Windows Embedded Compact

Window Compositor Developer’s Guide

Writer: Mark Horowitz

Technical Reviewer: Jay Treptow

Published: January 2012

Applies To: Windows Embedded Compact 7

Abstract

This paper describes how to use the Window Compositor to configure and manage multiple application windows on the device display. It covers:

- Alpha blending (transparency)
- Graphics device interface (GDI) API
- OpenGL for Embedded Systems API

Contents

Introduction 3
  How Window Compositor Works 3
  Alpha Blending Procedure 4
  Turning Off Output for Covered Windows 5

Window Compositor Alpha Blending Methods 6

Compositor APIs 7

Include Compositor in an OS Design 8
  Add Window Compositor from Platform Builder 8
  Add Window Compositor from the Command Line 9

Conclusion 10

Additional Resources 10

Introduction

Windows Embedded Compact developers can use an optional module, Window Compositor, to configure and manage the display of output windows from multiple applications on a single device display screen. Window Compositor uses a process called alpha blending to create a user experience in which overlapping windows appear partially transparent so that content on a covered window is fully or partially visible through the window that covers it. Alpha blending combines the graphical output from multiple applications to create this transparency effect. Window Compositor performs alpha blending by combining the color values for pixels in the overlapping and overlapped windows with a specified alpha value that determines the degree of transparency between the overlapping and overlapped windows.

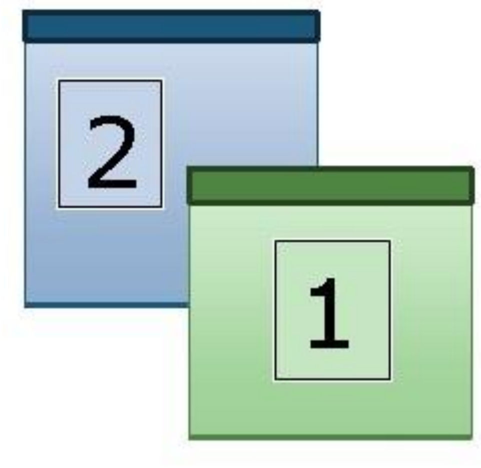
As part of its application programming interface (API), Window Compositor includes several functions that manage the alpha blending process, and with the API you can determine whether Window Compositor performs alpha blending for entire windows or for individual pixels within each window. Window Compositor also includes APIs and a Windows message (WM_OBSCURED (http://go.microsoft.com/fwlink/?LinkId=198384)) that applications can use to control the behavior and appearance of application windows. You add Window Compositor to an OS design by setting the appropriate SYSGEN and optional board support package (BSP) variables before building a BSP.

How Window Compositor Works

In a Windows Embedded Compact system, the Graphics, Windowing and Events Subsystem (GWES) (http://go.microsoft.com/fwlink/?linkid=198409) is the interface between applications and the system's video drivers. Each application that produces visual output updates its own display window through GWES as if it were the only application that writes to the device display. GWES responds to each application display request as it receives the request and then uses the driver to update the device display. This process is called updating the primary surface.

If two windows on a device overlap, GWES clips the underlying window area by updating only the primary display for the top window. As a result, the underlying window's output is hidden as illustrated by window 2 in Figure 1.

Figure 1: Window 1 clips Window 2



If you want overlapping application windows to be fully or partially transparent, you must add Window Compositor and its alpha blending features to the OS design. Window Compositor works with GWES to provide alpha blending between the overlapping windows. For instructions on adding Window Compositor to an OS, see Include Compositor in an OS Design.

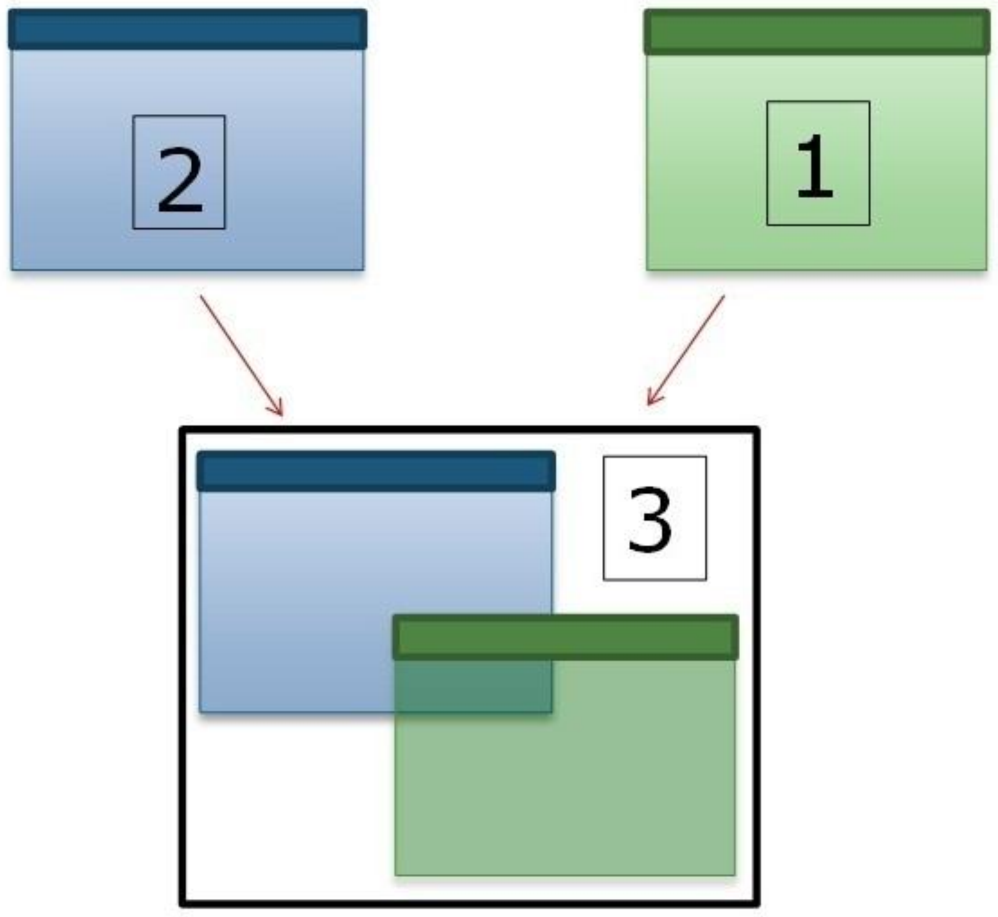
Alpha Blending Procedure

Window Compositor uses the following procedure to perform alpha blending.

1. When Window Compositor starts, GWES creates one virtual display surface for Window Compositor to use as a canvas to perform window composition, which is the process of combining the visual outputs of the multiple applications onto a single output display. This virtual display surface is called the compositor back surface.
2. As each application window is created, Window Compositor creates a virtual display surface for the window. This display surface is the window’s back surface.
3. When the application calls GWES to draw the window, GWES updates the window’s back surface instead of the primary surface. Because this back surface is exclusive to the application window, GWES does not do any clipping, which leaves the window’s back surface as a complete representation of its output.
4. Window Compositor combines all of the application back surfaces into its own back surface and then updates the primary surface with the resulting combination. To achieve opacity, use Window Compositor APIs to set opacity values either for an entire back surface or for individual pixels in the back surface.

Figure 2 illustrates the alpha blending of two windows (1 and 2), each with its own back surface, using the compositor back surface (3).

Figure 2: Alpha blending



1, 2: Back surfaces
3: Compositor back surface

Turning Off Output for Covered Windows

If one application window completely covers another application window, you can improve graphics performance by ensuring that the application with the covered output window turns off output until that window is uncovered. To do this, Window Compositor sends a **WM_OBSCURED** window message to inform an application when its display window is completely covered and then sends a second **WM_OBSCURED** message if any part of the window area (including the window’s borders and title bar) is uncovered. Window Compositor considers a window uncovered if the covering window has any level of transparency that results from alpha blending.

Window Compositor Alpha Blending Methods

The Window Compositor APIs use several methods, shown in the following table, to configure the degree of opacity for application windows. A window can be configured to have a single global opacity value for the entire window, or it can be configured so that opacity can be set for individual pixels.

The following table lists and describes Window Compositor methods for configuring opacity by using alpha blending.

Compositor method	Description
SetWindowCompositionFlags (http://go.microsoft.com/fwlink/?linkid=198373)	Defines the ability of the back surface of an application to set a per-pixel opacity value, or *alpha value*. To configure the back surface for a 24-bit-per-pixel (bpp) color value and an 8-bit alpha value, call this method with the **WCF_ALPHATRANSPARENCY** and **WCF_TRUECOLOR** flags set in the *dwFlags* parameter. Then the hexadecimal value for each pixel is defined as the following: *0xaarrggbb* In this hexadecimal value: • *aa* represents the alpha value of the pixel. • *rr* represents the intensity of red in the pixel. • *gg* represents the intensity of green in the pixel. • *bb* represents the intensity of blue in the pixel. To set the alpha value for each pixel, do the following: 1. Call the CreateSolidBrush (http://go.microsoft.com/fwlink/?linkid=198374) function to create a solid brush, and specify the 8-bit alpha value in the *aa* hex digits and 0 the for the hex digits *rrggbb* in the function’s *crColor* parameter. 2. Use the brush you created in the call to CreateSolidBrush (http://go.microsoft.com/fwlink/?linkid=198374) to paint the area that receives the back

Compositor method	Description
	surface opacity.
SetWindowOpacity (http://go.microsoft.com/fwlink/?linkid=198383)	Sets an opacity value that Window Compositor uses for the entire window when it composes its back surface.
GetWindowCompositionFlags (http://go.microsoft.com/fwlink/?linkid=198375)	Retrieves the window composition flags.
GetWindowOpacity (http://go.microsoft.com/fwlink/?linkid=198376)	Retrieves the window opacity value.

If you use XAML to create Silverlight for Windows Embedded device applications, you cannot use XAML attributes to set the opacity levels. Instead, you must call the IXRVisualHost::GetContainerHWND (http://go.microsoft.com/fwlink/?linkid=198439) method to obtain the window handle and then pass this handle to the **SetWindowCompositionFlags** or **SetWindowOpacity** function.

Compositor APIs

Window Compositor supports two graphics API libraries: graphics device interface (GDI) and OpenGL for Embedded Systems (OpenGL ES) 2.X. The system configuration determines the graphics API that Window Compositor uses to compose application back surfaces onto its own. Window Compositor has no effect on the API that each application uses to display output. The system must still provide the driver that supports the graphics APIs that each application calls.

When Window Compositor uses the GDI APIs to update its back surface, it's using the *GDI compositor*. When Window Compositor uses the OpenGL ES APIs to update its back surface, it's using the *OpenGL compositor*.

The compositor you choose to use depends on the features you need for the device. The following table compares the features in the two compositors.

GDI compositor	OpenGL ES compositor
Provides software alpha blending when it is not available from the system hardware.	Provides hardware alpha blending between system windows.
Can use hardware-accelerated bit blitting (copying large blocks of pixels from one part of graphics memory to another).	Can use hardware-accelerated bit blitting.

GDI compositor	OpenGL ES compositor
Is better suited for devices without a graphics processing unit (GPU).	Is better suited for devices with a GPU.
Does not need hardware acceleration.	Uses hardware acceleration to improve performance.

Include Compositor in an OS Design

To include Window Compositor in your OS, you must configure the required SYSGEN variable **SYSGEN_COMPOSITION** and the required board support package (BSP) environment variable **BSP_GLES2COMPOSITOR** before you create the BSP. For information about **SYSGEN** variables, see GWES Catalog Items and Sysgen Variables (http://go.microsoft.com/fwlink/?LinkId=198380). For information about BSP environment variables, see BSP Environment Variables (http://go.microsoft.com/fwlink/?LinkId=198382).

The following table defines the required settings.

Variable/Setting	Definition
SYSGEN_COMPOSITION = 1	Adds Window Compositor functionality to the BSP and adds the default GDI compositor to the OS.
BSP_GLES2COMPOSITOR = 1	Adds the OpenGL compositor instead of the GDI compositor to the OS. SYSGEN_COMPOSITION must also be set to 1 or Platform Builder ignores this setting.

Add Window Compositor from Platform Builder

Platform Builder offers a standard user interface (UI) to set required SYSGEN and BSP variables and to include Window Compositor in the OS build. After you create a project in Platform Builder, use the following steps to set the SYSGEN_COMPOSITION and BSP_GLES2COMPOSITOR variables and to add Window Compositor to the OS.

To add the GDI compositor to the OS from Platform Builder

1. In the **Solution Explorer** window, click the **Catalog Items View** tab.

   You can also click **View**, click **Other Windows**, and then click **Catalog Items View**.

2. In the **Catalog Items View** window, in the search box at the top of the catalog item tree, type **SYSGEN_COMPOSITION**. The Catalog Item tree appears and **Window Compositor** (in the path **Core OS > Shell and User Interface > Graphics, Windowing and Events**) is selected. This selection indicates that **SYSGEN_COMPOSITION** is correctly set to **1** for the build.

To add the OpenGL compositor to the OS from Platform Builder

1. Repeat steps 1 and 2 from the preceding procedure.
2. In the **Project** window, click **project-name Properties**, where *project-name* is the name of the project.
3. In the **Property Pages** pane, click **Configuration Properties**, and then click **Environment**.
4. If BSP_GLES2COMPOSITOR is not in the **Environment Variables** list, then do the following:
   a. Below the **Environment Variables** list, click the **New** button.
   b. In the **Environment Variable** dialog box, in the **Variable Name** list, type **BSP_GLES2COMPOSITOR**, and in the **Variable Value** list, type **1**.

      The variable name and value appear in separate columns in the **Environment Variable** list.
   c. Click **OK** or **Apply**.

Add Window Compositor from the Command Line

You can set the required values for SYSGEN and BSP variables and build the image from the command line without running Platform Builder, or you can open a command line to do this within Platform Builder. Then you use the following procedure to build the OS and add Window Compositor to it.

To add Window Compositor to an OS from the build command line

1. Open a Command Prompt window that points to the wince.bat file that builds the OS. For example, the following command opens a Command Prompt window that points to wince.bat in the default location, C:\WINCE700\public\COMMON\oak\, and sets options for the batch file.

   ```
   %SystemRoot%\system32\cmd.exe /k "set
   _WINCEROOT=C:\WINCE700\public\COMMON\oak\WinCE.bat x86 uldr CEPC"
   ```
2. At the command prompt, do one of the following:
   - To include the default GDI compositor in the OS, type the following command:

     ```
     SYSGEN_COMPOSITION=1
     ```
   - To include the OpenGL compositor in the OS, type both of the following commands at the command line:

     ```
     SYSGEN_COMPOSITION=1
     BSP_GLES2COMPOSITOR=1
     ```
3. To build the image, at the command prompt, type **blddemo** with any arguments you want.

Conclusion

Windows Embedded Compact developers can provide a rich UI by solving the problem of overlapping application windows through window transparency. The Windows Embedded Compact 7 operating system includes the optional Window Compositor module, which implements alpha blending between windows to enable window transparency and improve user experience. You can add Window Compositor to an OS design by setting the appropriate SYSGEN and optional BSP variables before building a BSP. Window Compositor also includes an API that you can use to set or get both the degree of transparency for an overlapping window and the area of the window, up to the entire window area, that is transparent.

Additional Resources

- Windows Embedded website (http://go.microsoft.com/fwlink/?LinkId=183524)