Windows XP Technical Articles

Using Windows XP Visual Styles

Windows User Experience Team
Microsoft Corporation

May 2001

This is preliminary documentation and is subject to change.

Summary: This document describes how to use Microsoft Windows XP to perform the more common tasks that are necessary to apply visual styles to your applications. (13 printed pages)

Contents

Introduction

Using Microsoft® Windows® XP, you can now define the visual style or appearance of controls and windows from simple colors to textures and shapes. You can control each defined part of a control as well as each part of the non-client (frame and caption) area of a window. The user can then use the Appearance tab in the Windows Control Panel to switch between the classic visual style and other available styles.

A visual style is included with the Windows XP release. Using helper libraries and application programming interfaces (APIs), you can incorporate the Windows XP visual style into your application with few code changes. For more information, see the Platform SDK documentation in the MSDN Library.

ComCtl32.dll Version 6

All applications running on the Windows XP operating system have a non-client area, which includes the window frame and non-client scrollbars. A visual style is applied to the non-client area by default. This means that the appearance of the non-client area is specified by the visual style that is currently installed. To apply a visual style to common controls in the client area, you must use ComCtl32.dll version 6 or later. Unlike earlier versions of ComCtl32.dll, version 6 is not redistributable. The only way you can use version 6 of the dynamic-link library (DLL) is to use an operating system that contains it. Windows XP ships with both version 5 and version 6. ComCtl32.dll version 6 contains both the user controls and the common controls. By default, applications use the user controls defined in User32.dll and the common controls defined in ComCtl32.dll version 5.

If you want your application to use visual styles, you must add an application manifest that indicates that ComCtl32.dll version 6 should be used if it is available. Version 6 includes some new controls and new options for other controls, but the biggest change is support for changing the appearance of controls in a window.

Visual Style Tasks

To add visual styles to your controls, you might need to perform some of the following tasks.

Using ComCtl32.dll Version 6 in an Application That Does Not Use Third-Party Extensions

The following are examples of applications that do not use third-party extensions.

- Calculator
- FreeCell
- Minesweeper
- Notepad
- Solitaire

To create a manifest and enable your application to use visual styles.

1. Link to ComCtl32.lib and call **InitCommonControls** (see the Platform SDK documentation in the MSDN Library).
2. Add a file called YourApp.exe.manifest to your source tree that has the following XML format:
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
version="1.0.0.0"
processorArchitecture="X86"
name="CompanyName.ProductName.YourApp"
type="win32"
/>
<description>Your application description here.</description>
<dependency>
<dependentAssembly>
<assemblyIdentity
type="win32"
name="Microsoft.Windows.Common-Controls"
version="6.0.0.0"
processorArchitecture="X86"
publicKeyToken="6595b64144ccf1df"
language="*"
/>
</dependentAssembly>
</dependency>
</assembly>
```
3. Add the manifest to your application's resource file as follows:
```
CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST "YourApp.exe.manifest"
```
   Note When you add the previous entry to the resource you must format it on one line. Alternatively, you can place the XML manifest file in the same directory as your application's executable file. The operating system will load the manifest from the file system first, and then check the resource section of the executable. The file system version takes precedence.

Using ComCtl32 Version 6 in an Application That Uses Extensions, Plugins, or a DLL That is Brought into a Process

The following are examples of applications that use extensions.

- Microsoft Management Console (Mmc.exe)
- Windows Shell
- Microsoft® Visual Studio®

To create a manifest and enable your application to use visual styles.

1. Use the Windows XP Beta 2 SDK or later.
2. Include the common controls header file as follows:
```
#include "commctrl.h"
```
3. Define a compiler variable preprocessor definition as follows:
```
#define SIDEBYSIDE_COMMONCONTROLS 1
```

Add a file called YourApp.manifest to your source tree that has the following XML format:
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
version="1.0.0.0"
processorArchitecture="X86"
name="CompanyName.ProductName.YourApp"
type="win32"
/>
<description>Your application description here.</description>
<dependency>
<dependentAssembly>
<assemblyIdentity
type="win32"
name="Microsoft.Windows.Common-Controls"
version="6.0.0.0"
processorArchitecture="X86"
publicKeyToken="6595b64144ccf1df"
language="*"
/>
</dependentAssembly>
</dependency>
</assembly>
```
4. Add the manifest to your application's resource file as follows:
```
CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST "YourApp.manifest"
```
Winuser.rh includes the following defines:
```
#define CREATEPROCESS_MANIFEST_RESOURCE_ID 1
#define CONTROL_PANEL_RESOURCE_ID 123
#define RT_MANIFEST 24
```

Using ComCtl32 Version 6 in Control Panel or a DLL That is Run by RunDll32.exe

To create a manifest and enable your application to use visual styles.

1. Link to ComCtl32.lib and call **InitCommonControls**.
2. Add a file called YourApp.cpl.manifest to your source tree that has the following XML format:
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
version="1.0.0.0"
processorArchitecture="X86"
name="CompanyName.ProductName.YourApp"
type="win32"
/>
<description>Your application description here.</description>
<dependency>
<dependentAssembly>
<assemblyIdentity
type="win32"
name="Microsoft.Windows.Common-Controls"
version="6.0.0.0"
processorArchitecture="X86"
publicKeyToken="6595b64144ccf1df"
language="*"
/>
</dependentAssembly>
</dependency>
</assembly>
```

3. Add the manifest to your application's resource file as follows:
   CONTROL_PANEL_RESOURCE_ID RT_MANIFEST "YourCpl.manifest"

   Note When you author a control panel application, place it in the appropriate category. Control Panel now supports categorization of control panel applications. This means that control panel applications can be assigned identifiers and separated into task areas such as Add or Remove Programs, Appearance and Themes, or Date, Time, Language, and Regional Options.

Using Visual Styles with a Custom Control

The UxTheme API referenced in the steps to add visual styles to a control and in the corresponding code examples are defined in the header file, Uxtheme.h. The elements of the API are documented in the Platform SDK. In addition to describing the steps for applying visual styles to a control, this section provides a drawing code sample and tips on drawing controls.

To enable a control to apply visual styles.

1. Call **OpenThemeData** and pass the hwnd of the control you wish to apply visual styles to and a class list that describes the control's type. The classes are defined in Tmschema.h. **OpenThemeData** returns an HTHEME handle, but if the visual style manager is disabled or the current visual style does not supply specific information for a given control, the function returns NULL. If the return value is NULL use non-visual style drawing functions.
2. To draw the control, call **DrawThemeBackground** and pass the following:
   - Theme handle returned from **OpenThemeData**, the HDC to use for rendering the control.
   - Part identifier that describes the part of the control to render. See Theme Parts and States for information on control parts and states.
   - State identifier that describes the current state of the part.
   - Pointer to the RECT structure that contains the coordinates of the rectangle where the control will be rendered.
3. Some parts can be partially transparent. You can determine this by calling **IsThemeBackgroundPartiallyTransparent** with the theme handle, a control part, and a control state.
4. If your control draws text, position the text within the content rectangle of the control and select a font.
   - To determine the location of the content rectangle call **GetThemeBackgroundContentRect**
   - Add your desired font to the device context (DC) then call **DrawThemeText**. This function enables visual effects such as shadowed text in some controls.
5. When your control receives a WM_THEMECHANGED message, it should do the following:
   - Call **CloseThemeData** to close the existing theme handle.
   - Call **OpenThemeData** to get the theme handle for the newly loaded visual style.
   - This code sample illustrates the two calls.

```
case WM_THEMECHANGED:
   CloseThemeData (hTheme);
   hTheme = OpenThemeData (hwnd, L"MyClassName");
```

6. When your control receives a WM_DESTROY message, call **CloseThemeData** to release the theme handle that was returned when you called **OpenThemeData**.

Example of Drawing Code

The following code sample demonstrates how to draw a button control.

```
HTHEME hTheme = NULL;
hTheme = OpenThemeData(hwndButton, "Button");
...
DrawMyControl(hDC, hwndButton, hTheme, iState);
...
if (hTheme)
{
    CloseTheme(hTheme);
}

void DrawMyControl(HDC hDC, HWND hwndButton, HTHEME hTheme, int iState)
{
    RECT rc, rcContent;
    TCHAR szButtonText[255];
    HRESULT hr;

    GetWindowRect(hwndButton, &rc);
    GetWindowText(hwndButton, szButtonText,
                  ARRAYSIZE(szButtonText));

    if (hTheme)
    {
        hr = DrawThemeBackground(hTheme, hDC, BP_BUTTON,
                iState, &rc, 0);
        // Always check your result codes.

        hr = GetThemeBackgroundContentRect(hTheme,
                BP_BUTTON, iState, &rc, &rcContent);
        hr = DrawThemeText(hTheme, hDC, BP_BUTTON, iState,
                szButtonText, lstrlen(szButtonText),
                DT_CENTER | DT_VCENTER | DT_SINGLELINE,
                0, &rcContent);
    }
    else
    {
        // Draw the control without using visual styles.
    }
}
```

Enabling Owner Drawn Controls to Use Visual Styles

With Windows XP controls no longer have to consist only of lines and color fills. Now they include rich textures and patterns that can vary according to the state of the control. This means that you cannot use already existing programming elements to describe owner-drawn controls that have the look and feel of controls that have a visual style applied. If you have code that already has owner-drawn controls, you have the following options.

- You can call **SetWindowTheme** to turn off theming on the control.

  ```
  SetWindowTheme (hwndButton, TEXT (" "), TEXT (" "));
  ```

  The control will render as it did in earlier versions of Windows.
- You can continue to use the owner-drawn methods in controls, but some of the properties will be ignored. For example, if you change the background color of a button, the button is drawn like a button with a visual style applied but the background color is not the color you specified. However if you change the font, the button is drawn with the specified font.

Making a Control Appear to Have No Visual Style in a Dialog Box or Window That Has a Visual Style

In some cases, applications have custom-drawn controls that should not use a visual style, even when the other controls in the window do have a visual style applied. If you want to mark any custom-drawn control as having no visual style, you must call **SetWindowTheme** and pass in the window handle for the control and an empty string for both the *pszSubAppName* and *pszSubIdList* parameters. Calling the **SetWindowTheme** function with empty strings causes the control to render without a visual style appearance. The following code snippet shows how to create a button control and then call **SetWindowTheme** to remove the visual style from the button.

```
HWND hwndButton;

hwndButton = CreateWindow (TEXT ("button"), ...);
if (hwndButton)
{
   SetWindowTheme (hwndButton, TEXT (" "), TEXT (" "));
}
```

Note If you decide to have the control change at run time to a visual style control you can call **SetWindowTheme** but pass NULL for the two string parameters.

Using UxTheme Manager to Render a Control Whose Parts Do Not Have a Visual Style

If you have a control that is not one of the common controls and you want it to look appropriate on a computer that has visual style files installed, you can take one of the following approaches.

- The basic approach is to use the system metric colors. Most system metric colors are set when a visual style file is applied.
- You can use parts from other controls and render each part separately. For example, for a calendar control that consists of a grid, you can treat each square formed by the grid as a toolbar button. Do the following to program the squares as toolbar buttons.
  Call **OpenThemeData** as follows:

  ```
  OpenThemeData (hwnd, "Toolbar");
  ```

  Use the returned theme handle to render each of the squares on the calendar.
- You can mix and match control parts by calling **OpenThemeData** multiple times for a given control and using the appropriate theme handle to draw different parts. In some visual styles, however, certain parts might not be compatible with other parts.
- Some specialized applications that need to match the Windows XP look above the control level might need to detect the currently loaded visual style and have custom renderings designed to match the look. This can be useful for applications that provide visual style architecture.

Using Visual Styles with HTML Content

There are many applications, written in HTML and deployed as an HTML Application (HTA) or a Win32 application, that use HTML or host the WebObject in their UI elements. When running on Windows XP, these applications can have the same look that is consistent with Win32-based applications running on the same operating system. The following are some considerations when applying visual styles to HTML content.

- Visual styles in relation to HTML content applies only to Intrinsic HTML Controls such as buttons, scrollbars, and select controls. The Windows XP visual style is applied automatically to controls on HTML pages. If you do not want your page to have a visual style applied, use the META tag with its attribute set to NO. The META tag is documented in a following section.
- HTML pages that modify the Cascading Style Sheets (CSS) properties such as background or border do not have a visual style applied to them. They display the specified CSS attribute.
- When specified as part of the content, most CSS properties do apply to elements that have visual styles applied.
- You must add a metatag to the <head> section of your pages. You need to add this tag only once; visual styles apply it to all the page content. This also applies to content packaged as HTAs. The metatag must be as follows:

  ```
  <META HTTP-EQUIV="MSThemeCompatible" CONTENT="Yes">
  ```

- Be aware that visual styles might change the layout of your content. Also, if you set certain attributes on Intrinsic HTML Controls, such as the width of a button, you might find that the label on the button is unreadable under certain visual styles.
- You must thoroughly test your content using visual styles to determine whether applying visual styles has an adverse impact on your content and layout. Make any changes required to make your content work successfully.

Causing the UxTheme Manager to Ignore Top-Level Windows

To avoid applying the new visual style to a top-level window, consider the following:

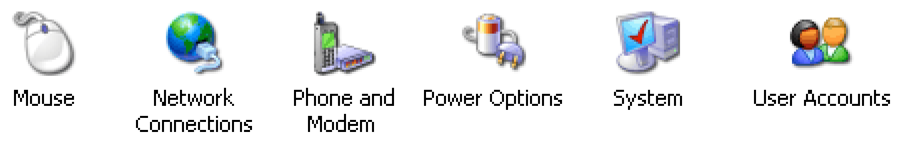
- If a window has a region applied to it, the UxTheme Manager assumes that this is a specialized window and the window will not use visual styles throughout its life. Child windows associated with a non-visual style top-level window may still apply visual styles even though the parent does not.
- If you apply a region to a top-level window and later remove the region, a visual style is not automatically applied to the window. To apply a visual style to the window, you must create a new window.

- If you want to disable the use of visual styles for all top-level windows in your application, call **SetThemeAppProperties** and do not pass the STAP_ALLOW_NONCLIENT flag.
- If an application does not call **SetThemeAppProperties**, the assumed flag values are STAP_ALLOW_NONCLIENT | STAP_ALLOW_CONTROLS | STAP_ALLOW_WEBCONTENT. The assumed values cause the non-client area, the controls, and Web content to have a visual style applied.

Using 32 Bit Anti-Aliased Icons

Windows XP imagelists, which are collections of images used with certain controls such as list-view controls, support the use of 32-bit anti-aliased icons and bitmaps. Color values use 24 bits, and 8 bits are used as an alpha channel on the icons. To create an imagelist that can handle a 32-bits-per-pixel (bpp) image, call the ImageList_Create function passing in an ILC_COLOR32 flag.

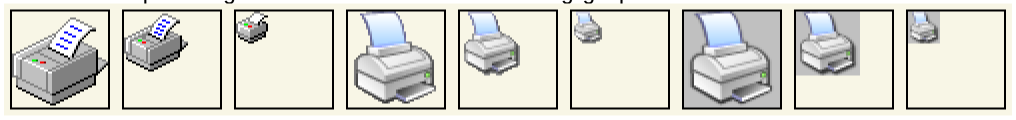
The following graphic shows that the icon format has not changed, but the way in which you author these icons has changed.



Icon Format

To author 32-bit icons correctly, do the following:

- Author multiple images for each icon. The following graphic shows this.



Multiple Images of the Same Icon

- In the previous graphic the first three images are in 16-color mode for use in safe mode.
- The next three icons are used in Windows XP 256-color mode.
- The last three icons have the alpha channel and can be used only in Windows XP or future operating systems running 24-bit color or higher.
- The order of images in the icon format does matter. If the order is wrong, older versions of Windows function poorly when extracting the icons. Extracting the icons incorrectly can cause memory corruption and improper rendering.
- Previous versions of Windows had a 10-icon resource limit, but Windows XP supports thousands of icon resources.

Note You can use third-party tools to generate icon files and bitmaps that contain an alpha channel.

Shipping Your Application Both on Windows XP and Earlier Versions of Windows

Much of the Windows XP visual style architecture is designed to make it simple to continue to ship your product on earlier versions of Windows that do not support changing the appearance of controls. When shipping an application for more than one operating system, be aware of the following:

- Installing your application's manifest in earlier versions will not affect the rendering of controls.
- You must test your application to make sure you are not relying on features of ComCtl32.dll version 6 without first checking for the current version.
- Do not link directly to the UxTheme.lib. If you use the UxTheme API to add visual styles to custom controls, load the library on demand.
- Write error-handling code for instances when visual styles do not work as expected.

- If you use the features in ComCtl32.dll version 6, such as the tile-view or link control, you must handle the case where those controls are not available on your user's computer. ComCtl32.dll version 6 is not redistributable.

Summary

This document described tasks that are necessary to apply visual styles to your applications. The document does not include all the tasks you may need to perform but rather discusses some of the more common ones.