST ERROR STACK** は **GET LAST SQL ERROR** コマンドの新名称です。

GET LAST ERROR STACK は、4D アプリケーションの現在のエラー " スタック " に関する情報を返します。4D のステートメントがエラーを起こしている場合、現在のエラースタックにはそのエラーの説明と、さらに生成された一連のエラーが含まれます。例えば、" ディスクがいっぱい " のようなエラーが発生すると、ファイルへの書き込みエラーとなり、さらにレコード保存コマンドのエラーとなります。スタックには 3 つのエラーが含まれます。直前の 4D ステートメントがエラーを生成していない場合、現在のエラースタックは空です。

この汎用コマンドを使用して、発生するすべてのエラーを処理できます。しかし、ODBC ソースが生成するエラーに関する詳細な情報を取得するには、**SQL GET LAST ERROR** コマンドを使用する必要があります。

GET LAST ERROR STACK は、**ON ERR CALL** コマンドでインストールされるエラー処理メソッド内で使用しなければなりません。

テーマ : 割り込み

Web サービス (クライアント)

4D Web サービス間の最適化

Web サービスを使用した 4D アプリケーション間のデータ交換が 4D v11 SQL r3 で最適化されました。この最適化に伴い、**SET WEB SERVICE OPTION** コマンドと **SET DATABASE PARAMETER** コマンドに新しいオプションが追加されました (ページ 23 "4D 環境 " の段落参照)。

SET WEB SERVICE OPTION

SET WEB SERVICE OPTION(option; value)

引数	型		説明
option	倍長整数	→	設定するオプションのコード
value	倍長整数 | テキスト	→	オプションの値

SET WEB SERVICE OPTION コマンドを使用して SOAP リクエストにおける内部的な圧縮メカニズムを有効にし、4D アプリケーション間のデータ交換速度を向上させることができます。

この目的のために、"Web Services (Client)" テーマに新しい 2 つの定数が追加されました : (**Web Service Deflate Compression** と **Web Service No Compression**)。

SOAP リクエストの圧縮を有効にするには、4D の Web サービスクライアントで以下のステートメントを実行します :

SET WEB SERVICE OPTION(Web Service HTTP Compression; Web Service Deflate Compression)

注 : "Deflate" は 4D の内部で使用される圧縮アルゴリズムの名前です。

この場合、Web サービスクライアントから送信される次回の SOAP リクエストのデータは、4D SOAP サーバに送信される前に、標準の HTTP メカニズムを使用して圧縮されます。サーバはリクエストを解凍・解析し、同じメカニズムを使用して自動で応答します。

注 : このメカニズムは 11.3 より前のバージョンの 4D SOAP サーバにリクエストを送信する際には使用できません。

SET WEB SERVICE OPTION コマンドに続く呼び出しのみに効果があります。つまり圧縮を行うリクエストのたびにこのコマンドをよびださなければなりません。

デフォルトで、4D は Web サービス HTTP リクエストを圧縮しません。

この動作を最適化するために、リクエストの圧縮率と敷居値を設定する追加のオプションがあります。これらのオプションは **SET DATABASE PARAMETER** で設定できます (ページ 23 "4D 環境 " の段落参照)。

AUTHENTICATE WEB SERVICE

AUTHENTICATE WEB SERVICE (name; password{; authMethod}{;*})

引数	型		説明
name	テキスト	→	ユーザ名
password	テキスト	→	パスワード
authMethod	倍長整数	→	認証メソッド
*	*	→	指定時 : proxy による認証

AUTHENTICATE WEB SERVICE は最後の引数としてアスタリスク (*) を受け入れます。この引数を渡すと、認証情報は HTTP Proxy に送信されます。

この設定は、Web サービスクライアントと Web サービスの間に認証を必要とする Proxy が存在するときに必要となります。Web サービス自身が認証される場合は、2 つの認証が必要です。

- Proxy の後ろにある Web サービスとの認証 :

```
  `DIGEST モードでの Web サービス認証
AUTHENTICATE WEB SERVICE("SoapUser";"123";2)
  ` デフォルトモードでの Proxy 認証
AUTHENTICATE WEB SERVICE("ProxyUser";"456";*)
CALL WEB SERVICE(...)
```

4D 環境

SET DATABASE PARAMETER, Get database parameter

SET DATABASE PARAMETER と **Get database parameter** コマンドは、4D 間の Web サービスによるデータ交換最適化に関連する、新しい 2 つのセレクタを受け入れます (ページ 22、"4D Web サービス間の最適化 " の段落参照)。

セレクタ = 50 (HTTP Compression Level)

- 値 : 1 to 9 (1 = 速度優先 , 9 = 圧縮優先 , -1 = 最適値)
- 説明: Webサービスで実行されるHTTP通信の圧縮レベルを設定します (クライアントのリクエストやサーバの応答)。圧縮した通信は、Web サービスで通信する 2 つの 4D アプリケーションがある時に使用できます (ページ 22、SET WEB SERVICE OPTION コマンド参照)。
  このセレクタでは圧縮率を下げて圧縮時間を早くするか、圧縮時間を

かけても圧縮率を上げるかの選択ができます。値の選択はデータのサイズやタイプにより異なります。value 引数に 1 から 9 の値を渡します。1 は早く圧縮し、9 は圧縮率を上げます。また -1 を指定して、圧縮の速度と率の妥協点を指定できます。
デフォルトで、圧縮レベルは 1 (速度優先) です。

セレクタ = 51 (HTTP Compression Threshold)

- 値 : 倍長整数値
- 説明: 最適化モード (前述) における 4D Webサービス間のフレームワークにおいて、通信を圧縮しないリクエストの敷居値を設定します。この設定により、小さな通信を圧縮してマシンの処理時間を無駄にするようなことを避けます。
value にバイト単位のサイズを渡します。
デフォルトで、圧縮の敷居値は 1024 バイトです。

XML

"XML" コマンドテーマは分割され、一部のコマンドの名称が変更されました。さらに 2 つのコマンドが追加されました。

XML コマンドの再構成

より機能を明確にするために、"XML" テーマのコマンドは 3 つの新しいテーマに分割されました :

- **XML DOM**: DOM コマンドがまとめられています。
- **XML SAX**: SAX コマンドがまとめられています。
- **XML ユーティリティ** : 汎用の XML コマンドがまとめられています (XSLT、エラー管理、そして SVG コマンド)。

また **DOM EXPORT TO PICTURE** コマンドは **SVG EXPORT TO PICTURE** に名称変更され、"XML ユーティリティ " テーマに置かれます。

最後に、2 つの新しい XML コマンドが 4D v11 SQL r3 で追加されました (以下の説明参照):

- DOM Get Root XML element が "XML DOM" テーマに、
- SVG Find element ID by coordinates が "XML ユーティリティ " テーマに追加されました。

DOM Get Root XML element

DOM Get Root XML element (elementRef) → String

引数	型		説明
elementRef	文字列	→	XML 要素参照
Function result	文字列	←	ルート要素参照 (16 文字) またはエラーの場合 ""

DOM Get Root XML element コマンドは、引数に渡された XML 要素が属するドキュメントのルート要素参照を返します。この参照は他の XML 解析コマンドで使用できます。

コマンドが正しく実行されると、OK システム変数に 1 が設定されます。エラーが発生すると、例えば要素参照が無効などでエラーが発生した場合、OK システム変数は 0 に設定され、エラーが生成されます。この場合、コマンドは空の文字列を返します。

テーマ : XML DOM

SVG Find element ID by coordinates

SVG Find element ID by coordinates ({*;} pictureObject; x; y) → String

引数	型		説明
*		→	指定時、pictureObject はオブジェクト名 (文字列) 省略時、pictureObject はフィールドまたは変数
pictureObject	ピクチャ	→	オブジェクト名 (* 指定時) またはフィールドや変数 (* 省略時)
x	倍長整数	→	X 座標 (ピクセル単位)
y	倍長整数	→	Y 座標 (ピクセル単位)
関数の戻り値	文字列	←	X, Y の場所で見つかった要素の ID

SVG Find element ID by coordinates コマンドは、pictureObject で指定した SVG ピクチャ内で (x,y) 座標で設定した位置に見つけた XML 要素の ID ("id" または "xml:id" 属性) を返します。このコマンドは特に、SVG オブジェクトを使用してインタラクティブなグラフィックインタフェースを作成するために使用できます。

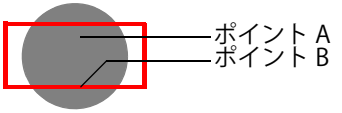
オプションの * 引数を渡すと、pictureObject 引数はオブジェクト名です。この引数を渡さない場合、pictureObject はフィールドあるいは変数です。この場合、文字列ではなくフィールドや変数の参照 (フィールドまたは変数オブジェクトのみ) を渡します。

ピクチャがフォームに表示されている必要はありません。この場合、" オブジェクト名 " タイプのシンタックスは使用できず、フィールドまたは変数名を渡さなければなりません。

x と y 引数に渡される座標は、ピクチャの左上隅 (0,0) からの相対座標をピクセル単位で指定します。フォームに表示されているピクチャのコンテキストでは、MouseX と MouseY システム変数から返される値を使用できます。これらの変数は **On Clicked**、**On Double Clicked**、および (v11.3 の新機能として) **On Mouse Enter** と **On Mouse Move** フォームイベントで更新されます。

注 : ピクチャの座標システムにおいて、繰り返しを除き、ピクチャ表示フォーマットに関わらず、[x;y] は常に同じポイントを指します。

コマンドからは、指定されたポイントで最初に見つかった要素の ID が返されます。例えば以下のケースで、ポイント A の座標が渡された場合コマンドは円の ID を返し、ポイント B の座標が渡された場合コマンドは四角の ID を返します。:



座標が重ね合わせや複合オブジェクトに対応する場合、コマンドは必要に応じて親要素に戻り、有効な ID 属性を持つ最初のオブジェクトの ID を返します。

コマンドは以下の場合空の文字列を返します :

- "id" 属性を持つ要素が見つからないうちに root に達してしまった場合。
- 座標がどのオブジェクトも指していない場合。
- "id" 属性値が空の時。

pictureObject が有効な SVG ピクチャでない場合、コマンドは空の文字列を返し、OK システム変数に 0 が設定されます。コマンドが正しく実行されれば、OK システム変数は 1 に設定されます。

テーマ : XML ユーティリティ

他の新機能

ドキュメントサイズの管理

デフォルトで、4D v11 SQL r3 のランゲージは実数型の値を使用してドキュメントサイズを処理します。以前のバージョンでは倍長整数型でした。この変更により、2GB を超えるドキュメントをランゲージで扱えるようになりました。

" システムドキュメント " テーマの以下のコマンドがこの変更に関係します :

- **Get document size** (このコマンドは実数値を返します)
- **SET DOCUMENT SIZE** (size 引数)
- **SET DOCUMENT POSITION** (offset 引数)

Mac OS での複数ローのデータの入力及びリストボックスの表示

Mac OS でも、同じリストボックス " セル " 内に複数行のテキストを表示、および入力することができるようになりました。4D の以前のバージョンでは、この機能を使用できるのは Windows だけでした。

Mac OS でリストボックスに行送りを追加するには、**Option+ 改行**キーを押すだけです。行の高さは自動的には変更されませんのでご注意下さい。

統計関数による OK 変数の更新

多大なセレクションデータで統計関数（例えば **Averege**）を実行する際、進捗サーモメータが表示されます。このサーモメータには、**停止**ボタンが備わっているので、演算を中止することができます。
そのため、統計関数は OK システム変数を更新するようになりました。演算が完了した場合変数は 1 に設定され、ユーザが**停止**ボタンをクリックした場合、OK 変数は 0 に設定されます。