4D v14 One Day Training

ジャン＝ピエール・リブロウ（Jean-Pierre Ribreau）

jpr@ribreau.net

4D v14 One Day Training

4D-PHP-JavaScript: 面倒なことが苦手な人は楽な道をみつける

---

1. 4D 以外のプログラミング言語を使用することは必要?

・ 欲しいから，ではなく，必要だから使用する。

・ （自分が）好きだから，ではなく，（ネットの世界で）好まれているから使用する。

・ ネットに進出したいから，ではなく，ネットに囲まれているから使用する。

世界は急速に変化しています。とりわけ，コンピューターの世界は，おおきな変化の渦中にあります。タブレット，スマートフォンなどのデバイスがクライアント端末に代わるものとして存在感を高め，スタンドアロン・クライアントは，その役割をブラウザに譲りつつあります。鍵となる言葉は，モバイル・コンピューティング，クラウド，分散コンピューティングです。この流れに逆らって泳ぐことなど，考えられません。ついてゆけないのであれば，近い将来，引退の道しか残されていないのです。

ブラウザには，ブラウザの言語があります。それは，Web 技術に基づいた言語で，4D とは用語や概念が違います。画面はフォームではなく，ページと呼ばれ，クライアント/サーバーの持続的なコンテキストがあるのではなく，セッション管理が求められます。データは，内部構造が不透明なレコード型ではなく，構造的かつ透過的なオブジェクトとして扱われます。ブラウザでサーバーに接続するシステムでは，ページの表示に HTML，プロシージャーの実行には，PHP，JavaScript，Perl などの言語を使用します。

そこでこのように自問しなければなりません。「私は，PHP，JavaScript などを習得する気があるだろうか。それとも…???」。第二の選択肢はありません。もう覚悟を決めるしかないのです。

「外国語」を使用するべきおもな理由は，ふたつ挙げることができます。

イ）4D だけでは大変な問題を手早く解決するため

たとえば，QR コードを印刷したい，という問題を考えてみましょう。考えられる手段は，次のとおりです。

がんばって 4D だけで QR コードを作成する。できないことはないと思いますが，

- かなり複雑なコードになることは必至
- 実行速度は芳しくない（かもしれない）
- いずれにしても，プログラミングとデバッグに多大な時間が求められる

プラグインを自作する。やはり，できないことはないと思いますが，

- C++の知識が必要
- デバッグが厄介
- プラットフォーム，OS，新バージョンに適応しなければならない
- いずれにしても，プログラミングとデバッグに多大な時間が求められる

そんなとき，PHP または JavaScript の定評あるライブラリの中から既存のソリューションを探し出して，アプリケーションに移植することができれば，とてもスマートです。

ロ）将来性のない，消滅しつつあるコードの代替品が必要なとき。4D Chart など。

2. もっとも適応性が高い PHP ライブラリを識別する

「PHP libraries」で Google 検索すれば，7,400 万件ほどのヒットが返されます。PHP のプログラミング人口が 500 万人以上ともいわれていることを考えれば，これは当然の話かもしれません。現在，自分が取り組んでいる問題は，すでに PHP デベロッパの誰かが直面し，解決した上で汎用的なライブラリとして公開している可能性はじゅうぶんにあります。ですから，ソリューションの密林の中から有用なライブラリをいかに少ない労力で安全に探し出すか，それがほんとうの課題であるということができるでしょう。

QR コードを作成したい，PDF ファイルからテキストを抽出したい，凝ったグラフィックを作成したい，といった課題に面したとき，はじめは 4D 本体のライブラリでどこまでできるのか，調査するのは道理にかなったことです。それでも良い方法がみつからないとき，Google のような Web サービスを活用し，調査の範囲を広げると良いでしょう。PHP であれば，定評のあるサイトからチェックすることができます。

http://www.w3schools.com/php/

http://php.net/

それでもまだ，良い方法がみつからないとき，「QR Code generator PHP libary」といった風に，検索条件を絞り込んでみてください。オンラインでサンプルを試し，気に入った（この段階では，最低限のレベル，つまりライブラリが期待どおりの出力をするという意味）ところで，ソースコードをダウンロードすることができます。はじめは，ダウンロードしたソースコードのファイル数に圧倒されるかもしれません。PHP デベロッパーの多くは，Unix 星から来た Unix 星人です。彼らは，ひとつのプロジェクトに大量のファイルを使用するという特徴があります。これは，Unix 星の習慣です。あまり気にしてはいけません。

実際，我々にとって興味があるのは，主要なファイル，普通は index.php という名称のファイルだけです。ここに挙げたのは，phpqrcode フォルダーから取られた例です。

//...は，この箇所でコードが省略されていることを示しています。

青のテキストは，重要ではないコードです。

赤のテキストは，筆者が加えたコメントです。

```
<?php
echo "<h1>PHP QR Code</h1><hr/>";
//set it to writable location, a place for temp generated PNG files
$PNG_TEMP_DIR =
dirname(__FILE__).DIRECTORY_SEPARATOR.'temp'.DIRECTORY_SEPARATOR;
//...
include "qrlib.php";
//ここに必要なファイルがすべて含まれている
//...
$filename = $PNG_TEMP_DIR.'test.png';
//ピクチャの保存先ファイル名
//...
```

```
$errorCorrectionLevel = 'L'; //デフォルト値
if (isset($_REQUEST['level']) && in_array($_REQUEST['level'],
array('L','M','Q','H')))
$errorCorrectionLevel = $_REQUEST['level'];
$matrixPointSize = 4; //デフォルト値
if (isset($_REQUEST['size']))
$matrixPointSize = min(max((int)$_REQUEST['size'], 1), 10);
if (isset($_REQUEST['data'])) {
if (trim($_REQUEST['data']) == '')
die('data cannot be empty! <a href="?">back</a>');
//G I G O...
$filename = $PNG_TEMP_DIR.'test'.md5
($_REQUEST['data'] .'|'.$errorCorrectionLevel.'|'.$matrixPointSize).'.png';
//書き出しファイルだが無視して良い
//ここからが大事！
QRcode::png($_REQUEST['data'], $filename, $errorCorrectionLevel,
$matrixPointSize, 2);
} else {
echo 'You can provide data in GET parameter: <a
href="?data=like_that">like that</a><hr/>';
QRcode::png('PHP QR Code :)', $filename, $errorCorrectionLevel,
$matrixPointSize, 2);
//QRcode::png はクラスの png メソッドを指している。定義しているファイルは phpqrcode.php
class QRcode {
...
public static function png($text, $outfile = false, $level = QR_ECLEVEL_L,
$size = 3, $margin = 4, $saveandprint=false)
...
}
```

このようなコードを見つけることができれば「ビンゴ」です。そのライブラリは，ほぼ原形のまま 4D に取り入れることができます。あとは，データの入出力方法を特定するだけです。

http://phpqrcode.sourceforge.net/#demo

3. PHP ライブラリの内部を研究する

PHP は，数あるプログラミング言語のひとつに過ぎません。ですから，人類が操るどんな言語よりもずっと単純です。それでも，落とし穴，偽りの友，文法上の例外など，デベロッパーに対する嫌がらせのために設置されたとしか思えない，厄介な邪魔者が存在します。強者以外を認めようとしない，狭量なマッチョ・コーディングの聖典は，「コードが読めないものに，コードを書く資格なし」という一節ではじまるのです。

4D デベロッパーのために，PHP 言語と 4D 言語を比較し，注意するべきポイントを簡単にまとめてみました。

PHP 変数のルール

変数は，$記号と変数名で記述します。

変数名の最初の文字は，英字またはアンダースコア記号です。

変数名には，英数字およびアンダースコア記号だけが使用できます。（A-z, 0-9, _）

変数名にスペースを含めることはできません。

変数前は，大文字と小文字を区別します。（$y と$Y は別々の変数）

配列:

PHP の添字配列は 4D の配列と基本的に同じですが，要素番号は 1 ではなく，0 から数えます。4D の配列にも 0 番要素は存在しますが，基本的には使用されません。

```
$age=array(35, 42, 56, 54, 43);
$zAge = $age[0];
```

PHP の連想配列は，JSON 配列や 4D の ARRAY OBJECT と基本的に同じです。

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");

$age['Peter']="35";
$age['Ben']="37";
$age['Joe']="43";

$zAge = $age['Ben'];
```

PHP 多次元配列は，4D の二次元配列と似ていますが，次元数に限定されていない上，連想配列や値タイプのまちまちな配列を含めることができます。

```
$cars = array //二次元配列
array: (
array("Volvo",100,96),
array("BMW",60,59),
array("Toyota",110,100)
);
```

PHP には，4 種類の変数スコープがあります。

ローカル: PHP 関数の内部で宣言された変数。4D のローカル変数に相当します。

グルーバル: 関数の外部で宣言された変数。プロセス変数に相当します。

スタティック: 関数の終了時に値が失われないローカル変数です。

パラメーター: 4D の引数に相当します。

関数の内部からグローバル変数にアクセスするには， キーワード global を使用します。

```
<?php//PHP のはじまり
  $x=5; //グローバルスコープ
  $y=10;
$  z=2
function myTest($p) //パラメーター
{
  $k=0;//ローカルスコープ
  static $n=0; //スタティック，クリアされない
  global $x,$y;
  $n++;
  $k=(($x+$y)*$p)+$n;
  return $k; //4D の$0 みたいなもの
}
echo (myTest($z)); //出力 31
echo (myTest($z)); //出力 32 ($n の影響)
?>//PHP のおわり
```

4D と PHP の共通点と相違点

クラスメソッド内のコード

非スタティックプロパティは，->（オブジェクト演算子）でアクセスします。下記の例で property はプロパティ名です。

```
$this->property
```

スタティックプロパティは，::（ダブルコロン）でアクセスします。

```
self::$property
```

疑似変数$this は，オブジェクトのコンテキスト内でコールされたクラスメソッド内であれば，どこでも使用することができ，それをコールしたオブジェクト（大抵はそのメソッドが

属するオブジェクトですが，別オブジェクトのコンテキストから静的にメソッドがコールされたのであれば，他方のオブジェクト）に対する参照です。4D の Self と似ています。

@について: @は，メールアドレスではないので注意してください。PHP には，エラー制御演算子がひとつだけ存在します。それが@です。PHP 式の冒頭に@が置かれた場合，その式を評価したことによってエラーが返されたとしても，エラーは無視されます。

式に@記号が使用されているときは，set_error_handler() でセットされたカスタムエラー処理関数の中で error_reporting() をコールしても 0 が返されます。PHP の設定で track_errors が有効にされているのであれば，式によって返されるエラーメッセージは $php_errormsg 変数に代入されます。この変数の内容は，エラーが返されるたびに更新されるので，できるだけ早くチェックするようにしてください。

```
<?php
//意図的なファイルエラー
$my_file = @file ('non_existent_file') or
  die ("ファイルを開くことができません: エラー'$php_errormsg'");
//@は，関数に限らずどんな式でも使用できる
$value = @$cache[$key];
//この場合，添字$key が存在しなかったとしてもエラーは返されない
?>
```

注記：@演算子の効力は，式に限定されています。式かそうでないかを見分ける基準として，値が返されるものは式である，よって@を使用することができる，と考えることができます。具体的には，変数，関数，定数，インクルードなどです。逆に，関数やクラスの定義，if や foreach などの条件分岐構文には@を使用することができません。

参照

4D のポインターは，オブジェクトに対する明示的な参照であると考えることができます。これに対し，リスト参照，メニュー参照などは，暗示的な参照です。

PHP には，ポインターとまったく同じような参照は存在しません。&演算子は，参照を作るものですが，これは複数の変数が同じ内容を指すようにするものです。

たとえば，4D ではこのようなコードを記述することができます。

```
$b := 10
$a := ->$b
```

$a はポインターであり，$b は変数です。10 という値を受け取るには，$a->を使用します。

PHP では，このようにコードを記述します。

```
<?php
$b=10;
$a =& $b;
?>
```

これは，$a と$b が同じ内容を指していることを意味しています。したがって，$a および $b は 10 を返します。

落とし穴に注意してください。

```
$a &= $b //$a = $a & $b の短縮形，つまりビットワイズ演算

$a =& $b //$a に$b の内容の対する参照を代入
```

PHP の演算子で迷ったときには，下記の URL を参照すると良いでしょう。

http://stackoverflow.com/questions/3737139/reference-what-does-this-symbol-mean-in-php

参照は，変数を渡すためにも使用されます。関数の中でローカル変数を宣言し，その関数をコールしたスコープの変数と同じ内容を参照させることができます。

```
<?php
function foo(&$var) {
  $var++; }
$a=5;
foo($a);
?>
```

この場合，$a には 6 が代入されます。

参照を値を返すこともできます。

```
<?php
class foo {
  public $value = 42;
  public function &getValue() {
    return $this->value;
  }
}
```

```
$obj = new foo;
$myValue = &$obj->getValue();
// $myValue は$obj->value に対する参照，つまり 42
$obj->value = 2;
echo $myValue;
//$obj->value の新しい値，つまり 2 が返される
?>
```

4D のようなポインターが必要な場面では，PHP の可変変数を使用します。可変変数とは，変数名を動的に指定することができる変数のことです。普通の変数は，つぎのように記述して値をセットします。

```
<?php
$a = 'hello';
?>
```

これに対し，可変変数は，変数の値をもってそれを自身の名前とします。上記の例では，hello という値が変数に代入されたので， $$演算子でその名前の変数を使用することができます。

```
<?php
$$a = 'world';
?>
```

これにより，PHP のシンボルツリーには，ふたつ変数が定義されたことになります。"hello"という内容の$a，そして"world"という内容の$hello です。

4D ポインターと比較するならば，$$a='b'は，$a:=->$b になぞらえることができます。$b の値を取得するには，4D では$a->，PHP では${$a}と記述します。

シングル引用符とダブル引用符

HTML とは違い，PHP では，シングル引用符とダブル引用符の働きが一緒ではありません。シングル引用符の中では，変数や特殊な文字などのエスケープ・シーケンスは展開されません。これに対し，ダブル引用符の中では，変数名が展開されます。

```
<?php
$a = 'hello';
echo 'a is $a';
//出力: a は $a echo "a is $a";
//出力: a は hello
?>
```

Heredoc（Here Document）テキストは，ダブル引用符で括られた文字列と同じように処理されますが，ダブル引用符を省略できる点が違います。heredoc 内では，引用符をエスケープする必要がありません。

```
$a = 10;
$b = 'hello';
$str = <<<END_STR
a is $a
and "b" is $b
END_STR;
echo $str;
//出力
a is 10
and "b" is hello
```

バックティック（`）は特に注意を要する文字です。これは，引用符などではなく，おそるべき実行演算子だからです。PHP は，バックティック内の文字列をシェルコマンドとして実行し，結果を変数あるいは標準出力に返します。バックティック演算子を使用することは，shell_exec() をコールするの同じです。

```
<?php
  $output = `ls -al`;
  echo "<pre>$output</pre>";
?>
```

演算子

==（equal，等価）と===（identical, 同一）を混同しないように注意してください。

```
<?php
if("22" == 22); // True
if("22" === 22); // False
?>
```

!（論理否定）と~（ビットワイズ排他）も混同しやすい演算子です。

ときおり!!（二重否定）演算子に出くわすかもしれません。これは，真偽値を boolean の TRUE または FALSE に型変換（キャスト）するため良く使われる手法です。非常に驚いているという意味ではありません。

4.　PHP を 4D に結合するときに苦労しないために

最重要チェックポイント

php.ini この初期化ファイルは，データベースの Resources フォルダーに作られます。.ini ファイルの中では，コメントは// ではなく，;で始めます。また，改行コードは，CR でも; でもなく，EOF です。仕様を決めた人は，一体，何を考えていたのでしょう？

ファイルには，ユーティリティ・スクリプト 4D_Execute_PHP.php に対するフルパスをしめた auto_prepend_file 定義が含まれているはずです。ファイル冒頭のコメントにも，有用な情報が含まれています。

前述のスクリプトの場所は，4D アプリケーション/ Resources/php/Windows（または /Mac）です。

このスクリプトは，include() というよりも，require() のような働きをします。これがみつからないと，4D はスクリプト全体を実行することができても，スクリプト内のルーチンが実行できないことになります。

display_errors は，stderr にセットされているべきです。そうなっていれば，4D は PHP コードの実行中に発生したエラーを処理することができます。

```
display_errors = "stderr"
error_reporting = E_ALL & ~E_NOTICE
```

簡単な例

4D におけるもっとも簡単な PHP の用法は，4D の PHP インタープリターにはじめから用意されている PHP 関数を直接コールする，というものです。

たとえば，ある年のイースター（復活祭）の日付が調べたい，としましょう。もちろん，日付を計算する方法を知っていれば，4D だけでも算定することができます。しかし，PHP の Calendar ライブラリには，そのための関数がすでに存在するのです。

http://php.net/manual/en/function.easter-date.php

マニュアルページによれば，関数名は easter_date() です。

関数からは，該当する日の真夜中に対応する UNIX タイムスタンプが返されます。

やってみましょう。

```
C_LONGINT($dayNb;$month;$day;$year)
C_TEXT($EasterDate;$UnixStamp)
$date:=Current date
$year:=Num(Request("年を入力:"; String(Year of($date))))

$path:="" //PHP 関数を直接コールするのでパスは不要
$isOK:=PHP Execute($path;"easter_date";$UnixStamp;$year)
If ($isOK)
  //UNIX タイムスタンプを日付に変換
  $isOK:=PHP Execute($path;"date";$EasterDate;"M-d-Y";$UnixStamp)
  ALERT($EasterDate)
End if
```

別の例として，テキストの中にそれぞれの文字が出現する回数を調べたいケースを考慮してみましょう。ラッキーなことに，PHP の String ライブラリには count_chars() という関数が用意されています。

```
mixed count_chars ( string $string [, int $mode = 0 ] )
```

この関数は，テキスト内のバイト値（0~255）の出現回数をカウントし，指定したフォーマットで返します。モードの選択肢は下記のとおりです。

0 - バイト値をキー，回数を値とした，キー/値の配列

1- 上記と同じだが，0 回の文字は省略

2- 上記とは逆に，0 回の文字のみ

3- 含まれている文字を文字列で

4- 含まれていない文字を文字列で

4D メソッドに組み込めばこのような感じになります。

```
C_TEXT($result)
$path:="" //PHP 関数を直接コールするのでパスは不要
$str:="Here is the text we want to process"
$mode:=1 //無難なフォーマット
$isOK:=PHP Execute($path;"count_chars";$result;$str;$mode)
If ($isOK)
  ALERT($result)
```

```
  //返り値 {"32":64, "40":1, "44":4,"45":5, "47":2,"48":1, "49":1, "50":1, ...
"51":1, "52":1}
  //これを 4D の配列に変換
  $result:=Substring($result;2) //最初の'{'を除去
  $result:=Substring($result;1;Length($result)-1)//最後の'}'を除去
  ARRAY TEXT($arTexts;0)
  $isOK:=PHP Execute($path;"explode";$arTexts;",";$result)
  //テキストを配列に展開する便利な PHP 関数
  If ($isOK)
    $n:=Size of array($arTexts)
    ARRAY INTEGER($arOccurences;$n)
    ARRAY TEXT($arChar;$n)
    For ($i;1;$n)
      $text:=Replace string($arTexts{$i};Char(Double quote);"")
      $pos:=Position(":";$text)
      If ($pos>0)
        $arChar{$i}:=Char(Num(Substring($text;1;$pos-1)))
        $arOccurences{$i}:=Num(Substring($text;$pos+1))
      End if
    End for
  End if
End if
```

このように，PHP の便利な関数は，とても簡単に 4D からコールすることができます。

PHP ファイルを使用する

ここに挙げたのは，http://wiki.pchart.net/doc.dataset.normalize.html に掲載されていた PHP のサンプルコードです。このままでは 4D に取り込めないので，多少，手を加えることにしましょう。

```
// Standard inclusions
include("class/pData.class.php");
include("class/pDraw.class.php");
include("class/pImage.class.php");
$MyData = new pData();
$MyData->addPoints(array(-4,VOID,VOID,12,8,3),"Frontend #1");
$MyData->addPoints(array(3,12,15,8,5,-5),"Frontend #2");
$MyData->addPoints(array(2,7,5,18,19,22),"Frontend #3");
$MyData->setAxisName(0,"Average Usage");
$MyData->addPoints(array("Jan","Feb","Mar","Apr","May","Jun"),"Labels");
$MyData->setSerieDescription("Labels","Months");
$MyData->setAbscissa("Labels");
$MyData->normalize(100,"%");
$myPicture = new pImage(700,230,$MyData);
$myPicture->drawGradientArea(0,0,700,230,
DIRECTION_VERTICAL,array("StartR"=>240,
"StartG"=>240,"StartB"=>240,"EndR"=>180,"EndG"=>180,"EndB"=>180,"Alpha"=>10
0));
$myPicture->drawGradientArea(0,0,700,230,
DIRECTION_HORIZONTAL,array("StartR"=>240,
"StartG"=>240,"StartB"=>240,"EndR"=>180,"EndG"=>180,"EndB"=>180,"Alpha"=>20
));
```

```
$myPicture->setFontProperties(array("FontName"=>"fonts/pf_arma_five.ttf",
"FontSize"=>6));
$myPicture->setGraphArea(60,20,680,190);
$myPicture-
>drawScale(array("DrawSubTicks"=>TRUE,"Mode"=>SCALE_MODE_ADDALL));
$myPicture-
>setShadow(TRUE,array("X"=>1,"Y"=>1,"R"=>0,"G"=>0,"B"=>0,"Alpha"=>10));
$myPicture->drawStackedBarChart(array("DisplayValues"=>TRUE,
"DisplayColor"=>DISPLAY_AUTO, "Rounded"=>TRUE,"Surrounding"=>60));
$myPicture->setShadow(FALSE);
$myPicture->drawLegend(480,210, array("Style"=>LEGEND_NOBORDER,
"Mode"=>LEGEND_HORIZONTAL));
$myPicture->Render("normalize.png");
```

この PHP を 4D からコールできるように，まずコード全体を関数に書き換えます。

```
<?php
// Standard inclusions
include("class/pData.class.php");
include("class/pDraw.class.php");
include("class/pImage.class.php");

function graph_in_PNG () {
$MyData = new pData();
$MyData->addPoints(array(-4,VOID,VOID,12,8,3),"Frontend #1");
$MyData->addPoints(array(3,12,15,8,5,-5),"Frontend #2");
$MyData->addPoints(array(2,7,5,18,19,22),"Frontend #3");
$MyData->setAxisName(0,"Average Usage");
$MyData->addPoints(array("Jan","Feb","Mar","Apr","May","Jun"),"Labels");
$MyData->setSerieDescription("Labels","Months");
$MyData->setAbscissa("Labels");
$MyData->normalize(100,"%");
$myPicture = new pImage(700,230,$MyData);
$myPicture->drawGradientArea(0,0,700,230,
DIRECTION_VERTICAL,array("StartR"=>240,
"StartG"=>240,"StartB"=>240,"EndR"=>180,"EndG"=>180,"EndB"=>180,"Alpha"=>10
0));
$myPicture->drawGradientArea(0,0,700,230,
DIRECTION_HORIZONTAL,array("StartR"=>240,
"StartG"=>240,"StartB"=>240,"EndR"=>180,"EndG"=>180,"EndB"=>180,"Alpha"=>20
));
$myPicture->setFontProperties(array("FontName"=>"fonts/pf_arma_five.ttf",
"FontSize"=>6));
$myPicture->setGraphArea(60,20,680,190);
$myPicture-
>drawScale(array("DrawSubTicks"=>TRUE,"Mode"=>SCALE_MODE_ADDALL));
$myPicture-
>setShadow(TRUE,array("X"=>1,"Y"=>1,"R"=>0,"G"=>0,"B"=>0,"Alpha"=>10));
$myPicture->drawStackedBarChart(array("DisplayValues"=>TRUE,
"DisplayColor"=>DISPLAY_AUTO, "Rounded"=>TRUE,"Surrounding"=>60));
$myPicture->setShadow(FALSE);
$myPicture->drawLegend(480,210, array("Style"=>LEGEND_NOBORDER,
"Mode"=>LEGEND_HORIZONTAL));

//返り値をファイルに書き出す場合
$myPicture->Render("normalize.png");


//返り値を stdOut として BLOB で受け取る場合
echo ($myPicture->Render("")); }
```

```
?>
```

こうすることにより，とりあえず 4D からこの PHP がコールできるようになります。

```
$result:=""
$fl_OK:=PHP Execute($path;"graph_in_PNG";*)//返り値を受け取らない
If ($fl_OK)
  PHP GET FULL RESPONSE(v_StdBlobOut)
  //stdOut に送られた値はこちらで受け取る
  C_PICTURE(vPNG)
  BLOB TO PICTURE(v_StdBlobOut;vPNG)
End if
```

つぎに，パラメーター化したい部分を変数に書き換えてテストします。

```
<?php
// Standard inclusions
include("class/pData.class.php");
include("class/pDraw.class.php");
include("class/pImage.class.php");

function graph_in_PNG () {

$pictWidth = 700;
$pictHeight = 230;
$gradStart = 240; //0..255
$gradEnd = 180; //0..255
$fontName = "fonts/pf_arma_five.ttf";
$fontSize = 6;

$MyData = new pData();
$MyData->addPoints(array(-4,VOID,VOID,12,8,3),"Frontend #1");

$MyData->addPoints(array(3,12,15,8,5,-5),"Frontend #2");
$MyData->addPoints(array(2,7,5,18,19,22),"Frontend #3");
$MyData->setAxisName(0,"Average Usage");
$MyData->addPoints(array("Jan","Feb","Mar","Apr","May","Jun"),"Labels");
$MyData->setSerieDescription("Labels","Months");
$MyData->setAbscissa("Labels");
$MyData->normalize(100,"%");
$myPicture = new pImage($pictWidth,$pictHeight,$MyData);
$myPicture->drawGradientArea(0,0,$pictWidth,$pictHeight,
DIRECTION_VERTICAL,array("StartR"=>$gradStart,
"StartG"=>$gradStart,"StartB"=>$gradStart,"EndR"=>$gradEnd,"EndG"=>$gradEnd,
"EndB"=>$gradEnd,"Alpha"=>10 0));
$myPicture->drawGradientArea(0,0,$pictWidth,$pictHeight,
DIRECTION_HORIZONTAL,array("StartR"=>$gradStart,
"StartG"=>$gradStart,"StartB"=>$gradStart,"EndR"=>$gradEnd,"EndG"=>$gradEnd,
"EndB"=>$gradEnd,"Alpha"=>20 ));
$myPicture->setFontProperties(array("FontName"=>$fontName,
"FontSize"=>$fontSize));

$myPicture->setGraphArea(60,20,680,190);
$myPicture-
>drawScale(array("DrawSubTicks"=>TRUE,"Mode"=>SCALE_MODE_ADDALL));
$myPicture-
>setShadow(TRUE,array("X"=>1,"Y"=>1,"R"=>0,"G"=>0,"B"=>0,"Alpha"=>10));
$myPicture->drawStackedBarChart(array("DisplayValues"=>TRUE,
"DisplayColor"=>DISPLAY_AUTO, "Rounded"=>TRUE,"Surrounding"=>60));
```

```
$myPicture->setShadow(FALSE);
$myPicture->drawLegend(480,210, array("Style"=>LEGEND_NOBORDER,
"Mode"=>LEGEND_HORIZONTAL));

//返り値をファイルに書き出す場合
$myPicture->Render("normalize.png");


//返り値を stdOut として BLOB で受け取る場合
echo ($myPicture->Render("")); }
?>
```

仕上げに，4D からパラメーターが渡されるようにコードを書き換えます。

```
function graph_in_PNG
($pictWidth ,$pictHeight ,$gradStart ,$gradEnd ,$fontName ,$fontSize) {
//...
}
```

配列データなども 4D から供給することができます。

```
function graph_in_PNG
($vSerie1,$vSerie2,$vSerie3,$Data1,$Data2,$Data3,$Categ,$Title,$sizeX,$sizeY
) {
$MyData = new pData();
$aSerie1=explode("_",$Data1);
$aSerie2=explode("_",$Data2);
$aSerie3=explode("_",$Data3);
$aCateg =explode("_",$Categ);
$MyData->AddPoint($aSerie1,$vSerie1);
$MyData->AddPoint($aSerie2,$vSerie2);
$MyData->AddPoint($aSerie3,$vSerie3);
$MyData->setAxisName(0,$ Title);
$MyData->addPoints(array($aCateg),"Labels");
$MyData->setSerieDescription("Labels","Months");
$MyData->setAbscissa("Labels");
$MyData->normalize(100,"%");
$myPicture = new pImage($sizeX,$sizeY,$MyData);
$grad1=$sizeY+10;
$grad2=$sizeY-50;
$myPicture->drawGradientArea(0,0, $sizeX, $sizeY,
DIRECTION_VERTICAL,array("StartR"=>$grad1,  "StartG"=>$grad1,"StartB"=>$grad1,
"EndR"=>$grad2, "EndG"=>$grad2, "EndB"=>$grad2,"Alpha"=>100));
//...
```

PHP に慣れないうちは，少しずつ，コードを書き換えてゆくのが良いと思います。一度にあまり多くの変更を加えないほうが無難です。

PHPExcel Library など，公開されているテクニカルノートも参考にすることができます。

http://kb.4d.com/assetid=76312

5. いつ JavaScript を使用するか

PHP に慣れたところで，今度は JavaScript にチャレンジしてみましょう。両言語には，かなりの共通点があります。とはいえ，落とし穴，偽りの友，文法上の例外など，デベロッパーに対する嫌がらせのために設置されたとしか思えない，厄介な邪魔者がやはり存在します。

JavaScript をはじめるべき理由は少なくともふたつ挙げることができます。

- Web エリアを通して多数の Web サービスにアクセスできる
- 4D-Mobile を使用するときに必要

PHP と同じように，JavaScript には多数のライブラリや関数が存在し，インタフェースの課題や，4D に存在しない関数を補う点で役立つことが少なくありません。

たとえば，QR コードの生成であれば，簡単な URL コールを Web エリアに渡すだけで実現することができます。

```
$url := "http://chart.googleapis.com/chart?"
$url := $url + "cht=qr" \\ask a DR Code
$url := $url + "&chs=200x200" \\ specifies the size
$url := $url + "&chl=" + $MyText2Encode + "/" \\ and text to encode
WA OPEN URL(MyWArea;$url)
```

あるいは，**WA Get page content** を使用し，HTML を挿入した後，**WA Set page content** で結果を受け取ることができます。

```
<img src="http://chart.googleapis.com/chart?cht=qr&chs=200x200&chl=" +
$MyText2Encode +" />
```

JavaScript のライブラリは，インターネットで探すことができます。

http://en.wikipedia.org/wiki/Category:JavaScript_libraries

上記ページには，100 以上の JavaScript ライブラリが列挙されています。

WA Execute JavaScript あるいは **WA EXECUTE JAVASCRIPT FUNCTION** でこうしたライブラリを活用してみてはいかがしょうか。

6. JavaScript と JSON: もはや避けては通れない道，心の準備をするように

JSON (JavaScript Object Notation) は，データ交換のためのテキスト型・オープン・スタンダードで，XML よりもずっとシンプルなフォーマットであることが特徴です。
JavaScript で構造的データや連想配列などのオブジェクトを表現するためのリテラル表記に由来しますが，PHP や 4D のようなプラットフォームでも使用することができます。

下記は JSON 構造体の例です。

```
{
"firstName": "John",
"lastName": "Smith",
"age": 25,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": 10021
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234" },
{
"type": "fax",
"number": "646 555-4567" }
]
}
```

これが連想配列であることは，一目で判断することができます。phoneNumbers など，一部の要素は，それ自体が連想配列になっています。{と}の間のある部分は，構造体データであり，[と]の間にある部分は，配列データです。

JSON オブジェクトは，4D，PHP，JavaScript，REST サーバー同士でデータを交換する上でとても都合が良いということができます。

http://php.net/manual/en/function.json-encode.php
http://php.net/manual/en/function.json-decode.php

7. プラグインに対する依存から脱却する

4D プラグインの歴史は，最初期の 4D にまで遡ることができます。寄せられる要望すべてに応えることはできないことに気づいた LR（ロホン・リバルディエール）は，誰もが自分でプラグインを書けるようにしました。私は，彼とのやり取りをはっきり思い出すことができます。

LR: 「4D は何か機能が足りないと感じることはない？」

聴衆: 「それはもう，いますぐにでも必要なものがあるのだ，これとこれと…」

彼は笑顔で返しました，

LR: 「じゃあ良い知らせだね，これからは自分で機能が追加できるようにしたよ」

聴衆: 「...。」

当時は，PASCAL でエクスターナルを書きました。その後，大量の汗を流して，C また C++でプラグインを書き直しました。その後，Windows に移植し，その後，Mac OS X に書き直し，システムが変わるたび，4D が変わるたび，Carbon を取り入れ，Carbon を取り除き，Altura に対応し，Altura に対応するのをやめ，Think Pascal，MPW (Macintosh Programmer's Workshop)，Object Master，CodeWarrior, Xcode とツールを変えながら，何度も，何度も，書き直したのです。

経験から導きだせる結論はこうです。プラグインに関わると大変なことになります。

もちろん，メンテナンスの行き届いた，優良なプラグインにおおきな価値があることは認めます。ただ，ちょっとした必要であれば，PHP か JavaScript などの既存のライブラリでまかなえるものです。あるいは，やる気があれば，それらの言語を習得し，自分でモジュールを作成できるかもしれません。それらの言語は，（少なくとも建前上は）システムに依存しないものだからです。

4D Server: 4 人組ユニット（ファンタスティック・フォー）

---

1. 統合サーバーそれぞれの能力を最大限に引き出すには

4D と 4D Server の間には，かなりの共通点があります。したがって，以下に続く内容の一部は，4D Server だけでなく，4D（ローカル・モード）にもあてはまるものです。それでも，サーバーの話が強調されているのは，サーバー側のコーディングにまずい点があった

場合，その影響はクライアント数，あるいはクライアント数の平方にさえ比例し，いっそう深刻なものになるからです。

4D Server には，4 種類の主要なサーバーが存在します。

A)　DB4D

これがデータベース・サーバーです。データに対するあらゆるアクセスは，その一部であるデータベース・エンジンが実行します。他のサーバーは DB4D と対話するようになっており，直接データにアクセスすることはありません。

B)　アプリケーション・サーバー

4D プロセスを起動し，4D メソッドを実行するサーバーです。4D クライアントをコントロールし，必要に応じて 4D プロセスを作成します。データに対するアクセスやクエリを実行するときには，DB4D と対話します。

C)　SQL サーバー

SQL クエリを実行するサーバーです。SQL エンジンに渡されたクエリを処理しますが，直接データにアクセスすることはなく，SQL クエリを解析して 4D 内部コールに変換した後は，DB4D と対話します。

D)　HTTP サーバー

HTTP リクエストおよび REST（Representational State Transfer）リクエストを処理するサーバーです。HTTP および REST の仕様に基づいた API であるともいえます。おもに Web サーバーとして使用されますが，Wakanda REST サービスが有効にされているときには，データサーバーとして機能します。それでも，直接データにアクセスすることはなく，データに対するアクセスやクエリを実行するときには，DB4D と対話します。

2.　コオペラティブ vs プリエンプティブ

この話題を考慮するときには，次のことを憶えておいてください。

スレッドとプロセスを取り違えてはいけない

4D デベロッパーは，しばしば「プロセス」を個別のメソッドを平行処理させるための，独立した世界（実行コンテキスト）のように考えます。それは間違いではありませんが，厳密に正確というわけでもありません。4D 内のプロセスは，いくつかのオブジェクトを共有しています。実際，インタープロセスと銘打たれたもの（変数，セット，命名セレクション）は，どれもプロセス間で共通のものです。そのようなわけで，4D のプロセスを個別のスレッドで実行させることはできません。プロセス群は，4D のスケジューラーにより管理されています。

スレッドは，システム・レベルで定義されているもので，オペレーション・システムのスケジューラーが管理できる，プログラムされた命令の独立した最小の単位です。

4D v11 SQL また 4D Server v11 SQL は，マルチスレッド・アプリケーションであり，マルチコア・アーキテクチャの恩恵にあずかることができます。4D デベロッパーは， 4D とオペレーション・システムそれぞれにより，スレッドがどのように処理されるのか理解していれば，データベースの速度をほんとうに向上させる目的でコードを調整することができます。また，CPU サイクルを監視し，ある程度のフィードバックをオペレーション・システムから得ることにより，自分のコードがほんとうにマルチスレッド・モデルを利用しているのかどうか，見極めることができます。

データに対するあらゆるアクセス，インデックス処理，データベース処理，および SQL エンジンに対するコールは，特に手を加えなくても，自動的にマルチスレッド・モデルの恩恵にあずかることができます。同じことは，キャッシュのディスク書き込み，インデックスの作成，ログファイルの作成など，4D が内部的に実行するプロセスの大多数にあてはまります。

このように，4D アプリケーションはいくつものプロセスを同時に実行しているのであり，マルチタスクであるということができます。あるスレッドが入出力を待機している間，別のスレッドではクエリが実行されるかもしれません。この種のマルチタスクでは，システムスレッドが使用されるため，マルチコアあるいはマルチプロセッサのプラットフォームでは，全体のパフォーマンス向上が見込めます。

これはスケーラビリティに貢献します。サーバーに接続した多数のクライアントは，同時に多数のデータベース処理を実行できるからです。そうしたデータベース処理は，いくつものプリエンプティブ・スレッドで同時に実行されます。この平行処理の利点は，CPU の数を増やすことにより，同時処理能力を伸ばすことができることにあります。特定のクライアント・プロセスが他のクライアント・プロセスよりも優遇されることはありません。

インデックスの作成，データキャッシュの管理，SQL サーバーに対する外部コールは，いずれもプリエンプティブ・スレッドが処理します。4D は，作成したプリエンプティブ・スレッドの実行スケジュールを管理する必要がなく，それはすべてオペレーション・システムに委ねられます。したがって，このモデルの受益者は，4D（ローカル）よりも 4D Server です。

DB4D の実行には，ローカルモードでコオペラティブ，サーバーモードではプリエンプティブ・スレッドが使用されます。スレッドとは，おおまかにいって，共通のメモリ・コンテキストで実行されるプロセスのことです。スレッドが軽量といわれているのは，スレッドの切り替えには，メモリ・コンテキストの切り替えが伴わないからです。

コオペラティブ・マルチタスキング（タイム・シェアリング，または非プリエンプティブ・マルチタスキング）は，オペレーション・システムが実行中のプロセスから別のプロセスにコンテキストを切り替えることをしないタイプの平行計算処理です。そのようなシステムでは，実行中のタスクは自発的に計算処理を中断し，他のタスクに制御を渡すことにより，つまり互いに協力することにより，マルチタスキングを可能にしています。

一方，プリエンプティブ・マルチタスキングは，オペレーション・システムがタスクの中断と再開を監督し， 各タスクの協力を必要としないタイプの平行計算処理です。

DB4D のタスクは，そのほとんどがプリエンプティブです。つまり，プリエンプティブ・スレッドで実行されます。クライアント側でローカルではないプロセスが作成されるたびに，サーバー側に対応するプロセスが作成されます。これらのプロセスは，コオペラティブ・スレッドで実行され，4D がプロセス同士の同期をコントロールします。4D サーバーには，クライアント側で作成されたプロセス毎に，対応するプリエンプティブ・プロセスも作成されます。DELAY PROCESS や GET PROCESS VARIABLE などのリクエストは，コオペラ

ティブ・プロセスのほうが処理します。クエリ，並び替え，インデックス操作などのデータベース・エンジン・リクエストは，プリエンプティブ・プロセスのほうが処理します。そのような処理は，複数のプロセッサ・コアに分散されます。加えて，Begin SQL/End SQL 構文で SQL リクエストを実行するときには，プリエンプティブ・スレッドがもうひとつ作成されます。

アプリケーションのメソッドを実行するプロセスは，コオペラティブであり，4D スケジューラーの配下にあります。すべてのプロセスは単一のスレッドで実行されており，4D エンジンに対するコールのない繰り返し処理などでこのスレッドがロックされることのないよう，メソッドを記述するときには配慮しなければなりません。

4D は，プリエンプティブ・プロセス毎に 1MB のメモリ・スタックを配分します。スタックのサイズは，SET DATABASE PARAMETER のセレクター53 番で変更することができます。このサイズを増量した場合，クライアントの同時接続数に比例して，プリエンプティブ・スレッドに配分されるメモリの量が増えるため，アプリケーション全体のパフォーマンスが低下する恐れがあります。デフォルト値は 1MB ですが，これを 512KB あるいは 256KB に引き下げることもできます。

3. 同期 vs 非同期

同期を取ることは，平行処理で頭を悩ませるおおきな問題のひとつです。ある種のタスクは，他に依存することなく完了できるのに対し，別のタスクは互いに関連しており，同期を取ることが求められます。あるプロセスが新しいプロセスを開始し，時間を要する何らかの処理をバックグラウンドで実行しなければならない場合，第一のプロセスがその結果を受け取る必要がない限り，第二のプロセスは非同期で実行させることができます。

この問題に対処するためのひとつの有効な手段はセマフォの使用です。第二のプロセスは，計算を開始する前にセマフォをセットし，結果が出た時点でそのセマフォを解除します。第一のプロセスは，結果が必要になるまで実行を続け，セマフォが解除されるまで待機します。

マルチプロセス・アプリケーションは，概して滑らかに動いているような印象を与えます。時間を要する処理は，バックグラウンドで実行されているからです。逆に，プロセス同士が

互いを待機しあう構図は，出口のない状況（キャッチ 22）を容易に発生させかねないため，デバッグの難易度はさらに高まります。また，タイミングが重要な鍵を握っているため，ロックの状況は，特定の条件下でしか発生しないかもしれません。しかも，マーフィーの法則どおり，そうした状況は，デバック中には決して起こらないのです。

4. 4D vs SQL: なぜ，いつ，どのような基準で選べば良い?

SQL は，データベース管理システムそのものでも，独立したソフトウェアでもなく，むしろデータベース管理システムの重要な構成部分のひとつであり，プログラミング言語とツールによってデータベースと対話するための方法を提供するものです。4D には，SQL コマンドを解析するための SQL サーバーが備わっていますが，実際のデータ処理は，DB4D，つまりデータベース・エンジンによって行なわれます。

4D のクエリは，どんな場合にも DB4D エンジンによって実行されます。SQL サーバーは，次のような目的で SQL エンジンを使用しています。

- SQL クエリやコマンドを解釈するため
- SQL を 4D の内部言語に変換するため
- DB4D に処理を委ねるため
- 結果を受け取って SQL 処理を完了するため

このことを考えれば，SQL クエリが 4D クエリよりも高速であるはずはないことは容易に理解することができます。実際のクエリを実行するのは，結局，同じエンジンだからです。

それでも，特定の場合には，SQL を使用することにメリットがあります。

- MySQL など，他のデータベースでコードを再利用したいとき
- 同時に複数のデータベースを開きたいとき
- SQL を書くことに慣れているから

・　速度が問題ではないとき

・　4D よりも SQL でクエリを記述したほうが簡単で済むとき

・　など

いずれにしても，最高のパフォーマンスが得たいのであれば，ODBC ではなく，4D 同士のダイレクト接続で SQL を実行するべきです。

5.　4D の内部的な最適化を踏まえ，その恩恵を最大限に活用する

最高のクエリ速度を追求するため，4D は経験則（ヒューリスティック）に基づいた複雑なアルゴリズムを使用しています。ヒューリスティックは，満足のゆく解に到達する過程で，思考上の近道を経由することにより，判断に必要な認識上の負担を軽減し，時間を短縮するという手法です。なぜ，厳格な論理ではなく， ヒューリスティックなのでしょうか。プログラミングの第一法則にもあるとおり，一概には言えない...（it all depends...）からです。

4D が判断材料に含めるのは，インデックスの有無，インデックスの種類，ディスクアクセスの回数，キャッシュの中身，などです。その後，4D は検索条件や比較演算子を並び替え，セレクションを作成するために最短の道を探ります。

たとえば，それができる場合，4D はシーケンシャルアクセスを始める前に，インデックスを使用してセレクションを絞り込もうとします。あるいは，特定のケースにおいては，敢えて逆の条件で検索を実行して結果を反転するかもしれません。

こうしたことは，クエリを記述する前に考慮に入れることが大切です。すでにじゅうぶん速いクエリをさらに速くする必要はないとしても，ユーザーが不便を感じるほど遅いクエリはなんとかしなければなりません。クエリの最適化が問題になるのは，ローカルモードよりも，リモートモードのほうです。

昔から，クライアント/サーバーの最大の弱点は，ネットワーク・アクセスでした。4D は，v11 以降，ネットワーク・アクセスの回数が少なくなるよう改良されています。それでも，クライアント/サーバーのシステムでは，クエリは必ずサーバー側で実行され，結果はクエリを要求したクライアントに返されなければなりません。一度で完了できない複雑なクエリ

の場合，セット演算を実行してセレクションを合成しなければならないかもしれません。たとえば次のようなクエリです。

(((Class="A@") & (Age>15)) | ((Class="Z@") & ((Age<15) | (Age>80)))) & (StillAlive | (NbKids>2))

4D のプログラミング言語でそのようなクエリを実行するためには，クエリをいくつものクエリに分解し，セットを作成し，セット演算を行ないます。そのような場合，クエリやセット演算は「サーバー側で実行」プロパティが有効にされたメソッドで処理し， 結果だけをクライアントに返すのが効果的です。

v11 以降，別の方法でもクエリが最適化できるようになり，わざわざセット演算を実行しなくても良くなりました。

QUERY BY FORMULA は，多くの場合において，クエリを実行するための最良の選択肢となりました。このコマンドは，できり限りインデックスを使用するようになり，また，フォーミュラがサーバーに転送できる限り，フォーミュラをサーバー側で実行するようになったからです。

前述の例は，次のように書き換えることができます。

QUERY BY FORMULA([Tbl]; ((([Tbl]Class="A@") & ([Tbl]Age>15)) | (([Tbl]Class="Z@") & (([Tbl]Age<15) | ([Tbl]Age>80)))) & ([Tbl]StillAlive | ([Tbl]NbKids>2))

引数が変数で渡されたとしても同じです。

```
<>vClass1:="A@"
<>vClass2:="Z@"
vAgeMin:=15
vAgeMax:=80
```

QUERY BY FORMULA([Tbl]; ((([Tbl]Class=vClass1) & ([Tbl]Age>vAgeMin)) | (([Tbl]Class=vClass2) & (([Tbl]Age<vAgeMin) | ([Tbl]Age>vAgeMax)))) & ([Tbl]StillAlive | ([Tbl]NbKids>2)))

このような場合，変数はクライアント側で評価され，その値が代入されたフォーミュラがサーバーに転送されます。

別の例を挙げましょう。

// <>ar_Array はクライアント側のインタープロセス配列
QUERY BY FORMULA([Tbl]; ([Tbl]Class= <>ar_Array{[Tbl]Age}))

この場合，クライアント側でフォーミュラ内の値を事前に評価することができないため，配列全体がサーバーに転送されます。

さらに別の例を挙げましょう。

QUERY BY FORMULA([Tbl]; (([Tbl]Nb= Log([Tbl]Age)| ([Tbl]Nb= Sin(Screen width)|([Tbl]Code= Current User)))

この場合，関数（**Current user**）がクライアント側で評価され，その値がフォーミュラとともにサーバーに転送されます。

最後に，このコマンドでは対処できない例を挙げましょう。

QUERY BY FORMULA([Tbl]; (***MyFunction***([Tbl]Age))

この場合，メソッド（***MyFunction***）はレコード毎にクライアント側で評価されなければなりません。このようなクエリは，「サーバー側で実行」プロパティが有効にされたメソッドの中に置くことにより最適化することができます。

セット演算と連続クエリを使用する従来の手法でも，次のような工夫をすることにより，最適化することができます。

SET QUERY DESTINATION(Into set;"Set1") を実行します。そうすれば，セレクションの作成を省略することができ，セットがそのままサーバー側で作成されるからです。

この場合，**UNION**("Set1"; "Set2"; "Set2") のようなセット演算は，セットがある場所，つまりサーバー側で実行されます。

さらに，**USE SET** コマンドも，セットがある場所，つまりサーバー側で実行されます。

したがって，**Execute on server** や「サーバー側で実行」プロパティを使用する必要はありません。

似たようなコマンド名ですが，**ORDER BY FORMULA** は，クライアント側で実行されるので注意してください。もちろん，「サーバー側で実行」プロパティを有効にしているのであれば，話は別です。

4D 自体もクエリを最適化していることに留意してください。たとえば，

QUERY([Tbl]; [Tbl]Fld # "Max")

4D は，多数のレコードが期待されているという前提により，大多数のレコードは値が"Max"ではないと推測します。そして，**QUERY**([Tbl]; [Tbl]Fld = "Max") というクエリを実行してから検索の結果を反転します。このことを知っていれば，そしてレコードの大多数は値が"Max"であることが事前に分かっているのであれば，むしろ別の値でクエリを実行したほうが，素早く目的のセレクションが作成できるかもしれません。

DESCRIBE QUERY EXECUTION, **Get Last Query Plan** および **Get Last Query Path** は，QUERY または SELECT で実行したクエリを 4D がどのようにアレンジしているのか，内部を調べるために有用なコマンドです。

```
C_TEXT($vResultPlan;$vResultPath)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
DESCRIBE QUERY EXECUTION(True) `解析モードのはじまり
Begin SQL
  SELECT ACTORS.FirstName, CITIES.City_Name FROM ACTORS, CITIES
  WHERE ACTORS.Birth_City_ID=CITIES.City_ID ORDER BY 1
  INTO :aTitles, :aDirectors;
End SQL
$vResultPlan:=Get Last Query Plan(Description in Text Format)
```

```
$vResultPath:=Get Last Query Path(Description in Text Format)
DESCRIBE QUERY EXECUTION(False) `解析モードのおわり
```

$vResultPlan および$vResultPath には，実行されたクエリに関する情報が代入されます。

```
$vResultPlan:
[Join] : ACTORS.Birth_City_ID = CITIES.City_ID
$vResultPath: And
[Merge] : ACTORS with CITIES
[Join] : ACTORS.Birth_City_ID = CITIES.City_ID
(1227 records found in 13 ms) --> 1227 records found in 13 ms
--> 1227 records found in 14 ms
```

4D: やさしい 4D の使い方

---

1. 基本原則: 流れに逆らうな，流れに身を任せよ

4D があまり乗り気ではないことをデベロッパーが強要させた例は，過去，枚挙に暇がありません。そうした試みは，概して残念な結果に終わります。

2. データを最適化する: 断片化の罠

データファイルは，128 バイトずつのブロックに組織されています。したがって，ディスク上の最小レコードサイズは，128 バイトです。

一方，データファイルの最大サイズ（インデックスファイル，ストラクチャファイルも同じ）は，65536 * 8192 * 16384 * 128 バイト，つまり，1,048,576 ギガバイト，1024 テラバイト，もしくは 1 ペタバイト，あるいは 2 の 50 乗バイトです。

データファイルは，いくつかの部分で構成されています。

- ヘッダー
- ブロック・アロケーション・テーブル
- レコード・アドレステーブル・ツリー
- BLOB アドレステーブル・ツリー

・　実際のレコード

重要: これらの部分は，データベース・オペレーションの種別と順序により，データファイル内で動的に配分されます。

どこか最適化できる箇所があるでしょうか。プログラミングの第一法則を思い起こしましょう。

第一法則：一概には言えないよ...（it all depends...）

データアクセス最適化の有無がアプリケーションのパフォーマンスにおおきなインパクトを及ぼす例として，データに対するシーケンシャルアクセスを挙げることができます。新しく作成されたデータファイルの場合，4D はレコードを順序どおりに保存してゆきます。つまり，レコード番号 1 の後にはレコード番号 2 が続きます。シーケンシャルアクセスを快適に実行するためには，この順序を崩さないことが大切です。言い換えるならば，データの断片化を避ける必要があります。もっとも，ほんとうに必要がないのであれば，そこまで神経遣う必要はありません。

昔（2004 以前）のバージョンでは，つぎのようなテクニックが使われていました。

・　エアバッグ（レコードサイズを固定するための可変長領域）

・　ビッグデータテーブル（BLOB，ピクチャ，テキストなど，レコード中，サイズが可変の部分を別テーブルに保存すること）

おおきなオブジェクトをどこに保存するかについて，どれかが絶対的に優れていると断言することはできません。それぞれのオプションには，長所と短所があるからです。

・　レコードと一緒：オブジェクトのサイズがそれほどおおきくなければ最速

・　データファイルの中：明示的にレコードをロードしたタイミングでオブジェクトがロードされるため，一定の効果が期待でき，ある程度，データファイルの断片化を緩和することができる

- データファイルの外：オブジェクトはデータファイルから分離されているため，整合性が失われるリスクがある
- ビッグデータテーブル：データがロードされるタイミングを完全にコントロールできる
- 別ファイル：データファイルとインデックスファイルが分けられているように，専用のファイルにおおきなデータを保存する

内部ストレージの最大サイズ（閾値）オプションは，飽くまで自動設定なので，データの断片化を避ける上での威力は限定的です。

4D v12 以降，ピクチャフィールドのメタデータを SET PICTURE METADATA および GET PICTURE METADATA で読み書きできるようになりました。v13 では，IPTC/Keywords メタデータをインデックスの対象にすることができ，そうすることによってピクチャの検索を最適化することができるようになりました。

可変長データを保存する法を選ぶときには，いろいろな要素を考慮しなければなりません。

- アクセスの種類（インデックス・シーケンシャル）
- テーブルの変異性（どれだけレコードが更新されるか）
- レコードの固定長部分と可変長部分の割合と程度
- レコードをロードする頻度
- 実行モード（リモート・ローカル）
- レコード数
- キャッシュサイズ
- プロセス数
- 許容できる最低の水準

3. インデックスタイプの決定: 舞台裏の事情を考慮する

4D には，用途別にいくつかのインデックスタイプが用意されています。

- B-Tree
- クラスター
- 複合（コンポジット）
- キーワード

それらは，基本的にどちらかのグループに属します。

- B-Tree
- クラスターB-Tree

インデックスの対象となるデータは，スカラー値とそれ以外に分類することができます。スカラー値は，不可分の量的情報で，一度にひとつの値だけを持つことができ，データの占有サイズは固定です。4D でいえば，ブール，実数，整数，倍長整数， 64 ビット整数，日付がこれに相当します。スカラー値には，通常，B-Tree インデックスを使用します。

クラスターインデックスは，基本的に B-Tree インデックスと同じロジックを使用し，その同じロジックの上に成り立っています。ただ，キー（検索の対象となる値）毎にひとつのレコード番号を保存する代わりに，下記いずれかの 4 バイト値を保存する点が違っています。

- 対応するレコードが 1 件であれば，そのレコード番号
- 対応するレコード 2 件以上であれば，クラスター番号

クラスターは，レコード番号のリストであり，それは下記どちらかの形を取ります。

- リストに含まれるレコードの件数が少なければ（テーブルのレコード総数の 32 分の 1 以下）レコード番号（4 バイト）のリスト。つまり，**セレクション**のようなもの。

- そうでなければ，4D は，テーブルのレコード 1 件に対して 1 ビットを割り当てた，ビットテーブル。つまり，**セット**のようなもの。

レコード総数のおおきなテーブルのクラスターは，かなりのサイズになるかもしれません。レコード数の上限は 10 億件なので，クラスターは，理論的に 119 メガバイト（10 億ビット）に達する可能性があります。おおきなクラスターはページに分けられ，ビットがすべて同じ値のページは圧縮することにより，最適化が計られています。

テキストのインデックスを作成する点で，4D はなかなか優秀ですが，自動的な処理ゆえに，まだまだ最適化の余地が残されています。

Google のような，フィーリング検索ができるようにするためには，ある程度の「仕込み」が不可欠です。具体的には，インデックスが作成される前段階で，一定の処理を加えることにより，インデックスを整えておきます。

- 4D がインデックスを作成する単語（ワード）の範囲を実際的な検索語だけに制限する
- インデックス作成用（データ加工済み）フィールドを用意する
- フィールドの特性により，ICU セパレーター文字を置換する
- 不要な単語を除去する
- 同義語・類義語を追加する
- 略語を展開する
- 特殊な表現（例：Vitamin C）に対応する
- 綴り違いに対する耐久性を高める（活用語・合成語）
- Soundex（綴りよりも発音を重視したアルゴリズム）を利用する
- 暗号化

クラスターは，基本的にビットテーブルであり，セットになぞらえることができます。アドレステーブルのエントリー毎に 1 ビットが使用され，ビットの位置は，レコード番号に対応しているからです。v11 以降のクラスターインデックスは，かつての「手作り」クラスターによる速度向上テクニックとよく似ています。

もっとも，4D のクラスターは，ビットテーブルとレコード番号リストを使い分けている点で，より最適化が計られているということができます。