Resizing forms

By Jean-Yves Fock-Hoon, Technical Support Engineer, 4D Inc.

Technical Note 06-16

Overview

The purpose of this technical note is to demonstrate the use of the new form commands in 2004.

A major change

The problem

There is no better way to illustrate a problem than to use an example. This technical note is provided with a very simple 6.8 database. In the Custom Menus environment, there is a "6.8 Version" menu bar with a unique menu item: Demonstration. If you select that item, it creates 2 processes. The first visible process is a process that displays a scrollable area. You may select an item and click on the Edit button. This will display the other process, which emulates data entry on the selected item in the scrollable area. The purpose of that example is to show how both processes interact when it comes to handle window sizes. If you cancel the data entry after increasing the size of the window, you may want the "Output form" to adopt that new size. It's a pretty simple technique that uses GET WINDOW RECT and SET WINDOW RECT commands.

The 2004 database provided with the technical note was built upon the 6.8 database. If we perform the same test, we can see that the interface does not respond anymore. Objects have been resized and the form becomes unusable.

The cause

The behavior has changed with version 2004 and the introduction of new commands as well as the introduction of the 'pusher' system. Since these commands have been modified, this is why the 6.8 example no longer works.
What can be done to emulate that 6.8 behavior? According to the documentation, we can no longer use the SET WINDOW RECT command and would have to use the RESIZE FORM WINDOW command.

In versions prior to 2004, SET WINDOW RECT would resize the window and the objects inside the window. This was a basic behavior but would not give you enough control on windows. You may need to redefine the borders of your windows without performing any resizing inside your dialog. With 2004 this is now possible. SET WINDOW RECT allows you to redefine the size of the window and does not resize the contents of that window.

The example

In the "2004 Version" menu bar, we can select the "Demonstration" menu item that will emulate what was done in prior versions. We can edit and resize both windows, they will still be synchronized and objects won't disappear from the screen. Let's se what was done behind the code.

In 6.8, we were just grabbing the coordinates of the window, storing them into some interprocess variables and applying them to the new window. The automatic resize will do the rest.

In 2004, we can see that more lines of code are required.

```
GET WINDOW RECT(<>LeftIn2004;<>TopIn2004;<>RightIn2004;<>BottomIn2004)
RESIZE FORM WINDOW(<>RightOut2004-<>LeftOut2004-<>RightIn2004+<>LeftIn2004;<>BottomOut2004-
<>TopOut2004-<>BottomIn2004+<>TopIn2004)
GET WINDOW RECT($NL1;$NT1;$NR1;$NB1)
SET WINDOW RECT(<>LeftOut2004;<>TopOut2004;$NR1-$NL1+<>LeftOut2004;$NB1-$NT1+<>TopOut2004)
SET WINDOW RECT(<>LeftOut2004;<>TopOut2004;<>RightOut2004;<>BottomOut2004)
```

Let's now analyze the code. You have now two sets of variables, one for the coordinates for each window. The computation is a little bit tricky. If you want to increase the size of window A to reach the size of window B, you need to know by how much. This is why you need to have coordinates from both windows. We can compute the difference in order to know by how much window A should be resized to reach the size of window B. This explains the 'tricky' parameters used with RESIZE FORM WINDOW.

The remaining lines can be optimized but it was duplicated for better understanding. The user was in the window A, and requested the window B. He increased the size of the window and moved it on the screen. When he's about to go back to window A, he's expecting to retrieve window A at the same place he left window B. We saw how to compute the window size; now we need to handle its move. To move a window, we would have to use the SET WINDOW RECT command and provide the same coordinates. We need to know the current position of our window and the new width of our window after resize. This is why we are performing that GET WINDOW RECT command to retrieve the coordinates. From these coordinates, we can compute the current width and height of the window. From the second set of variables, we know where the window B is. We can apply the same coordinates with the first SET WINDOW RECT. So, why do we have the second SET WINDOW RECT? We want window A to be at the same position as window B. Window A has now the same width and height than window B, but is not at the same position. We are looking at the new coordinates; they are the same as window B. Therefore, there is no need to compute them again; we can just use window B's coordinates since both windows have now the same size.

Therefore, the code should be something like this:

```
GET WINDOW RECT(<>LeftIn2004;<>TopIn2004;<>RightIn2004;<>BottomIn2004)
RESIZE FORM WINDOW(<>RightOut2004-<>LeftOut2004-<>RightIn2004+<>LeftIn2004;<>BottomOut2004-
<>TopOut2004-<>BottomIn2004+<>TopIn2004)
SET WINDOW RECT(<>LeftOut2004;<>TopOut2004;<>RightOut2004;<>BottomOut2004)
```

In place of the following code used in prior versions:

SET WINDOW RECT(<>LeftOut68;<>TopOut68;<>RightOut68;<>BottomOut68)

Resizing or not resizing

4D 2004 introduces a new concept, the ability to resize a window in a non-proportional way. In earlier versions you could resize your window and all objects inside that window would be resized according to their properties. This is in fact two features used in the same command. You might want to programmatically resize a window as if the user was using the grow box of the window or you might simply want to increase the window size in order to show some items that are currently outside of its boundaries.

Resizing a window

The command SET WINDOW RECT allows you to redefine the coordinates of the window. This is useful when you have some other object outside of the current boundaries of your window and when you want to extend its size in order to include these objects.

Example 1:

To illustrate this point, let's select the "Example 1" menu item from the "2004 version" menu bar. The dialog uses a form where all items have a fixed coordinates.
If you click on the Size 1, Size 2 and Size 3 buttons, you can see that the size of the window has been resized. Objects outside of the window are now visible.
But, you can already do that in prior versions, so what's new? Let's see our second example.

We have now a resizable dialog. We can resize it horizontally. In the form, the scrollable area has been defined as "Grow". When we increase the size of the window, we can see the scrollable area growing. We can see that there are other objects to the right of the scrollable area. These objects have a fixed position. Note that the form size is based on the Close button. Those other objects are outside of the visible form. In our dialog, we can manually resize our dialog, increasing its size. Our objects on the right side are still not displayed. They are pushed on the right side, still outside of the visible form. With prior versions, the resize would have displayed those objects. In the dialog, you can click on the "Advanced" button. This button retrieves the current coordinates of the window and increases them by 140 pixels, in order to include the hidden objects in the visible form using the SET WINDOW RECT command. Our objects are now visible. We can still resize or dialog and see the scrollable area moving.

Example 2:

In example 2, we can notice that the form has a minimal and maximal value for the width, 180 and 500. We have arbitrarily defined 500 as maximal value but we have defined 180 as minimal value because of the "Done" button that is movable. If we

reduce the dialog, the “Done” button will overlap the “Advanced” button. This is why we have a default value of 180 pixels. If we reduce our width down to 180 pixels and click on the “Advanced” button, the form will have now a width of 320 pixels. We can now reduce it and see our “Done” button interfering with the “Default” button.
This is why we are using the SET FORM HORIZONTAL RESIZING command in the object method. This will allow us to define programmatically the width of the window. Expanded or not, we can now be sure that the “Done” button will not interfere with the other buttons.

Example 3:

This example is very simple and is a mix of the first two examples. From a dialog where all items have a fixed size, we can click on the Default/Advanced button to increase or not the size of our window.
The unique difference is that the form has a fixed size that shows everything in the visible form. However, we are going to show only half of the form by executing the SET WINDOW RECT command during the On Load form event.

Example 4:

Our Example 4 is an example that will show you what you can do when dealing with constraints. Our first dialog is a very small dialog that can be resized vertically up to 300 pixels as defined in the “On load” form event of the form method. There is no horizontal resized allowed.

In the Advanced button, we can see that the size of the window is based on the position of the scrollable area. We are retrieving the coordinates of the object and add a value as margin (10) or as offset (145) in order to display other objects on the right side. If you click on the “Advanced” button, the dialog is now bigger, as expected. The dialog is now horizontally and vertically resizable, up to 500 pixels for both.
Increase the dialog to use 500 pixels as width and height, and click on the Default button to go back to a smaller dialog. Following the execution of the object method, a SET WINDOW RECT is performed based on the size of the scrollable area; too big for the small dialog whose boundaries have been defined by the SET FORM HORIZONTAL RESIZING and SET FORM VERTICAL RESIZING. This is why a RESIZE FORM WINDOW(0;0) has been executed. Without that line of code, the dialog would still be a large dialog. As soon as you interact with it, it will be resized to what it should be. If you execute this RESIZE FORM WINDOW (0;0) line of code, 4D will redraw the window and the dialog will be displayed within its boundaries, i.e. with the maximal values allowed.

Example 5:

The SET WINDOW RECT command is so powerful that it can make your interface unusable when it is misused with resizable forms and movable objects. With the dialog from Example 5, if you click on buttons, the interface will remain fine as long as you do not resize the window. If you resize the window and click on one of the buttons, such as Size 1 to reduce the width of the window, the width of the objects is going to get out of control. As you can see, you can no longer see all of these colored squares. Of course, you can still manipulate them though the language. Click on the Reset

button and these objects are displayed again as they were before. You can reset their position by using the MOVE OBJECT command.

It is highly recommended that you check the properties of your forms and objects. All of them must be compatible. You need to be cautious when defining these parameters. Be sure that the boundaries of the form are the appropriate ones. To do so, enable the "Display Limits" in your form. From the contextual menu in the form editor, select the sub-menu Display and then select Limits. This will display the real size of the form to be displayed based on the current window size. By default, the form will contain all objects since the default setting is "Automatic Size". You may limit the size of the form by setting the size to be based on one specific object or by simply defining a fixed size. If you know if this form can be resizable, you may define them as "Move" or "Grow" if needed. If objects are outside of the visible form, make sure that they will not interfere with other objects after a form is resized. If your form is not designed to be resized, do not set objects to Move or Grow.
One of the most important things is to define valid values for the minimal and maximal height and width. Do not keep the default values, 0 to 32000; otherwise, if the form is resized, your interface may become unusable.

Example 6:

This example illustrates the last point. Our form has some objects that can move and grow. There is also a rectangle with a fixed size. If we select our Example 6 menu item, a small dialog is displayed with 4 buttons. The first one displays the form within an "Open Form window", i.e. using the exact width and height of the form as window size. If you click on that button, you can see that the dialog has been displayed with all objects displayed at the right place. As defined in the form. If you click on the second or third buttons, we are about to display the same form in a bigger or smaller window. As you can see, objects are resized and their size has adjusted to match the change. If you check the widow with the smaller size, you might see that some objects' size has decreased but most importantly, the static text "Select an item" displayed on the left of the pop-up menu is no longer here. In fact, it has been moved; it is now outside of the visible window. This is because the static text has been defined as "Move". The last word of that text, item, is more than 300 pixels on the left of the bottom right corner of the window. If we resize the window and make it smaller, in this case, 300 pixel-wide, our static text is also moved to the left. Therefore, the coordinates of the word 'item' would become roughly -60. The static text "Click on Done" is also outside the screen. However, a border has been defined for that string. We can clearly see the edge of the border of that static string. Increasing the size of this window will allow us to see these static text objects again.

Resizing a window

The command RESIZE FORM WINDOW allows you to resize a window as if you were manually resizing the window by dragging its corner.

Example 7:

This example displays a dialog with movable objects. You can resize that dialog and see the objects moving. This dialog has a minimum and maximum size. The purpose

of this dialog is to show you how to allow a window to be resizable and how to resize it programmatically. From the window size area, you may redefine the boundaries, up to the maximum size of that dialog.
If the size has reached a specific value or based on some other conditions, you may also decide to freeze or disable the resizing of the window. You can use the SET FORM HORIZONTAL RESIZING and SET FORM VERTICAL RESIZING commands to do so.

If you want to enable the resizing, pass True as first parameter. If you provide only the first parameters, the values defined in the form will be used as delimiters. You may define your own by passing them as 2nd and 3rd parameter to those commands.

If you want to temporarily disable the resizing, you may execute these commands with False as the first parameter.

Both commands work only if the form has been defined as resizable in the Form editor. If the form has a fixed width or height, the equivalent command will not work at all for that fixed attribute. If you want to use that command, be sure that the form does not have set size. If you really need to have a fixed value and still want to be able to use those commands, compute the fixed value and define it as minimal and maximal values.

For example, if you plan on using this command to resize the form horizontally but require that the form is displayed with a fixed width and prevent manual resizing, compute the required width. Assuming that the value is now 447 (value required for the [MyTable]Dial7 form), define this value in Minimum Width and maximum Width. If you display the form, you will see that the width cannot be resized. However, the SET FORM HORIZONTAL RESIZING command can still be used to define the horizontal resizing.

Once the horizontal and vertical resizing has been defined and that we now have our boundaries, how far can a form be resized? As you can see, you cannot increase the size of the window further than allowed. We can now check the RESIZE FORM WINDOW command. This command allows you to resize your window and make your objects to move or to grow as if you were resizing manually the window. Let's reduce the size of the window and click on the "Maximum Size" button. This button will execute the following code:

```
RESIZE FORM WINDOW(1000;1000)
```

As we can see, we're about to resize our dialog to a 1000 pixels; however, the displayed dialog size is still limited to 600 pixels. This command is also bound to that limitation; it cannot allow you to increase the size of the window further than allowed.

This is also true when it comes to reduce the size of the window. Click on the Minimum Size button. Each width and Height will be decreased by 1000 pixels; the size of the window is now the minimum allowed.

If there was no limitation defined for that form, the dialog would have been unusable. The user would not necessarily have found a way to reduce the size of the variable, or worse, it may even not be visible.

Note: If you define a form to be resizable down to a 10 pixels width or height and cannot get the dialog to be resized to less than 100 pixels, this is because your form contains some objects that cannot be moved. The width and height required to display non movable objects is factually the real minimal width and height for that window. The default minimal values in the example form are 200. If you enter 200 in both minimal values and apply this setting, you can see that you cannot resize it down to 200 pixels. The width and height are blocked by the position of the 3 buttons and the position of the group box and the button for the height. If all objects are set to "Move", you will be able to resize your window down to 200 pixels, as expected.

Moving a window

The command SET WINDOW RECT allows you to redefine the coordinates of the window. You can use it to move a window. You can use the GET WINDOW RECT command to retrieve the coordinates of the current window in order to compute its width and height. You can then compute the next coordinates and use the SET WINDOW RECT. This will allow you to move the window where you want.

Summary

This technical note describes the difference between resizing a window whose objects grow and move to accommodate the changes, and resizing a window without altering the size of its contents.