[www.pudn.com](http://www.pudn.com) > [WTLGrid2\_src.zip](http://www.pudn.com/downloads129/sourcecode/windows/control/detail554156.html) > atlgdix.h, change:2003-05-29,size:23807b

Formularbeginn



Formularende

Formularbeginn

#ifndef \_\_ATLGDIX\_H\_\_

#define \_\_ATLGDIX\_H\_\_

/////////////////////////////////////////////////////////////////////////////

// Additional GDI/USER wrappers

//

// Written by Bjarke Viksoe (bjarke@viksoe.dk)

// Copyright (c) 2001-2002 Bjarke Viksoe.

// Thanks to Daniel Bowen for COffscreenDrawRect.

//

// This code may be used in compiled form in any way you desire. This

// file may be redistributed by any means PROVIDING it is

// not sold for profit without the authors written consent, and

// providing that this notice and the authors name is included.

//

// This file is provided "as is" with no expressed or implied warranty.

// The author accepts no liability if it causes any damage to you or your

// computer whatsoever. It's free, so don't hassle me about it.

//

// Beware of bugs.

//

#pragma once

#ifndef \_\_cplusplus

#error ATL requires C++ compilation (use a .cpp suffix)

#endif

#ifndef \_\_ATLGDI\_H\_\_

#error atlgdix.h requires atlgdi.h to be included first

#endif

namespace WTL

{

/////////////////////////////////////////////////////////////////////////////

// Macros

// The GetXValue macros below are badly designed and emit

// compiler warnings e.g. when using RGB(255,255,255)...

#pragma warning(disable : 4310)

#ifndef BlendRGB

#define BlendRGB(c1, c2, factor) \

RGB( GetRValue(c1) + ((GetRValue(c2) - GetRValue(c1)) \* factor / 100L), \

GetGValue(c1) + ((GetGValue(c2) - GetGValue(c1)) \* factor / 100L), \

GetBValue(c1) + ((GetBValue(c2) - GetBValue(c1)) \* factor / 100L) );

#endif

#ifndef COLOR\_INVALID

#define COLOR\_INVALID (COLORREF) CLR\_INVALID

#endif

/////////////////////////////////////////////////////////////////////////////

// CIcon

template< bool t\_bManaged >

class CIconT

{

public:

HICON m\_hIcon;

// Constructor/destructor/operators

CIconT(HICON hIcon = NULL) : m\_hIcon(hIcon)

{ }

~CIconT()

{

if( t\_bManaged && m\_hIcon != NULL ) ::DestroyIcon(m\_hIcon);

}

CIconT<t\_bManaged>& operator=(HICON hIcon)

{

m\_hIcon = hIcon;

return \*this;

}

void Attach(HICON hIcon)

{

if( t\_bManaged && m\_hIcon != NULL ) ::DestroyIcon(m\_hIcon);

m\_hIcon = hIcon;

}

HICON Detach()

{

HICON hIcon = m\_hIcon;

m\_hIcon = NULL;

return hIcon;

}

operator HICON() const { return m\_hIcon; }

bool IsNull() const { return m\_hIcon == NULL; }

// Create methods

HICON LoadIcon(\_U\_STRINGorID icon)

{

ATLASSERT(m\_hIcon==NULL);

m\_hIcon = ::LoadIcon(\_Module.GetResourceInstance(), icon.m\_lpstr);

return m\_hIcon;

}

HICON LoadIcon(\_U\_STRINGorID icon, int cxDesired, int cyDesired, UINT fuLoad = 0)

{

ATLASSERT(m\_hIcon==NULL);

m\_hIcon = (HICON) ::LoadImage(\_Module.GetResourceInstance(), icon.m\_lpstr, IMAGE\_ICON, cxDesired, cyDesired, fuLoad);

return m\_hIcon;

}

HICON LoadOEMIcon(UINT nIDIcon) // for IDI\_ types

{

ATLASSERT(m\_hIcon==NULL);

m\_hIcon = ::LoadIcon(NULL, MAKEINTRESOURCE(nIDIcon));

return m\_hIcon;

}

HICON CreateIcon(int nWidth, int nHeight, BYTE cPlanes, BYTE cBitsPixel, CONST BYTE\* lpbANDButs, CONST BYTE \*lpbXORbits)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(lpbANDbits);

ATLASSERT(lpbXORbits);

m\_hIcon = ::CreateIcon(\_Module.GetResourceInstance(), nWidth, nHeight, cPlanes, cBitsPixel, lpbANDbits, lpbXORbits);

return m\_hIcon;

}

HICON CreateIconFromResource(PBYTE pBits, DWORD dwResSize, DWORD dwVersion = 0x00030000)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(pBits);

m\_hIcon = ::CreateIconFromResource(pBits, dwResSize, TRUE, dwVersion);

return m\_hIcon;

}

HICON CreateIconFromResourceEx(PBYTE pbBits, DWORD cbBits, DWORD dwVersion = 0x00030000, int cxDesired = 0, int cyDesired = 0, UINT uFlags = LR\_DEFAULTCOLOR)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(pbBits);

ATLASSERT(cbBits>0);

m\_hIcon = ::CreateIconFromResourceEx(pbBits, cbBits, TRUE, dwVersion, cxDesired, cyDesired, uFlags);

return m\_hIcon;

}

HICON CreateIconIndirect(PICONINFO pIconInfo)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(pIconInfo);

m\_hIcon = ::CreateIconIndirect(pIconInfo);

return m\_hIcon;

}

HICON ExtractIcon(LPCTSTR lpszExeFileName, UINT nIconIndex)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(!::IsBadStringPtr(lpszExeFileName,-1));

m\_hIcon = ::ExtractIcon(\_Module.GetModuleInstance(), lpszExeFileName, nIconIndex);

return m\_hIcon;

}

HICON ExtractAssociatedIcon(HINSTANCE hInst, LPCTSTR lpIconPath, LPWORD lpiIcon)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(!::IsBadStringPtr(lpIconPath,-1));

ATLASSERT(lpiIcon);

m\_hIcon = ::ExtractAssociatedIcon(hInst, lpIconPath, lpiIcon);

return m\_hIcon;

}

// Operations

BOOL DestroyIcon()

{

ATLASSERT(m\_hIcon!=NULL);

BOOL bRet = ::DestroyIcon(m\_hIcon);

if( bRet ) m\_hIcon = NULL;

return bRet;

}

HICON CopyIcon()

{

ATLASSERT(m\_hIcon!=NULL);

return ::CopyIcon(m\_hIcon);

}

HICON DuplicateIcon()

{

ATLASSERT(m\_hIcon!=NULL);

return ::DuplicateIcon(NULL, m\_hIcon);

}

BOOL DrawIcon(HDC hDC, int x, int y)

{

ATLASSERT(m\_hIcon!=NULL);

return ::DrawIcon(hDC, x, y, m\_hIcon);

}

BOOL DrawIcon(HDC hDC, POINT pt)

{

ATLASSERT(m\_hIcon!=NULL);

return ::DrawIcon(hDC, pt.x, pt.y, m\_hIcon);

}

BOOL DrawIconEx(HDC hDC, int x, int y, int cxWidth, int cyWidth, UINT uStepIfAniCur = 0, HBRUSH hbrFlickerFreeDraw = NULL, UINT uFlags = DI\_NORMAL)

{

ATLASSERT(m\_hIcon!=NULL);

return ::DrawIconEx(hDC, x, y, m\_hIcon, cxWidth, cyWidth, uStepIfAniCur, hbrFlickerFreeDraw, uFlags);

}

BOOL DrawIconEx(HDC hDC, POINT pt, SIZE size, UINT uStepIfAniCur = 0, HBRUSH hbrFlickerFreeDraw = NULL, UINT uFlags = DI\_NORMAL)

{

ATLASSERT(m\_hIcon!=NULL);

return ::DrawIconEx(hDC, pt.x, pt.y, m\_hIcon, size.cx, size.cy, uStepIfAniCur, hbrFlickerFreeDraw, uFlags);

}

BOOL GetIconInfo(PICONINFO pIconInfo)

{

ATLASSERT(m\_hIcon!=NULL);

ATLASSERT(pIconInfo);

return ::GetIconInfo(m\_hIcon, pIconInfo);

}

};

typedef CIconT<true> CIcon;

typedef CIconT<false> CIconHandle;

/////////////////////////////////////////////////////////////////////////////

// CCursor

// Protect template against silly macro

#ifdef CopyCursor

#undef CopyCursor

#endif

template< bool t\_bManaged >

class CCursorT

{

public:

HCURSOR m\_hCursor;

// Constructor/destructor/operators

CCursorT(HCURSOR hCursor = NULL) : m\_hCursor(hCursor)

{ }

~CCursorT()

{

if( t\_bManaged && m\_hCursor != NULL ) ::DestroyCursor(m\_hCursor);

}

CCursorT<t\_bManaged>& operator=(HCURSOR hCursor)

{

m\_hCursor = hCursor;

return \*this;

}

void Attach(HCURSOR hCursor)

{

if( t\_bManaged && m\_hCursor != NULL ) ::DestroyCursor(m\_hCursor);

m\_hCursor = hCursor;

}

HCURSOR Detach()

{

HCURSOR hCursor = m\_hCursor;

m\_hCursor = NULL;

return hCursor;

}

operator HCURSOR() const { return m\_hCursor; }

bool IsNull() const { return m\_hCursor == NULL; }

// Create methods

HCURSOR LoadCursor(\_U\_STRINGorID cursor)

{

ATLASSERT(m\_hCursor==NULL);

m\_hCursor = ::LoadCursor(\_Module.GetResourceInstance(), cursor.m\_lpstr);

return m\_hCursor;

}

HCURSOR LoadOEMCursor(UINT nIDCursor) // for IDC\_ types

{

ATLASSERT(m\_hCursor==NULL);

m\_hCursor = ::LoadCursor(NULL, MAKEINTRESOURCE(nIDCursor));

return m\_hCursor;

}

HICON LoadCursor(\_U\_STRINGorID cursor, int cxDesired, int cyDesired, UINT fuLoad = 0)

{

ATLASSERT(m\_hCursor==NULL);

m\_hCursor = (HCURSOR) ::LoadImage(\_Module.GetResourceInstance(), cursor.m\_lpstr, IMAGE\_CURSOR, cxDesired, cyDesired, fuLoad);

return m\_hCursor;

}

HCURSOR LoadCursorFromFile(LPCTSTR pstrFilename)

{

ATLASSERT(m\_hCursor==NULL);

ATLASSERT(!::IsBadStringPtr(pstrFilename,-1));

m\_hCursor = ::LoadCursorFromFile(pstrFilename);

return m\_hCursor;

}

HCURSOR CreateCursor(int xHotSpot, int yHotSpot, int nWidth, int nHeight, CONST VOID \*pvANDPlane, CONST VOID \*pvXORPlane)

{

ATLASSERT(m\_hCursor==NULL);

m\_hCursor = ::CreateCursor(\_Module.GetResourceInstance(), xHotSpot, yHotSpot, nWidth, nHeight, pvANDPlane, pvXORPlane);

return m\_hCursor;

}

HICON CreateCursorFromResource(PBYTE pBits, DWORD dwResSize, DWORD dwVersion = 0x00030000)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(pBits);

m\_hIcon = ::CreateIconFromResource(pBits, dwResSize, FALSE, dwVersion);

return m\_hIcon;

}

HICON CreateCursorFromResourceEx(PBYTE pbBits, DWORD cbBits, DWORD dwVersion = 0x00030000, int cxDesired = 0, int cyDesired = 0, UINT uFlags = LR\_DEFAULTCOLOR)

{

ATLASSERT(m\_hIcon==NULL);

ATLASSERT(pbBits);

ATLASSERT(cbBits>0);

m\_hIcon = ::CreateIconFromResourceEx(pbBits, cbBits, FALSE, dwVersion, cxDesired, cyDesired, uFlags);

return m\_hIcon;

}

// Operations

BOOL DestroyCursor()

{

ATLASSERT(m\_hCursor!=NULL);

BOOL bRet = ::DestroyCursor(m\_hCursor);

if( bRet ) m\_hCursor = NULL;

return bRet;

}

HCURSOR CopyCursor()

{

ATLASSERT(m\_hCursor!=NULL);

return (HCURSOR) ::CopyIcon( (HICON) m\_hCursor );

}

#if(WINVER >= 0x0500)

BOOL GetCursorInfo(LPCURSORINFO pCursorInfo)

{

ATLASSERT(m\_hCursor!=NULL);

ATLASSERT(pCursorInfo);

return ::GetCursorInfo(pCursorInfo);

}

#endif

};

typedef CCursorT<true> CCursor;

typedef CCursorT<false> CCursorHandle;

/////////////////////////////////////////////////////////////////////////////

// CAccelerator

template< bool t\_bManaged >

class CAcceleratorT

{

public:

HACCEL m\_hAccel;

// Constructor/destructor/operators

CAcceleratorT(HACCEL hAccel = NULL) : m\_hAccel(hAccel)

{ }

~CAcceleratorT()

{

if( t\_bManaged && m\_hAccel != NULL ) ::DestroyAcceleratorTable(m\_hAccel);

}

CAcceleratorT<t\_bManaged>& operator=(HACCEL hAccel)

{

m\_hAccel = hAccel;

return \*this;

}

void Attach(HACCEL hAccel)

{

if( t\_bManaged && m\_hAccel != NULL ) ::DestroyAcceleratorTable(m\_hAccel);

m\_hAccel = hAccel;

}

HCURSOR Detach()

{

HACCEL hAccel = m\_hAccel;

m\_hAccel = NULL;

return hAccel;

}

operator HACCEL() const { return m\_hAccel; }

bool IsNull() const { return m\_hAccel == NULL; }

// Create methods

HACCEL LoadAccelerators(\_U\_STRINGorID accel)

{

ATLASSERT(m\_hAccel==NULL);

m\_hAccel = ::LoadAccelerators(\_Module.GetResourceInstance(), accel.m\_lpstr);

return m\_hAccel;

}

HACCEL CreateAcceleratorTable(LPACCEL pAccel, int cEntries)

{

ATLASSERT(m\_hAccel==NULL);

ATLASSERT(!::IsBadReadPtr(lpAccelDst, sizeof(ACCEL)\*cEntries));

m\_hAccel = ::CreateAcceleratorTable(pAccel, cEntries);

return m\_hAccel;

}

// Operations

int CopyAcceleratorTable(LPACCEL lpAccelDst, int cEntries)

{

ATLASSERT(m\_hAccel!=NULL);

ATLASSERT(!::IsBadWritePtr(lpAccelDst, sizeof(ACCEL)\*cEntries));

return ::CopyAcceleratorTable(m\_hAccel, lpAccelDst, cEntries);

}

BOOL TranslateAccelerator(HWND hWnd, LPMSG pMsg)

{

ATLASSERT(m\_hAccel!=NULL);

ATLASSERT(::IsWindow(hWnd));

ATLASSERT(pMsg);

return ::TranslateAccelerator(hWnd, m\_hAccel, pMsg);

}

};

typedef CAcceleratorT<true> CAccelerator;

typedef CAcceleratorT<false> CAcceleratorHandle;

/////////////////////////////////////////////////////////////////////////////

// CLogFont

class CLogFont : public LOGFONT

{

public:

CLogFont()

{

::ZeroMemory( (LOGFONT\*) this, sizeof(LOGFONT) );

}

CLogFont(const LOGFONT& lf)

{

Copy(&lf);

}

CLogFont(HFONT hFont)

{

ATLASSERT(::GetObjectType(hFont)==OBJ\_FONT);

::GetObject(hFont, sizeof(LOGFONT), (LOGFONT\*) this);

}

HFONT CreateFontIndirect()

{

return ::CreateFontIndirect(this);

}

void SetBold()

{

lfWeight = FW\_BOLD;

}

BOOL IsBold() const

{

return lfWeight >= FW\_BOLD;

}

void MakeBolder(int iScale = 1)

{

lfWeight += FW\_BOLD \* iScale;

}

void MakeLarger(int iScale)

{

if( lfHeight > 0 ) lfHeight += iScale; else lfHeight -= iScale;

}

void SetHeight(long PointSize, HDC hDC = NULL)

{

// For MM\_TEXT mapping mode...

// NOTE: MulDiv() gives correct rounding.

lfHeight = -MulDiv(PointSize, ::GetDeviceCaps(hDC, LOGPIXELSY), 72);

}

long GetHeight(HDC hDC = NULL) const

{

// For MM\_TEXT mapping mode...

// NOTE: MulDiv() gives correct rounding.

return ::MulDiv(-lfHeight, 72, ::GetDeviceCaps(hDC, LOGPIXELSY));

}

long GetDeciPointHeight(HDC hDC = NULL)

{

POINT ptOrg = { 0, 0 };

::DPtoLP(hDC, &ptOrg, 1);

POINT pt = { 0, 0 };

pt.y = abs(lfHeight) + ptOrg.y;

::LPtoDP(hDC,&pt,1);

return MulDiv(pt.y, 720, ::GetDeviceCaps(hDC,LOGPIXELSY)); // 72 points/inch, 10 decipoints/point

}

void SetHeightFromDeciPoint(long DeciPtHeight, HDC hDC = NULL)

{

POINT pt;

pt.y = MulDiv(::GetDeviceCaps(hDC, LOGPIXELSY), DeciPtHeight, 720); // 72 points/inch, 10 decipoints/point

::DPtoLP(hDC, &pt, 1);

POINT ptOrg = { 0, 0 };

::DPtoLP(hDC, &ptOrg, 1);

lfHeight = -abs(pt.y - ptOrg.y);

}

void SetCaptionFont()

{

NONCLIENTMETRICS ncm = { 0 };

ncm.cbSize = sizeof(ncm);

::SystemParametersInfo(SPI\_GETNONCLIENTMETRICS, sizeof(ncm), &ncm, 0);

Copy(&ncm.lfCaptionFont);

}

void SetMenuFont()

{

NONCLIENTMETRICS ncm = { 0 };

ncm.cbSize = sizeof(ncm);

::SystemParametersInfo(SPI\_GETNONCLIENTMETRICS, sizeof(ncm), &ncm, 0);

Copy(&ncm.lfMenuFont);

}

void SetStatusFont()

{

NONCLIENTMETRICS ncm = { 0 };

ncm.cbSize = sizeof(ncm);

::SystemParametersInfo(SPI\_GETNONCLIENTMETRICS, sizeof(ncm), &ncm, 0);

Copy(&ncm.lfStatusFont);

}

void SetMessageBoxFont()

{

NONCLIENTMETRICS ncm = { 0 };

ncm.cbSize = sizeof(ncm);

::SystemParametersInfo(SPI\_GETNONCLIENTMETRICS, sizeof(ncm), &ncm, 0);

Copy(&ncm.lfMessageFont);

}

void Copy(const LOGFONT\* lf)

{

ATLASSERT(lf);

::CopyMemory( (LOGFONT\*) this, lf, sizeof(LOGFONT) );

}

CLogFont& operator=(const CLogFont& src)

{

Copy(&src);

return \*this;

}

CLogFont& operator=(const LOGFONT& src)

{

Copy(&src);

return \*this;

}

CLogFont& operator=(HFONT hFont)

{

ATLASSERT(::GetObjectType(hFont)==OBJ\_FONT);

::GetObject(hFont, sizeof(LOGFONT), (LOGFONT\*) this);

return \*this;

}

bool operator==(const LOGFONT& logfont) const

{

return( logfont.lfHeight == lfHeight &&

logfont.lfWidth == lfWidth &&

logfont.lfEscapement == lfEscapement &&

logfont.lfOrientation == lfOrientation &&

logfont.lfWeight == lfWeight &&

logfont.lfItalic == lfItalic &&

logfont.lfUnderline == lfUnderline &&

logfont.lfStrikeOut == lfStrikeOut &&

logfont.lfCharSet == lfCharSet &&

logfont.lfOutPrecision == lfOutPrecision &&

logfont.lfClipPrecision == lfClipPrecision &&

logfont.lfQuality == lfQuality &&

logfont.lfPitchAndFamily == lfPitchAndFamily &&

::lstrcmp(logfont.lfFaceName, lfFaceName) == 0 );

}

};

/////////////////////////////////////////////////////////////////////////////

// CMemDC

class CMemDC : public CDC

{

public:

CDCHandle m\_dc; // Owner DC

CBitmap m\_bitmap; // Offscreen bitmap

CBitmapHandle m\_hOldBitmap; // Originally selected bitmap

RECT m\_rc; // Rectangle of drawing area

CMemDC(HDC hDC, LPRECT pRect = NULL)

{

ATLASSERT(hDC!=NULL);

m\_dc = hDC;

if( pRect != NULL ) m\_rc = \*pRect; else m\_dc.GetClipBox(&m\_rc);

CreateCompatibleDC(m\_dc);

::LPtoDP(m\_dc, (LPPOINT) &m\_rc, sizeof(RECT)/sizeof(POINT));

m\_bitmap.CreateCompatibleBitmap(m\_dc, m\_rc.right-m\_rc.left, m\_rc.bottom-m\_rc.top);

m\_hOldBitmap = SelectBitmap(m\_bitmap);

::DPtoLP(m\_dc, (LPPOINT) &m\_rc, sizeof(RECT)/sizeof(POINT));

SetWindowOrg(m\_rc.left, m\_rc.top);

}

~CMemDC()

{

// Copy the offscreen bitmap onto the screen.

m\_dc.BitBlt(m\_rc.left, m\_rc.top, m\_rc.right-m\_rc.left, m\_rc.bottom-m\_rc.top,

m\_hDC, m\_rc.left, m\_rc.top, SRCCOPY);

//Swap back the original bitmap.

SelectBitmap(m\_hOldBitmap);

}

};

/////////////////////////////////////////////////////////////////////////////

// COffscreenDraw

// To use it, derive from it and chain it in the message map.

template< class T >

class COffscreenDraw

{

public:

BEGIN\_MSG\_MAP(COffscreenDraw)

MESSAGE\_HANDLER(WM\_PAINT, OnPaint)

MESSAGE\_HANDLER(WM\_PRINTCLIENT, OnPaint)

MESSAGE\_HANDLER(WM\_ERASEBKGND, OnEraseBackground)

END\_MSG\_MAP()

LRESULT OnPaint(UINT /\*uMsg\*/, WPARAM wParam, LPARAM /\*lParam\*/, BOOL& /\*bHandled\*/)

{

T\* pT = static\_cast<T\*>(this);

if( wParam != NULL )

{

CMemDC memdc( (HDC) wParam, NULL );

pT->DoPaint(memdc.m\_hDC);

}

else

{

RECT rc;

::GetClientRect(pT->m\_hWnd, &rc);

CPaintDC dc(pT->m\_hWnd);

CMemDC memdc(dc.m\_hDC, &rc);

pT->DoPaint(memdc.m\_hDC);

}

return 0;

}

LRESULT OnEraseBackground(UINT /\*uMsg\*/, WPARAM /\*wParam\*/, LPARAM /\*lParam\*/, BOOL& /\*bHandled\*/)

{

return 1; // handled; no need to erase background; do it in DoPaint();

}

void DoPaint(CDCHandle dc)

{

ATLASSERT(false); // must override this

}

};

// To use it, derive from it and chain it in the message map.

template< class T >

class COffscreenDrawRect

{

public:

BEGIN\_MSG\_MAP(COffscreenDrawRect)

MESSAGE\_HANDLER(WM\_PAINT, OnPaint)

MESSAGE\_HANDLER(WM\_PRINTCLIENT, OnPaint)

MESSAGE\_HANDLER(WM\_ERASEBKGND, OnEraseBackground)

END\_MSG\_MAP()

LRESULT OnPaint(UINT /\*uMsg\*/, WPARAM wParam, LPARAM /\*lParam\*/, BOOL& /\*bHandled\*/)

{

T\* pT = static\_cast<T\*>(this);

if( wParam != NULL )

{

CMemDC memdc( (HDC) wParam, NULL );

pT->DoPaint(memdc.m\_hDC, memdc.m\_rc);

}

else

{

CPaintDC dc(pT->m\_hWnd);

CMemDC memdc(dc.m\_hDC, &dc.m\_ps.rcPaint);

pT->DoPaint(memdc.m\_hDC, dc.m\_ps.rcPaint);

}

return 0;

}

LRESULT OnEraseBackground(UINT /\*uMsg\*/, WPARAM /\*wParam\*/, LPARAM /\*lParam\*/, BOOL& /\*bHandled\*/)

{

return 1; // handled; no need to erase background; do it in DoPaint();

}

void DoPaint(CDCHandle dc, RECT& rcClip)

{

ATLASSERT(false); // must override this

}

};

/////////////////////////////////////////////////////////////////////////////

// CSaveDC

class CSaveDC

{

public:

HDC m\_hDC;

int m\_iState;

CSaveDC(HDC hDC) : m\_hDC(hDC)

{

ATLASSERT(::GetObjectType(m\_hDC)==OBJ\_DC || ::GetObjectType(m\_hDC)==OBJ\_MEMDC);

m\_iState = ::SaveDC(hDC);

ATLASSERT(m\_iState!=0);

}

~CSaveDC()

{

Restore();

}

void Restore()

{

if( m\_iState == 0 ) return;

ATLASSERT(::GetObjectType(m\_hDC)==OBJ\_DC || ::GetObjectType(m\_hDC)==OBJ\_MEMDC);

::RestoreDC(m\_hDC, m\_iState);

m\_iState = 0;

}

};

/////////////////////////////////////////////////////////////////////////////

// CHandle

#if (\_ATL\_VER < 0x0700)

class CHandle

{

public:

HANDLE m\_h;

CHandle(HANDLE hSrc = INVALID\_HANDLE\_VALUE) : m\_h(hSrc)

{ }

~CHandle()

{

Close();

}

operator HANDLE() const { return m\_h; };

LPHANDLE operator&()

{

ATLASSERT(!IsValid());

return &m\_h;

}

CHandle& operator=(HANDLE h)

{

ATLASSERT(!IsValid());

m\_h = h;

return \*this;

}

bool IsValid() const { return m\_h != INVALID\_HANDLE\_VALUE; };

void Attach(HANDLE h)

{

if( IsValid() ) ::CloseHandle(m\_h);

m\_h = h;

}

HANDLE Detach()

{

HANDLE h = m\_h;

m\_h = INVALID\_HANDLE\_VALUE;

return h;

}

BOOL Close()

{

BOOL bRes = FALSE;

if( m\_h != INVALID\_HANDLE\_VALUE ) {

bRes = ::CloseHandle(m\_h);

m\_h = INVALID\_HANDLE\_VALUE;

}

return bRes;

}

BOOL Duplicate(HANDLE hSource, bool bInherit = false)

{

ATLASSERT(!IsValid());

HANDLE hOurProcess = ::GetCurrentProcess();

BOOL b = ::DuplicateHandle(hOurProcess,

hSource,

hOurProcess,

&m\_h,

DUPLICATE\_SAME\_ACCESS,

bInherit,

DUPLICATE\_SAME\_ACCESS);

ATLASSERT(b);

return b;

}

};

#endif // \_ATL\_VER

/////////////////////////////////////////////////////////////////////////////

// Mouse Hover helper

#ifndef NOTRACKMOUSEEVENT

#ifndef WM\_MOUSEENTER

#define WM\_MOUSEENTER WM\_USER + 253

#endif // WM\_MOUSEENTER

// To use it, derive from it and chain it in the message map.

// Make sure to set bHandled to FALSE when handling WM\_MOUSEMOVE or

// the WM\_MOUSELEAVE message!

template< class T >

class CMouseHover

{

public:

bool m\_fMouseOver; // Internal mouse-over state

bool m\_fMouseForceUpdate; // Update window immediately on event

CMouseHover() :

m\_fMouseOver(false),

m\_fMouseForceUpdate(true)

{

}

BEGIN\_MSG\_MAP(CMouseHover)

MESSAGE\_HANDLER(WM\_MOUSEMOVE, OnMouseMove)

MESSAGE\_HANDLER(WM\_MOUSELEAVE, OnMouseLeave)

END\_MSG\_MAP()

LRESULT OnMouseMove(UINT /\*uMsg\*/, WPARAM wParam, LPARAM lParam, BOOL& bHandled)

{

T\* pT = static\_cast<T\*>(this);

if( !m\_fMouseOver ) {

m\_fMouseOver = true;

pT->SendMessage(WM\_MOUSEENTER, wParam, lParam);

pT->Invalidate();

if( m\_fMouseForceUpdate ) pT->UpdateWindow();

\_StartTrackMouseLeave(pT->m\_hWnd);

}

bHandled = FALSE;

return 0;

}

LRESULT OnMouseLeave(UINT /\*uMsg\*/, WPARAM /\*wParam\*/, LPARAM /\*lParam\*/, BOOL& bHandled)

{

T\* pT = static\_cast<T\*>(this);

if( m\_fMouseOver ) {

m\_fMouseOver = false;

pT->Invalidate();

if( m\_fMouseForceUpdate ) pT->UpdateWindow();

}

bHandled = FALSE;

return 0;

}

BOOL \_StartTrackMouseLeave(HWND hWnd) const

{

ATLASSERT(::IsWindow(hWnd));

TRACKMOUSEEVENT tme = { 0 };

tme.cbSize = sizeof(tme);

tme.dwFlags = TME\_LEAVE;

tme.hwndTrack = hWnd;

return \_TrackMouseEvent(&tme);

}

BOOL \_CancelTrackMouseLeave(HWND hWnd) const

{

TRACKMOUSEEVENT tme = { 0 };

tme.cbSize = sizeof(tme);

tme.dwFlags = TME\_LEAVE | TME\_CANCEL;

tme.hwndTrack = hWnd;

return \_TrackMouseEvent(&tme);

}

};

#endif // NOTRACKMOUSEEVENT

}; // namespace WTL

#endif // \_\_ATLGDIX\_H\_\_

[http://icon.cnzz.com/img/pic.gif](http://www.cnzz.com/stat/website.php?web_id=1236358)

Formularende