The AP PICT displayer Plug-in Area

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

Technical Note 06-22

Abstract

The purpose of this Technical Note is to explore the "AP PICT displayer" plug-in area, which is included with the 4D Pack plug-in. This plug-in area can be used to dynamically update a 4D form while the form method is being executed, a feature that is not possible with other form objects.

A sample database is provided that demonstrates the features of the "AP PICT displayer" plug-in area.

Introduction

The Problem

4D does not redraw the Graphical User Interface (GUI) during the execution of a method.

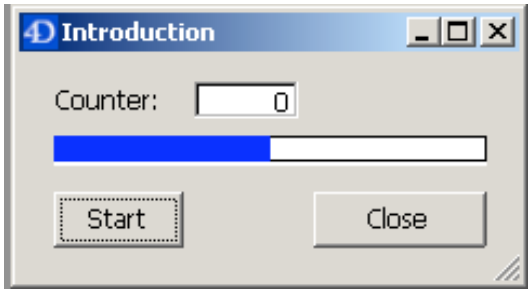
There is no better way to illustrate this problem than with an example. This Technical Note includes a sample database. Open this database with 4th Dimension 2004 (the structure is called "Pictures.4DB"). Make sure that the 4D Pack plug-in is installed since the "AP PICT displayer" plug-in area is part of the 4D Pack plug-in.

From Custom Menus mode select "Introduction" from the "Demonstration" menu. This will display a small dialog with a button labeled "Start" and a variable labeled "Counter". The object method for the "Start" button follows:

```
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
```

Notice a loop grabs some pictures from the picture library for the progress bar and another loop increments the counter and updates the progress bar based on the read picture.

Click on the “Start” button. Notice the progress bar is updated while the object method is being executed but the “Counter” variable is not. This happens because 4D does not redraw the GUI during the execution of any method.



The Solution

The “AP PICT UPDATER” command from 4D Pack enables on-the-fly changes to the picture displayed in an “AP PICT displayer” plug-in area. Unlike regular 4D form objects, which are redrawn according to form events, “AP PICT UPDATER” immediately redraws the area on the screen. Thus this command can be used within a loop to perform animation effects. This can also be a good alternative for displaying progress bars or for use in any window that displays the current status of the process.

Libraries and components are already available to implement a progress bar in your database. Most of these designs rely on displaying the progress bar in a new process. However displaying another process may not fit your needs. Doing this can generate some other redraw effects; you may not always want the progress bar to be modal; you might have some synchronization issues between the 2 processes; or it may be just that the progress bar process uses too much CPU time that you cannot afford. Using the “AP PICT displayer” plug-in area along with the “AP PICT UPDATER” command allows you better control in these situations.

AP PICT displayer Plug-in Area In-Depth

How to use the plug-in area

First of all be sure that the 4D Pack plug-in has been installed in your database. Then, from your form, draw a plug-in area and select “AP PICT displayer” as the plug-in “Type”.

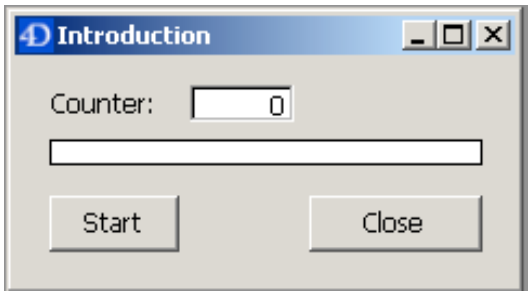
The “AP PICT displayer” plug-in area allows you to display a picture, which is set using the “AP PICT UPDATER” command or automatically loaded from a PICT resource. To automatically display a picture coming from a resource, give your

plug-in area a name with an alphabetic first character, followed by the resource ID number of the picture you want to see. For example, to display the PICT resource #128, name the area "v128". Note that the example database uses resource ID "15000". It is recommended that custom resources have a resource ID greater or equal than 15000.

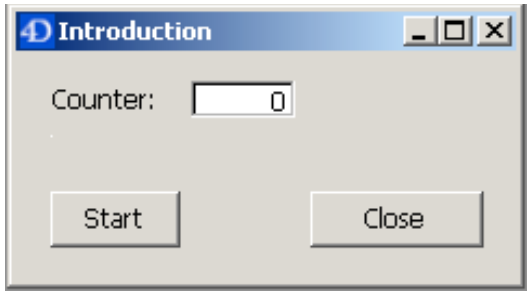
Some restrictions on the "AP PICT displayer" plug-in area:

- The plug-in area cannot be outside of the area.
- The plug-in area cannot be marked as "Invisible".
- The plug-in area must be visible when displaying the form.
- If the plug-in area has not been redrawn in the current interface (e.g. because it was invisible), the automatic update will not work.
- The automatic update will not work during the On Load form event. When displaying the form for the first time, the plug-in area will display the picture saved in the resource (this might be undesirable, see below).

If you go back to the "Introduction" example, you can see that the progress bar starts as an "empty" rectangle:



If you do not want to display the plug-in area initially you can assign an "empty" picture of 1 pixel to the resource. See "Example 2" under the "Demonstration" menu of the sample database for an example of this:



This example is similar to the first one except that the following code is executed during the "On Load" event of the form:

```
Case of
   : (Form event=On Load )
      C_PICTURE($aaa)
```

```
        GET PICTURE FROM LIBRARY(14999;$aaa)
        SET PICTURE RESOURCE(15000;$aaa)
    End case
```

A picture of 1 pixel wide and high is assigned to the resource #15000 and is used by the plug-in area. When displaying the dialog, 4D displays the plug-in area with that very small picture (you may notice a small white dot in the screenshot above). This one pixel is barely noticeable.

The restrictions that the plug-in area must be drawn before it can be used and cannot be used during the On Load form event are not a problem for an interface like an installer. Normally with an installer a dialog is displayed with some buttons to proceed to the installation/configuration or to cancel the dialog. On other words, the dialog is drawn but some user interaction is required to make it "advance". Using the above technique you can make the plug-in area part of this dialog so that it is ready to be used once the user takes the appropriate action.

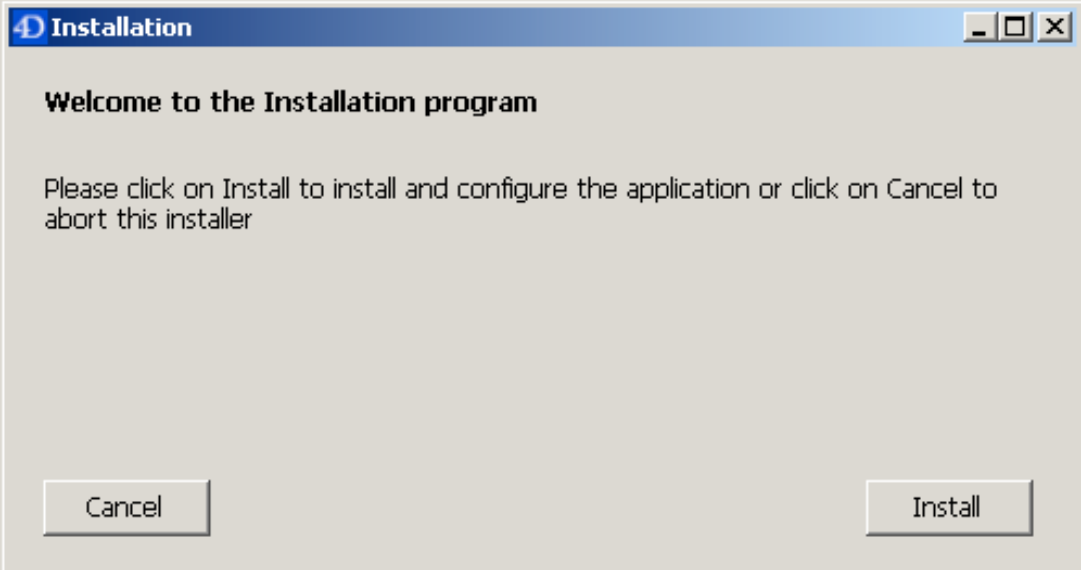
However, this design is quite limiting. The plug-in area cannot be limited to these specific actions. Since we cannot update it from the On Load form event, we can surely do it from the "On Timer" event.

More examples

This section covers the rest of the examples presented in the "Demonstration" menu of the example database. Each heading below corresponds to the menu selection for that example.

Installation:

This example illustrates the use of a regular dialog requiring a user action. Selecting the "Installation" menu item executes the "M_Installation" method, which uses the "Dial_Installation" form from the "Table 1" table. This dialog displays a welcome message and two buttons; one to cancel the dialog, the other to proceed:



In this form the plug-in area has been sized to take up the entire dialog. Additionally the upper-left coordinate of the plug-in area is set to (-1,-1). This is done so that the initial picture resource (a single white pixel) is not visible when the dialog is first displayed.

This form is designed to emulate an installer. Nothing will actually be installed; the purpose is to demonstrate the interface as a "slide show". The pictures are read from the picture library and stored into a picture array. Note that the "AP PICT displayer" plug-in area does not have any picture formatting option. The default format is truncated non-centered. This is why the pictures should have at least the same width and height of the plug-in area in order to avoid a grey redraw around the picture. In this example the plug-in area and the pictures have a width of 493 pixels and a height of 240 pixels.

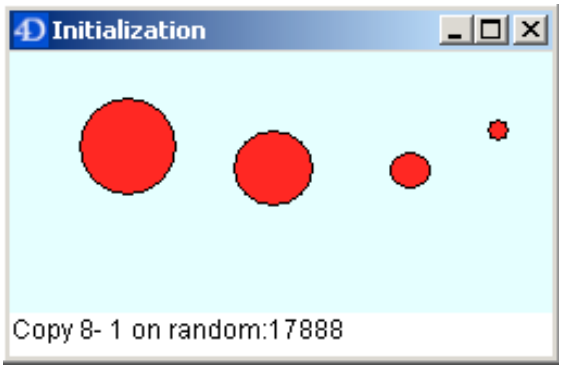
Clicking on the button labeled "Install" executes the following object method:

```
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
```

This method uses a loop to display 14 pictures, one picture per second. Of course, in a real case, you would want to define some milestones in your code so that a particular picture is displayed when reaching a specific point in the installer, or perhaps just perform and infinite loop on the array of pictures. You can perform a random number of calls to the picture updater method that will automatically display the next picture and restart from picture 1 once the last picture of the slide show has been displayed.

Initialization:

This example is similar to the one from "Installation". However in this example the dialog contents are updated without any user interaction:



The text at the bottom of the image is updated on-the-fly.

This example uses the "On Timer" form event to perform the changes. The "M_Initialization" method uses the "Dial_Initialization" form from "Table 1". This form uses the "On load" and "On Timer" form events and the following form method:

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

```
        AP PICT UPDATER (v15000;$Picture)
    End case
```

The "On Load" code initiates the timer. The "On Timer" code disables the timer and runs the code that initializes, updates or optimizes the plug-in area as needed.

This example does not use any pictures from the picture library except for the logo that is displayed when the "Dial_Initialization" form is shown. Instead, since any Text objects or Text variables cannot be redrawn while the method is executing, the text is "integrated" into the picture using the "GetPicture" method. This is accomplished with the use of 4D Chart commands.

Here is the "GetPicture" method:

```
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)
```

The method is creates an offscreen 4D Chart area and places the text in that area. The width of the text will depend on the width of the logo so the "PICTURE PROPERTIES" command is used to get the size of the logo picture.

Once the picture and the text have been inserted into the offscreen area the "CT Area to picture" command is used to return the new picture with the text integrated. Finally the offscreen area is cleared in order to avoid any memory leak.

Using this technique the form method can loop through its tasks and also keep the user updated on what is going on without generating two processes e.g. the "task" process, and the "progress" process.

Input form:

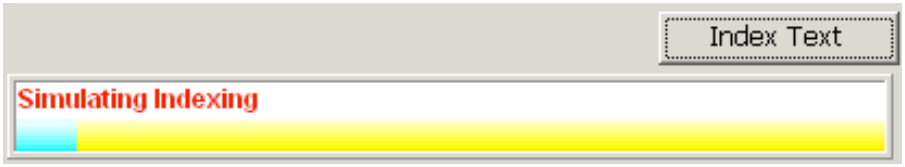
Of course the "AP PICT displayer" plug-in area is not useful for only for installations or initialization processes. It can also be used in input forms.

Imagine that you have a record loaded for editing. When the record is about to be validated, or following some other specific action, a task might be performed that

takes a long time finish. It would be nice to let the user know progress of such a task.

In a multi-process environment, you might have multiples of such tasks executing and therefore face multiple processes that might also need to display a progress bar for each task. This can generate a lot of work for 4D, especially if the database uses a lot of "On Activate"/"On Deactivate" events. Managing all of the processes could become quite a pain as well. The technique used in the following example is helpful in this situation.

The "Input form" example executes the "M_AddRec" method, which executes an "ADD RECORD" command on the table "Log". In the input form the ability to index the current text field is offered. Of course no indexing will actually be performed since this is just a simulation of a heavy and long task.



Clicking on the button labeled "Start Index" displays a progress bar and some blinking, red text that says "Simulating Indexing". The object method then performs a loop to update the progress bar and simulate the indexing task. When the task is complete the text is changed to a green colored message that says "Simulation Completed".

Since the progress bar is integrated into the input form no extra process is necessary to keep it updated.

On a side note, notice that more 4D Chart commands have been used in this example. This is done to create the "blinking" text effect. Using a counter, the text alternates between being "empty" or not. This alternation gives the impression that the text is blinking. Note that the text value is not really "empty" but contains a space. 4D Chart will not insert text into the area if there is no text to be inserted. A space is used as a work-around. Also notice the use of color, the definition of a font, and the use of the "CT SET TEXT ATTRIBUTES" command to give style to the text.

In the previous examples many pictures were used to simulate the animation of the progress bar. In this example, only two pictures are used to build the progress bar; a yellow spot for unused space in the progress bar and a blue drawing used to fill the progress bar. This is accomplished by using binary operators on the pictures to build the progress bar. Then, by saving a "copy" of the progress bar at the end of each iteration, the progress bar does not need to be generated from scratch each time nor be broken up into many pictures to simulate the animation.

Once the content of the progress bar has been computed there is no need to add text to it each time. Instead a copy of the picture containing the text is saved and the / operator is used to insert the progress bar picture at the end of the picture of the text, e.g.:

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

The / operator inserts the "$Thermobar" picture at the bottom of the "$pictTitle" picture.

Once the loop completed, the final message is displayed in green text and with a full blue progress bar:

For further details see the object method for the "bIndex" button of the form "[Log]Input".

Multiple dialogs:

Of course, multiple dialogs can also use this plug-in area. In this example, the "M_Multiples method" creates ten processes using the "M_OneProcess" method. This method uses the "Dial2" form that contains a simple progress bar:

If this example looks similar to the "Initialization" one, it worth noting how the progress bar is computed. Here is the relevant portion of the form method:

```
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)
```

This progress bar is not a full bar but just a "cursor" moving in the progress bar. Therefore some empty space must be drawn before drawing the cursor itself in order to show the movement. This is the purpose of the computation of the "$c_ThermoEmptyStep" variable. The *+ operator is used to change the width of the picture without changing its height. Since this picture is a "blank" space it can be used in order to compute the first part of the progress bar. The same can also be applied for the remaining blank space of the progress bar. The progress bar can then be built with the cursor nested between the beginning and the end.

Altering a picture is quite easy with these operators, however the size of the picture may no longer fit the plug-in area size. The "PICTURE PROPERTIES" command can be used to retrieve the width or height as needed. The width and height of the plug-in area can be computed based on the coordinates returned by the "GET OBJECT RECT" command. In this example each "slice" of the progress bar has a width of 4 pixels. Thus with a loop over 100 iterations, the picture will have a width of 400 pixels. The plug-in area has the same height of the picture but is 200 pixels wide only. Therefore the picture needs to be adjusted to fit the plug-in area (remember the plug-in area truncates the picture if it is too large).

In this example the picture needs to be reduced by half. To do so the *+ operator is used again. The expression "*+ 0.5" reduces the width of the picture by 50%. This scales the progress bar to fit within the plug-in area.

4D Derby:

Where is the pleasure of coding if there is no fun?

This example simulates a derby; a race between 8 horses. The "Start" button starts the race and the code randomly determines the results of the race. This example calls the "M_HorseRace" method and uses the "Horse Race" form from "Table 1".

This form contains 2 "AP PICT displayer" plug-in areas. The first one contains the pictures of 8 horses while the second one is almost invisible (notice the small white dot as in "Example 2"). The first plug-in area is named "v15000" and represents the race track. This plug-in area uses the PICT resource 15000. The second area is named "v15001". This area displays the finish order of the race. This area uses the PICT resource 15001.

It is important to note that all of the horse pictures are loaded in the "apict_Horses" array. The positions of each horse are stored in the "al_pos" array. When the position reaches 465, the race for that horse is over. The picture that represents one step performed by a horse is stored in the "$pictTrack" variable.

In the loop, for each horse, the picture that will represent the distance traveled by the horse is generated. The picture of the horse is then added to distance picture to indicate the horse's position. The distance picture is computed with the *+ operator:

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

To add the horse picture to the distance the + operator is used:

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

This is performed for all 8 horses.

To build the global view of the track the / operator is used:

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

It is important to note that the pictures of each horse do not have the same width. The / operator will resize the picture, since a unique picture will be returned by 4D.

When a horse reaches the finish line the picture of that horse is added to the results area:

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

Additionally some text is added to the track area to indicate the position of the finisher within the field:

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

An update of the track is executed for each step performed during one iteration:

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

A similar update is performed each time a horse reaches the Finish line:

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

Summary

This Technical Note described how to use the "AP PICT displayer" plug-in area. When combined with the picture library, the 4D Chart plug-in commands, and the picture operators available in 4th Dimension this plug-in area can be a powerful tool for building your GUI.

Notes

- The "AP PICT displayer" plug-in area is referred to as "%AP PICT displayer" in the 4D Pack documentation. Both names are equivalent.
- The sample database included with this Technical Note uses the QuickTime Container 1.2 plug-in, which was introduced in Technical Note 06-17, "QuickTime Container 1.2 – More Features".