

```

/*****
/*
/*----- P R I N T E R . C -----*/
/* Task      : Display and selection of default printer */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich */
/* developed on : 09/15/1995 */
/* last update  : 09/22/1995 */
/*****

#include <windows.h>
#include <shlobj.h>
#include <commctrl.h>

#include "resource.h"
#include "enumfold.h"
#include "taskbar.h"

//---- Constants -----
#define BTN_WIDTH 100 // Width of a button
#define BTN_HEIGHT 22 // Height of a button

#define ID_PRINTERBUTTONS 432 // Command ID of first button
#define MAX_PRINTERBUTTONS 99 // maximum 99 buttons

#define MY_CLASS "StdPrinter" // Window class for printer window

#define WM_MYTASKBAR WM_USER + 1 // Message through which the taskbar
// communicates with the application window
#define TB_PRINTER_ID 1 // ID of taskbar icon

#define TBCLASS "TaskBarController" // Class of taskbar controller

#define IDM_REMOVE 101 // ID of PopupMenu item

// Custom message for displaying Context menu -----
#define WM_MYCONTEXTMENU WM_USER + 2

//----- Global Variables -----
char g_szPrinter[ MAX_PATH ]; // Name of current default printer

//---- Typedefs -----
typedef struct tagPRINTERBUTTONS // Control structure for buttons
{
    HWND hWnd; // Window handle
    int x, y, cx, cy; // Location dimensions
    BOOL bPressed; // Button pressed?
    HICON hIcon; // Handle of icon to be displayed
    int iTextLen; // Length of output text (including ellipsis...)
    BOOL bEllipsis; // Display ellipsis?
} PRINTERBUTTONS;

//----- Global Variables -----
PRINTERBUTTONS g_Printers[ MAX_PRINTERBUTTONS ]; // For each button
// a structure

int g_iNumPrinters = 0; // How many buttons were received?
char g_szDefault[ MAX_PATH ]; // current default printer

/*****
/* SetStdPrinter - Sets default printer */
/*-----*/
/* Parameters: hPrinter - Handle of printer that will be */
/* default printer. */
/* Return value : TRUE - Default printer set */
/* FALSE - Error */
/*****
BOOL SetStdPrinter( HANDLE hPrinter )
{
    DRIVER_INFO_2 *di2;
    PRINTER_INFO_2 *pi2;
    DWORD dwNeeded;
    char szDeviceEntry[ MAX_PATH ];
    BOOL retval = FALSE;

    // Get required DRIVER_INFO2 size for printer -----

```

```

GetPrinterDriver( hPrinter, NULL, 2, NULL, 0 , &dwNeeded );

// Allocate memory and get DRIVER_INFO2 structure -----
di2 = malloc( dwNeeded );

if( di2 )
    GetPrinterDriver( hPrinter,
                      NULL,
                      2,
                      (LPBYTE)di2,
                      dwNeeded,
                      &dwNeeded );
                      // Environment
                      // Level = 2

// Get required PRINTER_INFO2 size for printer -----
GetPrinter( hPrinter, 2, NULL, 0 , &dwNeeded );

// Allocate memory and get PRINTER_INFO2 structure -----
pi2 = malloc( dwNeeded );
if( pi2 )
    GetPrinter( hPrinter,
                2,
                (LPBYTE)pi2,
                dwNeeded ,
                &dwNeeded );
                // Level = 2

if( pi2 && di2 )
{
    LPSTR lpDRV;
    // Put together WIN.INI entry (Printername,DriverDLL, Port) ---
    strcpy( szDeviceEntry, pi2->pPrinterName );
    strcat( szDeviceEntry, "," );

    // Search for '.' in driver names -----
    lpDRV = strchr( di2->pDriverPath, '.' );
    if( lpDRV )
    {
        int i = 0;
        // Only the name of the DLL is required as the driver DLL -
        // without the extension and without the path. -
        lpDRV[0] = '\\0';
        while( lpDRV[ i ] != '\\\\' ) i--;
        i++;
        strcat( szDeviceEntry, &lpDRV[ i ] );
    }
    else strcat( szDeviceEntry, di2->pDriverPath );

    // Append port name -----
    strcat( szDeviceEntry, "," );
    strcat( szDeviceEntry, pi2->pPortName );

    // Update WIN.INI entry -----
    WriteProfileString( "Windows", "device", szDeviceEntry );

    // Notify other applications about the change in default printer
    //This application also learns about the change from WM_WININICHANGE.
    SendMessage( HWND_BROADCAST, WM_WININICHANGE, 0, (LONG)"Windows" );

    retval = TRUE;
    // Everything's OK!
}

if( di2 ) free( di2 );
if( pi2 ) free( pi2 );

return retval;
}

/*****
/* GetStdPrinter - Retrieves current default printer from WIN.INI */
/* ----- */
/* Parameters:      lpText - Address of text buffer that is to receive */
/*                  the current default printer. */
/*                  cbText - Size of buffer */
/* Return value : none */
*****/
void GetStdPrinter( LPSTR lpText, int cbText )
{
    DWORD dw;

```

```

// Retrieve [Windows] "device" entry from WIN.INI -----
dw = GetProfileString( "Windows", "device", "", lpText, cbText );

if( ( dw ) && ( ( int ) dw <= cbText ) )          // Entry read?
{
    // Remove driver and port names -----
    int i = 0;
    while( lpText[i] != ',' ) i++;
    lpText[i] = '\0';
}
else lpText[0] = '\0';
}

/*****
/* EnumShellCallback - Callback function for enumerating Shell
/*
/*
/*-----*/
/* Parameters:      lpFolder      - Address of IShellFolder inter-
/*                      face of the current object
/*
/*                      pszPath      - Path name of the object or NULL
/*
/*                      pszDisplayName - DisplayName of object
/*
/*                      ulAttrib      - Attributes of object
/*
/*                      dwUser        - User data (parent window)
/*
/*                      pidlPath      - PIDL of object (relative to
/*                      Desktop)
/*
/*                      pidl          - PIDL of object (relative to
/*                      parent object)
/*
/*                      iLevel        - Nesting level
/*
/* Return value : TRUE - enumerate other objects of current nesting
/*                      level.
/*
/*                      FALSE - continue at parent level
/*
*****/
BOOL WINAPI EnumShellCallback( LPSHELLFOLDER lpFolder,
                                PSTR          pszPath,
                                PSTR          pszDisplayName,
                                DWORD          ulAttrib,
                                DWORD          dwUser,
                                LPITEMIDLIST  pidlPath,
                                LPITEMIDLIST  pidl,
                                int           iLevel )
{
    HWND      hParent;
    HANDLE    hPrinter;
    SHFILEINFO sfi;

    // Parent object is named "Printer" so we'll ignore it -----
    if( iLevel == 0 ) return TRUE;

    hParent = (HWND)dwUser;
    // This callback also gets a notification about the
    // "New Printer" of the Control Panel. To prevent it from
    // becoming available, the program first tests whether the
    // printer passed here can be opened.
    if( OpenPrinter( pszDisplayName, &hPrinter, NULL ) )
    {
        // Can other printers be added to window? -----
        if( g_iNumPrinters < MAX_PRINTERBUTTONS )
        {
            HDC      hDC;
            HFONT    hF;
            SIZE      sText;

            // Create buttons window if it doesn't yet exist -----
            if( !g_Printers[ g_iNumPrinters ].hWnd )
                g_Printers[ g_iNumPrinters ].hWnd =
                    CreateWindowEx( 0L,
                                    "BUTTON",
                                    NULL,
                                    BS_OWNERDRAW |
                                    BS_PUSHBUTTON |
                                    WS_CHILD,
                                    0, 0, 0, 0,
                                    hParent,
                                    (HMENU)(ID_PRINTERBUTTONS + g_iNumPrinters),
                                    (HINSTANCE)GetWindowLong( hParent, GWL_HINSTANCE ),

```

```

        0L );

// Text of button = Printer Name -----
SetWindowText( g_Printers[ g_iNumPrinters ].hWnd,
                pszDisplayName );

// Set font of button -----
SendMessage( g_Printers[ g_iNumPrinters ].hWnd,
              WM_SETFONT,
              (LPARAM)GetStockObject( DEFAULT_GUI_FONT ),
              0 );

// Find out whether complete text fits in the button.      ---
// If not, then cut text and output ellipsis              ---
// ("...")                                                  ---
hDC = GetDC( g_Printers[ g_iNumPrinters ].hWnd );

// Select button font in DC -----
hF = SelectObject( hDC, GetStockObject( DEFAULT_GUI_FONT ) );

// Get length of printer name -----
g_Printers[ g_iNumPrinters ].iTextLen = strlen( pszDisplayName );

// Get dimensions of text within the button -----
GetTextExtentPoint( hDC,
                    pszDisplayName,
                    g_Printers[ g_iNumPrinters ].iTextLen,
                    &sText );

// Able to display complete text ? -----
if( sText.cx > BTN_WIDTH - 24 )
{
    SIZE sEllipsis;

    // Ellipsis required for output -----
    g_Printers[ g_iNumPrinters ].bEllipsis = TRUE;

    // Get length of ellipsis -----
    GetTextExtentPoint( hDC, "...", 3, &sEllipsis );

    // Continue reducing text length, until text + "..." fits in
    // button.
    do
    {
        g_Printers[ g_iNumPrinters ].iTextLen -= 1;
        GetTextExtentPoint( hDC,
                            pszDisplayName,
                            g_Printers[ g_iNumPrinters ].iTextLen,
                            &sText );
        } while( sText.cx + sEllipsis.cx > BTN_WIDTH - 24 );

    // Extend length of text by length of the ellipsis -----
    g_Printers[ g_iNumPrinters ].iTextLen += 3;
    }
    else g_Printers[ g_iNumPrinters ].bEllipsis = FALSE;

SelectObject( hDC, hF ); // Reselect old font
ReleaseDC( g_Printers[ g_iNumPrinters ].hWnd, hDC ); //Release DC

// Remove existing button icon -----
if( g_Printers[ g_iNumPrinters ].hIcon )
    DestroyIcon( g_Printers[ g_iNumPrinters ].hIcon );

// Get printer icon and allocate -----
SHGetFileInfo( (LPSTR)pidlPath,
                0,
                &sfi,
                sizeof(sfi),
                SHGFI_PIDL |
                SHGFI_ICON |
                SHGFI_SMALLICON |
                SHGFI_SYSICONINDEX |
                SHGFI_ICONLOCATION);

```

```

    g_Printers[ g_iNumPrinters ].hIcon = sfi.hIcon;

    // Does the button represent the default printer ? -----
    g_Printers[ g_iNumPrinters ].bPressed =
        !strcmpi( g_szDefault, pszDisplayName );

    g_iNumPrinters++;

    // Redraw button -----
    InvalidateRect( g_Printers[ g_iNumPrinters-1 ].hWnd ,NULL,TRUE);
    ShowWindow( g_Printers[ g_iNumPrinters-1 ].hWnd , SW_SHOW );
    UpdateWindow( g_Printers[ g_iNumPrinters-1 ].hWnd );
}
ClosePrinter( hPrinter );
}
return TRUE;
}

/*****
/* DeletePrinterButtons - Deletes all the printer buttons          */
/*-----*/
/* Parameters:      none                                          */
/* Return value : none                                          */
*****/
void DeletePrinterButtons( void )
{
    int i;
    for( i = 0; i < g_iNumPrinters; i++ )
    {
        // Destroy icon and window -----
        DestroyIcon( g_Printers[ i ].hIcon );
        DestroyWindow( g_Printers[ i ].hWnd );
        g_Printers[ i ].hWnd = 0;
    }
    g_iNumPrinters = 0;
}

/*****
/* FillWindowWithPrinters - Inserts buttons for all the available  */
/*                          printers in window.                    */
/*-----*/
/* Parameters:      hWnd - Handle of window that is to receive the */
/*                          buttons.                                */
/* Return value : none                                          */
*****/
void FillWindowWithPrinters( HWND hWnd )
{
    int i;
    RECT r;

    g_iNumPrinters = 0;

    // Enumerate all printers -----
    EnumFolders( CSIDL_PRINTERS, EnumShellCallback, (LPARAM)hWnd );

    // In case printers were removed in the meantime, now we have to
    // release the structures of the printers that are no longer required.
    i = g_iNumPrinters;
    while( ( i < MAX_PRINTERBUTTONS ) &&
        ( g_Printers[ i ].hWnd || g_Printers[ i ].hIcon ) )
    {
        // Destroy window and icon -----
        DestroyIcon( g_Printers[ i ].hIcon );
        g_Printers[ i ].hIcon = 0;
        DestroyWindow( g_Printers[ i ].hWnd );
        g_Printers[ i ].hWnd = 0;
        i++;
    }

    // Rearrange buttons -----
    GetWindowRect( hWnd, &r );
    SendMessage( hWnd,
        WM_SIZE,
        0,
        MAKELPARAM( r.right - r.left, r.bottom - r.top ) );
}

```

```

/*****
/* DrawBevel - Draw elevated or depressed rectangle */
/*-----*/
/* Parameters:   HDC           - DeviceContext, in which rectangle is to */
/*               be drawn                                           */
/*               lpRect        - rectangle to be drawn               */
/*               bPressed      - draw depressed (FALSE) or elevated (TRUE)*/
/*               rectangle.                                           */
/* Return value : none                                              */
/*****
void DrawBevel( HDC hDC, LPRECT lpRect, BOOL bPressed )
{
    HPEN hOuterLightPen,           // Pens for inner and outer
        hOuterDarkPen,           // rectangle
        hInnerLightPen,
        hInnerDarkPen,
        hOldPen;                 // For restoring the current pen

    // Create pens -----
    hOuterLightPen = CreatePen( PS_SOLID,
                                1,
                                bPressed ?
                                GetSysColor( COLOR_3DDKSHADOW ) :
                                GetSysColor( COLOR_3DHILIGHT ) );
    hInnerLightPen = CreatePen( PS_SOLID,
                                1,
                                bPressed ?
                                GetSysColor( COLOR_3DSHADOW ) :
                                GetSysColor( COLOR_3DFACE ) );

    hOuterDarkPen = CreatePen( PS_SOLID,
                                1,
                                bPressed ?
                                GetSysColor( COLOR_3DHILIGHT ) :
                                GetSysColor( COLOR_3DDKSHADOW ) );
    hInnerDarkPen = CreatePen( PS_SOLID,
                                1,
                                bPressed ?
                                GetSysColor( COLOR_3DFACE ) :
                                GetSysColor( COLOR_3DSHADOW ) );

    // draw outer rectangle -----
    hOldPen = ( HPEN )SelectObject( hDC, hOuterLightPen );
    MoveToEx( hDC, lpRect->left, lpRect->bottom - 2, NULL );
    LineTo( hDC, lpRect->left, lpRect->top );
    LineTo( hDC, lpRect->right - 1, lpRect->top );

    SelectObject( hDC, hOuterDarkPen );
    LineTo( hDC, lpRect->right - 1, lpRect->bottom - 1 );
    LineTo( hDC, lpRect->left - 1, lpRect->bottom - 1 );

    // draw inner rectangle -----
    hOldPen = ( HPEN )SelectObject( hDC, hInnerLightPen );
    MoveToEx( hDC, lpRect->left + 1, lpRect->bottom - 3, NULL );
    LineTo( hDC, lpRect->left + 1, lpRect->top + 1 );
    LineTo( hDC, lpRect->right - 2, lpRect->top + 1 );

    SelectObject( hDC, hInnerDarkPen );
    LineTo( hDC, lpRect->right - 2, lpRect->bottom - 2 );
    LineTo( hDC, lpRect->left - 2, lpRect->bottom - 2 );

    SelectObject( hDC, hOldPen );

    DeleteObject( hInnerLightPen );
    DeleteObject( hInnerDarkPen );
    DeleteObject( hOuterLightPen );
    DeleteObject( hOuterDarkPen );
}

/*****
/* PrinterWndProc - Default window procedure */
/*-----*/
/* Parameters:   Default parameters */

```

```

/* Return value : Default return value */
/*****
LRESULT CALLBACK PrinterWndProc( HWND    hWnd,
                                UINT     uMsg,
                                WPARAM   wp,
                                LPARAM    lp )
{
    switch( uMsg )
    {
        // Adapt size of window to the buttons -----
        case WM_SIZE:
        {
            int w,                                // Number of button columns
                h,                                // Number of button rows
                i;
            RECT r, rw;

            // How many buttons can be arranged horizontally?
            w = LOWORD( lp ) / BTN_WIDTH;

            // Define boundaries -----
            if( w == 0 ) w = 1;
            if( w > g_iNumPrinters ) w = g_iNumPrinters;

            // How many buttons can be arranged vertically? -----
            h = (g_iNumPrinters / w);

            // Are there any buttons left over that have to be placed in a -
            // new row? -
            if( g_iNumPrinters % w ) h++;

            // Calculate and set positions of buttons -----
            for( i = 0; i < g_iNumPrinters; i++ )
            {
                g_Printers[ i ].x = ( i % w ) * BTN_WIDTH;
                g_Printers[ i ].y = ( i / w ) * BTN_HEIGHT;
                g_Printers[ i ].cx = BTN_WIDTH;
                g_Printers[ i ].cy = BTN_HEIGHT;

                MoveWindow( g_Printers[ i ].hWnd,
                           g_Printers[ i ].x,
                           g_Printers[ i ].y,
                           g_Printers[ i ].cx,
                           g_Printers[ i ].cy,
                           TRUE );
            }

            // Calculate size of window's client area -----
            r.left = 0;
            r.top = 0;
            r.right = w * BTN_WIDTH;
            r.bottom = h * BTN_HEIGHT;

            // Include caption, borders and menu bar in calculation -----
            AdjustWindowRect( &r,
                             GetWindowLong( hWnd, GWL_STYLE ),
                             GetMenu( hWnd ) ? TRUE : FALSE );

            // Leave window at current location, but set ---
            // new size ---
            GetWindowRect( hWnd, &rw );
            MoveWindow( hWnd,
                       rw.left,
                       rw.top,
                       r.right-r.left,
                       r.bottom-r.top,
                       TRUE );
        }
        break;

        // A button must be redrawn -----
        case WM_DRAWITEM:
            // Check Command ID to determine whether a printer button needs -
            // to be drawn -
            if( ( (int)( LOWORD(wp) ) >= ID_PRINTERBUTTONS ) &&
                ( (int)( LOWORD(wp) ) < ID_PRINTERBUTTONS + g_iNumPrinters ) )

```

```

{
    LPDRAWITEMSTRUCT lpDIS;    // Information for drawing the button
    int               iButtonIndex;    // Which button?
    char              szText[ MAX_PATH ];    // Button text
    int               iOffset;
    RECT              r;

    lpDIS = ( LPDRAWITEMSTRUCT )lp;

    // Get index of printer button -----
    iButtonIndex = ( ( UINT ) wp ) - ID_PRINTERBUTTONS;

    // Save rectangle of area to be redrawn -----
    CopyRect( &r, &lpDIS->rcItem );

    // Draw elevated or depressed border -----
    DrawBevel( lpDIS->hDC,
                &lpDIS->rcItem,
                g_Printers[ iButtonIndex ].bPressed );

    // Move "inner life" of button ?
    iOffset = g_Printers[ iButtonIndex ].bPressed ? 1 : 0;

    // Draw icon of button -----
    DrawIconEx( lpDIS->hDC,
                2 + iOffset,
                2 + iOffset,
                g_Printers[ iButtonIndex ].hIcon,
                16,
                16,
                0,
                NULL,

    // Get rectangle for text output (subtract icon and border)  -
    r.right -= 2;
    r.left += 16 + 4;
    r.top += 2;
    r.bottom -= 2;

    // Get text to be output -----
    SendMessage( g_Printers[ iButtonIndex ].hWnd,
                WM_GETTEXT,
                sizeof( szText ),
                ( LPARAM ) szText );

    // Does the complete text fit in the button, or do we need  --
    // to end the text with an ellipsis  --
    if( g_Printers[ iButtonIndex ].bEllipsis )
    {
        szText[ g_Printers[ iButtonIndex ].iTextLen - 3 ] = '.';
        szText[ g_Printers[ iButtonIndex ].iTextLen - 2 ] = '.';
        szText[ g_Printers[ iButtonIndex ].iTextLen - 1 ] = '.';
    }

    // Output text -----
    ExtTextOut( lpDIS->hDC,
                r.left + iOffset,
                r.top + iOffset,
                ETO_CLIPPED | ETO_OPAQUE,
                &r,
                szText,
                g_Printers[ iButtonIndex ].iTextLen,
                NULL );
}
break;

// Default printer changed or printer removed/added? -----
case WM_WININICHANGE:
    if( ( lstrcmpi( (LPSTR)lp, "windows") != 0 ) &&
        ( lstrcmpi( (LPSTR)lp, "devices") != 0 ) )
        break;
    // Fallthrough!
case WM_CREATE:
    // Get name of default printer -----
    GetStdPrinter( g_szDefault, sizeof( g_szDefault ) );

```



```

    // Feed in all installed printers -----
    FillWindowWithPrinters( hWnd );
break;

// User clicked printer button to set a new -----
// default printer
case WM_COMMAND:
    if( ( (int)LOWORD(wp) ) >= ID_PRINTERBUTTONS ) &&
        ( (int)LOWORD(wp) ) < ID_PRINTERBUTTONS + g_iNumPrinters ) )
    {
        char szPrinter[ MAX_PATH ];
        int iButtonIndex;                // Index of clicked button
        int iOldPressed;                 // Index of button currently being pressed
        int i;
        HANDLE hPrinter;

        if( HIWORD( wp ) == BN_CLICKED )
        {
            iOldPressed = -1;
            iButtonIndex = ( (UINT) wp ) - ID_PRINTERBUTTONS;

            // Get index of old pressed button -----
            for( i = 0; i < g_iNumPrinters; i++ )
                if( g_Printers[ i ].bPressed ) iOldPressed = i;

            // Has a new button been clicked? -----
            if( iOldPressed != iButtonIndex )
            {
                // Get name of new printer to be set -----
                SendMessage( g_Printers[ iButtonIndex ].hWnd,
                    WM_GETTEXT,
                    sizeof( szPrinter ),
                    ( LPARAM ) szPrinter );

                // Open printer... -----
                if( OpenPrinter( szPrinter, &hPrinter, NULL ) )
                {
                    // ...and set as default printer -----
                    SetStdPrinter( hPrinter );
                    CloseHandle( hPrinter );
                }
                else
                    // Printer could not be opened
                    MessageBox( hWnd,
                        "Error opening printer!",
                        "Error",
                        0 );
            }
        }
    }
break;

// Destroy window, buttons and icons -----
case WM_CLOSE:
    ShowWindow( hWnd, SW_HIDE );
    return 0;

case WM_DESTROY:
    DeletePrinterButtons();
    PostQuitMessage( 0 );
break;
}
return DefWindowProc( hWnd, uMsg, wp, lp );
}

/*****
/* CreateStdPrinterWindow - Create default printer control window */
/*-----*/
/* Parameters:      hInst - Instance of application */
/* Return value : Handle of window */
/*****
HWND CreateStdPrinterWindow( HINSTANCE hInst )
{
    WNDCLASS wc;
    HWND hWnd;

```

```

// Initialize window class -----
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfWndProc = PrinterWndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInst;
wc.hIcon = LoadIcon( hInst, MAKEINTRESOURCE( IDI_APPLICATION ) );
wc.hCursor = LoadCursor( NULL, IDI_APPLICATION );
wc.hbrBackground = (HBRUSH)COLOR_BACKGROUND;
wc.lpszMenuName = NULL;
wc.lpszClassName = MY_CLASS;

RegisterClass( &wc ); // Register window class -----

// Zero out button array -----
memset( g_Printers, 0, sizeof( g_Printers ) );

hWnd = CreateWindowEx( WS_EX_PALETTEWINDOW,          // Create window
                      MY_CLASS,
                      "PC-Intern",
                      WS_THICKFRAME |
                      WS_SYSMENU,
                      0,0,
                      0,0,
                      NULL,
                      0,
                      hInst,
                      0L );

return hWnd;
}

/*****
/* TaskbarWndProc - Default window procedure */
/* Expanding by editing of TaskBar messages */
/* via user-defined message */
/* WM_MYTASKBAR */
/*-----*/
/* Parameters: Default parameters */
/* Return value : Default return value */
/*-----*/
/* Info : The Taskbar communicates with the application through this */
/* invisible window. */
*****/
LRESULT FAR PASCAL TaskbarWndProc( HWND hWnd,
                                   UINT msg,
                                   WPARAM wp,
                                   LPARAM lp )
{
    static HWND hPrinter; // Handle of default printer selection window
                          // Set in WM_CREATE and then always
                          // available

    switch( msg )
    {
        case WM_CREATE: // Create default printer selection window
            hPrinter = CreateStdPrinterWindow( (HINSTANCE)
                                                GetWindowLong( hWnd,
                                                                GWL_HINSTANCE ) );
            break;
        case WM_WININICHANGE: // Tooltip = current default printer
            GetStdPrinter( g_szPrinter, sizeof( g_szPrinter ) );
            TaskBarSetToolTip( hWnd, TB_PRINTER_ID, g_szPrinter );
            break;

        // Edit messages of Taskbar icon -----
        case WM_MYTASKBAR:
            switch( LOWORD(wp) )
            {
                case TB_PRINTER_ID: // My Task icon
                    switch( LOWORD(lp) ) // Which mouse message?
                    {
                        case WM_LBUTTONDOWN: // Left double-click
                            // Display default printer selection and in foreground --
                            ShowWindow( hPrinter, SW_SHOW );
                            BringWindowToTop( hPrinter );
                            break;
                    }
            }
    }
}

```

```

        case WM_RBUTTONDOWN:
            PostMessage( hWnd, WM_MYCONTEXTMENU, 0, 0 );
            break;
    }
    break;
}
break;
case WM_COMMAND:                                // Edit menu command
    if( LOWORD( wp ) == IDM_REMOVE )
    {
        DestroyWindow( hPrinter );           // Destroy default printer window
        DestroyWindow( hWnd );                // Destroy myself
    }
    break;
case WM_MYCONTEXTMENU:                            // Display Context menu
{
    HMENU hMenu;
    POINT p;

    // Create Popup menu -----
    hMenu = CreatePopupMenu();

    // Insert only "Remove" menu item -----
    AppendMenu( hMenu,
                MF_STRING | MF_ENABLED,
                IDM_REMOVE,
                "Remove" );

    GetCursorPos( &p );                        // Get current mouse position...

    // Window must be active so that it can process Popup menu -
    // messages correctly! -
    SetActiveWindow( hWnd );

    //... to display the menu there -----
    TrackPopupMenu( hMenu,
                    TPM_RIGHTBUTTON,
                    p.x,
                    p.y,
                    0,
                    hWnd,
                    NULL );

    DestroyMenu( hMenu );                      // Delete menu
}
break;
case WM_DESTROY:                                // Window has been destroyed
    PostQuitMessage( 0 );                      // End application
    break;
}
return DefWindowProc( hWnd, msg, wp, lp );
}

```

```

/*****
/* WinMain - Start function */
/*-----*/
/* Parameters:    Default parameters */
/* Return value : Default return value */
/*****
int WINAPI WinMain( HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpszCmdLine,
                   int        nCmdShow )
{
    HWND     hTaskBarController;    // Handle of Taskbar controller window
    WNDCLASS wc;

    wc.style          = 0;           // Initialize window class of
    wc.lpfnWndProc    = TaskbarWndProc; // Taskbar controller
    wc.cbClsExtra     = 0;
    wc.cbWndExtra     = 0;
    wc.hInstance      = hInstance;
    wc.hIcon          = NULL;
    wc.hCursor        = NULL;
    wc.hbrBackground  = NULL;
    wc.lpszMenuName   = NULL;

```

```

wc.lpszClassName = TBCLASS;

if( RegisterClass( &wc ) )
{
    // Create invisible Taskbar controller window -----
    hTaskBarController = CreateWindow( TBCLASS, "", 0,0,0,0,0,0,
                                      NULL, hInstance, 0 );

    if( hTaskBarController )
    {
        MSG msg;

        // Load printer icon -----
        HICON hTaskBarIcon = LoadIcon( hInstance,
                                       MAKEINTRESOURCE( IDI_ICON ) );

        CoInitialize(NULL);           // Initialize OLE (for EnumFolders)

        // Get default printer -----
        GetStdPrinter( g_szPrinter, sizeof( g_szPrinter ) );

        // Add Taskbar icon to Taskbar -----
        TaskBarAddIcon( hTaskBarController,
                       WM_MYTASKBAR,
                       TB_PRINTER_ID,
                       hTaskBarIcon,
                       g_szPrinter );

        DestroyIcon( hTaskBarIcon );           // Icon no longer required

        // Message loop -----
        while( GetMessage( &msg, 0,0,0 ) )
        {
            TranslateMessage( &msg );
            DispatchMessage( &msg );
        }
        // Remove Taskbar icon -----
        TaskBarDeleteIcon( hTaskBarController, TB_PRINTER_ID );

        CoUninitialize();               // Uninstall OLE
    }

    // Remove window class -----
    UnregisterClass( TBCLASS, hInstance );
}
return 0;
}

```