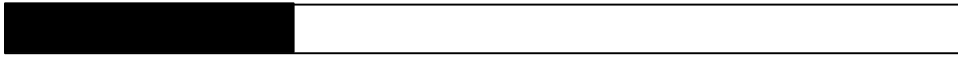


Construct an ActiveX control that acts as a progress bar.
For something 25% done, it might look like this:



There is no need to draw the bounding rectangle. We will draw only the background and the colored portion of the bar.

Use the following steps to create your control.

1. Create a new control. (Don't modify any of your previous efforts, start from scratch!)
2. Add a property *PerCent* to your control. Make this property persistent, and set its default value to zero. In the *OnPerCentChanged* routine, if a value smaller than zero is entered, replace it with a zero. If a value greater than 100 is entered, replace it with 100. Call *Invalidate()* to force a redraw of the window.
3. Even though we are going to do animation, we will not draw successive pictures. Instead, we will make the *OnDraw* routine responsible for drawing *one frame* of the animation.
4. In the *OnDraw* routine, retain the line starting *cdc->FillRect(...*, because this will be used to draw the background.
5. We need to add a second *cdc->FillRect(...* to draw the filled portion of the rectangle. To accomplish this we must first create a blue brush (all progress bars use blue, don't they?) and then we must do a transformation of coordinates.
6. The height of the filled rectangle will be the height of the window. The width will be from 0 to 100, in Bar Coordinates. We must transform the variable *m_perCent* from bar coordinates to screen coordinates which run from *rcBounds.left* to *rcBounds.right*. So 0 must correspond to *rcBounds.left* and 100 must correspond to *rcBounds.right*.
7. In GUI programming we must do transformation of coordinates all the time, so lets just "do the drill." The coordinate transformation is linear, so the equation must be of the form $y = Ax + B$. We must determine A and B. First, 0 must transform to *rcBounds.left*, so we get the following equation.

$$\text{rcBounds.left} = A * 0 + B = B$$

Now 100 must transform to *rcBounds.right*, so we get the following derivation.

$$\text{rcBounds.right} = A * 100 + \text{rcBounds.left}$$

$$\text{rcBounds.right} - \text{rcBounds.left} = A * 100$$

$$A = (\text{rcBounds.right} - \text{rcBounds.left}) / 100$$

We have to be careful with A, because in many cases (*rcBounds.right* – *rcBounds.left*) will be smaller than 1. We wish to avoid floating point arithmetic, but we also want to avoid the loss of precision that an integer division would cause. Therefore, we cannot compute A directly, but must use the numerator and denominator of A in our calculations, as follows.

```
long BarRight = m_perCent * (rcBounds.right – rcBounds.left);
```

```
BarRight = BarRight / 100;
```

```
BarRight = BarRight + rcBounds.left;
```

8. To create the blue brush we use the following statements.
 `CBrush BlueBrush;`
 `BlueBrush.CreateSolidBrush(RGB(0,0,255));` // learn how to do RGBs!
9. To create the filled rectangle use the following statements.
 `RECT FRect = rcBounds;` //everything the same except right
 `FRect.right = BarRight;` // we don't really need the intermediate variable
10. Now we have to do animation. We will do that in Visual Basic, using the timer control. Add a timer control to your VB program, use the default name, "Timer1" and make sure that the property "Interval" is set to 0.
11. Add a button to start the animation. We will assume the default name "Command1." We will assume further that your progress bar has the default name "ProgressBar1".
12. Double-click on the button and add the following code to the "click" event. This will cause a redraw of the control every 1/10 second.
 `Timer1.Interval = 100`
 `ProgressBar1.Percent = 0`
13. Double-click on the timer control, and add the following code to the timer event.
 `If ProgressBar1.Percent >= 100 Then`
 `Timer1.Interval = 0`
 `Else`
 `ProgressBar1.Percent = ProgressBar1.Percent + 1`
 `End If`
14. Run your program. You should notice the left-hand side of the bar flickering as the animation progresses. Instead of this, we would like to see the bar grow smoothly from left to right. Here's how to get rid of the flicker. (Note, in this simple case, we could do something simpler, but the following technique will work for almost all small-window animations.)
15. Instead of drawing directly to the screen, we will draw into a memory-based bitmap, and then snap the completed drawing into the window. This will eliminate the flicker caused by erasing the old bar.
16. The first step is to make a copy of your OnDraw Routine. First, go to the left-hand window that contains the list of classes, and right click on the CxCtrl class, where x is your project name. Select "Add Member Function" from the pop-up menu.
17. The new function type is void, and the new function declaration is
 `OnDraw2(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);`
18. Double-click on the name OnDraw2 to go to the code for the new function. Delete the 2 from the name of this function, then find the old OnDraw function and add a 2 after the "w". In other words, the body of the new function becomes the OnDraw routine, and the OnDraw routine becomes the body of the function you just created.
19. Now we will begin to add code to the empty OnDraw routine. The first step is to create a bitmap that is exactly the same size as our drawing window. To do this, we use the following statements.
 `CBitmap MyMap;`
 `MyMap.CreateCompatibleBitmap(pdc,rcBounds.Width( ),rcBounds.Height( ));`

20. Now we must create a drawing context that allows us to use drawing commands in the bitmap. We want this drawing context to have the same properties as the screen, so we will use the following statements.
- ```
CDC MyDc;
MyDc.CreateCompatibleDC(pdc);
CbitMap * OldMap = MyDc.SelectObject(&MyMap);
```
21. The bounding rectangle of the bitmap is not the same as the bounding rectangle of our control's window, so we need to create a new bounding rectangle using the following statements.
- ```
CRect MyRect;  
MyRect.left = 0;  
MyRect.right = rcBounds.Width( ) - 1;  
MyRect.top = 0;  
MyRect.bottom = rcBounds.Height( ) - 1;
```
22. Finally, we call our old drawing routine to draw into the bitmap we just created.
- ```
OnDraw2(&MyDc,MyRect,MyRect);
```
23. Now we need to snap the pre-drawn picture onto the screen. To do this we use the BitBlt function.
- ```
pdc->BitBlt(rcBounds.left,rcBounds.top,rcBounds.Width( ),rcBounds.Height( ),  
            &MyDc,0,0,SRCCOPY);
```
24. The only thing left is to clean up the stuff we created.
- ```
MyDc.SelectObject(OldMap);
MyDc.DeleteDC();
MyMap.DeleteObject();
```
25. Ok, so now test your control, and observe that the flickering is gone.
26. Turn in "the usual."