

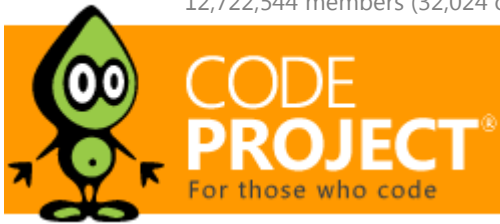
Not quite what you are looking for? You may want to try:

- [MFC Grid control 2.27](#)
- [An MFC Chart Control with Enhanced User Interface](#)

highlights off

12,722,544 members (32,024 online)

jash.liao ▼ 1.1K Sign out



[home](#)

[articles](#)

[quick answers](#)

[discussions](#)

[features](#)

[community](#)

[help](#)

MFC USER CONTROL

Articles » Desktop Development » Button Controls » Owner-draw buttons



## CButtonST v3.9 (**MFC** Flat buttons)



**Davide Calabro**, 29 Mar 2003



4.95 (401 votes)

Rate: ☆☆☆☆☆

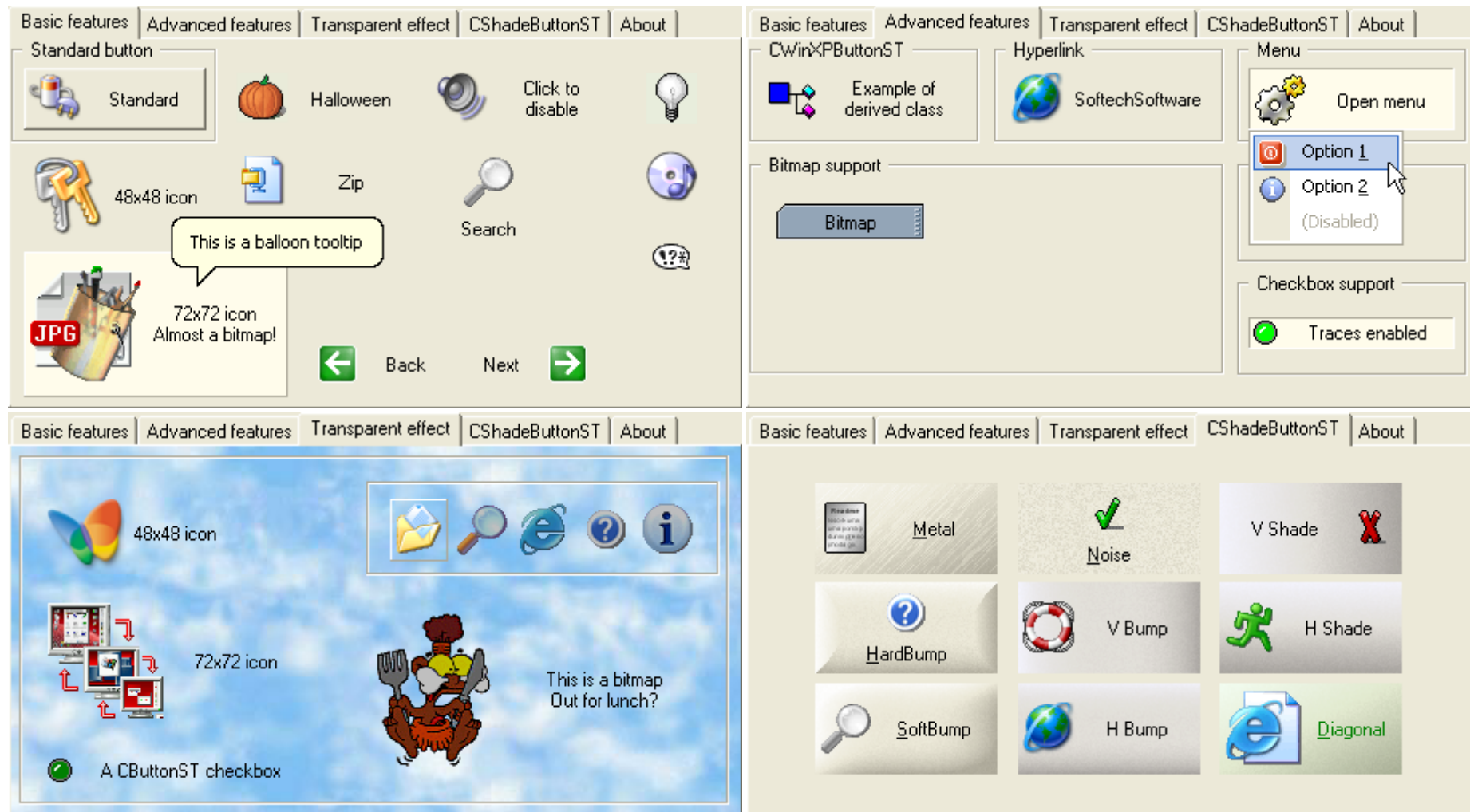
A fully featured owner-draw button class - it's got the lot!



[Download demo project - 902 Kb](#)



[Download source - 19 Kb](#)



[SoftechSoftware homepage](#)

[SoftechSoftware Email](#)

<!--STEP 3 --> <!-- Add the article text. Please use simple formatting (

/

etc) -->

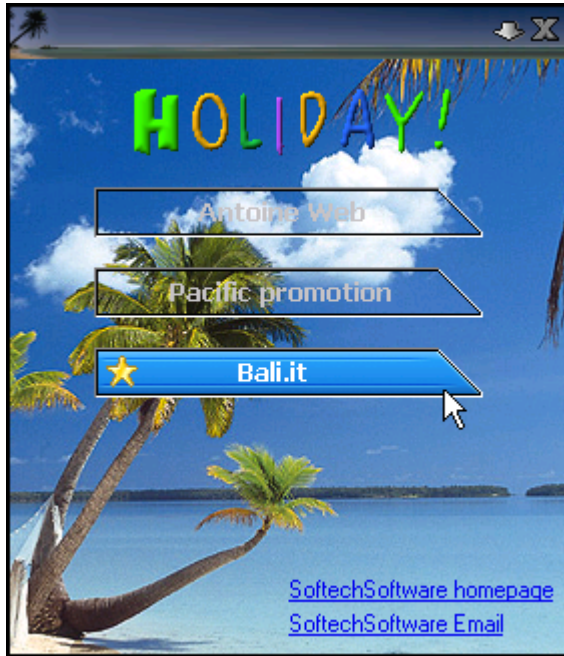
# Abstract

CButtonST is a class derived from **MFC** CButton class.

With this class your applications can have standard buttons or new and modern buttons with "flat" style!

Main CButtonST features are:

- Standard CButton properties
- Text and icon (or bitmap) on the same button
- Only text or only icon/bitmap buttons
- Support for any size icons (max. 256 colors)
- Support for bitmaps
- Support for transparent buttons (for bitmapped applications)
- Standard or flat button style
- Change runtime from flat to standard style
- Buttons can have two images. One when the mouse is over the button and one when the mouse is outside (only for "flat" buttons)
- Every color can be customized
- Can be used via DDX\_ calls
- Can be used in DLLs
- Can be dinamically created
- Each button can have its own mouse pointer
- Button is hilighted also when the window is inactive, like happens in Internet Explorer
- Built-in support for multiline tooltips
- Built-in basic support for menus
- Built-in support for owner draw menus (using BCMenu class)
- Built-in basic support for sounds
- Can be derived to create other button styles not supplied by default
- Full source code included!
- UNICODE compatible
- Cost-less implementation in existing applications



Click [here](#) to see a real-world application made using CButtonST.

## How to integrate CButtonST in your application

In your project include the following files:

- BtnST.h
- BtnST.cpp

Starting from version 3.5, CButtonST now supports menus created using the BCMenu class. This 3rd party class gives you the ability to show menus using the most recent visual styles as seen on the latest Microsoft products or even in Windows XP. Latest BCMenu version can be found [here](#).

To enable support for BCMenu the following two lines in BtnST.h must be enabled:

```
#define BTNST_USE_BCMENU
#include "BCMenu.h"
```

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

Also, the following files must be included in your project:

- BCMenu.h

- BCMenu.cpp

*Note: please note that when BCMenu support is enabled the parameters accepted by the **SetMenu** method are different!*

Starting from version 3.6, CButtonST can play sounds on particular button states.

To enable support for sound the following line in BtnST.h must be enabled:

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

```
#define BTNST_USE_SOUND
```

This gives access to the **SetSound** method.

### Create a CButtonST object statically

With dialog editor create a standard button called, for example, IDOK (you don't need to make it owner drawn) and create a member variable for this button:

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

```
CButtonST m_btnOk;
```

Now attach the button to CButtonST. For dialog-based applications, in your OnInitDialog:

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

```
// Call the base-class method
CDialog::OnInitDialog();

// Create the IDOK button
m_btnOk.SubclassDlgItem(IDOK, this);
```

Or in your DoDataExchange:

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

```
// Call the base method
CDialog::DoDataExchange(pDX);

// Create the IDOK button
DDX_Control(pDX, IDOK, m_btnOk);
```

### Create a CButtonST object dynamically

In your application, create a member variable for the button. Please note that this variable is a *pointer*:

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

```
CButtonST* m_pbtnOk;
```

Now create the button. For dialog-based applications, in your OnInitDialog:

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

```
// Call the base-class method
CDialog::OnInitDialog();
```

```
// Create the IDOK button
m_pbtnOk = new CButtonST;
m_pbtnOk->Create(_T("&Ok"), WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, <BR>CRect(10, 10, 200, 100), this, IDOK);
// Set the same font of the application
m_pbtnOk->SetFont(GetFont());
```

Remember to destroy the button or you will get a memory leak. This can be done, for example, in your class destructor:

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

```
if (m_pbtnOk) delete m_pbtnOk;
```

## Class methods

### SetIcon (using multi-size resources)

Assigns icons to the button.

Any previous icon or bitmap will be removed.

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

```
// Parameters:
//     [IN]   nIconIn
//             ID number of the icon resource to show when the mouse is over the<BR>// button. Pass NULL to remove any icon from the button.
//     [IN]   nCxDesiredIn
//             Specifies the width, in pixels, of the icon to load.
//     [IN]   nCyDesiredIn
//             Specifies the height, in pixels, of the icon to load.
//     [IN]   nIconOut
//             ID number of the icon resource to show when the mouse is outside <BR>// the button. Can be NULL.
//             If this parameter is the special value BTNST_AUTO_GRAY (cast to int) <BR>// the second icon will be automatically created
//             starting from nIconIn <BR>// and converted to grayscale.
//             If this parameter is the special value BTNST_AUTO_DARKER (cast <BR>// to int) the second icon will be automatically created 25%
//             <BR>// darker starting from nIconIn.
//     [IN]   nCxDesiredOut
//             Specifies the width, in pixels, of the icon to load.
//     [IN]   nCyDesiredOut
//             Specifies the height, in pixels, of the icon to load.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//     BTNST_INVALIDRESOURCE
//             Failed loading the specified resource.
//
DWORD SetIcon(int nIconIn, int nCxDesiredIn, int nCyDesiredIn, <BR>int nIconOut = NULL, int nCxDesiredOut = 0, int nCyDesiredOut = 0)
```

### SetIcon (using resources)

Assigns icons to the button.

Any previous icon or bitmap will be removed.

Hide Copy Code

```
// Parameters:
//      [IN]  nIconIn
//            ID number of the icon resource to show when the mouse is over the <BR>//      button.
//            Pass NULL to remove any icon from the button.
//      [IN]  nIconOut
//            ID number of the icon resource to show when the mouse is <BR>//      outside the button. Can be NULL.
//            If this parameter is the special value BTNST_AUTO_GRAY (cast to int) <BR>//      the second icon will be automatically created
starting from <BR>//      nIconIn and converted to grayscale. If this parameter is the <BR>//      special value BTNST_AUTO_DARKER (cast to
int) the second
//            icon will be automatically created 25% darker starting from nIconIn.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//
DWORD SetIcon(int nIconIn, int nIconOut = NULL)
```

#### **SetIcon** (using handles)

Assigns icons to the button.

Any previous icon or bitmap will be removed.

Hide Copy Code

```
// Parameters:
//      [IN]  hIconIn
//            Handle fo the icon to show when the mouse is over the button.
//            Pass NULL to remove any icon from the button.
//      [IN]  hIconOut
//            Handle to the icon to show when the mouse is outside the button. <BR>//      Can be NULL.
//            If this parameter is the special value BTNST_AUTO_GRAY the second
//            icon will be automatically created starting from hIconIn and <BR>//      converted to grayscale.
//            If this parameter is the special value BTNST_AUTO_DARKER the second
//            icon will be automatically created 25% darker starting from hIconIn.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//
DWORD SetIcon(HICON hIconIn, HICON hIconOut = NULL)
```

#### **SetBitmaps** (using resources)

Assigns bitmaps to the button.

Any previous icon or bitmap will be removed.

Hide Copy Code

```
// Parameters:
//      [IN]  nBitmapIn
//            ID number of the bitmap resource to show when the mouse is <BR>//            over the button.
//            Pass NULL to remove any bitmap from the button.
//      [IN]  crTransparentColorIn
//            Color (inside nBitmapIn) to be used as transparent color.
//      [IN]  nBitmapOut
//            ID number of the bitmap resource to show when the mouse <BR>//            is outside the button.
//            Can be NULL.
//      [IN]  crTransparentColorOut
//            Color (inside nBitmapOut) to be used as transparent color.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//      BTNST_FAILEDMASK
//            Failed creating mask bitmap.
//
DWORD SetBitmaps(int nBitmapIn, COLORREF crTransparentColorIn, <BR>int nBitmapOut = NULL, COLORREF crTransparentColorOut = 0)
```

**SetBitmaps** (using handles)

Assigns bitmaps to the button.

Any previous icon or bitmap will be removed.

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

```
// Parameters:
//      [IN]  hBitmapIn
//            Handle fo the bitmap to show when the mouse is over the button.
//            Pass NULL to remove any bitmap from the button.
//      [IN]  crTransparentColorIn
//            Color (inside hBitmapIn) to be used as transparent color.
//      [IN]  hBitmapOut
//            Handle to the bitmap to show when the mouse is outside the button.
//            Can be NULL.
//      [IN]  crTransparentColorOut
//            Color (inside hBitmapOut) to be used as transparent color.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//      BTNST_FAILEDMASK
//            Failed creating mask bitmap.
//
DWORD SetBitmaps(HBITMAP hBitmapIn, COLORREF crTransparentColorIn, <BR>HBITMAP hBitmapOut = NULL, COLORREF crTransparentColorOut = 0)
```



**SetFlat**

Sets the button to have a standard or flat style.

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

```
// Parameters:
//      [IN]   bFlat
//             If TRUE the button will have a flat style, else
//             will have a standard style.
//             By default, CButtonST buttons are flat.
//      [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//             Function executed successfully.
//
DWORD SetFlat(BOOL bFlat = TRUE, BOOL bRepaint = TRUE)
```

**SetAlign**

Sets the alignment type between icon/bitmap and text.

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

```
// Parameters:
//      [IN]   byAlign
//             Alignment type. Can be one of the following values:
//             ST_ALIGN_HORIZ      Icon/bitmap on the left, text on the right
//             ST_ALIGN_VERT       Icon/bitmap on the top, text on the bottom
//             ST_ALIGN_HORIZ_RIGHT Icon/bitmap on the right, text on the left
//             ST_ALIGN_OVERLAP    Icon/bitmap on the same space as text
//             By default, CButtonST buttons have ST_ALIGN_HORIZ alignment.
//      [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//             Function executed successfully.
//      BTNST_INVALIDALIGN
//             Alignment type not supported.
//
DWORD SetAlign(BYTE byAlign, BOOL bRepaint = TRUE)
```

**SetPressedStyle**

Sets the pressed style.

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

```
// Parameters:
//      [IN]   byStyle
//             Pressed style. Can be one of the following values:
//             BTNST_PRESSED_LEFTRIGHT Pressed style from left to right (as usual)
```

```
//      BTNST_PRESSED_TOPBOTTOM    Pressed style from top to bottom
//      By default, CButtonST buttons have BTNST_PRESSED_LEFTRIGHT style.
//      [IN]  bRepaint
//      If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//      Function executed successfully.
//      BTNST_INVALIDPRESSEDSTYLE
//      Pressed style not supported.
//
DWORD SetPressedStyle(BYTE byStyle, BOOL bRepaint = TRUE)
```

### SetCheck

Sets the state of the checkbox.

If the button is not a checkbox, this function has no meaning.

Hide Copy Code

```
// Parameters:
//      [IN]  nCheck
//      1 to check the checkbox.
//      0 to un-check the checkbox.
//      [IN]  bRepaint
//      If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//      Function executed successfully.
//
DWORD SetCheck(int nCheck, BOOL bRepaint = TRUE)
```

### GetCheck

Returns the current state of the checkbox.

If the button is not a checkbox, this function has no meaning.

Hide Copy Code

```
// Return value:
//      The current state of the checkbox.
//      1 if checked.
//      0 if not checked or the button is not a checkbox.
//
int GetCheck()
```

### SetDefaultColors

Sets all colors to a default value.

Hide Copy Code

```
// Parameters:
//      [IN]  bRepaint
//      If TRUE the control will be repainted.
```

```
//
// Return value:
//     BTNST_OK
//     Function executed successfully.
//
DWORD SetDefaultColors(BOOL bRepaint = TRUE)
```

### SetColor

Sets the color to use for a particular state.

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

```
// Parameters:
//     [IN]   byColorIndex
//             Index of the color to set. Can be one of the following values:
//             BTNST_COLOR_BK_IN      Background color when mouse is over the button
//             BTNST_COLOR_FG_IN      Text color when mouse is over the button
//             BTNST_COLOR_BK_OUT     Background color when mouse is outside the button
//             BTNST_COLOR_FG_OUT     Text color when mouse is outside the button
//             BTNST_COLOR_BK_FOCUS   Background color when the button is focused
//             BTNST_COLOR_FG_FOCUS   Text color when the button is focused
//     [IN]   crColor
//             New color.
//     [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//     BTNST_OK
//     Function executed successfully.
//     BTNST_INVALIDINDEX
//     Invalid color index.
//
DWORD SetColor(BYTE byColorIndex, COLORREF crColor, BOOL bRepaint = TRUE)
```

### GetColor

Returns the color used for a particular state.

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

```
// Parameters:
//     [IN]   byColorIndex
//             Index of the color to get.
//             See SetColor for the list of available colors.
//     [OUT]  crpColor
//             A pointer to a COLORREF that will receive the color.
//
// Return value:
//     BTNST_OK
//     Function executed successfully.
//     BTNST_INVALIDINDEX
//     Invalid color index.
```

```
//
DWORD GetColor(BYTE byColorIndex, COLORREF* crpColor)
```

### OffsetColor

This function applies an offset to the RGB components of the specified color.

This function can be seen as an easy way to make a color darker or lighter than its default value.

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

```
// Parameters:
//     [IN]   byColorIndex
//             Index of the color to set.
//             See SetColor for the list of available colors.
//     [IN]   shOffsetColor
//             A short value indicating the offset to apply to the color.
//             This value must be between -255 and 255.
//     [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//     BTNST_INVALIDINDEX
//             Invalid color index.
//     BTNST_BADPARAM
//             The specified offset is out of range.
//
DWORD OffsetColor(BYTE byColorIndex, short shOffset, BOOL bRepaint = TRUE)
```

### SetAlwaysTrack

Sets the highlight logic for the button.

Applies only to flat buttons.

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

```
// Parameters:
//     [IN]   bAlwaysTrack
//             If TRUE the button will be highlighted even if the window that owns it, is
//             not the active window.
//             If FALSE the button will be highlighted only if the window that owns it,
//             is the active window.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//
DWORD SetAlwaysTrack(BOOL bAlwaysTrack = TRUE)
```

### SetBtnCursor

Sets the cursor to be used when the mouse is over the button.

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

```
// Parameters:
//     [IN]   nCursorId
//             ID number of the cursor resource.
//             Pass NULL to remove a previously loaded cursor.
//     [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//     BTNST_INVALIDRESOURCE
//             Failed loading the specified resource.
//
DWORD SetBtnCursor(int nCursorId = NULL, BOOL bRepaint = TRUE)
```

### DrawBorder

Sets if the button border must be drawn.

Applies only to flat buttons.

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

```
// Parameters:
//     [IN]   bDrawBorder
//             If TRUE the border will be drawn.
//     [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//
DWORD DrawBorder(BOOL bDrawBorder = TRUE, BOOL bRepaint = TRUE)
```

### DrawFlatFocus

Sets if the focus rectangle must be drawn for flat buttons.

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

```
// Parameters:
//     [IN]   bDrawFlatFocus
//             If TRUE the focus rectangle will be drawn also for flat buttons.
//     [IN]   bRepaint
//             If TRUE the control will be repainted.
//
// Return value:
//     BTNST_OK
//             Function executed successfully.
//
DWORD DrawFlatFocus(BOOL bDrawFlatFocus, BOOL bRepaint = TRUE)
```

**SetTooltipText** (Using resource)

Sets the text to show in the button tooltip.

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

```
// Parameters:
//      [IN]   nText
//             ID number of the string resource containing the text to show.
//      [IN]   bActivate
//             If TRUE the tooltip will be created active.
//
void SetTooltipText(int nText, BOOL bActivate = TRUE)
```

**SetTooltipText**

Sets the text to show in the button tooltip.

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

```
// Parameters:
//      [IN]   lpszText
//             Pointer to a null-terminated string containing the text to show.
//      [IN]   bActivate
//             If TRUE the tooltip will be created active.
//
void SetTooltipText(LPCTSTR lpszText, BOOL bActivate = TRUE)
```

**EnableBalloonTooltip**

Enables the tooltip to be displayed using the balloon style.

This function must be called before any call to SetTooltipText is made.

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

```
// Return value:
//      BTNST_OK
//             Function executed successfully.
//
DWORD EnableBalloonTooltip()
```

**ActivateTooltip**

Enables or disables the button tooltip.

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

```
// Parameters:
//      [IN]   bActivate
//             If TRUE the tooltip will be activated.
//
void ActivateTooltip(BOOL bEnable = TRUE)
```

**GetDefault**

Returns if the button is the default button.

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

```
// Return value:
//     TRUE
//     The button is the default button.
//     FALSE
//     The button is not the default button.
//
BOOL GetDefault()
```

### DrawTransparent

Enables the transparent mode.

Note: this operation is not reversible.

DrawTransparent should be called just after the button is created.

Do not use transparent buttons until you really need it (you have a bitmapped background) since each transparent button makes a copy in memory of its background.

This may bring unnecessary memory use and execution overload.

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

```
// Parameters:
//     [IN]  bRepaint
//           If TRUE the control will be repainted.
//
void DrawTransparent(BOOL bRepaint = FALSE)
```

### SetURL

Sets the URL that will be opened when the button is clicked.

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

```
// Parameters:
//     [IN]  lpszURL
//           Pointer to a null-terminated string that contains the URL.
//           Pass NULL to removed any previously specified URL.
//
// Return value:
//     BTNST_OK
//           Function executed successfully.
//
DWORD SetURL(LPCTSTR lpszURL = NULL)
```

### SetMenu

Associates a menu to the button.

The menu will be displayed clicking the button.

This method is available only if **BTNST\_USE\_BCMENU** is not defined.

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

```
// Parameters:
//     [IN]  nMenu
//           ID number of the menu resource.
//           Pass NULL to remove any menu from the button.
```

```

//      [IN]   hParentWnd
//            Handle to the window that owns the menu.
//            This window receives all messages from the menu.
//      [IN]   bRepaint
//            If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//
//
DWORD SetMenu(UINT nMenu, HWND hParentWnd, BOOL bRepaint = TRUE)

```

### SetMenu

Associates a menu to the button.

The menu will be displayed clicking the button.

This method is available only if **BTNST\_USE\_BCMENU** is defined. The menu will be handled by the BCMenu class.

Hide Shrink ▲ Copy Code

```

// Parameters:
//      [IN]   nMenu
//            ID number of the menu resource.
//            Pass NULL to remove any menu from the button.
//      [IN]   hParentWnd
//            Handle to the window that owns the menu.
//            This window receives all messages from the menu.
//      [IN]   bWinXPStyle
//            If TRUE the menu will be displayed using the new Windows XP style.
//            If FALSE the menu will be displayed using the standard style.
//      [IN]   nToolBarID
//            Resource ID of the toolbar to be associated to the menu.
//      [IN]   sizeToolBarIcon
//            A CSize object indicating the size (in pixels) of each icon <BR>//            into the toolbar.
//            All icons into the toolbar must have the same size.
//      [IN]   crToolBarBk
//            A COLORREF value indicating the color to use as background <BR>//            for the icons into the toolbar.
//            This color will be used as the "transparent" color.
//      [IN]   bRepaint
//            If TRUE the control will be repainted.
//
// Return value:
//      BTNST_OK
//            Function executed successfully.
//      BTNST_INVALIDRESOURCE
//            Failed loading the specified resource.
//
//
DWORD SetMenu(UINT nMenu,
              HWND hParentWnd,
              BOOL bWinXPStyle = TRUE,

```



```

UINT nToolBarID = NULL,
CSize sizeToolBarIcon = CSize(16, 16),
COLORREF crToolBarBk = RGB(255, 0, 255),
BOOL bRepaint = TRUE)

```

### SetMenuCallback

Sets the callback message that will be sent to the specified window just before the menu associated to the button is displayed.

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

```

// Parameters:
//      [IN]  hWnd
//            Handle of the window that will receive the callback message.
//            Pass NULL to remove any previously specified callback message.
//      [IN]  nMessage
//            Callback message to send to window.
//      [IN]  lParam
//            A 32 bits user specified value that will be passed to the <BR>//            callback function.
//
// Remarks:
//      the callback function must be in the form:
//      LRESULT On_MenuCallback(WPARAM wParam, LPARAM lParam)
//      Where:
//      [IN]  wParam
//            If support for BCMenu is enabled: a pointer to BCMenu
//            else a HMENU handle to the menu that is being to be <BR>//            displayed.
//      [IN]  lParam
//            The 32 bits user specified value.
//
// Return value:
//      BTNST_OK
//      Function executed successfully.
//
DWORD SetMenuCallback(HWND hWnd, UINT nMessage, LPARAM lParam = 0)

```

### SizeToContent

Resizes the button to the same size of the image.

To get good results both the IN and OUT images should have the same size.

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

```
void SizeToContent()
```

### SetSound

Sets the sound that must be played on particular button states.

This method is available only if **BTNST\_USE\_SOUND** is defined.

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

```

// Parameters:
//      [IN]  lpszSound

```

```

//      A string that specifies the sound to play.
//      If hMod is NULL this string is interpreted as a filename, <BR>//      else it is interpreted as a resource identifier.
//      Pass NULL to remove any previously specified sound.
//      [IN] hMod
//      Handle to the executable file that contains the resource to <BR>//      be loaded.
//      This parameter must be NULL unless lpszSound specifies a <BR>//      resource identifier.
//      [IN] bPlayOnClick
//      TRUE if the sound must be played when the button is clicked.
//      FALSE if the sound must be played when the mouse is moved over <BR>//      the button.
//      [IN] bPlayAsync
//      TRUE if the sound must be played asynchronously.
//      FALSE if the sound must be played synchronously. The <BR>//      application takes control after the sound is completely played.
//
// Return value:
//      BTNST_OK
//      Function executed successfully.
//
DWORD SetSound(LPCTSTR lpszSound,
              HMODULE hMod = NULL,
              BOOL bPlayOnClick = FALSE,
              BOOL bPlayAsync = TRUE)

```

### OnDrawBackground

This function is called every time the button background needs to be painted.

If the button is in transparent mode this function will **NOT** be called.

This is a virtual function that can be rewritten in CButtonST-derived classes to produce a whole range of buttons not available by default.

Hide Copy Code

```

// Parameters:
//      [IN] pDC
//      Pointer to a CDC object that indicates the device context.
//      [IN] pRect
//      Pointer to a CRect object that indicates the bounds of the
//      area to be painted.
//
// Return value:
//      BTNST_OK
//      Function executed successfully.
//
virtual DWORD OnDrawBackground(CDC* pDC, CRect* pRect)

```

### OnDrawBorder

This function is called every time the button border needs to be painted.

This is a virtual function that can be rewritten in CButtonST-derived classes to produce a whole range of buttons not available by default.

Hide Copy Code

```
// Parameters:
//     [IN]   pDC
//           Pointer to a CDC object that indicates the device context.
//     [IN]   pRect
//           Pointer to a CRect object that indicates the bounds of the
//           area to be painted.
//
// Return value:
//     BTNST_OK
//           Function executed successfully.
//
virtual DWORD OnDrawBorder(CDC* pDC, CRect* pRect)
```

### GetVersionI

Returns the class version as a short value.

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

```
// Return value:
//     Class version. Divide by 10 to get actual version.
//
static short GetVersionI()
```

### GetVersionC

Returns the class version as a string value.

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

```
// Return value:
//     Pointer to a null-terminated string containig the class version.
//
static LPCTSTR GetVersionC()
```

## History

- v3.9 (03/March/2003)
  - Added support for Windows XP icons
  - Added support for multi-size icons
  - Added **BTNST\_AUTO\_DARKER** as a special value for second icon
  - Fixed the grayscale icon bug under Win9x/Me
  - Class is now indipendent from TTS\_BALLOON
- v3.8 (25/November/2002)
  - Added support for balloon tooltips
  - Added **EnableBalloonTooltip** method
  - OnDrawBorder** virtual method is now called also for non-flat buttons
  - OnDrawBackground** and **OnDrawBorder** now receive a correct **CRect\*** parameter

Fixed a little color bug

Correctly works under **MFC** 7.0

- v3.7 (22/July/2002)
  - Added **SetPressedStyle** method
  - Added **BTNST\_INVALIDPRESSEDSTYLE** return value
  - Added **ST\_ALIGN\_OVERLAP** align style
- v3.6 (09/July/2002)
  - Added **SetMenuCallback** method to give the ability to modify the associated menu just before is it displayed
  - Added basic support for sounds
  - Added **SetSound** method
  - Added **ResizeToContent** method
- v3.5 (18/April/2002)
  - Second icon can be automatically created in grayscale
  - Added **BTNST\_AUTO\_GRAY** as a special value for second icon
  - Bitmap is draw disabled if the button is disabled
  - Added support for owner draw menus (using BCMenu class)
  - Added an overloaded **SetMenu** method to support BCMenu class
  - Added **OffsetColor** method
  - Added support for **DDX\_Check** calls
- v3.4 (17/October/2001)
  - Added basic support for menus
  - Added **SetMenu** method
- v3.3 (20/September/2001)
  - Default button is now handled correctly
  - Removed some rarely used methods
  - Other optimizations
- v3.2 (14/June/2001)
  - Added support for bitmaps
  - Added **SetBitmaps** methods
- v2.6
  - Added an overloaded version of the **SetIcon** method
  - Added **ST\_ALIGN\_HORIZ\_RIGHT** flag to **SetAlign** function
  - Fixed a bug when used in **MFC** extension DLLs
  - Improved code for transparent buttons
- v2.5
  - Support for 16x16 32x32 and 48x48 icons
  - Buttons can be dinamically created
  - Added support for transparent buttons
  - Auto-detect default button (useful only for standard buttons)
  - Auto-detect icon's dimension
  - Added **DrawTransparent** method
  - Added **GetDefault** method
  - Modified **SetIcon** method

- v2.4
  - Added support for tooltips
  - Added **SetTooltipText**, **ActivateTooltip** members
  - The "Double-click bug" should be fixed
- v2.3
  - The class should now work from within a DLL
  - The "Spacebar-Bug" should be fixed
  - Added RedrawWindow() as the last line of **SetIcon** member
  - The focus rectangle is now the last thing drawn
  - The focus rectangle can now be drawn also for "flat" buttons
  - Added **SetFlatFocus**, **GetFlatFocus** members
  - Added **SetBtnCursor** member
  - Flat buttons can now work like in IE
- v2.2
  - Removed **SubclassDlgItem** member (this is transparent for the **user**)
  - Added **PreSubclassWindow** member (this allows DDX\_ calls)
  - Added **SetDefaultActiveFgColor**, **SetActiveFgColor**, **GetActiveFgColor** members
  - Added **SetDefaultActiveBgColor**, **SetActiveBgColor**, **GetActiveBgColor** members
  - Added **SetDefaultInactiveFgColor**, **SetInactiveFgColor**, **GetInactiveFgColor** members
  - Added **SetDefaultInactiveBgColor**, **SetInactiveBgColor**, **GetInactiveBgColor** members
  - When the mouse is over a button the focus now remains to the **control** that owns it!
  - The flat buttons now work properly also in windows not derived from CDialog!
  - A memory DC (CMemDC) is used to draw the button. This should speeds up the graphic operations.
- v2.1
  - Support for two icons
  - Modified **SetIcon** member
  - Added **SetShowText/GetShowText** members
  - Fixed a bug dealing with the left mouse button
  - Little optimizations
- v2.0
  - Changed the class name for name convention
  - Support for 256 colors icons
  - Removed a stupid memory leak!
  - Removed support for CImagelists
  - Documentation in HTML format
- ST\_CButton v1.1
  - Some minor changes
- ST\_CButton v1.0
  - First release

## Remarks

The demo application shows nearly all the features of the **CButtonST** class.

**CButtonST** architecture makes possible to produce a whole range of buttons not available by default. If someone implements new button styles I will be happy to include his code in the next **CButtonST** demo application.

## Thanks

Thanks very much to the dozens of **users** that are using **CButtonST** in their applications.

Thanks also to all the people that finds and fixes bugs. **Thanks!**

If possible, please send a screenshot of your application where CButtonST is used. These screenshots will be collected for personal interest.

## Disclaimer

THE SOFTWARE AND THE ACCOMPANYING FILES ARE DISTRIBUTED "AS IS" AND WITHOUT ANY WARRANTIES WHETHER EXPRESSED OR IMPLIED. NO REPOSNSIBILITIES FOR POSSIBLE DAMAGES OR EVEN FUNCTIONALITY CAN BE TAKEN. THE **USER** MUST ASSUME THE ENTIRE RISK OF USING THIS SOFTWARE.

## License

This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below.

A list of licenses authors might use can be found [here](#)

## Share



## About the Author

**Davide Calabro**

Web Developer

Italy 



No Biography provided

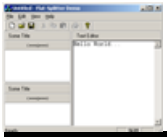
## You may also be interested in...



[Davide Calabro's CButtonST class port to WTL](#)



[IoT Reference Implementation: How to Build a Transportation in a Box Solution](#)



[A Flat Splitter Window](#)



[10 Ways to Boost COBOL Application Development](#)



[Intel® Quark™ SE Microcontroller C1000 Developer Kit - Accelerometer Tutorial](#)



[SAPrefs - Netscape-like Preferences Dialog](#)

## Comments and Discussions

Add a Comment or Question



Search Comments

Go

Spacing

Relaxed ▼

Layout

Open All ▼

Per page

25 ▼

Update

First Prev Next

## **Compiling VS2015 x64 bit**

 **Andrew Truckle****26-Aug-15 6:13**

Hi

I can't compile this now in x64 in VS2015. It is giving warnings about casting from int to HICON and/or HBITMAP.

Please help!

Andrew

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (1 vote)

## **API GetWindowText**

 **SebAub****8-Oct-14 10:45**

Hi,

I have to analyze the content of the button in another process. So I have to use the Windows API to acquire the title. It seems to correspond to the original text rather than the current. Did I miss something?

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## **Number of the Buttons**

 **Member 10832967****8-Jun-14 14:03**

First of all, thanks you very much! But I have a few questions, for example, are the numbers of the Buttons limited?

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (1 vote)

## **My vote of 1**

 **lanmanck****9-May-14 10:54**

fliker!!!

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## **Abnormal focus behaviour**

 **EricMic****25-Feb-13 22:23**

Hi,



Thank you very much for this !

I however had to change some code :

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

```
void CButtonST::OnEnable(BOOL bEnable)
{
    CButton::OnEnable(bEnable);

    if (bEnable == FALSE)
    {
        //CWnd* pWnd = GetParent()->GetNextDlgTabItem(this);
        //if (pWnd)
        //    pWnd->SetFocus();
        //else
        //    GetParent()->SetFocus();
        CancelHover();
    } // if
} // End of OnEnable
```

I commented the SetFocus part because it bring some abnormal button behavior. I believe setting the focus to the next tab item might be done only if the disabled button is the current control with focus.

thx.

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## My vote of 4

 shetkarabhijeet

11-Dec-12 14:05

thanks for this!

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (1 vote)

## Grayscale not working correctly on 32 bit alpha - (fixed)

 namezero111111

10-Oct-12 3:25

Hello folks,

I had a problem with this class that grayscale icons were not correctly created because the CreateGrayScaleIcon() method sometimes created black pixels.  
Here's the solution (copied from [here](#)):

In BtnST.h:

To protected section, add:

Hide Copy Code

```
HICON CreateGrayscaleIcon(HICON hIcon, COLORREF* pPalette);
```

In BtnST.cpp replace the whole CButtonST::CreateGrayscaleIcon() method with:

Hide Expand ▼ Copy Code

```
HICON CButtonST::CreateGrayscaleIcon(HICON hIcon, COLORREF* pPalette)
{
    if (hIcon == NULL)
    {
        return NULL;
    }

    HDC hdc = ::GetDC(NULL);

    HICON      hGrayIcon      = NULL;
    ICONINFO   icInfo         = { 0 };
    ICONINFO   icGrayInfo     = { 0 };
    LPDWORD    lpBits         = NULL;
    LPBYTE      lpBitsPtr     = NULL;
    SIZE       sz;
    DWORD      c1 = 0;
    BITMAPINFO bmpInfo        = { 0 };
    bmpInfo.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);

    if (::GetIconInfo(hIcon, &icInfo))
    {
        if (::GetDIBits(hdc, icInfo.hbmColor, 0, 0, NULL, &bmpInfo, DIB_RGB_COLORS) != 0)
        {
            bmpInfo.bmiHeader.biCompression = BI_RGB;

            sz.cx = bmpInfo.bmiHeader.biWidth;
            sz.cy = bmpInfo.bmiHeader.biHeight;
            c1 = sz.cx * sz.cy;
        }
    }
}
```

Reply · Email · View Thread · Permalink · Bookmark

1.00/5 (1 vote)

 **Re: compile warning.**

 **Mark Potter**

**6-Sep-12 0:53**

I replied to an [earlier thread](#) on this subject, but with the organization of this discussion board I think no one would notice a reply to an 8-year old message, so to repeat myself...

When building this code, one gets a compiler warning:

BtnST.cpp(19) : warning C4005: 'BS\_TYEMASK' : macro redefinition c:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK\Include\WinUser.h(8749) : see previous definition of 'BS\_TYEMASK'

To which Nirav replied:

I have also encountered the same warning.

Is it because of the MS-Platform SDK that we have installed, as the warning is because the constant is already defined in the WinUser.h in the Platform SDK.

Is this inevitable, or there's something wrong that we're doing?

To which I reply and ask:

The problem isn't just that it is being defined again, but that it is being defined again with a different value. In BtnST.cpp **BS\_TYEMASK** is defined as **SS\_TYEMASK**, on my current development system **SS\_TYEMASK** is defined as **0x0000001FL**; however before this line **BS\_TYEMASK** was **0x0000000FL**. So the general questions are:

- Should it be **0x1F** or **0x0F**?
- Should it use the SDK's value if defined, but **0x1F** if not?
- Why **SS\_TYEMASK** (a Static Control Style value) for **BS\_TYEMASK** (a Button Style value)?

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

## My vote of 5

 Jignesh9969

15-Mar-12 20:39

This code use in my application and its works fine.

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

2.00/5 (1 vote)

## Get color of the button

 Member 8544673

10-Feb-12 13:42

How to get the color of the clicked button.... i am new to mfc could u please explain the use of `GetColor(BYTE byColorIndex, COLORREF* crpColor)` with an example.

Thank You

*modified 10-Feb-12 1:06am.*

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (2 votes)

## GetCheck() error

 yl\_b

1-Dec-11 10:18

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

```
<code></code>
```

ON\_BN\_CLICKED(ID\_CHECK\_GROUP7, OnCheckRec7),in OnCheckRec7,can not get the CheckBox state!

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

5.00/5 (1 vote)

## problem with repaint [modified]

 Member 1913146

17-Aug-11 20:52

hello thanks alot

I think if you add this function to (BtnST.cpp) ,when paint

(image button) has been problem , may be solve with this code .

for example if before create (image button) top of it have been a window , after create (button) and move window to other side for see (button) , (button) colored with window color and this problem stay until lifetime. 😊

excuse me for bad english

hessamini@gmail.com

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

```
//BtnST.cpp
LRESULT CButtonST::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_NCPAINT:
```

```
    {
        m_bmpBk.DeleteObject();
        m_dcBk.DeleteDC();
        m_dcBk.m_hDC=NULL;
    }
}
return CButton::WindowProc(message, wParam, lParam);
}
void CButtonST::OnLButtonDown(UINT nFlags, CPoint point)
{
    GetParent()->Invalidate();
    CButton::OnLButtonDown(nFlags, point);
}
```

salam

*modified on Saturday, August 20, 2011 5:00 AM*

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## My vote of 4

 lzmthm

21-Jul-11 22:38

so goog for me !

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## Get24BitPixels problem

 jph.torcy

14-Apr-11 23:13

Hi,

There is just a little limitation in the Get24BitPixels() function (Vander Nunes function).

It is written :

```
wBmpWidth -= (wBmpWidth%4);           // width is 4 byte boundary aligned.
```

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

It should be :

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

```
dwBmpWidth = (dwBmpWidth+3) & ~3;    // 4 bytes padding
```

example:

The width of my image is 279 pixels

In the formula you have a width result equals to 276, it should be 280.

The formula only works fine when the width is a multiply of 4.

Bye,

Jean-Philippe

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

### My vote of 5

 Joe Sonderegger

9-Sep-10 0:43

It works

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (1 vote)

### My vote of 5

 yangpanpan

7-Sep-10 15:33

This control is useful for me in decorating my program, and it is also powerful.

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

### how to set font

 batsword

5-Sep-10 11:17

how to set font?thank you

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

### Re: how to set font

 batsword

9-Sep-10 10:34

sorry it is so easy to set font,now ,the problem is ,how is gif?

3q very much 😊

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

## My vote of 5

 **xiamec**

**31-Aug-10 9:05**

很好很强大，我做课程设计要用！  
提个建议，能不能设定按钮的形状，比如随意设置形状为圆形或方形？？  
Thank you!

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

## Re: My vote of 5

 **batsword**

9-Sep-10 10:32

大哥，老外看不懂你说的  
这个你要做任意形状就用透明的ICON或者指定一种底色的BMP  
国内也有任意形状的代码

这个BUTTON的缺点在两个按钮覆盖后面的显示不了



[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

---

## How to change the show time of the function SetTooltipText()? [modified]

 **VincentWong800**

**22-Aug-10 0:12**

This class is useful and wonderful! thanks!

I want to know how can i to change the show time of the SetTooltipText()?  
I mean is from mouse-in to display the text.

It's about 1 second. I want it no delay.

Thanks a lot!

---

*modified on Saturday, August 21, 2010 3:35 PM*

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

2.00/5 (1 vote)

## Call stack corruption?

 namezero111111

8-Aug-10 11:36

Hello folks,

I have long used this class in VCC6, and have recently ported a project to VS2k10.

However, I got call stack corruptions on every dialog in the application that used this class.


For this to occur, I merely have to add a button instance of CShadeButtonST or CButtonST at the end of the dialog.

Upon further investigation, I found that adding a dummy `int[4]` at the end of this class' definition masks the problem with the call stack corruption.

Has anyone else had this problem and found the true cause of it?

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## Re: Call stack corruption?

 namezero111111

9-Aug-10 10:53

For everyone here, the problem is in BalloonHelp.h (although I don't know if the balloon help comes with this project or was added afterwards).

The problem in there is a `"#pragma pack(push,1)"` that is never restored at the end of the header and may cause havoc or strange program behavior elsewhere where the file might be included. By deleting that line the problem can be fixed.

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

## thanks !

 Nice Gom

21-Jun-10 9:25

it's simple but powerful.

[Reply](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)

1.00/5 (1 vote)

## How To Change the Button Text?

 a1rex2003


12-May-10 6:18

I would greatly appreciate if someone could tell me how to change the button text. I have buttons without text and I would like to change/ add text dynamically.  
Thank you for your help!

[Reply](#) · [Email](#) · [View Thread](#) · [Permalink](#) · [Bookmark](#)



Last Visit: 7-Feb-17 9:49    Last Update: 7-Feb-17 4:49

[Refresh](#)[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [Next »](#)[General](#) [News](#) [Suggestion](#) [Question](#) [Bug](#) [Answer](#) [Joke](#) [Praise](#) [Rant](#) [Admin](#)[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)  
Web02 | 2.8.170203.1 | Last Updated 29 Mar 2003 請選取語言 ▼Layout: [fixed](#) | [fluid](#)Article Copyright 1999 by Davide Calabro  
Everything else Copyright © [CodeProject](#), 1999-2017

