



DOWNLOAD FREE E-BOOKS

100 Titles | 100 Pages | Kindle & PDF formats



[home](#) [articles](#) [quick answers](#) [discussions](#) [features](#) [community](#) [help](#)

Search for articles, questions, tips

Articles » Desktop Development » Toolbars & Docking windows » Docking Windows



Automatic resizing controls



drice, 28 Jan 2005

CPOL

Rate:



4.61 (23 votes)

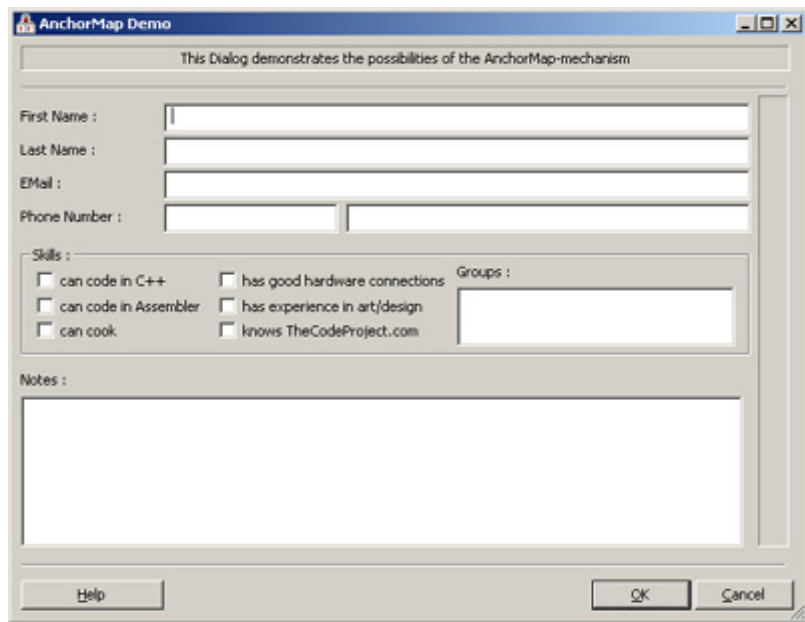
Mechanism to automatically dock/anchor your controls in a window or dialog.



Is your email address OK? You are signed up for our newsletters but your email address is either unconfirmed, or has not been reconfirmed in a long time. Please **click here to have a confirmation email sent** so we can confirm your email address and start sending you newsletters again. Alternatively, you can **update your subscriptions**.

[Download source files - 29.6 Kb](#)

[Download demo project - 83.4 Kb](#)



Introduction

In the following article, I will describe a mechanism that makes it possible to use anchoring and docking for your controls within dialogs and windows.

Docking and anchoring of controls means that some or all controls in a dialog or just a normal window adjust their size and position according to the size of the parent-window (a dialog or view for example). You've already seen this mechanism in many applications. The contact-form in MS Outlook is a good example. If you resize the form, the input-lines, memo-boxes and other controls will also shrink or increase their height and/or width to fit perfectly into the new area of the window.

You may also know this feature from the windows-forms designer in the .NET development environment. If you're a .NET developer by nature and will or have never used C or C++ to write Windows-applications, then this article will not help you very much. But for those who still write ATL/MFC-Applications or have to maintain such, the code presented herein may be a nice improvement for your applications.

The idea behind anchoring

The idea behind docking and anchoring is very simple. You give the user a resizable window or dialog and the elements within that window adjust their size and position to keep their logical position and size or adjust their size to fit into the area of the parent-window so that you don't have wasted space or have vanishing controls when the window becomes to small. With this mechanism, the user can adjust the size of the windows and controls and experiences more flexibility with your application.

If you'd write this code for every dialog and control, and if you have many of them, you'll surely need some time for this task. And in addition, it will become a boring task after the second or 3rd dialog.

The BPCtrlAnchorMap

The docking/anchoring-mechanism is solved by initializing a control-map with the initial size and position of every child-window. When the size of the parent-window changes, a handler-function is called which looks up the control-map and calculates the new size and position for the controls based on the relative size-change of the parent-window.

The BPAncorControlMap (that's what this article is about) provides this control-map-mechanism and all required functions along with some useful macros. It provides a quick and easy way to implement anchoring/docking in your dialogs and reduces your code-writing efforts to a minimum. All you have to do is to call two functions and declare the so called anchor-map.

Implementing the BPCtrlAnchorMap

The BPCtrlAnchorMap can be implemented in any *MFC* or *ATL*-based application. With some minor modifications, it can also be used in pure *win32* C or C++ applications that have no object-oriented framework.

The following example is based on an MFC-Dialog-Application and introduces these required steps to make it work.

- Include the file *bpctrlanchormap.h* in the same header-file where your class is declared or include it in a common header-file of your application like *stdafx.h*.
- Call the macro **DECLARE_ANCHOR_MAP()** within your class declaration. This macro produces the declarations for our two required functions named **InitAnchors** and **HandleAnchors**. We have to call these functions later in our code.
- Call the **InitAnchors()** function within the **OnInitDialog**-Handler or the **OnCreate**-Handler of your window. It is important that, the child-windows of your dialog or window have already been created and have a valid window handle assigned to it when you call **InitAnchors()**. If this condition is met, you can call **InitAnchors()** from anywhere in your code, but the two functions mentioned above are the best place.

[Hide](#) [Copy Code](#)

```
BOOL CAnchorMapDemoDlg::OnInitDialog() {  
  
    CDialog::OnInitDialog();  
    // some initialization code  
    // ...  
    // ...  
  
    InitAnchors();  
  
    return(TRUE);  
};
```

The **InitAnchors()** function can be called with a flags-parameter to specify additional functionality:

ANIF_CALCSIZE (0x0001)

If you specify this flag in the call to **InitAnchors**, the function will calculate the required size for the window by itself. This is useful for FormViews where the dialog-window is resized to fit into the area of the parent-window before the call to **InitAnchors**. (see also "Using the anchor-map with Form-Views")

ANIF_SIZEGRIP (0x0002)

This flag will tell the initialization function to add a sizing-grip into the bottom-right edge of the window. The sizing-grip will automatically be hidden when the window is maximized, and will be shown when the window is in "normal" or restored state.

- Add an **OnSize(WM_SIZE)** message-handler to your class (claswizard will do this for you) and call **HandleAnchors(&rcWnd)** from it. The **&rcWnd** parameter is a pointer to a **RECT**-structure which must contain the actual window-coordinates of your dialog or window. You can get these coordinates by calling **GetWindowRect(&rcWnd)** before.

[Hide](#) [Copy Code](#)

```
void CAnchorMapDemoDlg::OnSize(UINT nType, int cx, int cy) {  
  
    CDialog::OnSize(nType, cx, cy);  
  
    CRect rcWnd;  
    GetWindowRect(&rcWnd);  
  
    HandleAnchors(&rcWnd); // you can alternatively pass NULL for &rcWnd  
  
};
```

You can also call **HandleAnchors(NULL)** to let the function get the coordinates for you. In that case, your window-class must be derived directly or indirectly from **CWnd** (this is because **m_hWnd** is required).

- Define the *anchor-map* within your C or CPP file. (You have to do this outside of your class). The anchor-map-definition looks like a message-map definition. We will discuss anchor-maps in the next section. A simple example of an anchor-map looks like this.

[Hide](#) [Copy Code](#)

```
BEGIN_ANCHOR_MAP(CMyDialog)  
    ANCHOR_MAP_ENTRY(IDC_MYCONTROL, ANF_BOTTOM | ANF_RIGHT)  
END_ANCHOR_MAP()
```

- Compile and run your application

Defining anchor-maps

The anchor-map definition tells the code in *BPCtrlAnchorMap.h* which controls in your window should take part in docking/anchoring and how these controls should behave.

You begin the definition of an anchor-map with the macro **BEGIN_ANCHOR_MAP(theClass)** where you replace **theClass** with the name of your window-class (CMyDialog for example).

After this you can define one or more anchor-map entries with the macro **ANCHOR_MAP_ENTRY(nIDCtrl, nFlags)**. The **nIDCtrl**-parameter is the *control-id* of the child-window that should be added to the docking/anchoring mechanism. The **nFlags**-Parameter is a combination of one or more of the following constants and specifies the docking-behaviour for the control.

The following **docking-flags** are used to dock a control to one border of a window. A docked control will be moved to the border-side of the window to which it is docked and will adjust its width or height to the same width or height of the border while the other dimension (width or height) of the control will stay constant. Note that the **ANF_DOCK** flags should not be combined together or combined with anchoring-flags.

ANF_DOCK_TOP (0x0001)

This flag docks the control to the top of the window.

ANF_DOCK_BOTTOM (0x0002)

This flag docks the control to the bottom of the window

ANF_DOCK_LEFT (0x0004)

This flag docks the control to the left-side of the window

ANF_DOCK_RIGHT (0x0008)

This flag will dock the control to the right-side of the window

ANF_DOCK_ALL (0x000F)

This flag will dock the control to all border-sides of the window

ANF_DOCK_TOP_EX (0x0200)

Docks the control to the top of the window but keeps its original width and height.

ANF_DOCK_BOTTOM_EX (0x0400)

Docks the control to the bottom of the window but keeps its original with and height.

ANF_DOCK_LEFT_EX (0x0800)

Docks the control to the left-side of the window but keeps its original width and height

ANF_DOCK_RIGHT_EX (0x1000)

Docks the control to the right-side of the window but keeps its original width and height.

The following are the **anchoring-flags** and they are used to define the borders of a control to have a constant distance to the border-sides of the parent-window. If the parent-window changes its size, an anchored control will move its edges along with the edges of the parent-window. The following flags can be combined together in any combination.

ANF_TOP (0x0010)

The distance of the control to the top of the parent-window will be constant

ANF_BOTTOM (0x0020)

The distance of the control to the bottom of the parent-window will be constant

ANF_LEFT (0x0040)

The distance of the control to the left-side of the parent-window will be constant

ANF_RIGHT (0x0080)

The distance of the control to the right-side of the parent-window will be constant

ANF_AUTOMATIC (0x0100)

This is a special flag which lets the code determine the best anchoring-method. You should not combine this flag with any other flags.

Additionally, there are some **special flags** which you can combine with the docking/anchoring flags:

ANF_ERASE (0x02000)

This is a special flag which tells the **EraseBackground()** function to erase the area occupied by the control. The **EraseBackground()** function is a special function used to reduce the flickering in windows with many controls. For more information about this topic, see "Removing the flickering"

Special-case is NULL

Note that there is a special case when you pass **NULL** or **0** as *Contro-ID* to the **ANCHOR_MAP_ENTRY** macro. A value of **0** as Control-ID has the effect that all controls in the parent-window which are not already added to the control-map will now be added with the specified nFlags-parameter. The quickens way to define an anchor-map for a dialog is to add the entry **ANCHOR_MAP_ENTRY(NULL, ANF_AUTOMATIC)**.

This will add all controls automatically and determine the best anchoring-method for each-control.

ANCHOR_MAP_ENTRY_RANGE macro

In the latest version, I have added the **ANCHOR_MAP_ENTRY_RANGE** macro which can be used to add a range of control-IDs to the control-map. This macro is used exactly the same way as **ANCHOR_MAP_ENTRY**, except that you have to pass two Control-IDs as parameters, one to specify the first ID in the range and one for the last ID. The third parameter is the normal flags-value.

Hide Copy Code

```
ANCHOR_MAP_ENTRY_RANGE(IDC_B_SKILL1, IDC_B_SKILL6, ANF_TOP | ANF_LEFT)
```

Now, after you have defined all the anchor-map entries, you end the anchor-map definition with a call to the macro **END_ANCHOR_MAP()**.

You're now ready to test and run your application.

Using the anchor-map with Form-Views

The **BPCControlAnchorMap**-mechanism was originally designed for dialogs where the dialog or parent-window and its controls, both have constant dimensions. A problem arises when you use the mechanism with **CFormView**. A **CFormView** resizes itself to fit into the area of the parent-window in which it is contained. Since this process happens before you can call **InitAnchors()**, the control-map will be empty and the controls will not be resized correctly.

A workaround for this is to use the flag **ANIF_CALCSIZE** when calling the **InitAnchors**-function. If you call **InitAnchors(ANIF_CALCSIZE)**, then the **InitAnchors**-function will try to find the original size of the parent-window (your **CFormView**) by taking the bottom-right-most coordinate of all controls which are in the control-map.

Removing the flickering

Under certain circumstances or if you have large dialogs with many controls, the controls and the window may flicker when it is resized. This is because the default-implementation of the **WM_ERASEBKGD** message-handler erases the whole dialog-window and therewith our controls. This works in standard dialogs where the controls do not move or change their size.

To get rid of the flickering and reduce it to a minimum, I have added a special **EraseBackground**-function that erases only the area which is not occupied by controls. The mechanism uses a **Region (HRGN)**-object which is initialized to the client-rectangle of the window. After the region-object has been initialized, the **EraseBackground** function cycles through all child-windows (not just the ones in the control-map) and removes their areas from the region. Finally, the region is filled with the background-color. This prevents the child-windows from being overpainted with the background-color.

A problem that I have experienced with this technique is when using Group-Boxes. Group-Boxes do not draw their inner background and because the whole area of the group-box is removed from the region, its background is not filled and the group-box becomes "transparent". To force the area of the group-box not to be removed from the region, use the **ANF_ERASE**-flag when you add the group-box control to the control-map.

If you want to use this custom **EraseBackground** function, add a **WM_ERASEBKGD**-Handler to your window and call **m_bpfxAnchorMap.EraseBackground()** instead of the default implementation.

Hide Copy Code

```
BOOL CAnchorMapDemoDlg::OnEraseBkgnd(CDC* pDC) {
    // Here we call the EraseBackground-Handler from the
    // anchor-map which will reduce the flicker.
    return(m_bpfxAnchorMap.EraseBackground(pDC->m_hDC));
}
```

Behind the scenes

The following lines will try to give you an insight into what all the macros do. If you're interested in how the code works, download it and have a look into *bpctrlanchormap.h*.

The whole mechanism takes place within a special class which will be embedded into your dialog-class when you call **DECLARE_ANCHOR_MAP**. The file *bpctrlanchormap.h* includes the declaration of this class (**CBPCtrlAnchorMap**) along with the code. The macro **DECLARE_ANCHOR_MAP()** which you call in your class declaration finally extends to the following C++ code:

Hide Copy Code

```
CBPCtrlAnchorMap m_bpfxAnchorMap;  
void InitAnchors(BOOL bFindCtrlEdges = 0);  
void HandleAnchors(RECT *pRect);
```

Since you call **DECLARE_ANCHOR_MAP** within the declaration of your class, the member **m_bpfxAnchorMap** and the functions **InitAnchors** and **HandleAnchors** are added to your class.

The implementation (code) for the **InitAnchors** and **HandleAnchors** functions is generated with the anchor-map macros **BEGIN_ANCHOR_MAP**, **ANCHOR_MAP_ENTRY** and **END_ANCHOR_MAP**.

The **BEGIN_ANCHOR_MAP(theClass)** macro finally extends to the following code :

Hide Copy Code

```
void theClass::HandleAnchors(RECT *pRect) {  
    m_bpfxAnchorMap.HandleAnchors(pRect);  
}  
  
void theClass::InitAnchors(BOOL bFindCtrlEdges) {
```

Note that the **InitAnchors**-function is not closed with a curly-brace. This is because the code for the function is still missing. It comes with the **ANCHOR_MAP_ENTRY(nCtrlID, nFlags)** which extends to the following code:

Hide Copy Code

```
m_bpfxAnchorMap.AddControl(nIDCtrl, nFlags);
```

The **END_ANCHOR_MAP** macro contains some additional code for **InitAnchors** and finally outputs the closing-brace for the function:

Hide Copy Code

```
m_bpfxAnchorMap.Initialize(m_hWnd, bFindCtrlEdges);  
RECT rcWnd;  
::GetWindowRect(m_hWnd, &rcWnd);  
m_bpfxAnchorMap.HandleAnchors(&rcWnd);
```



```
};
```

Now that we have our functions ready for use, we can call the **InitAnchors** and **HandleAnchors**-function from our class. The rest is done by the code of CBPCtrlAnchorMap.

Well, that's it.

I hope you'll find this code useful and maybe someone can help me to reduce the flicker when large dialogs with many controls are resized.

bye, drice !

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



drice

Software Developer (Senior) BluePearl Software
Germany 

I started in the late 80's with a VIC 20 and wrote my first computer programs. In the 1990's I developed games for a published diskmag and also became active in the demo-scene. That was the time where I got a deep understanding about computers, Bits and Bytes and programmed a lot of different things (Games, Compilers, Tools and a lot of other stuff - It was just pure Fun !). In that time I was also reading a lot of books. I've used Turbo Pascal, Assembler, C++ and finally arrived in the business and C# world. In 2001 I founded my own company "BluePearl Software" and I develop business and technical applications for small and mid-sized companies as a freelancer, both industrial and non-industrial and in a wide range of business sectors.

I am still interested in the nitty-gritty of PCs and software and how to code/design software in an elegant way. I like to face and solve problems and design good software.

You may also be interested in...



Automatic Layout of Resizable Dialogs



Mobile Messaging with Twilio



Resizing Controls When Resizing Form



Announcing Ubuntu and Wind River Pulsar support with Intel® IoT Developer Kit 5.0



Visual COBOL New Release: Small point. Big deal



SAPrefs - Netscape-like Preferences Dialog

Comments and Discussions

<div>Add a Comment or Question ?</div>		<div>Search Comments</div>		<div>Go</div>					
		Spacing	Relaxed ▼	Layout	Normal ▼	Per page	25 ▼	<div>Update</div>	
							First	Prev	Next
<div></div>	<div>More simpler and more powerful way..</div>					<div></div>	<div>Mizan Rahman</div>		<div>25-Feb-11 21:25</div>
<div></div>	<div>Re: More simpler and more powerful way..</div>					<div></div>	<div>Flaviu2</div>		<div>22-Dec-11 19:14</div>
<div></div>	<div>Auto-mode</div>					<div></div>	<div>johnnyk4277</div>		<div>6-Feb-10 6:33</div>

Simply Amazing	bleubleu	12-Oct-08 10:02
Minor enhancement code	Mydraal	16-Jan-07 6:15
really perfect	kalaveer	20-Sep-06 14:14
Controls getting vanished	Raj Patil	29-May-06 20:44
Comment	vbachmut	4-Feb-06 5:26
Re: Comment	vietdoorgroup	27-Jul-06 16:58
Another ResizableLib?	Alexander Shevchenko	1-Feb-05 13:01
Flickering	Stephane Rodriguez.	30-Jan-05 22:46
The flickering problem	W. Kleinschmit	28-Jan-05 16:26
Re: The flickering problem	prcarp	28-Jan-05 22:02
Re: The flickering problem	drice	29-Jan-05 6:01
Re: The flickering problem	drice	29-Jan-05 6:16
question	.dan.g.	28-Jan-05 7:08
Re: question	drice	29-Jan-05 3:35
<div> Last Visit: 18-Dec-16 17:03 Last Update: 20-Dec-16 6:11 <div>Refresh</div> <div>1</div> </div>		

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
Web02 | 2.8.161218.1 | Last Updated 27 Jan 2005

請選取語言 ▼

Layout: [fixed](#) | [fluid](#)

Article Copyright 2005 by drice
Everything else Copyright © [CodeProject](#), 1999-2016

