The AP PICT displayer Plug-in Area

By Jean-Yves Fock-Hoon, Quality Assurance Manager, 4D Inc.

TN 06-22

Abstract

このテクニカルノートでは、4D Pack プラグインに含まれている AP PICT displayer プラグインエリアの可能性を探ることに主眼が置かれています。他のオブジェクトでは不可能ですが、このプラグインエリアを使用すれば、メソッドの実行中に 4D のフォームをダイナミックに更新することができます。

AP PICT displayer プラグインエリアを使用した簡単なサンプルデータベースも収録しました。

Introduction

The Problem

4D には、メソッドの実行中にグラフィカルユーザインタフェース(GUI)を更新できないという制限があります。

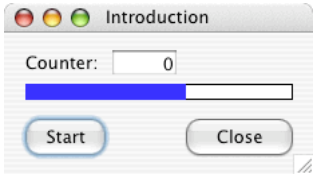
これが何を意味するかは、サンプルデータベースを起動してみればすぐに分かると思います。テクニカルノートに収録されているサンプルデータベース Pictures.4DB を 4th Dimension 2004 で起動してみてください。AP PICT displayer プラグインエリアは 4D Pack プラグインの一部なので、4D Pack プラグインがインストールされていなければなりません。

カスタムモードの「Demonstration」メニューから「Introduction」を選択します。Start ボタン、Counter 変数が配置された簡単なダイアログ画面が表示されます。Start ボタンのオブジェクトメソッドは次のようなものです：

```
ARRAY PICTURE(ap_Thermo;11)
For ($i;1;11)
   GET PICTURE FROM LIBRARY(14999+$i;ap_Thermo{$i})
End for
SET PICTURE RESOURCE(15000;ap_Thermo{1})
vCounter:=0
For ($i;1;11)
   DELAY PROCESS(Current process;60)
   AP PICT UPDATER (v15000;ap_Thermo{$i})
   vCounter:=vCounter+1
End for
```

上記は最初のループではプログレスバーの連続画像がピクチャライブラリから取り出され、次のループではカウンタの変数とプログレスバーの画像が更新されてゆくというメソッドです。

Start ボタンをクリックしてみましょう。プログレスバーの画像は更新されてゆきますが、オブジェクトメソッドの実行中に Counter 変数が更新されることはありません。これは、メソッドの実行中は GUI を再描画しないという 4D の特性による現象です。



The Solution

4D Pack の AP PICT UPDATER コマンドは、AP PICT displayer プラグインエリアに表示されたピクチャを即座に更新するためのコマンドです。フォームイベント終了時に更新される一般のフォームオブジェクトとは違い、AP PICT UPDATER コマンドで更新されたピクチャは、すぐに変更が反映されるため、アニメーション効果を表現したり、プログレスバーを表示したり、その他いろいろなタイプのウインドウで現在のプロセス状態を表示するために使用することができます。

これまでプログレスバーを表現するため数々のライブラリやコンポーネントが発表されてきましたが、ほとんどが新規プロセスを起動してプログレスバーを表示するというものでした。しかしながら新規プロセスによる方法は、必ずしもベストとはいえない場合があります。プログレスバーをモーダルウインドウで表示したくない、複数プロセス間の同期がうまくゆかない、といった再描画に関する問題、あるいは単純に余計な CPU パワーを消費するということが問題になるかもしれません。そのような場合、AP PICT displayer プラグインエリアと AP PICT UPDATER コマンドを組み合わせれば、思い通りにピクチャが制御できるようになるはずです。

AP PICT displayer Plug-in Area In-Depth

How to use the plug-in area

まずデータベースに 4D Pack プラグインがインストールされていることを確認してください。次に、フォームにプラグインエリアを追加し、プロパティリストでプラグインタイプを「AP PICT displayer」に設定します。

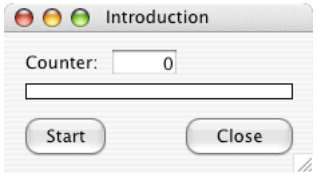
AP PICT displayer プラグインエリアに表示する画像は AP PICT UPDATER コマンド、あるいは PICT リソースにエリアをリンクさせて設定します。プラグインに特定の PICT リソースが表示されるようにする場合、プラグインエリアの変数名をアルファベット 1 文字+ピクチャのリソ

ース ID に設定してください。例えば、PICT リソース#128 を表示する場合、プラグインエリアの変数名を「v128」などに設定します。ちなみにサンプルデータベースでは v15000 に設定されています。一般にカスタムリソース ID は 15000 以降を使用することが推奨されています。

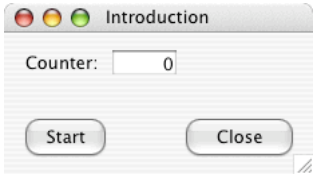
AP PICT displayer プラグインエリアには、いくつかの制限事項があります：

- プラグインエリアをフォームの領域外に配置することはできません。
- プラグインエリアをデフォルトで非表示にすることはできません。
- プラグインエリアはフォームとともに表示されていなくてはなりません。
- プラグインがカレントインタフェースで再描画できなかった場合(つまり非表示の場合)、自動更新は適用されません。
- 自動更新は On Load フォームイベントでは動作しません。フォームをはじめて表示するとき、プラグインエリアにはリソースに保存されたピクチャが表示されます。(これはデザイナーの意図に反する動作かもしれません。後述を参照してください。)

「Introduction」サンプルでは、フォームロード時に空白の四角形が表示されていました：



フォームロード時にプラグインエリアを表示したくないのであれば、サイズが 1 ピクセルの画像をあらかじめ設定しておきます。サンプルデータベース「Demonstration」メニューの「Example 2」でこの動作を確認することができます：



このサンプルは最初のものとほとんど同じですが、On Load イベントで次のコードを実行しているところが少し違っています：

```
Case of
¥ (Form event=On Load )
   C_PICTURE($aaa)
   GET PICTURE FROM LIBRARY(14999;$aaa)
   SET PICTURE RESOURCE(15000;$aaa)
End case
```

1 ピクセル幅のピクチャが読み込まれ、リソース#15000 に設定されているので、プラグインエリアにはその画像が読み込まれます。ダイアログが表示されたとき、プラグインに 1 ピクセル幅のピクチャが表示されています。(スクリーンショットをよく見ると、中央に白い点があるのが分かるかもしれません。)このような 1 ドットの画像は完全に無視できると思います。

使用前にプラグインが描画されていなければならない、On Load イベントでは使用できない、という制限事項があっても、インストーラのようなインタフェースは問題なく作成することができます。通常のインストーラは、実行・設定・キャンセルなどのボタンを備えたダイアログインタフェースで提供されるからです。すぐに動作を開始するのではなく、何らかの操作がなければ先へ進まないインタフェースであれば問題ありません。そのようなインタフェースは、前述のテクニックを利用し、ユーザアクション後に作動するようデザインすることができます。

とはいえこの特性が決定的な制限になると考える必要はありません。On Load で更新することはできなくても、On Timer を使用すれば良いからです。

More examples

この項目は、サンプルデータベース「Demonstration」メニューで提供されている他の例題に関する説明です。それぞれの見出しは、メニューアイテムに対応しています。

Installation:

ユーザアクションで起動する標準的なダイアログのサンプルです。「Installation」メニューを選択すると、M_Installation メソッドがコールされ、[Table 1]の Dial_Installation フォームがダイアログとして表示されます。ダイアログには歓迎のメッセージ、次へ進むためのボタン、キャンセルするためのボタンが配置されています。

このフォームでは、プラグインエリアがフォームサイズいっぱいに配置されています。加えて左上の座標が(-1,-1)に設定されており、フォームロード時に読み込まれる 1 ピクセルのピクチャがフォーム上には現れないようにしています。

このフォームはインストーラの動作を模倣しているだけです。実際に何かがインストールされるわけではありません。ここではスライドショー形式インタフェースのサンプルとして提供されています。画像はピクチャライブラリから読み込まれ、ピクチャ配列に収められます。AP PICT displayer プラグインエリアには、ピクチャフォーマットの設定がなく、デフォルトで「トランケート(中央合わせ)」のフォーマットが適用されます。そのようなわけで、ピクチャの周囲に灰色の余分が出ないようにするためには、プラグインエリアと同一サイズのピクチャを使用しなければなりません。サンプルでは、いずれも幅 492 ピクセル、高さ 240 ピクセルのピクチャを使用しています。

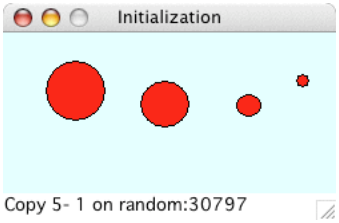
Install ボタンをクリックすると、次のオブジェクトメソッドが実行されます :

```
SET VISIBLE(*;"b@";False)
For ($i;1;14)
   For ($i_copy;1;10)
   ` copy. copy, copy
   End for
   For ($i_delete;1;10)
   ` delete. delete, delete
   End for
   For ($i_opt;1;10)
   ` optimize,   modify, validate
   End for
   DELAY PROCESS(Current process;60)
   AP PICT UPDATER (v15000;ap_Thermo{$i})
End for
SET VISIBLE(bCancel;True)
NEXT PAGE
```

このメソッドでは、毎秒 1 枚、合計 14 枚のピクチャが表示された時点でループが終了します。もちろん、本物のインストーラであれば、特定の段階まで到達したときにピクチャを変化させるようにします。あるいは、単純に無限ループでピクチャの配列を表示し続けるだけでもよいかもしれません。そのような場合、自動的に次のピクチャを表示し、スライドショー最後のピクチャまで到達すれば、最初に戻るようなピクチャ更新メソッドをコールすることができます。

Initialization:

このサンプルは「Installation」に似ていますが、ユーザのアクションがなくても自動的にダイアログの内容が自動的に更新される点が異なっています：



画像の下にあるテキストがメソッドの実行に連動して更新される点に注目してください。

このサンプルでは On Timer フォームイベントでダイアログを更新しており、M_Initialization メソッドが実行されると、[Table 1]の Dial_Initialization フォームが表示されます。このフォームには、On Load および On Timer フォームイベントの設定された次のようなフォームイベントが記述されています：

```
Case of
   ¥ (Form event=On Load )
      SET TIMER(10)
   ¥ (Form event=On Timer )
      SET TIMER(0)
      For ($i_loop;1;2)
         For ($i_copy;1;10)
         ` copy. copy, copy
         $Picture:=GetPicture ("Copy "+String($i_copy)+"- "+String($i_loop)+" on
         random:"+String(Random);MyLogo)
         DELAY PROCESS(Current process;10)
         AP PICT UPDATER (v15000;$Picture)
         End for
         For ($i_delete;1;10)
            ` delete. delete, delete
         $Picture:=GetPicture ("Delete "+String($i_delete)+"- "+String($i_loop)+"
         onrandom:"+String(Random);MyLogo)
         DELAY PROCESS(Current process;10)
         AP PICT UPDATER (v15000;$Picture)
         End for
```

```
        For ($i_opt;1;10)
            ` optimize,   modify, validate
            $Picture:=GetPicture ("Opt/Mod/Val "+String($i_opt)+"- "+String($i_loop)
            +" onrandom:"+String(Random);MyLogo)
            DELAY PROCESS(Current process;10)
        End for
    End for
    $Picture:=GetPicture ("Initialization completed";MyLogo)
    AP PICT UPDATER (v15000;$Picture)
End case
```

On Load イベントでは、タイマーが設定されており、On Timer イベントでは、そのタイマーを無効にした上でプラグインエリアの初期化、更新、最適化に必要なコードを実行しています。

このサンプルでは、Dial_Initialization フォームに表示されているロゴ以外はピクチャライブラリの画像を使用していません。むしろ、メソッド実行中はテキスト変数を含むテキストオブジェクトが更新できないことを踏まえ、ロゴ画像にテキストを合成しています。この処理をしているのが GetPicture メソッドです。同メソッドは 4D Chart コマンドを使用しています。

以下は GetPicture メソッドのコードです：

```
C_TEXT($1)
C_PICTURE($2;$0)
$MyLogo:=$2
$text:=$1

$oa:=CT New offscreen area
$NewPictID:=CT Place picture ($oa;$MyLogo;1;1)
PICTURE PROPERTIES($MyLogo;$width;$height)
$left:=1
$top:=$height+1
$right:=$width
$bottom:=$top+24

$TextID:=CT Draw text ($oa;$left;$top;$right;$bottom;$text)
$0:=CT Area to picture ($oa;-1)
CT DELETE OFFSCREEN AREA ($oa)
```

上記のメソッドは作成した 4D Chart オフスクリーンエリアにテキストを出力しています。テキストを合成するロゴのサイズは PICTURE PROPERTIES コマンドで取得し、テキストの幅はその値に基づいてを算出しています。

オフスクリーンエリアでテキストとロゴが合成されると、CT Area to picture で合成されたピ

クチャが取り出されます。最後にオフスクリーンエリアをクリアしているのは、メモリーリークを避けるためです。

このようなテクニックを使用すれば、「タスク」プロセスと「進行」プロセスを別個に起動しなくても、フォームメソッドがループ処理をしている間、進行状況が逐次画面に表示されるようにできます。

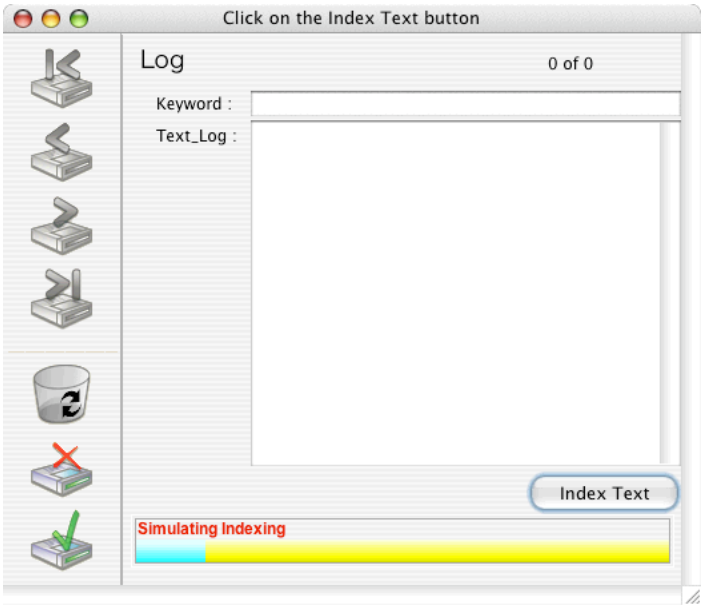
Input form:

当然、AP PICT displayer プラグインエリアの用途はインストーラや初期化プロセスに限られているわけではありません。入力フォームにも応用することができます。

仮に編集目的でレコードがロードされているとしましょう。レコードが確定されようとしているとき、あるいは特定のアクションが起こされたとき、いくらか時間のかかる処理が発生するかもしれません。そのようなタスクが実行されるときはユーザに知らせるのが親切です。

マルチプロセス環境では、そのようなタスクが同時にいくつも実行され、それぞれのタスクごとにプログレスバーの表示が求められるかもしれません。これでは 4D データベースに相当の負荷をかける恐れがあり、多数の On Activate/On Deactivate イベントが使用されているようであれば、なおさら負荷がかかることになります。大体、それだけの数のプロセスを管理するのはけっこう大変です。そのようなシチュエーションでは、次のサンプルで使用されているテクニックが役に立つかもしれません。

Input form サンプルでは、M_AddRec メソッドが実行され、[Log]テーブルに対して ADD RECORD コマンドが発行されます。テーブルの入力フォームには、テキストフィールドに対してインデックスを作成するというオプションが設けられています。もちろん、これはただのサンプルなので実際にインデックスが作成されるわけではありません。重くて長いタスクの代表例ということでそのようになっています。



Start Index ボタンをクリックすると、プログレスバー、そしてインデックス構築のシミュレーションを実行中であることを示す赤い点滅テキストが表示されます。ボタンのオブジェクトメソッドでは、プログレスバーの更新とインデックス構築のシミュレーションコードが実行され、タスクが終了するとそのことを知らせるテキストが緑色で表示されます。

プログレスバーの制御がフォームに統合されているため、新規プロセスは必要ありません。

ちなみにこのサンプルでは点滅するテキストの処理に 4D Chart コマンドが活用されています。テキストの内容をカウンタで切り替えているだけですが、画面ではテキストが点滅しているようになります。なお、4D Chart では空のテキストを渡した場合には何も出力されないため、代わりにスペースが使用されています。テキストのカラー、フォント、スタイルなども 4D Chart コマンドで設定しています。

これまでのサンプルでは、何枚もピクチャを使用してプログレスバーのアニメーション効果を表現していましたが、このサンプルのプログレスバーは、2 枚のピクチャだけで表現しています。プログレスバーは、黄色のピクチャに水色のピクチャをバイナリ演算で補完することによって更新されてゆくので、はじめにいろいろな状態の画像を用意する必要はありません。

プログレスバーの画像が作成された後、毎回 4D Chart でテキストオブジェクトを追加する必要はありません。テキストのピクチャが変数に納められているので、/演算子を使用してふたつのピクチャを合成することができるからです。

```
$Thermobar:=$pictTitle/$Thermobar
```

/演算子は、$Thermobar ピクチャの上に$pictTitle ピクチャを重ねる効果があります。

ループが完了すると最後のメッセージと 100%のプログレスバーが残ります：

詳細については bIndex ボタンおよび[Log]Input フォームのメソッドを参照してください。

Multiple dialogs:

もちろん、複数のダイアログが AP PICT displayer プラグインエリアを使用しても構いません。このサンプルの M_Multiples は M_OneProcess を 10 回コールして 10 個のプロセスを起動し、それぞれのメソッドによって Dial2 フォームの簡単なプログレスバーが表示されます。

基本的には Initialization サンプルに似ていますが、メソッドのプログレスバーを処理している部分は次のようにっています：

```
If ($i>1)
   $c_ThermoEmptyStep1:=$c_ThermoEmptyStep*+($i-1)
Else
   $c_ThermoEmptyStep1:=$c_ThermoEmptyStep1*0
```

```
End if
   $c_Thermo:=$c_ThermoEmptyStep1+$c_ThermoStep
If ($i<100)
   $c_ThermoEmptyStep1:=$c_ThermoEmptyStep*+(100-$i)
Else
   $c_ThermoEmptyStep1:=$c_ThermoEmptyStep1*0
End if
$c_Thermo:=$c_Thermo+$c_ThermoEmptyStep1
$c_Thermo:=$c_Thermo*+0.5
$tempo:=Mod(Random;30)
DELAY PROCESS(Current process;$tempo)
AP PICT UPDATER (v15000;$c_Thermo)
```

今回のプログレスバーは、長さが変化するグラフではなく、むしろ移動するカーソルのようなものです。このような移動を表現するためには、一定幅の空白に隣接してカーソルの画像を描画しなければなりません。その処理に使用されているのが$c_ThermoEmptyStep 変数です。ピクチャの高さを変化することなく幅だけを大きくするには、*+演算子を使用します。この変数は空白のピクチャなのでカーソルの左側部分および右側部分に使用できます。両者にはさまれる格好でカーソルの画像を合成することにより、移動する要素を表現できるというわけです。

演算子による画像の処理は簡単ですが、プラグインエリアとサイズが合わなくなるかもしれないので注意が必要です。ピクチャの幅と高さは PICTURE PROPERTIES コマンドで調べることができます。それに対してプラグインエリアのサイズは GET OBJECT RECT で返される座標から算出することができます。このサンプルでは幅は 4 ピクセルずつの画像を 100 回ループして合成しているのでピクチャのサイズは 400 ピクセルです。プラグインエリアの幅は 200 ピクセルなのでピクチャを修正しなくてはなりません。(高さはどちらも同じなので、プラグインエリアにピクチャが表示されると縮小されてしまうためです。)

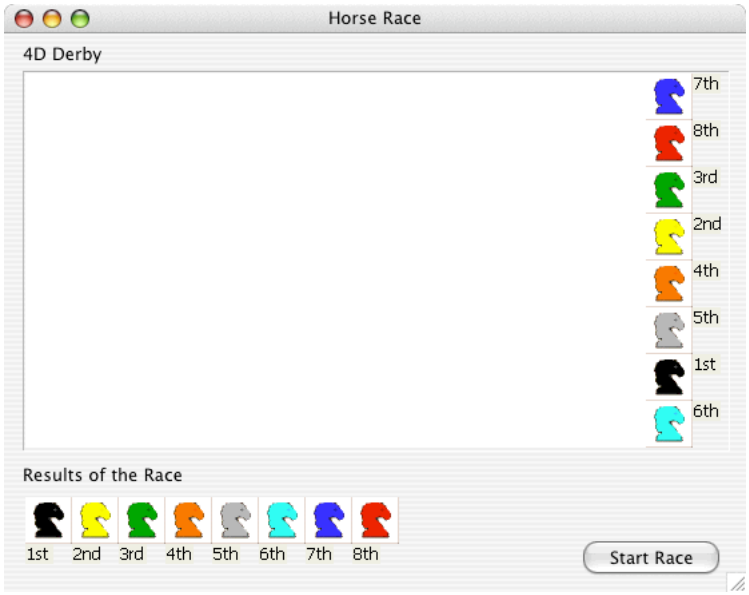
›

このサンプルの場合、ピクチャの幅を 50%にする必要がありますが、ここでも*+演算子を用いることができます。ピクチャ*+0.5 という演算は、ピクチャの幅を半分にするからです。これにより、プログレスバーがプグインエリアにちょうど収まるようになります。

4D Derby:

楽しみがなければ、プログラミングなどやってゆけないのではないでしょうか。

これは競馬のサンプルです。Start ボタンをクリックすると M_HorseRace メソッドが実行され、[Table 1]の Horse Race フォーム上を 8 頭の競走馬が疾走します。



フォームにはふたつの AP PICT displayer プラグインエリアが配置されています一方のエリアには競走馬のピクチャが表示されていますが、もうひとつのエリアは何も表示されていないように見えます。(Example 2 で使用した 1 ドットだけのピクチャを思い起こしてください。) 競走馬が表示されているほうのプラグインエリアの変数名は v15000 なので PICT リソース #15000 に対応しています。もうひとつのエリアは着順を表示するために使用されており、変数名が v15001 なので PICT リソース#15001 に対応しています。。

競走馬の画像は、すべてピクチャ配列 apict_Horses に読み込まれます。それぞれの走行距離は al_pos 配列に保存され、走行距離が 465 に達すればゴールとみなされます。一歩の移動を表わしている画像は、$pictTrack ピクチャ変数です。

ループでは、競走馬の走行距離が計算され、その画像に馬の画像が追加されます。走行距離のピクチャは*+演算子で延長して作成します：

```
$horsepict:=$pictTrack*+al_pos{$i}
```

走行距離の画像に馬の画像を追加しているのは+演算子です：

```
$horsepict:=$horsepict+apict_Horses{$i}
```

このような処理が 8 頭それぞれについて実行されます。

トラック全体のピクチャは/演算子で作成します：

```
vTrack:=vTrack/$horsepict
```

競走馬と走行距離の画像は、それぞれ幅が異なっていることに注意してください。新しいピクチャが返されるため、/演算子でピクチャがリサイズされることになります。

ゴールに着いた馬の画像は、+演算子でレース結果のエリアに追加されてゆきます：

```
pictArrival:=pictArrival+$pict_Horse
```

トラックエリアのピクチャには、それぞれの着順を示すテキストも追加されます：

```
apict_Horses{$i}:=apict_Horses{$i}+apict_Order{$i_Rank}
```

ループが実行されるたびにレーストラックのピクチャが更新されます：

```
AP PICT UPDATER (v15000;vTrack)
```

同じように競走馬がゴールするたびにレース結果のピクチャが更新されます：

```
AP PICT UPDATER (v15001;pictArrival)
```

Summary

このテクニカルノートでは、AP PICT displayer プラグインエリアを使用する方法、特にピクチャライブラリ、4D chart プラグインコマンド、および 4th Dimension のピクチャ演算子と組み合わせて使用することにより効果的な GUI をデザインする方法を説明しました。

Notes

- AP PICT displayer プラグインエリアは 4D Pack のドキュメントの中では%AP PICT displayer と呼ばれています。どちらも同じものです。
- このテクニカルノートのサンプルデータベースは、06-17 に収録されているプラグイン QuickTimeContainer 1.2 を使用しています。