How can I prevent the keyboard focus rectangle from appearing on a control I created?

devblogs.microsoft.com/oldnewthing/20171206-00

December 6, 2017

Raymond Chen

A customer wanted to prevent keyboard focus rectangles from appearing on controls they have created, like button controls. Presumably the reason is that they have customized the control's appearance to the point where the focus rectangle is no longer necessary. Drawing a focus rectangle in addition to the customized focus appearance would be redundant.

Recall how the focus indicator is managed: The system sends `WM_UPDATEUISTATE` messages to each control to tell it to alter its appearance and either add or remove keyboard focus and accelerator indicators.

Therefore, if you don't want a control to show keyboard focus indicators, you can intercept this message before it reaches the control and prevent it from learning about the change in state.

```
#define STRICT
#include <windows.h>
#include <windowsx.h>
#include <commctrl.h>

LRESULT CALLBACK AlwaysHideFocusRectangleSubclassProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam,
    UINT_PTR idSubclass, DWORD_PTR dwRefData)
{
 switch (uMsg) {
 case WM_NCDESTROY:
  RemoveWindowSubclass(hwnd,
   AlwaysHideFocusRectangleSubclassProc, idSubclass);
 break;

 case WM_UPDATEUISTATE:
  // do not let them modify UISF_HIDEFOCUS
  wParam &= ~MAKELONG(0, UISF_HIDEFOCUS);
  break;
 }
 return DefSubclassProc(hwnd, uMsg, wParam, lParam);
}

INT_PTR CALLBACK DialogProc(HWND hdlg, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 case WM_INITDIALOG:
  SendDlgItemMessage(hdlg, IDCANCEL, WM_UPDATEUISTATE,
    MAKELONG(UIS_SET, UISF_HIDEFOCUS), 0);
  SetWindowSubclass(GetDlgItem(hdlg, IDCANCEL),
    AlwaysHideFocusRectangleSubclassProc, 0, 0);
  return TRUE;

  case WM_COMMAND:
   if (GET_WM_COMMAND_ID(wParam, lParam) == IDCANCEL) {
    EndDialog(hdlg, 0);
   }
   break;
  }
  return FALSE;
}

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
    LPSTR lpCmdLine, int nShowCmd)
{
 DialogBox(hinst, MAKEINTRESOURCE(1), nullptr, DialogProc);
 return 0;
}

// resource file
#include <windows.h>
```

```
1 DIALOGEX 0, 0, 305, 280
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION |WS_SYSMENU
BEGIN
  DEFPUSHBUTTON "Button &1", IDOK, 0,0,50,50
  PUSHBUTTON "Button &2", IDCANCEL, 0,60,50,50
END
```

This sample program creates a dialog box with two buttons. Button 1 behaves normally, but we customized the focus indicators for Button 2:

When the dialog box initializes, we tell Button 2 to set the `UISF_HIDEFOCUS` flag in its UI state. Note that this creates a window tree with an inconsistent UI state, which would normally get you into trouble, but let's play this out.

After forcing the focus rectangle to be hidden on Button 2, we subclass the button, and in our subclass function, we clear the `UISF_HIDEFOCUS` flag when we receive the `WM_UPDATE-UISTATE` message, which blocks all attempts to initialize, set, or clear that state. This means that the focus rectangle will remain hidden on Button 2.

When you run this program, you'll see that the focus rectangle won't appear on Button 2 even if it is enabled on Button 1.

Notice that we did not block the `WM_CHANGEUISTATE` message. This means that if focus is on Button 2 and the user hits the `Tab` key to call up focus indicators, the request to show indicators will still bubble out to the dialog box root, and then propagate down to all the dialog box controls as a `WM_UPDATEUISTATE` message. Button 1 will honor the request, but Button 2 will ignore it. This is probably the behavior the user wants: They asked for focus indicators, so we will show focus indicators in all the controls that wish to support them.

Raymond Chen

Follow