






















4D v14 R3 - アップグレード

4Dの継続的なデリバリープログラムの最新バージョンとなる、4D v14 R3へようこそ。このマニュアルでは、この新しいリリースの中に含まれる全ての新機能や実装について説明しています。

-  ランゲージリファレンス
-  デザインモード
-  4D Server
-  4D SQL Server
-  4D Mobile

ランゲージリファレンス

-  DELAY PROCESS
-  GET ACTIVITY SNAPSHOT
-  Get database parameter と SET DATABASE PARAMETER
-  Is License available
-  MAXIMIZE WINDOW
-  OPEN URL (OPEN WEB URLの新しい名前)
-  OPEN WEB URL renamed
-  QR Get text property と QR SET TEXT PROPERTY
-  WEB GET OPTION と WEB SET OPTION
-  WEB SET HTTP HEADER
-  サブレコードの変換
-  リストボックスにてダイナミック変数でカラムを作成
-  Get database measures
-  Get locked records info
-  MOBILE Return selection
-  OPEN DATABASE

DELAY PROCESS

DELAY PROCESS (process ; duration)

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

説明

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

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

GET ACTIVITY SNAPSHOT

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

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

新しいプロパティ

このコマンドは4D v14 R3において、4D Server管理ウィンドウのReal Time Monitor (RTM)ページの進化に伴い、追加の情報を返すようになりました。

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

GET ACTIVITY SNAPSHOT(arrActivites{; *})

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

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

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

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

す。この配列の構造は以下のようになっています:

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

説明

dbContextInfo

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

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

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

dbOperationDetails

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

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

📄 Get database parameter と SET DATABASE PARAMETER

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

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

説明

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

定数	型	値
SQL Server Port ID	倍長整数	88

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

Is License available

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

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

説明

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

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

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

新しい4D Mobileライセンスの詳細な情報に関しては、[4D Mobile](#)セクションを参照して下さい。

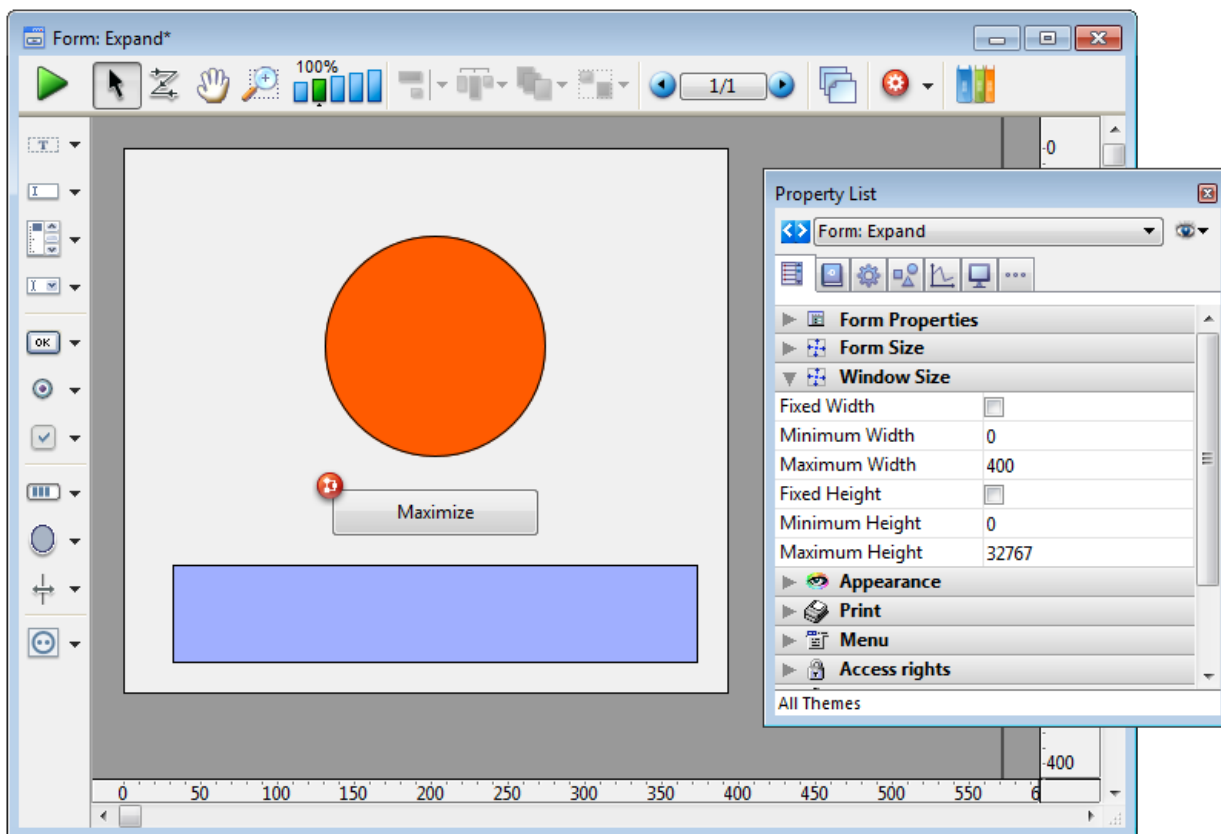
MAXIMIZE WINDOW

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

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

例題

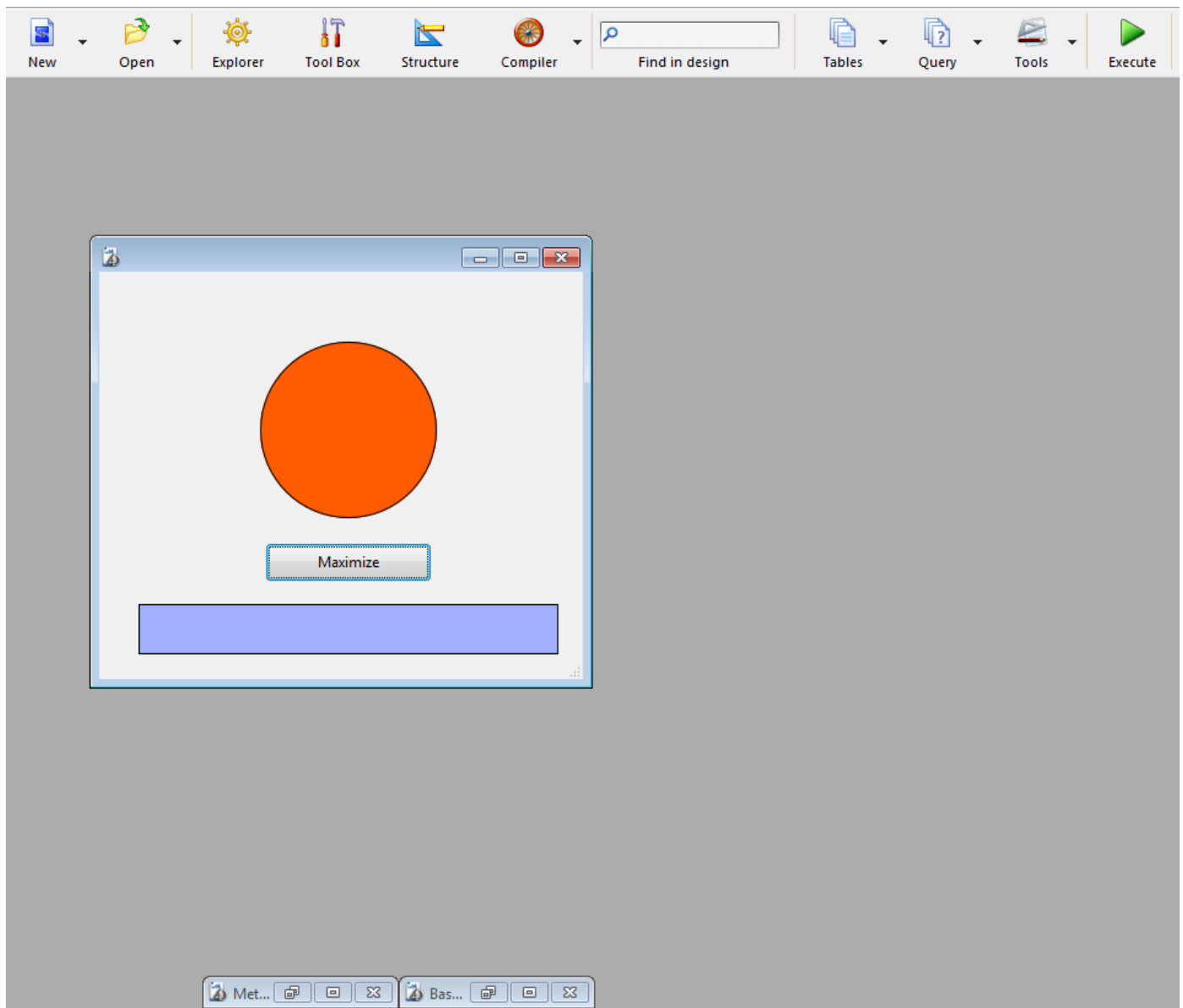
以下のフォームはサイズ制約付きで定義されています(最大幅=400):



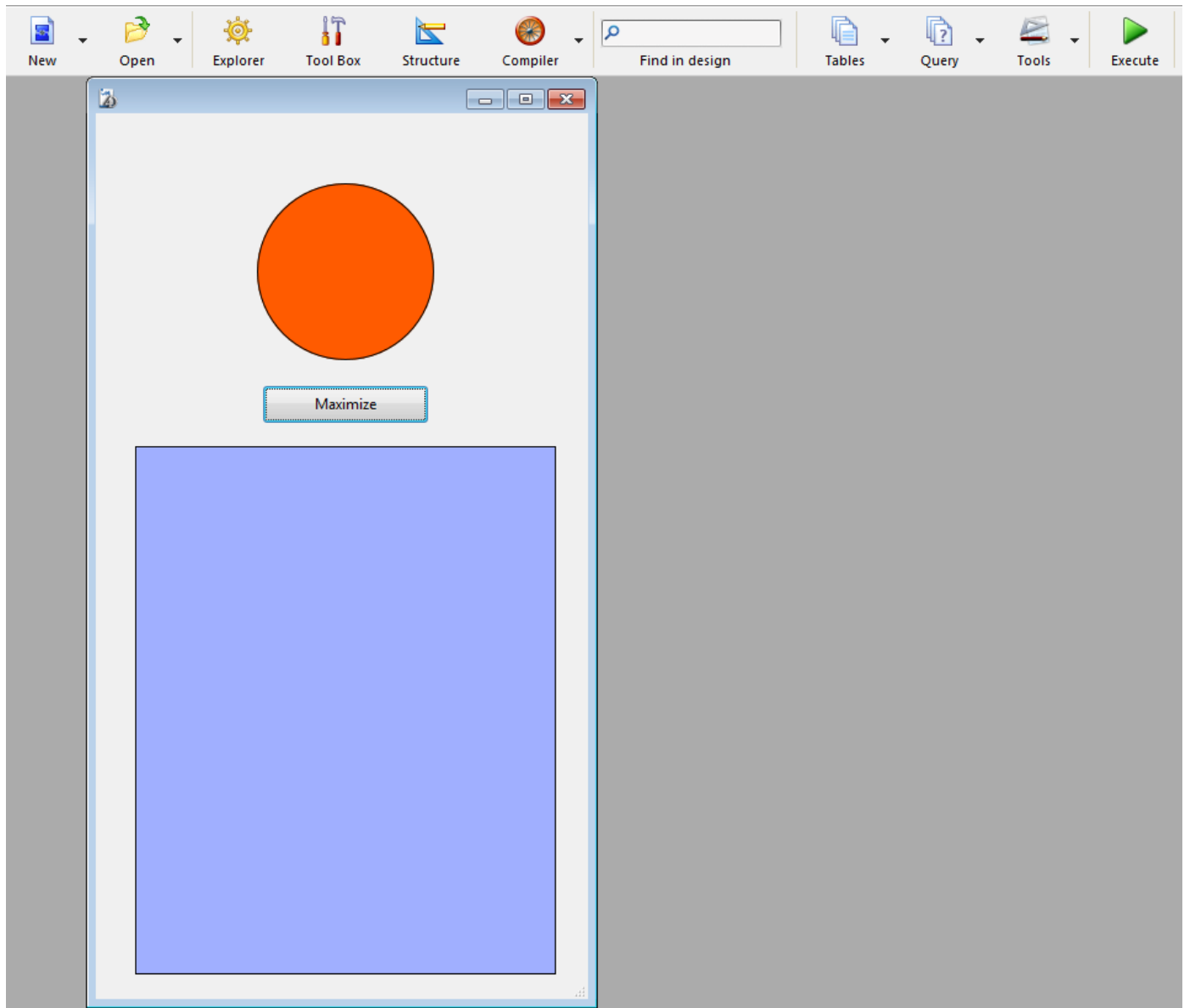
ボタンのコードには以下のように書かれています:

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

この状態において、ユーザーがボタンをクリックすると:



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



📄 OPEN URL (OPEN WEB URLの新しい名前)

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

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

説明

護岸性に関する注意: **OPEN URL** は、**OPEN WEB URL** コマンドの新しい名前です("システム環境"テーマ)。このコマンドは、4D v14 R3で昨日が拡張されました。

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

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

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

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

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

HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥App Paths

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

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

等です。

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

例題

以下の様に、テキストファイルを異なるアプリケーションで開く事ができます:

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

OPEN WEB URL renamed

In 4D v14 R3, the **OPEN WEB URL** command ("Utilities" theme) has been renamed **OPEN URL** and now accepts a new parameter. For more information, please refer to the **[OPEN URL \(new name for OPEN WEB URL\)](#)** section.

QR Get text property と QR SET TEXT PROPERTY

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

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

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

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

説明

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

property 引数は以下の様に挙動が変更されました:

- `qr font` 定数は `_O_qr font.` と改名されました。これは廃止予定の定数であり、今後は使用されるべきではありません (互換性は保持されますが、将来のリリースではサポートされなくなります)。
- 新しい `qr font name` 定数 (10) が "QR Text Properties" テーマに追加されました。今後フォントを指定する際には、この定数と文字列の値を使用して下さい。渡す事の出来るフォント名は **FONT LIST** コマンドを使用した際に返される名前です。

例題

以下のメソッドは、最初のカラムタイトルのフォント属性を定義します:

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

WEB GET OPTION と WEB SET OPTION

WEB GET OPTION (*selector* ; value)

WEB SET OPTION (*selector* ; value)

説明

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

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

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

例題

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

WEB SET HTTP HEADER

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

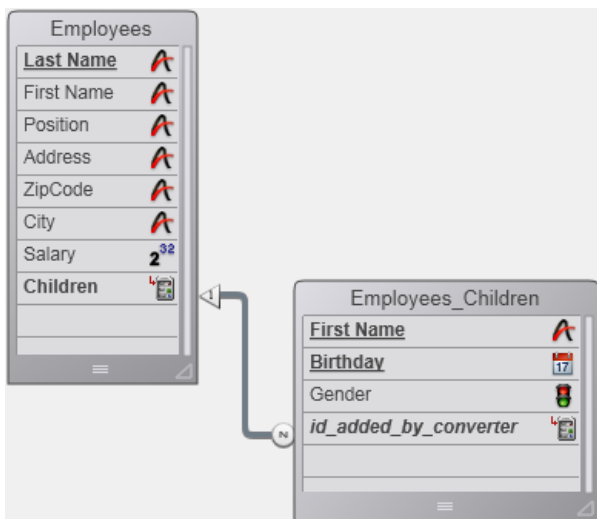
サーバーのヘッダーが変更可能に

4D v14 R3から、WEB SET HTTP HEADER コマンドを使うことによって4DからWebクライアントへとリプライを送る際のサーバーフィールドを設定することが可能になりました。以前のリリースでは、このヘッダーは4Dによって自動的に設定されており、変更は不可能でした。

サブレコードの変換

4D v14 R3では、v11以前のバージョンのデータベースを変換する際に4Dが自動的に追加していた特殊なフィールド "id_added_by_converter" の値を、ユーザーが割り当てることができるようになりました。以前のリリースでは、この値は4D によってのみ割り当てることができ、変換されたサブテーブルに新しいレコードを追加する際には **CREATE SUBRECORD** のような"廃止予定の"コマンドを使用する必要がありました。

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



これを使用すると、例えば以下の様を書くことができます:

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

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

📄 リストボックスにてダイナミック変数でカラムを作成

4D v14 R3 の新機能の一つとして、ランタイムで動的に新しいカラムを追加または挿入できるようになりました。この新機能によって、4D は必要な変数の定義(カラム、フッター、そしてヘッダー)を自動的に行います。

その結果、以下の三つの4Dコマンドが一部新しくなりました:

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

新シンタックス(Nilポインター)

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

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

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

4D v14 R3 では、これらのコマンドは、*colVariable* (適用可能時のみ)、*headerVar* と *footerVar* 引数において、Nil ポインター (->[]) を値として受け取るようになりました。この場合、4D は、コマンドが実行された時に必要な変数を動的に作成します(詳細な情報に関しては、4Dランゲージマニュアルの"ダイナミック変数"のセクションを参照して下さい)。

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

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

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

🔧 Get database measures

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

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

説明

テーマ: 4D環境

Get database measures コマンドを使うと、4D データベースエンジンイベントに関する詳細な情報を取得できます。返された情報にはディスクまたはメモリーキャッシュからのデータの読み込み・書き込みアクセス、ディスクまたはキャッシュへのデータの読み込み・書き込みアクセスに加え、データベースのインデックスへのアクセスも含まれます。

Get database measures は関連した情報を格納する単一のオブジェクトを返します。options オブジェクト引数を使用すると、返される情報のオプションを設定することができます。

返されるオブジェクトの概要

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

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

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

    "dataSegment1": {...},
    "indexSegment": {...},

    "tables": {...},
    "indexes": {...}
  }
}
```

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

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

要素プロパティ

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

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

8つの要素プロパティは全て同じオブジェクト構造を持ちます。例えば:

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

- **"value"** (数字): この"value"プロパティにはバイトの量がアクセスの回数で格納されます。基本的には、この値は"history"オブジェクトの合計値となっています("history"オブジェクトが存在しない場合も同様です)。
- **"history"** (オブジェクト配列): "history" オブジェクト配列は、秒ごとにグループ分けされたイベント値の集合です。"history"プロパティはoptions 引数を通してリクエストされた場合にのみ表示されます(以下を参照して下さい)。
"history" 配列は最大で200アイテムを格納することができます。配列内の各要素はそれ自体がオブジェクトであり、二つのプロパティを格納します:"value" と "time"です。
 - "value" (数字): 関連付けられた"time"プロパティで指定された時間内に扱われたバイトまたはアクセスの量。
 - "time" (数字): そのファンクションが呼び出されてから経過した秒数。上記の例の("time": -1649) は、1649秒前(厳密には1649秒前から1650秒前の間)を意味します。この1秒間に、54,202 バイトがディスク上から読み出されました。

この例でのhistoryの配列には連続した値(-1650,-1651,-1652, 等)は格納されていません。これの前の値は -1665で、これはつまり1650秒前から1665秒前までの15秒間はディスクから何も読み出されていないことを意味します。

注: デフォルトでは、配列には意味のある情報しか含まれません。

配列の最大サイズが200なので、データベースが頻繁に使用されている(例えばディスク上から毎秒何かを読み出されている)場合、historyの長さの上限は200秒となります。反対に、3分に1度だけ読み込みが発生する以外には何もデータベース起こらない場合には、historyの長さは600分(3×200)となります。

この例の場合は、以下の表のようにあらわされます:

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

dataSegment1 と indexSegment

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

```
"dataSegment1": {
```

```

    "diskReadBytes": {...},
    "diskWriteBytes": {...},
    "diskReadCount": {...},
    "diskWriteCount": {...}
  },
  "indexSegment": {
    "diskReadBytes": {...},
    "diskWriteBytes": {...},
    "diskReadCount": {...},
    "diskWriteCount": {...}
  }
}

```

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

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

例えば、以下のオブジェクトが返ってきます:

```

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

```

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

diskReadBytes.value = dataSegment1.diskReadBytes.value + indexSegment.diskReadBytes.value

diskWriteBytes.value = dataSegment1.diskWriteBytes.value + indexSegment.diskWriteBytes.value

diskReadCount.value = dataSegment1.diskReadCount.value + indexSegment.diskReadCount.value

diskWriteCount.value = dataSegment1.diskWriteCount.value + indexSegment.diskWriteCount.value

tables

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

```

"tables": { "Employees": {...} "Companies": {...} }

```

それぞれのテーブルプロパティには、10個のプロパティが格納されています:

- 最初の8つのプロパティはアクセスのあったテーブルに関連した値を格納した要素プロパティ(上記参照)です。
- 残り二つのプロパティ"records" と "blobs"には、それぞれ同じように8つの要素プロパティがあり、特定のフィールドの型に関連する情報だけが格納されています:
 - "records" プロパティはテーブル全体のフィールド(文字列、日付、数字、等)のうち、テキスト、ピクチャー、Blobのフィールドを除いた情報が格納されています
 - "blobs" プロパティには、テーブルのうちテキスト、ピクチャー、Blobフィールドの情報が格納されています

indexes

これがもっとも複雑なオブジェクトです。一つ以上のインデックスを使用してアクセスされた全てのテーブルがプロパティとして保存され、そのプロパティ内には使用されたインデックス名もプロパティとして格納されています。さらに、それぞれのインデックス名のプロパティオブジェクトには、そのインデックスに関連した8つの要素プロパティが格納されています。

例:

```

"indexes": { "Employees": { "EmpLastName": { "diskReadBytes": {...},
  "cacheReadBytes": {...}, "cacheMissBytes": {...}, "diskWriteBytes": {...},
  {...}, "diskReadCount": {...}, "cacheReadCount": {...},
  "cacheMissCount": {...}, "diskWriteCount": {...} } "index2Name": {...},
  "index3Name": {...}, ... } table2Name: {...} table3Name: {...} }

```

options 引数

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

プロパティ	型	詳細
"withHistory"	ブール	"true" を渡した場合、historyは返されたオブジェクト内で呼び出されるファンクションによって返されることを表します。"false"を渡した場合、ファンクションによって返されるオブジェクトにはhistoryが格納されないことを意味します。
"historyLength"	数値	返される history 配列のサイズを秒数で定義します(*)
"path"	文字列	取得したい特定のプロパティへの完全なパスを指定します (例:"DB.tables.Employees.records.diskWriteBytes")。このプロパティ内に有効なパスを渡した場合、それに対応する値だけがファンクションからの戻り値のオブジェクトに含まれます。

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

例:データベースはつい20秒前に開かれ、historyが60秒前をリクエストしたとします。返された値のうち0から-20は値が0に設定され、他のものは-1へと設定されます。"-1"の値が返された場合、これはリクエストされた時間が古すぎるか、または値が内部history配列内に残っていないことを意味します(200アイテムという上限に達したため、古い値は削除されています)。

クライアント/サーバー、そしてコンポーネントについて

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

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

コマンドがリモートの4Dから呼び出された場合、オブジェクトは空のまま返されます。

このコンテキストにおいて、サーバー側にあるデータベースの情報を取得したい場合には、もっとも簡単な手段は"execute on server"オプションが有効化されているメソッドを作成してしまう事です。

この原理はコンポーネントに対しても同様です:もしコンポーネントがローカルの4Dにおいて呼び出された場合、返されるのはホストデータベースについての情報です。4D リモートの場合には、サーバーデータベースについての情報を返します。

例題 1

返されたオブジェクト内にhistoryのログを残したい場合:

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

例題 2

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

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

返されたオブジェクトには、以下の様な情報が含まれます:

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

例題 3

直近の2分間に読み込みされたキャッシュのバイトサイズを取得したい場合:

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

🔧 Get locked records info

Get locked records info (aTable) -> 戻り値

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

説明

テーマ: レコードロック

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

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

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

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

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

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

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

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

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

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

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

例題

以下のコードを実行します:

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

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

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


MOBILE Return selection (aTable) -> 戻り値

引数	型		説明
aTable	テーブル	→	セレクションを取得したいテーブル
戻り値	Object	↩	Wakanda準拠のセレクション

説明

テーマ: セレクション

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

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

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

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

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

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

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

例題

[Countries] テーブルのクエリに基づいたカレントセレクションを Wakanda のグリッドに表示させたい場合を考えます。まず、以下の様な4Dメソッドを作成します:

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

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

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

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

ShortName	Name	Capital
Angola	Republic of Angola	Luanda
Argentina	Argentine Republic	Buenos
Australia	Commonwealth of Au...	Canberr
Brazil	Federative Republic of...	Brasilia
Canada	Canada	Ottawa
Chile	Republic of Chile	Santiago
China	People's Republic of C	Beijing

24 item(s)

Find Countries

ボタンに記述されているコードは以下の通りです:

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

その結果、グリッドが更新され以下の様になります:

ShortName	Name	Capital
India	Republic of india	New D
Italy	Italian Republic	Rome

2 item(s)

Find Countries

OPEN DATABASE (filePath)

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

説明

テーマ: 4D環境

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

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

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

ファイル名のみを渡す場合、現在開いているデータベースのストラクチャーファイルと同じ階層にそのファイルが置かれている必要があります。

アクセスパスが有効なデータベースを設定していた場合、4Dは開いているデータベースを閉じ、指定されたデータベースを開きます。シングルユーザーモードにおいては、閉じられた方のデータベースの **On Exitデータベースメソッド** と、新たに開かれた方のデータベースの **On Startupデータベースメソッド** が順番に開かれます。

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

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




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

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

例題

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

デザインモード

-  ランタイムエクスプローラーインターフェースの変更
-  利用不可能なピクチャー形式に対する新アイコン
-  コマンドラインインターフェースの拡張

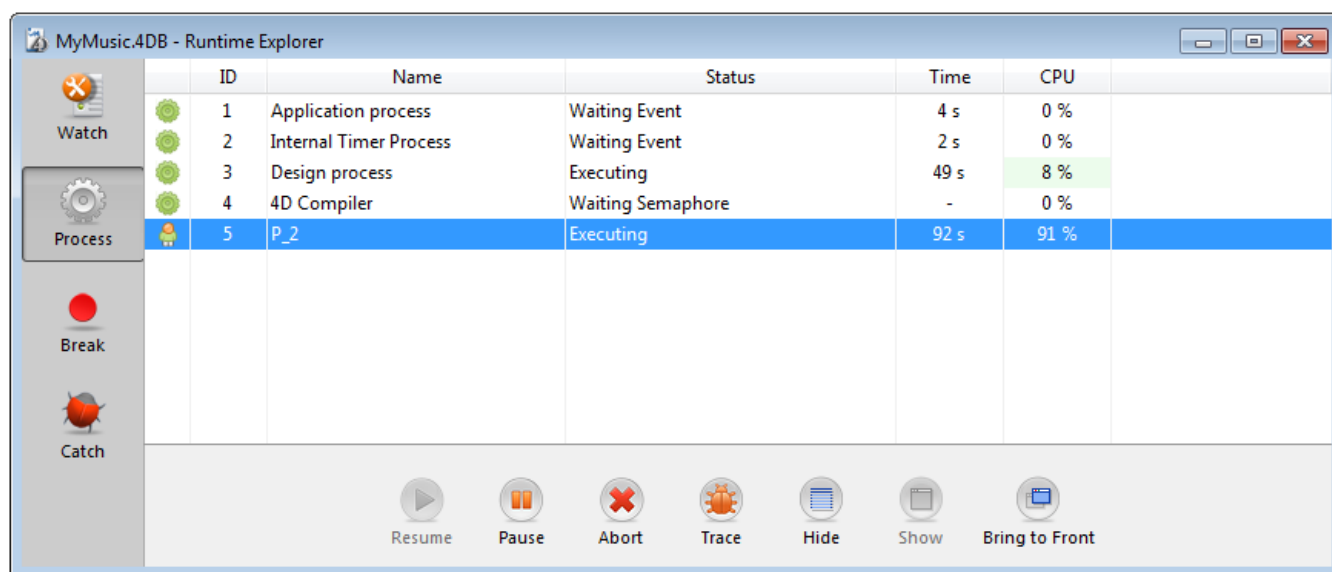
ランタイムエクスプローラーインターフェースの変更

4D v14 R3において、ランタイムエクスプローラーインターフェースが全体的に一新されました。

これに加え、**Process** ページの設計が一新され、それぞれのプロセスのタイプを表すのにアイコンを使用するようになりました。

プロセスページ

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



ID	Name	Status	Time	CPU
1	Application process	Waiting Event	4 s	0 %
2	Internal Timer Process	Waiting Event	2 s	0 %
3	Design process	Executing	49 s	8 %
4	4D Compiler	Waiting Semaphore	-	0 %
5	P_2	Executing	92 s	91 %

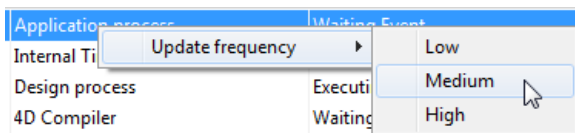
プロセスリストは4D Server管理ウィンドウの**プロセスページ**に近くなりました。

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

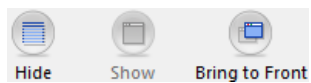
- CPU: グラフィカルビューはなくなりました。その代り、背景の赤色のグラデーションがプロセスのアクティビティレベルを表します。
- それぞれのプロセスはアイコンで表示されるようになりました。色と形がプロセスの型を表します：

- アプリケーションサーバー
- SQL サーバー
- DB4D サーバー(データベースエンジン)
- Web サーバー
- SOAP サーバー
- プロテクトされた4Dクライアントプロセス(接続している4Dの開発プロセス)
- メインの4Dクライアントプロセス(接続している4Dのメインプロセスまたはコラボラティブプロセス。クライアントマシンで作成されたプロセスのサーバー上と同じ)
- 4D クライアントベースプロセス(4Dクライアントプロセスと対等なプロセス。対応する4Dクライアントプロセスを管理するためのプリエンティブプロセス)
- スペアプロセス(以前の"4D クライアントデータベースプロセス")
- 4Dクライアントプロセス(接続した4Dで実行中のプロセス)
- ストアドプロシージャ(接続した4Dによって起動されたプロセスでサーバーで実行中のもの)
- Web メソッド(例えば4DACTIONによって起動されたもの)
- SOAP メソッド(Webサービスによって起動されたもの)
- SQL メソッド(SQLクエリによって起動されたもの)

- コンテキストメニューを使用して更新頻度を設定することが出来るようになりました(行を右クリックして下さい):



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



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

利用不可能なピクチャー形式に対する新アイコン

4D v14 R3 以降、マシン上でレンダリングできない形式で保存されているピクチャーに対しては新しいアイコンが表示されるようになりました。その対応していないフォーマットの拡張子は、アイコンの下部に書かれています：



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

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

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

コマンドラインインターフェースの拡張


以前はOS Xでのみサポートされていたアプリケーションコマンドラインパーサーが、4D v14 R3で改善され、OS XとWindows、両方で使用できるようになりました。

Windowsでも、例えば以下の様に記述する事ができるようになります:

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


4D Server

 OS X用4D Serverの64ビット版について

 リアルタイムモニターページ

OS X用4D Serverの64ビット版について

v14 R3 のリリースから、OS X用の新しい4D Server(64ビット版)のプレビュー版をご利用いただく事ができます。最終バージョンにて完全に利用可能になった時には、この新製品によって、4D Serverアプリケーションで64-ビットAppleマシンの実力を全て発揮する事ができるようになります。

64-ビット版の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、という点です。このセクションでは、64ビット版の4D ServerをMac OS Xで導入・使用する際の注意点を扱っていきます。

OS Xの必須バージョン

OS X用の64ビット版4D Serverを使用するためには、**10.9 (Mavericks)**以上のOSが必要になります。

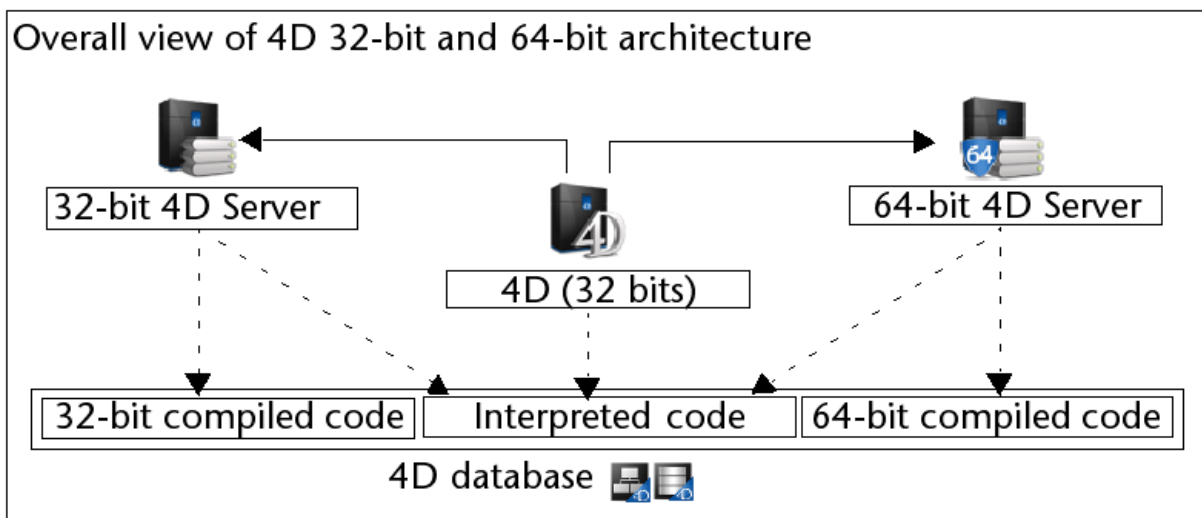
アーキテクチャー

64ビットアーキテクチャー用の4D Serverはその環境専用のバージョンです(32ビットOS上では動作しません)。

クライアント側 j からは、4Dアプリケーションは(OS X用・Win用どちらでも)64ビット版の4D Serverにアクセスする事ができます。この場合接続するのに使用する4Dは標準の32ビット版です(以下のダイアグラムを参照して下さい)。

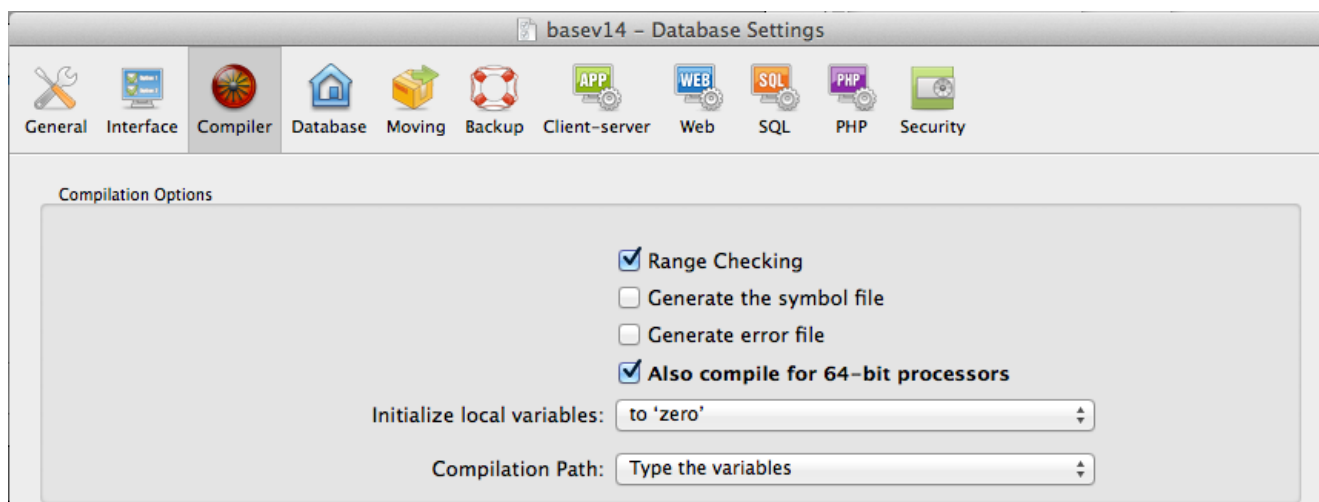
インタープリタモードでは、同じ4Dデータベースを64ビット版の4D Serverでも32ビット版4D Serverでも実行する事ができます。開発は、使用するアプリケーションを問わず、同じように進めることが可能です(ただし以下の制約が付随します)。

コンパイルモードでは、64ビット版4D Serverで実行するためには64ビットプロセッサ用にコンパイルされている必要があります。32ビット用にコンパイルされていて、インタープリタコードを含まないデータベースは64ビット版4D Serverで実行することはできません。



64-bit 用コンパイル

4Dアプリケーションは、32ビット用と64ビット用とにコンパイルすることができます。これをするためには、データベース設定の「コンパイラー」ページにある、**64-bit プロセッサ用にもコンパイルするオプション**をチェック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るとき、コンパイラーは64ビット用のコードと32ビット用のコードを .4DC と .4DBファイルに含めます。それにより、これらのファイルは32ビット版・64ビット版、両方の4D Serverで実行することが出来るようになります。デフォルトでは、このオプションはチェックされていません。

注: データベースの64ビット版をコンパイルするためには、Unicodeモードで動作している必要があります。そうでない場合にはコンパイル時にエラーが発生します。

コンパイルされたコードの互換性

4D v14 R3 では、OS X 64-bit アーキテクチャーをサポートするために4D内蔵のコンパイラーが変更されました。その結果、4D v14 R3(またはそれ以降のバージョン)でコンパイルされたデータベースのみが OS X 64-bitで実行可能となります。つまり:

- 既存の4DデータベースをOS X 64-bitにおいてコンパイルモードで使用したい場合、そのデータベースを4D v14 R3(またそれ以降のバージョン)で再コンパイルする必要があります。
- データベースがコンパイルされたのコンポーネントを使用する場合、そのコンポーネントを4D v14 R3(またそれ以降のバージョン)で再コンパイルする必要があります。

しかしながら、4D v14 R3を使用して生成された64-bit (または32-bit)にコンパイルされたコードは、4D v14.x または4D v14 Rxと互換性があります。つまり、一度4D v14 R3で再コンパイルしてしまえば、14.x と 14 Rx の全てのバージョンにおいて、32-bit、64-bit、どちらでも使用することができます。

サポート対象外の機能

以下の機能・技術はOS X用の64bit版4D Serverではサポートされません:

機能/技術	コメント
XSLT と Xalan	XSLT APPLY TRANSFORMATION 、 XSLT SET PARAMETER 、そして XSLT GET ERROR は動作しません。代わりに PHP <i>libxslt</i> モジュールを使用して下さい。
PICT フォーマット	画像の代わりに、'サポート外の画像フォーマット'画像 + ファイル拡張子が表示されます([#title id="3575"/])を 利用不可能なピクチャー形式に対する新アイコン を参照して下さい)。PICTフォーマットは4D全体で使用廃止になっています。 AP Is Picture Deprecated も参照して下さい。
cicn アイコン	GET ICON RESOURCE コマンドはサーバーではサポートされていません(*)。
データベース .RSR ファイル	データベース .RSR ファイルは、自動的に開かれませんが、 Open resource file を使用する必要があります。
書き込み可能なリソースファイル	Create resource file はサーバー上で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専用でのみ開く事ができます。 注意: Mac OS リソースファイルは、既に4D v11から廃止予定となっていました。
Font number	このコマンドはサーバー上ではサポートされていません。(*)
ASCII 互換モード	Unicodeモードのみがサポートされています

無効化されている機能

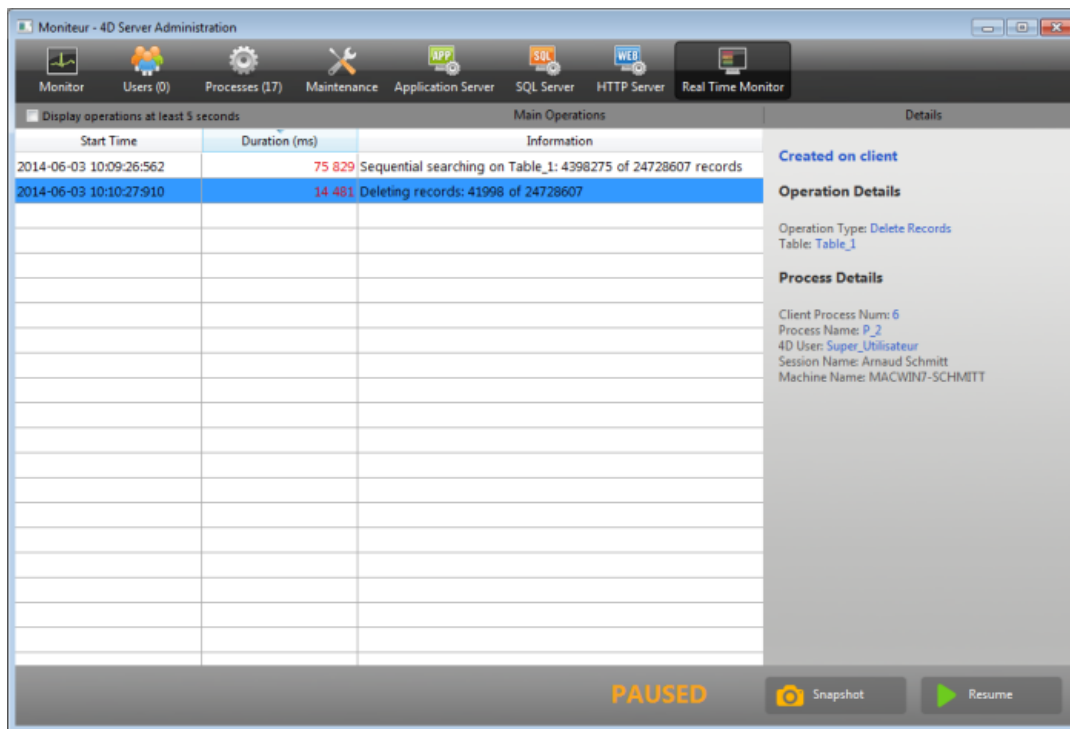
OS X用の64ビット版4D Serverの最初のリリースでは、一部の機能が一時的に無効化されています。64ビット版への対応が完了し次第、順を追って復帰していく予定です。

機能/技術	コメント
シリアルポート通信	無効化されています - 将来のリリースでは再度サポートされます。
フォームエディター(サーバー上で実行)	無効化されています - 将来のリリースでは再度サポートされます。
ダイアログボックスのインポート/エクスポート	無効化されています - 将来のリリースでは再度サポートされます。
ラベルエディター	無効化されています - 将来のリリースでは再度サポートされます。
チャートをサーバー上で生成	無効化されています - 将来のリリースでは再度サポートされます。
HTTP クライアント	クライアントの証明書の管理は無効化されています

リアルタイムモニターページ

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

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



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

アドバンスドモード

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

以下は、標準モードで表示される情報の例です:

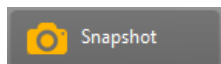
Start Time	Duration (ms)	Information	
2014-05-27 16:25:56	2 870	Sequential searching on Table_1: 438712 of 15128756 records	Created on client
			Operation Details
			Operation Type: Query Query Plan: 'Table_1]Champ_2'='Table_1]Champ_2'
			Process Details
			Client Process Num: 7 Process Name: P_3 4D User: Super_Utilisateur Session Name: Arnaud Schmitt Machine Name: MACWIN7-SCHMITT

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

Start Time	Duration (ms)	Information	
2014-05-27 16:25:56	2 870	Sequential searching on Table_1: 438712 of 15128756 records	<p>Created on client</p> <p>Operation Details</p> <p>Operation Type: Query Query Plan: "Table_1]Champ_2"="Table_1]Champ_2"</p> <p>Process Details</p> <p>Client Process Num: 7 Process Name: P_3 4D User: Super_Utilisateur Session Name: Arnaud Schmitt Machine Name: MACWIN7-SCHMITT Client UID: B0C9071B0C9071B0C9071B0C9071B0C9071 Client Version: v14 R2 Beta</p>

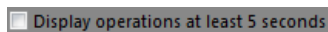
スナップショットボタン

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



オペレーションを最低5秒間表示する

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



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

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

新しく追加される情報

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

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

進捗メッセージはオフに


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

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

4D SQL Server

 ALTER DATABASE DISABLE TRIGGERS

 ALTER TABLE DISABLE TRIGGERS

 ポートIDをプログラミングで管理する

ALTER DATABASE DISABLE TRIGGERS

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

説明

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

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

例題

インポートのためにSQLオプションを全て一時的に無効化する例です:

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


ALTER TABLE DISABLE TRIGGERS

```
ALTER TABLE sql_name
{ADD column_definition [PRIMARY KEY] [TRAILING] |
DROP sql_name |
ADD primary_key_definition |
DROP PRIMARY KEY |
ADD foreign_key_definition |
DROP CONSTRAINT sql_name |
[{{ENABLE | DISABLE}} REPLICATE] |
[{{ENABLE | DISABLE}} LOG] |
[{{ENABLE | DISABLE}} AUTO_INCREMENT] |
[{{ENABLE | DISABLE}} AUTO_GENERATE] |
[{{ENABLE | DISABLE}} TRIGGERS] |
SET SCHEMA sql_name}
```

説明


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


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

ポートIDをプログラミングで管理する

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

4D Mobile

 [RESTは4D Mobileへ](#)

 [コレクションを返す新コマンド](#)

4D v14 R3 以降、4D Mobile接続を管理するために新しいライセンススキームが提案されました。これまでのリリースでは、4D Mobile 接続は標準の4Dクライアントライセンスを使用していました。今後はそれとは別に専属の**4D Mobile エクステンションパック**をご購入いただけます。4D Mobileライセンススキームについての詳細な情報については、4D Webサイトを参照して下さい。

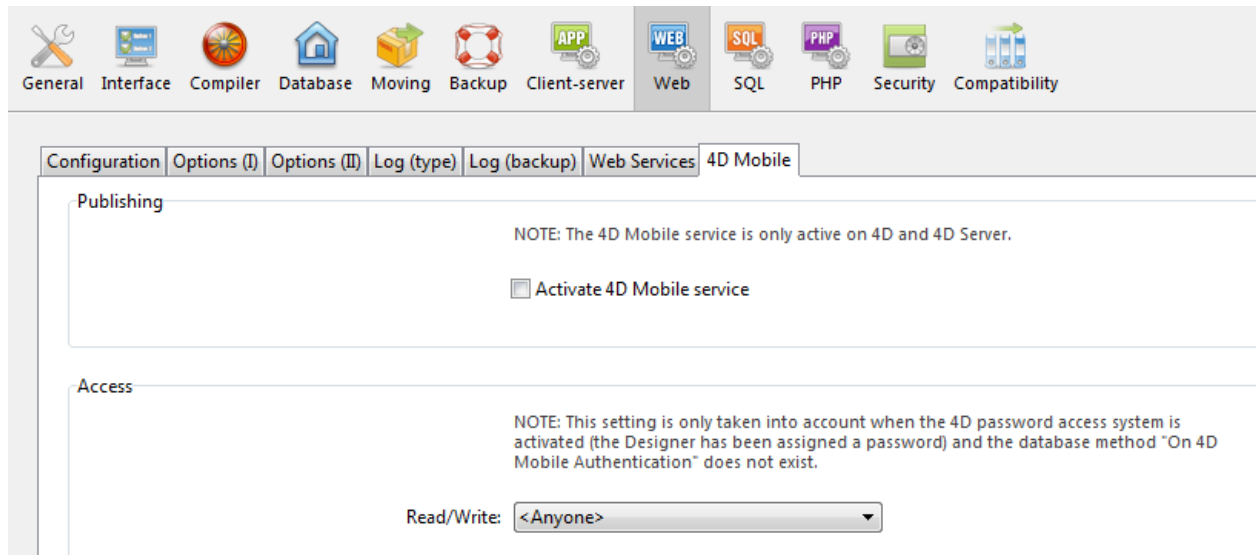
この新しいスキームをサポートするため、既存のオプションは4D v14 R3にて改名され、新しい定数が追加されています(その効果自体に変化はありません)。

改名された機能

以下のインターフェース要素は、新しい4D Mobileライセンススキームに合致するために改名されています。

データベース設定/Web

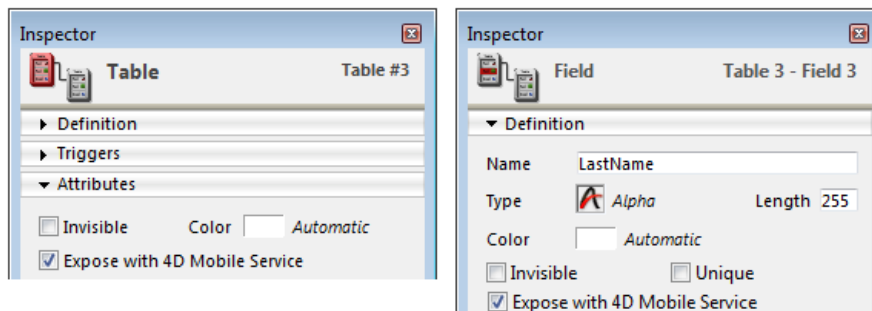
REST タブは、**4D Mobile** へと改名され、**Wakanda REST サービスを有効化**オプションは、**4D Mobile サービスを有効化**と変更されています：



ダイアログ内にある他の文字列も同じく更新されています。

ストラクチャーエディター

テーブルインスペクターとフィールドインスペクター両方において、**REST サービスで公開**のオプションが、**4D Mobileサービスで公開**に改名されています：



メソッドプロパティダイアログボックス

メソッドプロパティダイアログボックスにおいて、**REST 呼び出しからの利用を許可**のオプションが、**4D Mobile 呼び出しからの利用を許可**へと改名されています：


Available through 4D Mobile call

Table:

Scope:

エクスプローラー/メソッド

4Dエクスプローラーの"RESTメソッド"セクションは、"4D Mobileメソッド"へと改名されています:

- ▲ 4D Mobile Methods
 - ▲  Employees

On REST Authentication データベースメソッド

On REST Authentication データベースメソッドはOn 4D Mobile Authentication と改名されました。挙動は以前のリリースと同様です。

Is license availableでの新定数

Is License available コマンドは二つの新しい定数が追加されました:

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

注: 4D Developer Proにはテスト目的の4D Mobileテストライセンスがデフォルトで3つ提供されています。

コレクションを返す新コマンド

新しい **Mobile Return selection** コマンドは、4D Mobileを通してセレクションをWakandaコレクションとして直接返します。詳細な情報に関しては、**MOBILE Return selection** を参照して下さい。