

MRCEXT4.DLL (version 1.0)

Introduction

MRCEXT.DLL was originally an MFC 3.1+ extension DLL and is now available for MFC4.0. The main functionality it provides are sizeable control bars, both when floating and docked, similar to the Visual C++ 2.0 IDE. It also includes classes for 256-color bitmaps, playing AVI files, and bitmap buttons.

Full source code, and an example application, DOCKTEST is included.

History

Since I uploaded the first MRCEXT sample to Compuserve at the beginning of the year, there has been considerable interest in the source code. Micro Focus have now kindly agreed to make this available.

I don't claim the implementation is perfect, but at least now you have the source to code, you can look at the problems.

Licensing

Micro Focus intend this code to be Freeware similar to that proposed by the Gnu freeware license. The restriction that you do not charge for your software does not apply, provided that MRCEXT is not a significant part of your application. (ie you can't just bundle MRCEXT with a 5-line app, and sell it for 100 bucks a copy), nor does the section that you must supply source code on request.

You must accredit Micro Focus Inc for in the About Box and/or banner of your application.

You are free to modify the source code and distribute modified source code, provided you do not remove our copyright notices from the source code.

There is NO WARRANTY. Micro Focus make no commitment to maintain or support this software. It is supplied strictly as is, WITHOUT WARRANTY. You may of course, supply your own warranty.

There is nothing to stop you using MRCEXT in commercial applications, provided you accredit Micro Focus Inc.

Other Notes:

What's new over previous versions ?

I've attempted to improve the layout and drag/docking algorithms to preserve the sizes of control bars as they are dragged around. Most of the NT problems have now been fixed - floating control bars now even have a system menu.

What doesn't work ?

CMRCSIZEToolBar cannot be dragged. I cannot find a simple solution to this, and the hope is that with MFC4.0 people will take the direction of using MFC's CToolBar, and implementing the configuration support (I believe there are several examples of this already on Compuserve). If there is enough demand I may supply this as part of MRCEXT in the near future. For now CMRCSIZEToolBar remains, but I suspect it'll be fixed.

Testing

Micro Focus intend the sizeable docking functionality to be part of their forthcoming Visual Object COBOL product, so I think the code is pretty stable here, though I cannot claim this is the best way to implement the functionality. It was primarily designed to run on Windows'95, but has been casually tried on NT 3.51, and there are some known problems (mainly cosmetic as MRCEXT only draws in the 3D Windows'95 style). Generally it works ok.

Reference Guide

Hopefully most of this documentation is accurate. I've not changed it much from the MFC3.1 version.

MRCEXT includes the following advertised classes:

CMRCSizeControlBar	Sizeable control bar.
CMRCSizeToolBar	Sizeable tool bar - largely compatible with CToolBar.
CMRCSizeDialogBar	Sizeable dialog bar - compatible with CDialogBar.
CMRCFrameWndSizeDock	CFrameWnd class that supports sizeable control bars.
CMRCMDIFrameWndSizeDock	CMDIFrameWnd derived class that supports sizeable control bars.
CMRCColorLabel	Static control capable of displaying either colored text in any font or a bitmap (including limited support for 256-color bitmaps).
CMRCAnimateCtrl	Controls that plays .AVI files using Video for Windows MCI.
CMRCBitmapButton	Bitmap button where you just specify a single bitmap, and the pushed state is drawn automatically.

Using MRCEXT

Your project should link dynamically to MFC40(D).DLL ie check the "Use MFC in a shared DLL (mfc40(d).dll)" radio button in MFC App Wizard when creating your project.

You need to include the header file, "mrcext.h". This can usually be included in the pre-compiled headers. It contains #pragma's to link either with MRCEXT4D.LIB for debugging or MRCEXT4.LIB for release, so you don't have to make any changes to your makefile. In the unlikely event that you need to specify the link line explicitly, these pragma's can be disabled, by #defining _MRC_NOFORCELIBS, before including mrcext.h. Using it should be pretty much the same as using any MFC40(D).DLL. You may wish to copy MRCEXT4(D).DLL to your windows directory, and MRCEXT4(D).LIB to your \msvc\lib directory if you are not intending to change the code in MRCEXT.

The sizeable toolbar class is based around the Windows'95 common control. If you do not include the common control header file "afxcmn.h", the definitions of this class are not defined.

class CMRCSizeControlBar : public CControlBar

This class provides the basic behaviour for control bars that can be resized, both when docked and floating. The resizing is free-form, ie there are no discrete steps, and the bar has difference sizes, depending on whether it is docked vertically, horizontally or floating.

To implement this feature in your application you must:

1. Derive you main frame window (usually CMainFrame) from either CMRCFrameWndSizeDock (instead of CFrameWnd, for SDI apps), or CMRCFrameWndSizeDock (instead of CMDIFrameWnd, for MDI apps).
2. Derive any specific control bars you need from CMRCSizeControlBar, and/or make use of CMRCSizeDialogBar and CMRCSizeToolBar.

You may combine both sizeable (CMRCSizeControlBar-derived) and non-sizeable (CControlBar-derived) controls bars in the same application. To best get to grips with writing to these classes, read the implementation summary and look at the example. In general the steps you'll go through are pretty much the same as with MFC's CControlBar.

1. Construct the MFC CMRCSize...Bar object, either using new, or on the stack. Note that the constructor takes a parameter. If you wish to create an object on the stack, then you may find it easier to use the SetSizeDockStyle() function, immediately after constructing the object, than specifying a parameter to the constructor.
2. Use Create() to create the bar.
3. Use EnableDocking().
4. Float or Dock the bar. Generally, CControlBars don't work with much sense until you done this step.

One feature you may find useful in deriving from CMRCSizeControlBar, is that WM_COMMAND notifications to be handled locally inside the derived class, as well as via the standard routing mechanism. E.g. if you have a button control on a CMRCSizeControlBar derived class, you can implement the ON_COMMAND handler either in your derived class, OR, in the view, frame, etc. as with normal CControlBars.

Construction

CMRCSizeControlBar()	constructs a CMRCSizeControlBar
----------------------	---------------------------------

Attributes

m_FloatSize	Size of window when floating
m_HorzDockSize	Size of window when docked horizontally (to top or bottom of frame)
m_VertDockSize	Size of window when docked vertically (to left or right of frame)
IsProbablyFloating()	Indicates if control bar is floating

Operations

Create()	Creates the underlying windows object
SetSizeDockStyle()	Sets/modifies the style of the control bar
EnableDocking()	Enables docking

Overridables

OnSizedOrDocked()	Called when the control bar is sized or docked/undocked
-------------------	---

CMRCSizeControlBar:: CMRCSizeControlBar

CMRCSizeControlBar(int *Style* = 0);

Constructs a CMRCSizeControlBar object.

Parameters:

Style specifies additional behaviour flags for the object:

<i>SZBARF_DESTROY_ON_CLOSE</i>	When set, clicking the close button on the floating frame causes the control bar to be closed and destroyed. Normally, the control bar would just be hidden.
<i>SZBARF_AUTOTIDY</i>	MFC CControlBar objects cannot be deleted from PostNCDestroy(). Normally CControlBar objects are allocated within another object (e.g. CMainFrame) so the destructors are called when that object is deleted. If you allocate control bars dynamically (as is often useful with sizeable control bars), then you will get memory leaks. Setting this flag causes all allocated CMRCSizeControlBars to be tracked, and deleted automatically when the frame window is destroyed.
<i>SZBARF_STDMOUSECLICKS</i>	The control bar will respond to standard mouse events, such as double click on the title bar to dock, pop up a context menu to hide or enable/disable docking.
<i>SZBARF_DLGAUTOSIZE</i>	Use in conjunction with dialog bars. The gadgets on the dialog bar will resize to fit the new window. This is a simple linear mapping based on the original sizes & positions in the dialog resource, but it does work surprisingly well.
<i>SZBARF_ALLOW_MDI_FLOAT</i>	When set, the context menu will allow the user to float the bar as an MDI child window. The MDI floating code does seem to work, but there are some problems, so use with care !

CMRCSizeControlBar::Create()

```
BOOL Create(CWnd * pParent, LPCTSTR lpszTitle, UINT nID = 1, DWORD  
dwStyle = WS_CHILD | WS_VISIBLE | CBRs_BOTTOM,  
const RECT & rect = CFrameWnd::rectDefault);
```

Creates the control bar. You should use the WS_VISIBLE style when creating your toolbar, otherwise the initially size may be incorrect.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters

<i>pParent</i>	Points to the parent frame window of the control. Should be derived from either CMRCMDIFrameWndSizeDock, or CMRCFrameWndSizeDock.
<i>lpszTitle</i>	Specifies the control's caption - used in the floating frame window, when the bar is undocked.
<i>nID</i>	Specifies control's ID
<i>dwStyle</i>	Style of the control
<i>rect</i>	Initial size and position of the control.

CMRCSizeControlBar::SetSizeDockStyle()

void SetSizeDockStyle(DWORD dwStyle);

Sets the internal style flag for the control bar. Use this as an alternative to specifying a parameter on the constructor (except where specifying SZBARF_AUTOTIDY), and call it immediately after construction, but before Create().

Parameters

dwStyle

Style for the control bar. See CSizeControlBar() constructor for details of possible values
Points to the parent frame window of the control.

CMRCSizeControlBar::IsProbablyFloating()

BOOL IsProbablyFloating();

Return value:

TRUE if the control bar is floating (inside a floating frame window). Unlike CControlBar::IsFloating(), it returns TRUE if the bar is in transition between floating and non-floating. This function proves more useful inside the OnSizedOrDocked() overridable.

CSize CMRCSizeControlBar::m_FloatSize

CSize CMRCSizeControlBar::m_HorzDockSize

CSize CMRCSizeControlBar::m_VertDockSize

These 3 attributes determine the size of the bar when it is floating, docked horizontally or vertically. They are update when the bar is resized by the user. The main use of exposing them as public is to allow their sizes to be set explicitly immediately after creation, to determine how much space they will occupy when docked/undocked.

CMRCSizeControlBar::EnableDocking()

void EnableDocking(DWORD dwDockStyle);

This function is equivalent to CControlBar::EnableDocking(), but applies to sizeable control bars. Note that this function is NOT virtual, so you must take care that you do not inadvertently call CControlBar::EnableDocking() on a CSizeControlBar object.

CMRCSizeControlBar::OnSizedOrDocked()

virtual void OnSizedOrDocked(int *cx*, int *cy*, BOOL *bFloating*, int *flags*);

This overridable is called whenever the control bar is sized or docked/undocked. It is probably the easiest place to respond to resizing the control.

Parameters

<i>cx</i>	New width of the control bar.
<i>cy</i>	New height of the control bar.
<i>bFloating</i>	TRUE if the bar is now floating.
<i>flags</i>	Reserved.

CMRCSIZEDialogBar : public CSizeControlBar

This class implements a sizeable dialog bar, that should be entirely compatible with CDialogBar. If you specify the SZBARF_DLGAUTOSIZE on construction, the controls on the dialog bar will be moved and sized to fit within the current size of the dialog bar. This provides a very simple, yet effective form of geometry management. If you don not specify this style, then there is no geometry management facilities, so the controls within the dialog bar remain fixed, and will simply be clipped if the bar is resized too small. You may or course, override OnSizedOrDocked to move the controls around.

This control should be 100% compatible with CDialogBar. Please see the MFC documentation on CDialogBar for details. No further documentation is included here.

CMRCSIZEToolBar : public CSizeControlBar

This class implements a sizeable toolbar that is largely compatible with CToolBar. It responds to the standard MFC ON_UPDATE_COMMAND_UI handling and routing, tooltips and prompting, and provides the benefits of the common control, such as configuration.

This class is actually implemented as a Windows'95 CToolBarCtrl placed on a CSizeControlBar. Due to restrictions in the underlying common control it is not (and don't thin it ever can be) 100% compatible with the CToolBar class. However, enough compatibility is provided for AppWizard generated apps to work with minimal change (i.e. altering CMainFrame::m_wndToolBar to be a CMRCSIZEToolBar).

Some additional functionality has been added to utilise the Windows'95 common control configuration. To use this class, you must include the Visual C++ 2.1 header file "afxcmn.h" before including "mrccext.h".

Construction

CMRCSIZEToolBar()	constructor
-------------------	-------------

Attributes

GetToolBarCtrl()	Returns the underlying CToolBarCtrl
------------------	-------------------------------------

Operations

Create()	Creates windows objects for the toolbar
LoadBitmap()	Loads bitmap for the toolbar images
SetSizes()	Sets sizes of the buttons/image
SetButtons()	Sets command ID's of the buttons

CMRCSIZEToolBar::GetToolBarCtrl()

CToolBarCtrl * GetToolBarCtrl();

Return Value:

Returns a pointer to the underlying CToolBarCtrl object (Windows'95 common control). You should manipulate this object explicitly to obtain more refined toolbar behaviour - e.g. setting buttons, etc.

CMRCSIZEToolBar::Create()

```
BOOL Create(CWnd *pParent, DWORD dwStyle = WS_CHILD |  
           WS_VISIBLE | CBRS_TOP, UINT nID = AFX_IDW_TOOLBAR,  
           LPRECT pRect = NULL);
```

Creates the underlying windows objects for the toolbar.

Parameters:

<i>pParent</i>	Parent window
<i>dwStyle</i>	Style.
<i>nID</i>	ID.
<i>pRect</i>	Points to a rectangle for the initial size of the toolbar. If NULL, the a rectangle big enough for 1 row of standard sized toolbar buttons is used.

Return Value:

TRUE if successful, FALSE otherwise.

CMRCSIZEToolBar::LoadBitmap()

BOOL LoadBitmap(LPCTSTR lpszResourceName);

BOOL LoadBitmap(UINT nID);

Sets the underlying bitmap to be used for the images on the toolbar buttons.

Parameters:

<i>nID</i>	Resource ID of bitmap to load
<i>lpszResourceName</i>	Resource string of bitmap to load

Return Value:

TRUE if successful, FALSE otherwise.

Non-Compatibility with CToolBar

1. LoadBitmap() may only be called once for CMRCSIZEToolBar.
 2. If you wish to use a toolbar with non-standard sizes, you should call SetSizes() before LoadBitmap().
- These are underlying restrictions in CToolBarCtrl.

CMRCSIZEToolBar::SetSizes()

void SetSizes(SIZE sizeButton, SIZE sizeImage);

Sets the sizes of the images in the underlying bitmap, and of the buttons in the toolbar.

Parameters:

<i>sizeButton</i>	Sizes of the buttons in the underlying toolbar.
<i>sizeImage</i>	Sizes of images in the underlying toolbar

Non-Compatibility with CToolBar

1. SetSizes() must be called after Create()
 2. SetSizes() should be called before LoadBitmap() to have any effect on the image sizes.
- These are underlying restrictions in CToolBarCtrl.

CMRCSizeToolBar::SetButtons()

BOOL SetButtons(UINT * pButtons, int nButtons);

Sets the command id's for the buttons in the toolbar. This function is equivalent to CToolBar::SetButtons().

Parameters:

<i>pButtons</i>	Points to an array of button command Ids.
<i>nButtons</i>	Number of buttons in the above array.

Non-Compatibility with CToolBar

1. SetButtons() must be called after Create() and LoadBitmap().

CMRCSIZEToolBar::SetBitmapIds()

BOOL SetBitmapIds(UINT * pButtons, int nButtons);

This function enables the CToolBarCtrl configuration features. It specifies an array of command IDs that the images in the bitmap represent. Call this function before Create(), and SetButtons() and the toolbar will be created with the CCS_ADJUSTABLE style. SetButtons() then uses this specified array to determine which bitmap images to use, and the common control customization dialog uses it to determine which buttons could possibly be on the toolbar. The idea is that you call SetBitmapIds() to set the ID's that correspond to the images in the bitmap, and SetButtons() to determine the command ID's that are actually on the toolbar.

The common control customization allows buttons you to drag buttons, by holding the shift button and clicking and dragging the button, either to a new position on the toolbar, or off it (to remove it from the toolbar). Double-clicking displays the configuration dialog. In the latest version of Windows'95 I have there seem to be some problems when using this dialog.

Parameters:

pButtons

Points to an array of button command Ids.

nButtons

Number of buttons in the above array.

class CMRCFrameWndSizeDock : public CFrameWnd

class CMDIMRCFrameWndSizeDock : public CMDIFrameWnd

These 2 classes provide the frame window functionality necessary to support sizeable control bars. If you wish to use sizeable control bars, you should derive your main frame class (CMainFrame) from them (CMRCMDIFrameWndSizeDock if an MDI app, CMRCFrameWndSizeDock if an SDI app). The two classes have almost identical functions.

Construction

CMRCFrameWndSizeDock()	constructor
CMRCMDIFrameWndSizeDock()	constructor

Operations

ArrangeFloatingBars()	Arranges the floating control bars
TileDockedBars()	Tiles the docked sizeable control bars
EnableDocking()	Enables docking
LoadSizeBarState()	Loads the current state of the control bars, including sizes
SaveSizeBarState()	Saves the current state of the control bars, including size.
FloatControlBarInMDIChild()	Floats the specified control bar in an MDI child window, (instead of a mini-frame window). (CMRCMDIFrameWndSizeDock only).

CMRCFrameWndSizeDock::ArrangeFloatingBars()

void ArrangeFloatingBars(*dwOrient*);

Arranges the floating control bars within a frame.

Parameters:

dwOrient specifies the orientation of the arrangement. It can be one of:

- CBRS_ARRANGE_TOPLEFT
- CBRS_ARRANGE_TOPRIGHT
- CBRS_ARRANGE_BOTTOMLEFT
- CBRS_ARRANGE_BOTTOMRIGHT

CMRCFrameWndSizeDock::TileDockedBars()

void TileDockedBars(*dwDockStyle*);

Tiles the sizeable control bars within a frame so they take up equal size. You should call RecalcLayout() after calling this function.

Parameters:

dwDockStyle specifies which sides of the frame window, the control bars should be tiled for. It can be a combination of CBRs_ALIGN_TOP, CBRs_ALIGN_BOTTOM, CBRs_ALIGN_LEFT, CBRs_ALIGN_RIGHT or CBRs_ALIGN_ANY.

CMRCFrameWndSizeDock::LoadSizeBarState()

void LoadSizeBarState(LPCTSTR *pszProfileName*);

Loads the saved state of the bars from the Registry, including sizes. This function calls CFrameWnd::LoadBarState() to do most of it's work.

Note: To use this function, you must be using the Win32 Registry and NOT a .INI file to save your profile information. You should call CWinApp::SetRegistryKey() in InitInstance() of your application, before using this function.

Parameters:

pszProfileName specifies the key to use within the registry.

CMRCFrameWndSizeDock::SaveSizeBarState()

void SaveSizeBarState(LPCTSTR *pszProfileName*);

Saves the state of the bars to the Registry, including sizes. This function calls CFrameWnd::SaveBarState() to do most of it's work.

Notes:

1. This function destroys any windows constructed with the SZBARF_AUTOTIDY flag. CControlBars created with this flag will not generally be around next time in.
2. Positions and sizes are saved in a binary format, based on the ID of the control bars. You should ensure all your CControlBars have unique ID's.
3. See LoadSizeBarState regarding using the Win32 registry.

Parameters:

pszProfileName specifies the key to use within the registry.

CMR CMDIFrameWndSizeDock::FloatControlBarInMDIChild

void FloatControlBarInMDIChildEnableDocking (CControlBar* pBar, CPoint point, DWORD dwStyle = CBRS_ALIGN_TOP);

void UnFloatInMDIChild(CControlBar* pBar, CPoint point, DWORD dwStyle = CBRS_ALIGN_TOP);

This function floats the specified control bar as an MDI Child window (as opposed to a mini-frame window) in the MDI client area (similar to the docking and non-docking views in the MS Visual C++ IDE).

Parameters:

pBar	Points to the control bar to be floated.
point,	Position in screen co-ordinates of the top-left corner of the control bar
dwStyle	Orientation of the control bar within the parent window. Provided for consistency.

CMRCFrameWndSizeDock::EnableDocking()

void EnableDocking(*dwDockStyle*);

This function is equivalent to CFrameWnd::EnableDocking(), but should be called if your frame window is to support sizeable docking. NB: CFrameWnd::EnableDocking() is NOT virtual, so you must ensure you do not inadvertently call it for a CMRCFrameWndSizeDock or CMRCMDIFrameWndSizeDock object.

Parameters:

dwDockStyle specifies which sides of the frame window control bars can be docked to. It can be a combination of CBRS_ALIGN_TOP, CBRS_ALIGN_BOTTOM, CBRS_ALIGN_LEFT, CBRS_ALIGN_RIGHT or CBRS_ALIGN_ANY. Note that CBRS_FLOAT_MULTI is not supported.

class CMRCBitmapButton : public CButton

This class provides similar behaviour to CBitmapButton, except that you need only specify the bitmap for the button face. The pushed bitmap is generated automatically. Currently the “disabled” state is not supported. The colors of the bitmap are also adjusted, using the same algorithm that MFC uses for coloring toolbar buttons.

Construction

CMRCBitmapButton()	constructor
--------------------	-------------

Member Variables

m_bitmap	Bitmap on the button
----------	----------------------

Operations

Create()	Creates the underlying windows object
AutoLoad()	Creates, and loads the bitmap
LoadBitmap()	Loads the bitmap

CMRCBitmapButton::Create()

BOOL Create(**DWORD** *dwStyle*, **const** **RECT** & *rect*, **CWnd** **pParentWnd*, **UINT** *nID*);

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>dwStyle</i>	Style of the button. See CButton for possible styles.
<i>rect</i>	Specifies the control's size and position.
<i>pParentWnd</i>	Specifies the control's parent window. It must not be NULL.
<i>nID</i>	Specifies the control's ID.

CMRCBitmapButton::AutoLoad()

BOOL AutoLoad(UINT *nID*, CWnd **pParentWnd*, BOOL *bSetSize* = TRUE);

This function is designed for use within OnInitDialog to subclass a button in the dialog resource. It is similar to CBitmapButton::AutoLoad(). To use it you must:

1. Create a button in the dialog resource. The “owner draw” style should be set. The caption should be the resource name of a bitmap resource.
2. Create a bitmap resource with the name specified in the caption of the bitmap window.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>nID</i>	Specifies the ID of a button control that exists in the parent window.
<i>pParentWnd</i>	Specifies the parent window, usually a CDialog. It must not be NULL.
<i>bSetSize</i>	If TRUE, the control is automatically resized to match the size specified by the bitmap.

CMRCBitmapButton::LoadBitmap()

BOOL LoadBitmap(UINT *nID*);

BOOL LoadBitmap(LPCTSTR *lpszResourceName*);

Loads the bitmap that is drawn on the face of the button.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>lpszResourceName</i>	Specifies the resource name of the bitmap to load.
<i>nID</i>	Specifies the resource ID of the bitmap to load.

CMRCBitmapButton::AutoSize()

BOOL AutoSize();

Sets the size of the button to match the size of the loaded bitmap.

Return Value:

TRUE if successful, FALSE otherwise.

class CMRCAnimateCtrl : public CWnd

This class provides a simple AVI playing class which makes use of Video for Windows MCI interface. This means that it'll play any format of AVI and that performance is good, compared to the Windows'95 common control, which will only handle small, raw AVI files acceptably. Since it uses MCI, it is best used in a fairly modal manner, e.g. on an About dialog box.

Construction

CMRCAnimateCtrl()	constructor
-------------------	-------------

Operations

Play()	Creates the underlying windows object
Open()	Creates, and loads the bitmap
Rewind()	Loads the bitmap

CMRCAanimateCtrl::Open()

BOOL Open(LPCTSTR lpszFileName);

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

lpszFileName specifies the AVI file name to open. You may need to obtain the full path of your AVI file. The Win32 API function GetModulePath() is often useful for this.

CMRCAanimateCtrl::Close()

BOOL Close();

Closes any open AVI file in the control.

Return Value:

TRUE if successful, FALSE otherwise.

CMRCAanimateCtrl::Play()

BOOL Play(DWORD dwFlags, int nFirst = 0, int nLast = -1, int nRepeatDelay = 0);

Closes any open AVI file in the control.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>dwFlags</i>	Specifies optional flags controlling the playback. It can be a combination of ACF_REPEAT Repeats the clip, with or without a delay ACF_AUTOSIZE Sets the window to the size of the movie
<i>nFirst</i>	Specifies the first frame to play in the clip.
<i>nLast</i>	Specifies the last frame to play in the clip, -1 means play the entire file.
<i>nRepeatDelay</i>	Specifies the delay, in milliseconds, between repeating playback of the AVI clip.

class CMRCColorLabel : public CWnd

This class provides a simple static control, that either displays text in a specified foreground/background color and font, or displays a bitmap, with some limited support for 256-color bitmaps. It registers it's own window class and provides styles where it can load the specified bitmaps automatically. This means it is often possible to use it in dialog resources, without any other code.

Construction

CMRCColorLabel()	constructor
------------------	-------------

Operations

Create()	Creates the underlying windows object
LoadDIB()	Creates, and loads the bitmap
LoadDIBFromFile()	Loads the bitmap from a file
CreateSplash()	Creates and displays the underlying windows object suitable for a banner, or splash screen
SetColors()	Sets the foreground and background colors
SetFont()	Sets the font (if the control is displaying text)

CMRCColorLabel::Create()

**BOOL Create(LPCTSTR *lpszCaption*, DWORD *dwStyle*, RECT & *rect*,
CWnd **pParentWnd*, UINT *nID*);**

Creates the underlying windows object for the CMRCColorLabel.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>lpszCaption</i>	Specifies the caption of the window. This text can also be used to specify a resource name of a bitmap to load into the window.
<i>dwStyle</i>	Specifies the style of the control. In addition to the common window style, additional styles may be specified - see below.
<i>rect</i>	Specifies the control's size and position.
<i>pParentWnd</i>	Specifies the control's parent window.
<i>nID</i>	Specifies the control's ID. This can also be used to specify the resource ID of a bitmap that is loaded into the control

Additional Styles

The following styles can be specified

0x0001L	CLBS_CENTER	Centers the text or bitmap in the control. If the control is displaying text, and no specified, the text is drawn using the DT_WORDBREAK style.
0x0100L	CLBS_BITMAP_STRETCH	The bitmap is stretched to fit the size of the control.
0x0200L	CLBS_BITMAP_FROMTITLE	On creation, the control loads the bitmap specified by the Window text (caption) of the parent window. If the text begins with a number, this resource number is loaded. If no caption does not exist, the control fails to be created.
0x0400L	CLBS_BITMAP_FROMID	On creation, the control loads the bitmap specified by the ID of the control.
0x1000L	CLBS_BITMAP_PALETTE	Use the bitmaps palette when drawing the bitmap. Settings this style allows 256 bitmaps to be rendered.
0x2000L	CLBS_BITMAP_AUTOSIZE	On loading the bitmap, the window is resized to the size of the bitmap. If the CLBS_CENTER style is also specified, the window is repositioned so the bitmap is at the center of the previous window rectangle.
0x8000L	CLBS_BITMAP	The control contains a bitmap - not text..
0x0040L	CLBS_FILLBACKGROUND	The background of the control is explicitly filled in the background color. If this style will mean the area between the text, and/or bitmap is not drawn.
0x0010L	CLBS_BORDER_RAISED	A raised border is drawn just inside the client area of the control. You may have a conventional WS_BORDER style in addition.
0x0020L	CLBS_BORDER_SUNKEN	A sunken border is drawn just inside the client area of the control. You may have a conventional WS_BORDER style in addition

The use of these window styles allows bitmaps (even 256-color bitmaps) to be placed on dialog boxes without any additional code.

CMRCColorLabel::CreateSplash()

BOOL CreateSplash(LPCTSTR lpszResName, DWORD dwStyle, BOOL bBorder);

BOOL CreateSplash(UINT nID, DWORD dwStyle, BOOL bBorder);

Creates the control as a topmost window, suitable for use as a splash screen or banner, displaying the specified bitmap.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>lpszResName</i>	Specifies resource name of bitmap to load
<i>nID</i>	Specifies ID of bitmap to load
<i>dwStyle</i>	Style of window. Generally, the only style you should use here is CLBS_BITMAP_PALETTE if the bitmap is 256-color and should be drawn with a palette.
<i>bBorder</i>	If TRUE, the window is drawn with a small border surrounding the bitmap.

CMRCColorLabel::LoadDIB()

BOOL LoadDIB(LPCTSTR lpszResource, DWORD dwStyle);

BOOL LoadDIB(UINT nID, DWORD dwStyle);

BOOL LoadDIB(LPCTSTR lpszResource);

BOOL LoadDIB(UINT nID);

Creates the underlying windows object for the CMRCColorLabel.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>lpszResource</i>	Specifies the resource name of the bitmap to load.
<i>nID</i>	Specifies the resource ID of the bitmap to load.
<i>dwStyle</i>	Specifies the bitmap orientated styles to apply to the window, before the bitmap is loaded. These styles override any specified on window creation.

CMRCColorLabel::LoadDIBFromFile()

BOOL LoadDIBFromFile(LPCTSTR lpszFileName, DWORD dwStyle);

BOOL LoadDIBFromFile(LPCTSTR lpszFileName);

Loads a bitmap for the control to display, from the specified .bmp filename.

Return Value:

TRUE if successful, FALSE otherwise.

Parameters:

<i>lpszFileName</i>	Specifies the filename of the bitmap file to open.
<i>dwStyle</i>	Specifies the bitmap orientated styles to apply to the window, before the bitmap is loaded. These styles override any specified on window creation.

CMRCColorLabel::SetColors()

void SetColors(COLORREF foreground, COLORREF background);

Sets the foreground and background colors of the control.

Parameters:

foreground

Specifies the foreground color of the text, if the control is displaying text. This parameter is not significant if the control is displaying a bitmap.

background

Specifies the background color of the control. If the control is displaying a bitmap, this color is only significant if the control has the `CBSL_FILLBACKGROUND` style, and the bitmap is smaller than the control.

CMRCColorLabel::SetFont()

void SetFont(CFont * pFont);

Parameters:

pFont

Points to the font to be used to display the text in the window. You should ensure the CFont object exists for at least the lifetime of this control.

Mark Conway.
mrc@mfltd.co.uk
CIS:74664,434
January '95