Clarion Magazine

Multiline Radio Buttons And Checkboxes

by Carl Barnes

Published 2007-09-18

Many times I'll have a RADIO or CHECK control with text longer than will fit on a single line. I would like the text to wrap over multiple lines exactly the way it does on a PROMPT control, but the Clarion radio and check controls do not support multiline text. The workaround I use is to put the first line of text in the control and the additional lines in a PROMPT control positioned below. This workaround functions just fine and may be visually acceptable to some users, but typically does not look perfect. The text does not line up exactly horizontally or vertically. The focus rectangle only wraps the first line of text which not only looks poor, but can be a little confusing. The final flaw is that when the user clicks on the workaround prompt text the control does not get selected. This lack of functionality can make user think the control is disabled or read-only.

Figure 1 shows the difference between the workaround and a true multiline control. In each case, the lower example shows the appearance with an XP manifest. As you can see, the workaround can have problems with text alignment.

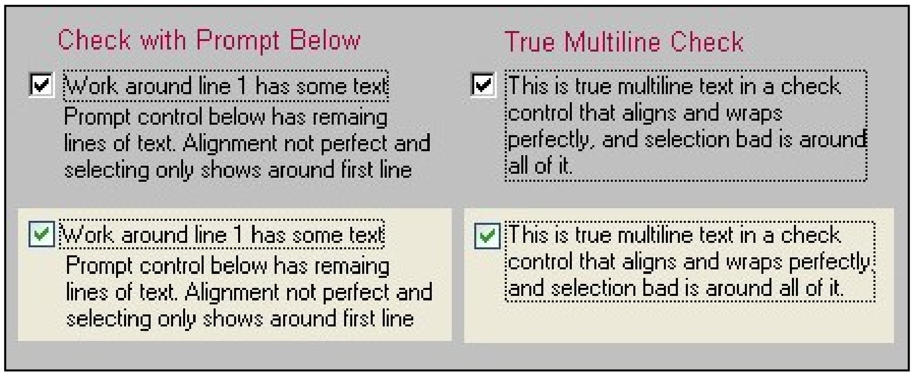

Check with Prompt Below
Work around line 1 has some text
Prompt control below has remaing lines of text. Alignment not perfect and selecting only shows around first line
True Multiline Check
This is true multiline text in a check control that aligns and wraps perfectly, and selection bad is around all of it.
Work around line 1 has some text
Prompt control below has remaing lines of text. Alignment not perfect and selecting only shows around first line
This is true multiline text in a check control that aligns and wraps perfectly and selection bad is around all of it.

Figure 1. Workaround and true multiline controls

Clarion may not natively support multiline controls, but that doesn't mean it can't be done. In this article I'll show you how to easily add multiline button, radio, and check controls to your applications by changing one style bit via the API.

Start with the button

Most Clarion controls are based on some form of standard Windows control. The Clarion BUTTON, CHECK and RADIO controls are all based on the Windows Button control class. (Yes, a check and radio are special forms of a button.) Looking up the styles available for the Button class on MSDN I found BS_MULTILINE (0x00002000), with the description "Wraps the button text to multiple lines if the text string is too long to fit on a single line in the button rectangle".

You can modify control styles with the Window API function SetWindowLong(), protyped as follows:

```
SetWindowLong(Unsigned,SIGNED,LONG),LONG,|
  PASCAL,DLL(1),NAME('SetWindowLongA'),proc
```

The following code is all it takes to turn on the multiline style:

```
BS_MULTILINE    EQUATE(2000h)
GWL_STYLE       EQUATE(-16)
BtnStyle        LONG,AUTO
    CODE
    BtnStyle=GetWindowLong(?Check1{PROP:Handle},GWL_STYLE)
    BtnStyle=BOR(BtnStyle,BS_MULTILINE)
    SetWindowLong(?Check1{PROP:Handle}, GWL_STYLE, BtnStyle)
```

Why didn't TopSpeed or SoftVelocity implement multiline controls in Clarion? Probably because this style is part of the Win32 API. Clarion supported 16 bit compilation through version 5.5, and the Clarion 6 IDE remains 16-bit. The 16-bit issues would have made it difficult to implement the feature. And maybe the feature was never requested by a developer. Searching the web for BS_MULTILINE information I also found that Borland never implemented multiline in Delphi or C++ Builder, so those developers also have to add multiline support via the Windows API.

New Win32 Alignment Styles

Besides multiline buttons there were also button text alignment styles added in Win32 that the Clarion language and IDE do not accommodate. The vertical position of the text can be specified as top, center (default) or bottom. For a check or radio this also specifies the location of the check box or radio circle. The horizontal alignment of the text can be specified as left (default), center or right. I don't care for the default vertical alignment of centered, so I typically change it to Top. Figure 2 shows some of the alignment combinations.

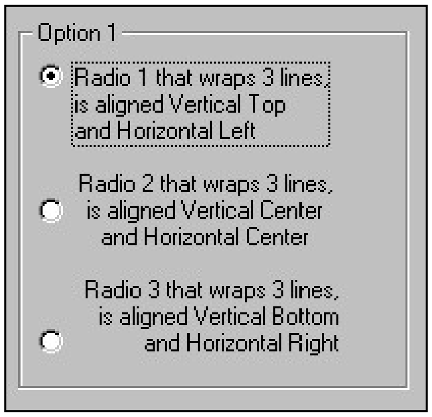

Option 1
Radio 1 that wraps 3 lines, is aligned Vertical Top and Horizontal Left
Radio 2 that wraps 3 lines, is aligned Vertical Center and Horizontal Center
Radio 3 that wraps 3 lines, is aligned Vertical Bottom and Horizontal Right

Figure 2. Left/top, center/center, and right/bottom alignments

The API also allows changing the placement of circle or box to be on the right side of the text with the style BS_RIGHTBUTTON (or you can use the alias BS_LEFTTEXT). Clarion does supports this style by adding the

LEFT attribute to the control. But Clarion does not allow you to specify right or centered text, so you end up with left aligned text as in Figure 3. I wish TopSpeed Clarion hadn't used the LEFT attribute to specify the location of the box/circle; now the keywords LEFT, CENTER and RIGHT cannot be used to specify text alignment.

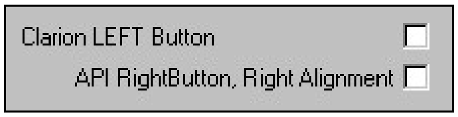


Figure 3. Clarion vs API right-side checkboxes

The complete list of Windows button style equates for setting these values are shown in the below table. Note that left, right, top and bottom are all separate bits. To turn on one you'll need to be sure the opposite is turned off, i.e. to turn on left you need to be sure right is turned off or you may end up with centered.

To turn off a bit you need to bitwise AND the complement. For example, to turn off Right the code is Style=BAND (Style,BXOR(-1,BS_RIGHT)).

Equate	**Value**	**Description**
GWL_STYLE	-16	Sets a new window style using SetWindowLong
BS_LEFT	0100h	Left-justifies the text in the button rectangle
BS_RIGHT	0200h	Right-justifies text in the button rectangle
BS_CENTER	0300h	Centers text horizontally in the button rectangle (Note Center = Left +Right)
BS_TOP	0400h	Places text at the top of the button rectangle
BS_BOTTOM	0800h	Places text at the bottom of the button rectangle
BS_VCENTER	0C00h	Places text in the middle (vertically) of the button rectangle (Note Center = Top+Bottom)
BS_RIGHTBUTTON	0020h	Positions the radio circle or check square on the right side of the button rectangle
BS_LEFTTEXT	0020h	Alias for BS_RIGHTBUTTON. Same as Clarion LEFT on RADIO or CHECK.
BS_PUSHLIKE	1000h	Makes a check box or radio look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. Clarion CHECK or RADIO will render "Push Like" if they have an ICON or FLAT attribute.
BS_MULTILINE	2000h	Wraps the button text to multiple lines if the text string is too long to fit on a single line in the button rectangle
BS_FLAT	8000h	Specifies that the button is two-dimensional; it does not use the default shading to create a 3D image.

The BS_PUSHLIKE style renders a CHECK or RADIO as a "latch button" instead of a box or circle graphic. When unchecked the button appears in its normal up position, when checked it appears as latched down. This can be done in

Clarion code by adding an ICON() to a CHECK or RADIO. Specifying ICON(ICON:None) will render the latch effect as a text button without an icon showing. The FLAT attribute will also render as latch button, but the button will appear flat when unchecked (Figure 4).

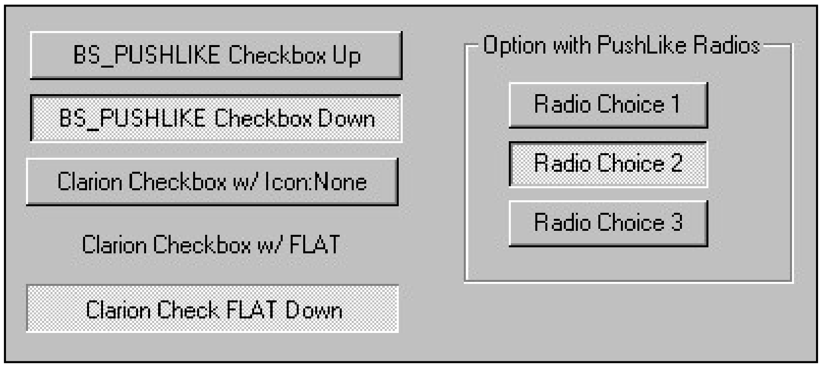


Figure 4.BS_Check/Radio's with BS_PushLike style

The API BS_Flat style is very different from the Clarion FLAT attribute. BS_Flat turns off 3D and renders a button, check or radio in a 2D effect that has an interesting retro look (Figure 5).

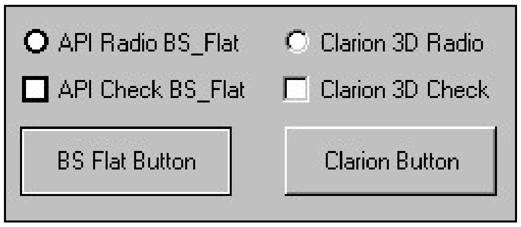


Figure 5.BS_Flat style

Multiline Text Buttons

Clarion supports multiple lines of text on a button as long as the button has an icon. For a multiline text only button with no icon at all use Icon:None:

```
BUTTON('With multiple<13,10>lines of text'),ICON(ICON:None)
```

In the Clarion C6 Window Formatter button icon drop list select "None" instead of "No Icon"; in C5.5. and prior you must add the attribute via the Window ellipsis [...] button on the procedures property. You can also skip the ICON attribute and apply the API BS_MULTILINE style via the API call. In my testing the results were identical so I would tend to use Icon:None and not have to write any extra code.

Specifying a font color using the FONT(,,COLOR:xxx) attribute for the text on the button will also cause Clarion to support multiline text on a button. If you don't want colored text you can use the equate COLOR:BtnText to use the Windows default color. COLOR:BtnText worked correctly at runtime but did not display correctly in the IDE. And finally, the FLAT attribute will also wrap the button text.

Manifest Issues

If you have an XP Manifest and the user's Windows appearance is set for the Classic look, then the horizontal alignment of center and right does not quite work the same. It appears the first line is done using the requested alignment, but the rest of the lines are always left -aligned under that. Figure 6 is a screen capture showing without and with an XP manifest in the Classic look.

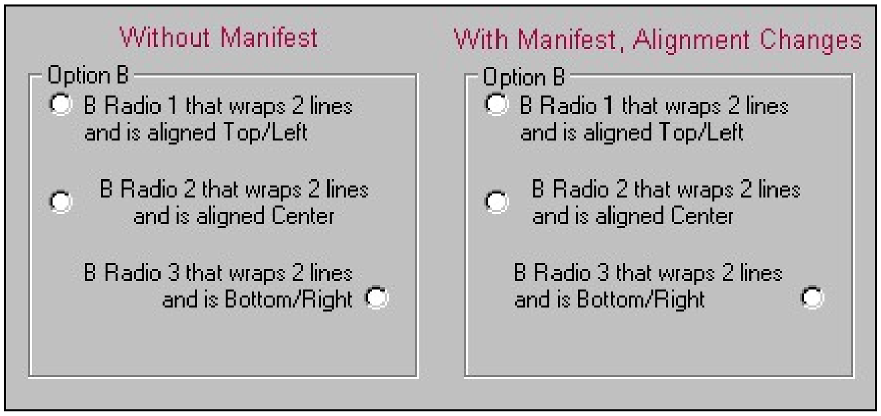

Without Manifest
With Manifest, Alignment Changes
Option B
B Radio 1 that wraps 2 lines and is aligned Top/Left
B Radio 2 that wraps 2 lines and is aligned Center
B Radio 3 that wraps 2 lines and is Bottom/Right

Figure 6. Manifest alignment differences with the Classic look

Multiline in the Clarion IDE Window Designer

The Clarion Window Designer does not support any of these API multiline text or alignment features at design time or in preview mode, so what you see in the IDE is not what you'll get at runtime. You'll just see one line of text centered in the height of the control and additional lines will be truncated. Trying to size the control will require running the program to verify it displays as desired.

One way to get a design time view is to use a PROMPT control as a design-time tool, but not as a runtime element on the form. You put the full text into a PROMPT control next to the CHECK / RADIO, then at runtime transfer the text and resize the control; once CHECK / RADIO control is set up you disable and hide the PROMPT.

The PROMPT supports specifying the Alt key with an ampersand and this hot key will also transfer to the CHECK / RADIO control. The PROMPT also supports embedding manual line breaks with a <13,10> in the text, and this will work correctly with multiline button controls.

Figure 7 is a screen shot from the Window Designer showing both methods. I selected all the controls so you can see that on the left the text is a separate prompt control next to the check control.

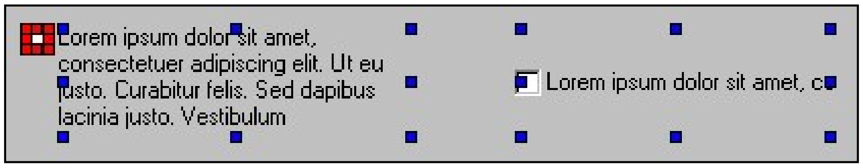

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut eu justo. Curabitur felis. Sed dapibus lacinia justo. Vestibulum
Lorem ipsum dolor sit amet, c

Figure 7. The Window Designer showing both approaches to multiline controls

In most of my work the text fits into two lines so I won't bother with the separate prompt. I temporarily add a prompt to get an idea of the size required to fit and wrap the text. Then I'll put the text into the check/radio, resize it using the prompt, and delete the prompt. I size the control wider then the first line of text and insert a <13,10> where I want the break to occur.

A Function to Make it Easy

Included in the download example is a function named SetBS_Multiline that makes it easy single call to implement all of the styles discussed above. The prototype for the function is as follows:

```
SetBS_MULTILINE procedure(  |
    LONG ButtonFEQ,        | ! FEQ BUTTON, CHECK or RADIO
    LONG PromptFEQ=0,      | ! FEQ PROMPT for text and size
    BYTE xLeftCntrRght=0,  | ! 1=Left 2=Center 3=Right
    BYTE yTopCntrBttm=1,   | ! 1=Top  2=Center 3=Bottom
    BYTE bRghtSideBtn=0,   | ! 1=Button on Right
    BYTE bMultiLine=1)     | ! 1=Multiline
```

This function is designed to optionally support your window design having a PROMPT containing the multiline text next to the CHECK / RADIO. The prompt must have the desired width and be aligned to where the text typically starts. The CHECK / RADIO will be resized to the height and width of the PROMPT, and it will allow width for the box/radio button as the prompt x position minus the control x position. Buttons are not resized by the function.

Figure 8 shows a number of controls in the Window designer (with all controls selected).

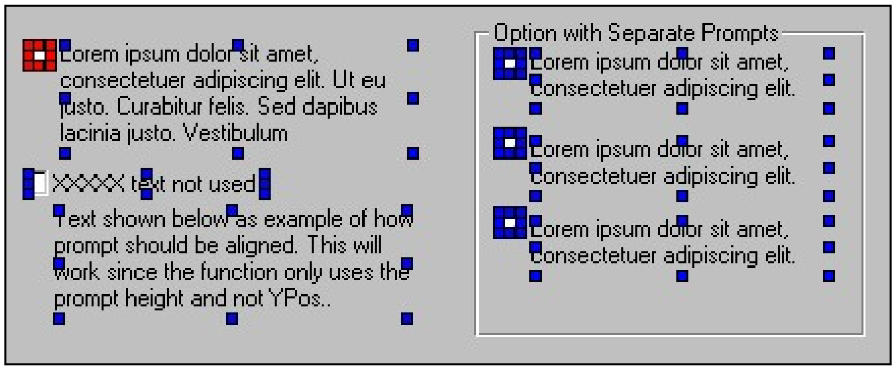

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut eu justo. Curabitur felis. Sed dapibus lacinia justo. Vestibulum
XXXXX text not used
Text shown below as example of how prompt should be aligned. This will work since the function only uses the prompt height and not YPos..
Option with Separate Prompts
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Figure 8. Various controls in the Window Designer

SetBS_Multiline is designed to be called after OPEN(Window) and before the ACCEPT. If you call it inside the ACCEPT loop the controls will not repaint immediately. Issuing a ?Check{PROP:Right}='1' should force the RTL to repaint.

The source code for this function is fairly simple, as follows:

```
SetBS_MULTILINE procedure(  |
    LONG ButtonFEQ,        | ! FEQ BUTTON, CHECK or RADIO
    LONG PromptFEQ=0,      | ! FEQ PROMPT for text and size
    BYTE xLeftCntrRght=0,  | ! 1=Left 2=Center 3=Right
    BYTE yTopCntrBttm=1,   | ! 1=Top  2=Center 3=Bottom
    BYTE bRghtSideBtn=0,   | ! 1=Button on Right
    BYTE bMultiLine=1)     | ! 1=Multiline
```

```
WindStyle       LONG,AUTO
Style_ON        LONG(0)
Style_OFF       LONG(0)
CtrlHWnd        UNSIGNED,AUTO
ControlType     LONG,AUTO
Prmt_X          LONG,AUTO   !Prompt Position
Prmt_Y          LONG,AUTO
Prmt_W          LONG,AUTO
Prmt_H          LONG,AUTO
Btn_X           LONG,AUTO   !Button Control's X
Btn_W           LONG,AUTO   !Control's X
  CODE
  CtrlHWnd = ButtonFEQ{prop:handle}
  WindStyle=GetWindowLong(CtrlHWnd,  GWL_STYLE)
  IF bMultiline
    Style_ON=BOR(Style_ON,BS_MULTILINE)
  end
  IF INRANGE(yTopCntrBttm,1,3)
    Style_OFF=BOR(Style_OFF,BS_VCENTER)
    Style_ON=BOR(Style_ON, |
          CHOOSE(yTopCntrBttm, BS_TOP,BS_VCENTER,BS_BOTTOM))
  END
  IF INRANGE(xLeftCntrRght,1,3)
    Style_OFF=BOR(Style_OFF,BS_CENTER)
    Style_ON=BOR(Style_ON, |
           CHOOSE(xLeftCntrRght, BS_LEFT, BS_CENTER, BS_RIGHT))
  END
  IF bRghtSideBtn
    Style_ON=BOR(Style_ON,BS_RIGHTBUTTON)
  END
  WindStyle=BAND(WindStyle,BXOR(-1,Style_OFF))
  WindStyle=BOR(WindStyle,Style_ON)
  SetWindowLong(CtrlHWnd, GWL_STYLE, WindStyle)
  IF PromptFEQ
    CASE (ButtonFEQ{prop:Type})
    OF Create:Radio OROF Create:Check
      GETPOSITION(PromptFEQ,Prmt_X,,Prmt_W,Prmt_H)
      GETPOSITION(ButtonFEQ,Btn_X,,Btn_W)
      IF Btn_X < Prmt_X           !Button Left of Prompt?
```

```
          Prmt_W += (Prmt_X-Btn_X)
        ELSE                    !Button Right of Prompt?
          Prmt_W =  Btn_X + Btn_W - Prmt_X
          Btn_X  =  Prmt_X
        END
        SETPOSITION(ButtonFEQ,Btn_X,,Prmt_W,Prmt_H)
      OF Create:Button
        !Buttons assumed to be the sized right
      END
      ButtonFEQ{prop:Text}=PromptFEQ{prop:Text}
      HIDE(PromptFEQ)
      DISABLE(PromptFEQ)
    END
    RETURN WindStyle
```

Double Click Radio

A somewhat common dialog offers the user a single choice implemented as an option with radio buttons, and a button to execute the choice. Figure 9 shows an example of such a window.

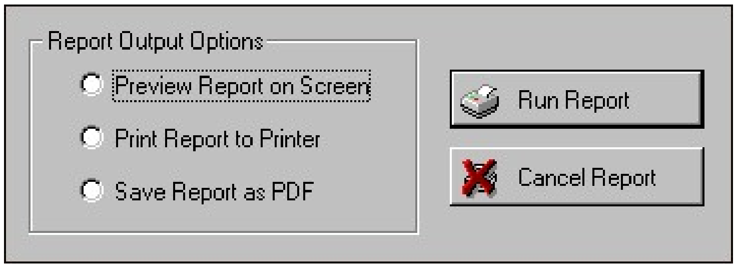


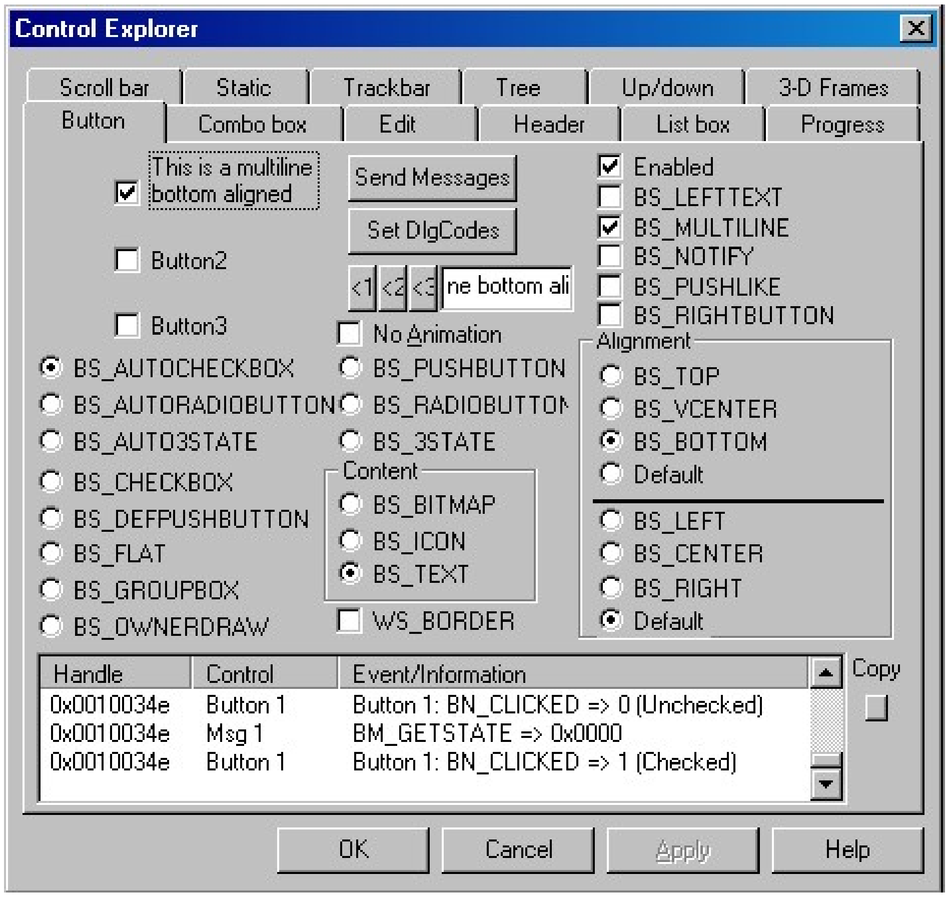
Figure 9. Single Option Dialog

After selecting the radio I find it annoying to have to mouse to the right side and click the button. What I want to do is double click on the radio button and have the logical choice, pushing the run button (default button), done for me. This behavior is easy to program in Clarion by alerting the MouseLeft2 key on the OPTION and processing the EVENT:AlertKey in each RADIO. Note the detail that only the radio buttons get the alert events. An example of double-click radios is included in the download.

Further Reading

The book *Win32 Programming* by Brent Rector and Joseph Newcomer has excellent coverage on programming dialogs using the standard Windows controls and the Windows GDI. It's a 1500 page book and deals with most of the Win32 API. One nice thing for Clarion programmers is it highlights differences between Win16 and Win32 in prominent tip boxes. It was in this book I found out BS_MULTILINE was not available under Win16. Most Win16 information is gone from MSDN.

The CD included with the book contains many useful utility programs. Some of the coolest are the "Explorers" which let you easily configure the many options available for the Windows programming concepts explained in the book. The Control Explorer will let you test all of the Windows standard controls and configure any option as well as send and view messages. This utility is useful for finding other features supported by the API but not by Clarion. Figure 10 is a screen shot of Control Explorer in which I have specified a checkbox as multiline:



[✓] This is a multiline bottom aligned
[ ] Button2
[ ] Button3

[✓] Enabled
[ ] BS_LEFTTEXT
[✓] BS_MULTILINE
[ ] BS_NOTIFY
[ ] BS_PUSHLIKE
[ ] BS_RIGHTBUTTON

[✓ BS_AUTOCHECKBOX] BS_AUTORADIOBUTTON BS_AUTO3STATE BS_CHECKBOX BS_DEFPUSHBUTTON BS_FLAT BS_GROUPBOX BS_OWNERDRAW BS_PUSHBUTTON BS_RADIOBUTTON BS_3STATE

[ ] No Animation

Content: BS_BITMAP BS_ICON [✓ BS_TEXT]

[ ] WS_BORDER

Alignment: BS_TOP BS_VCENTER [✓ BS_BOTTOM] Default

BS_LEFT BS_CENTER BS_RIGHT [✓ Default]

Handle	Control	Event/Information
0x0010034e	Button 1	Button 1: BN_CLICKED => 0 (Unchecked)
0x0010034e	Msg 1	BM_GETSTATE => 0x0000
0x0010034e	Button 1	Button 1: BN_CLICKED => 1 (Checked)

Figure 10. The Control Explorer

Clarion 7

The Clarion 7 Beta build 2430 does not implement multiline or any of the other styles discussed in this article. The Structure Designer will correctly display multiline check/radio text if it contains <13,10> line breaks, but the RTL does not render them as multiline at runtime. All code supplied with this article has been tested and works correctly under 7.2430, as well as Clarion 6, 5.5 and 5b.

Summary

That's about everything I know about button styles supported by the Windows API. There are three-state check boxes (checked, unchecked, and partly checked), but that's a story for another time. You may find it interesting to explore MSDN for other Window control styles not supported by Clarion.

Be sure to download and try the example project. There are examples of every button style I discussed in this article, and a few extended styles that were notI didn't covered. The function it contains is all you'll need to easily implement these new styles. This function could be built into an ABC compliant class or utility function DLL to make it easy to reuse in all your applications.

Resources

- MSDN Button Reference:
- MSDN Button Styles

Download the source

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of a number of Clarion utilities including CW Assistant, The CHM help class CHM4Clarion, and Clarion Source Search.

Reader Comments

Add a comment