Resize/Reposition the Controls in a Dialog at your Pleasure

Xia Xiongjun, 2 Mar 2006 CPOL

Rate:

★★★★½ 4.54 (38 votes)

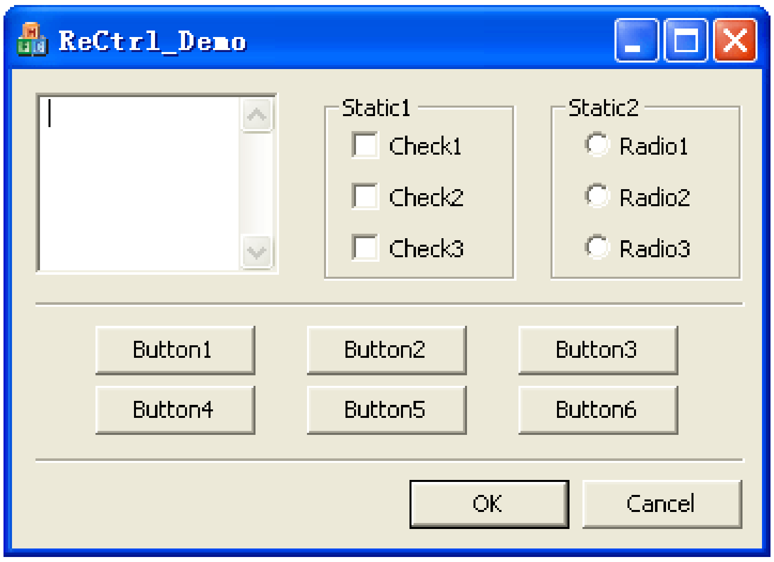
You can resize or reposition the controls in your dialog derived from CSizingDialog to anywhere you wish just by specifying some simple strings or numbers. In addition, most kinds of controls almost don't flicker when moving, which is often a problem in some other solutions.

Download demo project V1.2 (Modeless Dialog) - 58.93 KB

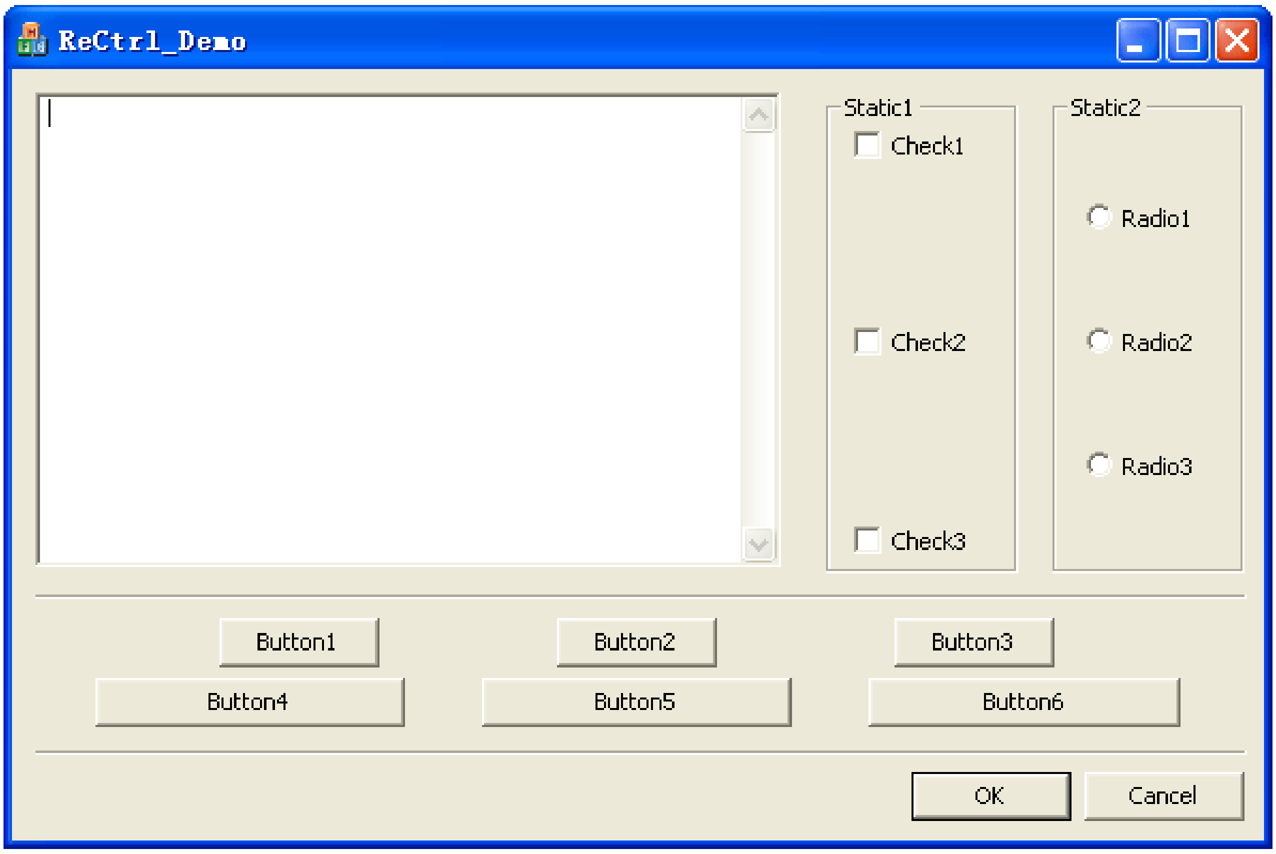
Download demo project V1.1 (Modal Dialog) - 33.36 KB

Download source code V1.2 - 7.23 KB

Download demo project and source code V2.4 - 61.24 KB



1. Original Dialog



2. Dialog after being resized

Introduction

A dialog is a simple but also important user interface which can be used as the complementarities in a SDI or MDI project. In many cases, it's also used as the main window of the project. Unfortunately the MFC architecture doesn't provide methods for resizing the controls in a dialog. When you change the size of a resizable dialog, all the controls stay still in the top-left corner.

Many people put forward several kinds of solutions, but they hardly meet my need. Some solutions use very complex techniques that are hard to understand and so it's hard to improve and extend. Some solutions are too simple to meet even a slightly complex need. Some are not very convenient for secondary developers to use. And most of them move the controls to the destination positions without thinking of the flicker of controls.

The following class, `CSizingDialog`, gives you some simple but powerful methods to change the situation in a way which can make your dialogs more usable and professional. I thank many people for their intelligent work of giving me many good elicitations.

Background

The movement of controls in a dialog can be simply regarded as the movement of the four borders of controls, which make the whole control resize and reposition in a dialog. The class `CSizingDialog` is just based on such a simple fact and is encapsulated for better use.

Using the Code

Let's start with how to use the class. The details will be discussed later:

1. Change the base class of your dialog class (for example `CYourDlg`) from `CDialog` to `baseCYourDialog` where `CDialog` occurs as the base class of `CYourDlg`. Add the following code to header file *YourDlg.h*:

Hide Copy Code

```
#include <span class="code-string">"SizingDialog.h"</span>
#ifndef baseCYourDialog
#define baseCYourDialog CSizingDialog
#endif
```

Of course, you can use `CSizingDialog` to take the place of `CDialog` directly.

2. Copy files *SizingDialog.h* and *SizingDialog.cpp* to your project directory, and add *SizingDialog.cpp* to your project.
3. Now you can specify the controls you want to move or size in the function `CYourDlg::OnInitDialog()` just as the following examples do. The functions will be explained below.

Hide Copy Code

```
this->AddResizableCtrl(IDC_EDIT1, _T("(-0.5)C"));
this->AddResizableCtrl(IDC_STATIC1, _T("X+CY"));
this->AddResizableCtrlArray(IDC_CHECK1, IDC_CHECK3, 0.0, 1.0, FALSE);
```

That's all to use the code.

Functions to Define the Moving Way of Controls

1. Add a single control specified by `nID`.

Hide Copy Code

```
void AddResizableCtrl(UINT nID, LPCTSTR lpszString = NULL);
void AddResizableCtrl(UINT nID,
        double dRateLeft, double dRateRight,
        double dRateTop, double dRateBottom);
```

`lpszString` and `dRateLeft`, `dRateRight`, `dRateTop`, `dRateBottom` are parameters to describe the moving way of controls. They will be discussed in details below.

Tips: If `nID == NULL`, all the controls in the dialog will be added using the same parameters.

2. Add (or modify) a control-range where IDs range from `nIDStart` to `nIDEnd` or from `nIDEnd` to `nIDStart`.

Hide Copy Code

```
void AddResizableCtrlRange(UINT nIDStart, UINT nIDEnd,
        LPCTSTR lpszString = NULL);
void ModifyResizableCtrlRange(UINT nIDStart, UINT nIDEnd,
        LPCTSTR lpszString = NULL);
void AddResizableCtrlRange(UINT nIDStart, UINT nIDEnd,
        double dRateLeft, double dRateRight,
        double dRateTop, double dRateBottom);
```

3. Add a control-range where IDs range from `nIDStart` to `nIDEnd` or from `nIDEnd` to `nIDStart`.

Hide Copy Code

```
void AddResizableCtrlArray(UINT nIDStart, UINT nIDEnd,
        double dSelfGain = 0.0, double dBorderIntervalRate = 1.0,
            BOOL bHori = TRUE);
```

The difference between this control-set definition method and the former methods is that in the former cases, all controls use the same moving parameters so they all behave the same while in the last case the moving parameters are not given directly and have to be calculated from other parameters. The typical result is that the controls' moving parameters are gradually changed from the first control to the last control.

The parameters of this function will be discussed separately below. You may also refer to moving parameters explanation.

`dSelfGai`	

n	The **size** gain of a control relative to the dialog window **size** gain in the dimension specified by bHori.
dBorderIntervalRate	The proportion between the "border" size, offset between the first control's left border and its original position, and the interval size between two controls in horizontal direction. The case of vertical direction is similar.
bHori	The direction in which the controls are rearranged.

Tips:

- Set dBorderIntervalRate = 0.0 to keep Ctrls near the border Close To Border
- Set dBorderIntervalRate = 1.0 to keep Ctrls Rearranged Uniformly
- Set dBorderIntervalRate = 1.0e20 to keep Intervals between Ctrls Constant

Moving Parameters Explanation

The moving offset of the right/bottom border of the dialog window relative to its original position is regarded as 1.0 in the follow definition.

1. Defined using numbers:

dRateLeft	The moving rate of a control's left border to the right
dRateRight	The moving rate of a control's right border to the right

   The vertical case is similar.

2. Defined using strings:

   A right defined string can be divided into different sections which are connected by the character '+' (other characters can also be specified). Each section is made up of a coefficient and a key word such as 0.5X, (-0.3)C, (-15.5RXY). If the coefficient is not specified, it is set to 1.0. The string is not sensitive to upper/lower characters.

   As you can guess, in a key word, X represents the horizontal dimension, while Y represents the vertical dimension. And if there is no X or Y, the definition is applied to both the horizontal dimension and vertical dimension.

X, Y, XY	The whole control moves following the dialog's right/bottom border.
C, CX, CY, CXY	Only the right/bottom border of the control moves following the dialog's right/bottom border. The left/upper border of the control stays still when the dialog is resizing.
R, RX, RY, RXY	The whole control will keep constant proportion to the dialog's client rectangle when the dialog is resizing.

Now, all the resizing and repositioning problems are discussed. **Note** that if a control's moving way in some direction is defined more than once, only the last definition can work.

Guess what the following sentences can bring about?

Hide Copy Code

```
this->AddResizableCtrl(NULL);
```

or

Hide Copy Code

```
this->AddResizableCtrl(NULL, _T("R"));
```

If you are not sure, add one of them to your CYourDlg::OnInitDialog() just before the sentence return TRUE; to see the effect. Of course, the first two steps in using the code are required to compile correctly.

Points Of Interest

Now I will discuss something about how to avoid control flicker when moving the dialog as far as possible. Anyone who just wants to use the class can jump over this section.

The first improvement is to set the `redraw` flag to `FALSE` before moving controls and restore it to `TRUE` after moving the controls as follows:

Hide Copy Code

```
pCtrl->SetRedraw(FALSE);
pCtrl->MoveWindow(&rcCtrl);
pCtrl->SetRedraw(TRUE);
```

(The original code is different since the visible state of the control is considered.)

The second improvement is in the way of erasing the old rectangle (for instance, `rcCtrlOld`) of a control. Instead of using `this->InvalidateRect(&rcCtrlOld)`, we just update the area which belongs to `rcCtrlOld` and at the same time DOESN'T belong to `rcCtrl` where the control stays currently. This is implemented by the following function:

Hide Copy Code

```
void CSizingDialog::InvalidateCtrlBorder(     // XR: erase lpRectOld - lpRectNew
        LPCRECT lpRectOld, LPCRECT lpRectNew, BOOL bErase /*= TRUE*/)
```

There are still problems when you move a `groupbox` control. The previous two steps can't solve the problems. There are some old traces which remain IN the new `groupbox` control. But if you don't apply these two steps, the `groupbox` control and all the other controls in the `groupbox` control will flicker severely.

My solution is to divide the `groupbox` control's area into 9 parts. Each part is an individual rectangle. The work is done by the following function:

Hide Copy Code

```
void GetGroupBoxRgn(const CWnd* pGBox, const CWnd* pParent, CRect* pRc)
```

`pRc` points to a `CRect` array with 9 elements. The meaning of `pRc`'s elements in a `GroupBox` is illustrated as follows, 0 means whole rectangle.

Hide Copy Code

```
//        ______________9____________
//       |                           |
//       7                           8
//       |_____5_____[ 4 ]_____6_____|
//       |                           |
//       |                           |
//       |                           |
//       |                           |
//       2                           3
//       |                           |
//       |                           |
//       |                           |
//       |                           |
//       |_____________1_____________|
```

Note that 7th, 8th and 9th part of the `groupbox` control do not exist in a `groupbox` with the default style.

So a `Groupbox` can be regarded as 9 "controls". Each of them can be applied the first two steps and the problem is solved. But I don't consider it as a good solution.

Of the 24 kinds of typical controls, the following kinds of controls still flicker more or less when moving even after applying the two steps. They are:

- `Combobox`
- `Listbox`
- Slider control
- List control

- Tree control
- Tab control

More effort is needed to solve this problem.

Revision History

- **8th October, 2005 - V1.2**: A critical problem has been fixed when the class is used in a modeless dialog.
- **7th September, 2005 - V1.1**: Four statements which are not supported by VC6 were modified, so the class now can be used in a project developed by VC6.
- **5th September, 2005 - V1.0**

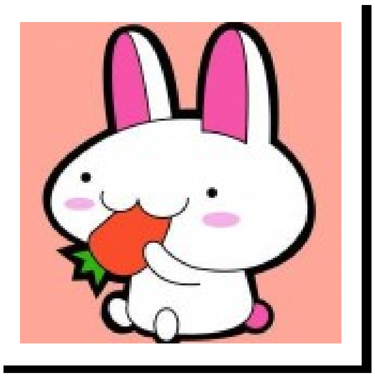
License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)

Share

EMAIL TWITTER

About the Author

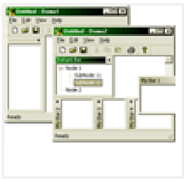


Xia Xiongjun

Software Developer
China

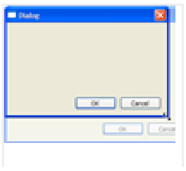
Xia Xiongjun loves this site.

You may also be interested in...

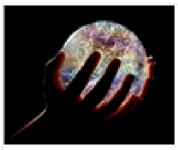


CSizingControlBar - a resizable control bar

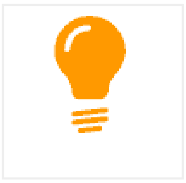
Office 365 SharePoint Online Architectural considerations

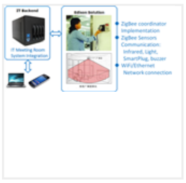


Automatic Layout of Resizable Dialogs



The Past, Present, and Future of IoT


VB.NET - Dynamically Resize and Reposition All Controls when Form is Resized, Including Font Sizes


Case Study：Build a Smart Conference System by Enabling ZigBee on the Intel® Edison Platform

Comments and Discussions

Add a Comment or Question

Search Comments Go

First Prev Next

Very useful piece of work
Siddharth R Barman 9-Jan-15 4:51

CDockablePane support
Andrew Truckle 23-May-14 19:18

Thanks for the solution..
mahigce 26-Aug-13 15:31

Re: Thanks for the solution..
Xia Xiongjun 29-Aug-13 23:34

This class can be applicabled on activeX control?
iishero 28-Dec-11 19:49

Re: This class can be applicabled on activeX control?
Xia Xiongjun 28-Dec-11 23:06

When background of main dialog is erased for applying new color, the static text controls colors are not applied properly. How to proceed?
SharanyaMahi 18-Oct-11 20:50

Adding Invisible Items makes them visible. *solution [modified]
brwx 25-Jul-08 21:08

Resizing a custom control
Hatachnat 16-Oct-07 9:14

What about using this in a CPropertyPage / Sheet scenario?
andrewtruckle 16-Jun-07 18:49

Fix for Visual C++ 6.0!
cristitomi 19-Mar-07 20:08

Re: Fix for Visual C++ 6.0!
Xia Xiongjun 22-Mar-07 2:34

Re: Fix for Visual C++ 6.0!
cristitomi 22-Mar-07 16:56

Problem with Combo box control
123rajesh 26-Oct-06 15:03

Re: Problem with Combo box control
andrewtruckle 22-Mar-07 0:34

RestoreWindowPosition
andrewtruckle 26-Sep-06 20:15

What about SDI - Applications?
enne87 17-Sep-06 20:45

Re: What about SDI - Applications?
Xia Xiongjun 17-Sep-06 21:26

Re: What about SDI - Applications?
enne87 18-Sep-06 6:35

How can i create a resizeable dialog in my existing project
premkamalg 29-Jun-06 16:29

Wonderful, bravo ! Added a little correction for VC8
andré_k 14-Jun-06 21:03

Re: Wonderful, bravo ! Added a little correction for VC8
andrewtruckle 29-Feb-08 18:37

Re: Wonderful, bravo ! Added a little correction for VC8
Priya_Sundar 21-Jul-08 17:57

Hi, works fine. NEED ONE MORE THING
kevikev2020 30-May-06 19:05

Re: Hi, works fine. NEED ONE MORE THING
kevikev2020 30-May-06 19:34

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink | Advertise | Privacy | Terms of Use | Mobile
Web01 | 2.8.160410.1 | Last Updated 1 Mar 2006

請選取語言

Layout: fixed | fluid

Article Copyright 2005 by Xia Xiongjun
Everything else Copyright © CodeProject, 1999-2016