Windows Embedded Compact

Performance Tuning Guide for Silverlight for Windows Embedded

Published: January 2012

Applies To: Windows Embedded Compact 7

Abstract

This paper describes how to optimize graphics performance on devices that are running Silverlight for Windows Embedded. When you optimize graphics performance, you improve the user experience on your device by providing faster load times and smoother animations. This paper describes:

- How to test graphics performance on the device hardware
- How to choose hardware components and drivers to improve performance
- How to optimize OS components such as binary shaders, graphics renderers, and Window Compositor
- How to optimize code in C++, XAML, and OpenGL Shading Language

© 2012 Microsoft. All rights reserved.

Contents

Introduction: SilverlightPerformance 4

Overview 4

Testing Graphics Performance 5
- Verifying the Graphics Performance of Silverlight XAML on the Device Hardware 5
- Testing the Shell and Applications to Obtain Performance Data 7
  - Silverlight for Windows Embedded Performance Tool (XRMon.exe) 9
    - Performance Counters 9

Tuning Graphics Performance 11
- Optimizing Your BSP, Hardware, or Drivers for Graphics Performance 11
  - Adding Minimum Graphics Hardware Components to the Hardware Board 12
  - Adding Support for Binary Shaders to the BSP 12
- Optimizing Operating System Components for Graphics Performance 14
  - Adding Support for Hardware Acceleration to the OS 14
  - Customizing the Implementation of the Graphics Renderer (Optional) 14
  - Tuning Performance of the Window Compositor 15
- Optimizing UI Elements and XAML Files for Graphics Performance 15
  - Using UI Elements in Your Application that are Optimized for Hardware Acceleration 16
    - Using Optimized UI Elements to Create Animations 16
    - Using Optimized UI Elements when you Display Images 17
    - Using VisualState and ControlTemplate Elements 18
  - Setting Bitmap Caching on UI Elements (Optional) 18
  - Reordering XAML Elements and Setting Z-Order for Optimal Display Buffer Usage 18
    - Changing the Order of XAML Elements 18
    - Grouping XAML Elements into a Panel Element 19
      - Grouping XAML Elements into User Controls 19
    - Separating XAML into Several Smaller Files 20
    - Setting the Z-Order on UI Elements 20
  - Removing Unnecessary XAML Elements 20
  - Using a VirtualizingStackPanel (Optional) 21
  - Setting the TargetType for ControlTemplate and Style Elements 21
- Optimizing Images for Graphics Performance 22
  - Modifying the File Type of Graphic Images to Optimize Load Time 22
  - Loading Images from Resource Binaries 23

Conclusion 24

© 2012 Microsoft. All rights reserved.

Additional Resources 24

© 2012 Microsoft. All rights reserved.

Introduction: SilverlightPerformance

The visual design of a UI based on Silverlight for Windows Embedded defines the user experience with your Windows Embedded Compact 7 device. However, a UI with graphics performance issues can detract from a good visual design. Therefore, testing and tuning graphics performance is important for improving the user experience.

This guide can help Windows Embedded Compact developers make hardware design decisions, use programming techniques, implement development processes, and use tests that can improve graphics performance of a UI based on Silverlight for Windows Embedded. These guidelines include code optimizations in both C++ and XAML, code optimizations in OpenGL Shading Language, hardware components for the device hardware, and a development environment that requires a hardware board, Microsoft Platform Builder, and Microsoft Expression Blend 3.

Overview

On desktop computers, graphics performance is of less concern than on compact devices. Desktop computers typically have virtually unlimited graphics processing power and so, for these UIs, testing and tuning graphics performance is less of an issue. Because Silverlight for Windows Embedded runs on Windows Embedded Compact 7, Silverlight for Windows Embedded has graphics performance considerations that are different from Silverlight for the desktop browser.

Graphics performance considerations are different for Silverlight for Windows Embedded because it runs on devices that can use different microprocessor types, that have custom hardware components, and that have memory size constraints. In contrast, Silverlight for the desktop browser runs on desktop computers and uses the central processing unit (CPU), graphics processing unit (GPU), and memory that is available on the desktop computer.

When you tune graphics performance for compact devices, you are trying to improve two major performance measurements in the UI.

- Load time (the time it takes to render the first screen), which depends on the following:
  - The number of UI elements in the design.
  - The types of UI elements in the design.
  - The visual design of the UI.
- Animation speed (the smooth and realistic appearance of movement on the screen), which depends on the following:
  - The number of UI elements that move.
  - The size of the UI elements that move, or the portion of the screen that is covered by moving elements.
  - Whether the UI elements are changing or oscillating as they move.

© 2012 Microsoft. All rights reserved.

For an in-depth overview of graphics performance concepts, see Improving Performance in Rich User Interfaces for Embedded Systems (http://go.microsoft.com/fwlink/?linkid=197420) at MSDN.

Testing Graphics Performance

Before you tune the graphics performance of Silverlight for Windows Embedded, you must test your shell and applications to find out what your key performance issues are.

To test graphics performance correctly during design and development, do the following:

- Verify the performance of Microsoft Silverlight 3 XAML on the device hardware while you design the XAML.
- Use the Silverlight for Windows Embedded Performance test suite to gather performance data such as load time, frames per second (FPS), and memory and CPU usage data.

These techniques are described in the following paragraphs.

Verifying the Graphics Performance of Silverlight XAML on the Device Hardware

To verify the graphics performance of Silverlight XAML, you interact with the UI that you created with XAML while the UI is running on the device hardware.

Do not wait until you finish the design of the Silverlight XAML to test it on the device hardware. Instead, test the XAML UI periodically on the device hardware as you would test the XAML periodically in Microsoft Expression Blend 3.

By testing the performance of the XAML on a regular basis, you can refine it to improve the user experience with the visual design and graphics performance.

When you download the XAML to the device hardware, make sure that the hardware board contains the components you expect to use in the production device.

Note

Alternatively, you can load the XAML on a virtual CEPC. However, the virtual CEPC does not support hardware acceleration, which is a requirement for OpenGL or DirectDraw graphics rendering and bitmap caching. However, you can still use the virtual CEPC to analyze resource usage of the XAML.

To verify graphics performance of Silverlight XAML, you must have Windows Embedded Compact 7 installed on the same computer on which you use Expression Blend 3. You must also have a hardware board connected to the computer and a display screen for the hardware board, and you must set up a build configuration in which you can update the Silverlight application project, rebuild the application, and then download the application to the hardware board.

© 2012 Microsoft. All rights reserved.

For more information about the tools that you use to create and build Silverlight for Windows Embedded applications in Microsoft Platform Builder, see A Sample Application Tutorial Using Windows Embedded Silverlight Tools (http://go.microsoft.com/fwlink/?LinkID=189508).

To verify graphics performance of Silverlight XAML

1. In Expression Blend 3, save the changes to your XAML project.
2. In Platform Builder, open your OS design project.
3. Open the subproject for the Silverlight for Windows Embedded application for which you have designed XAML.
4. Open a source file that belongs to the Silverlight for Windows Embedded application.
5. On the **Tools** menu, point to **Windows Embedded Silverlight Tools**, and then click **Update Silverlight for Windows Embedded Project**.
6. On the **File** menu, click **Save All**.
7. Rebuild your Silverlight for Windows Embedded application as follows:
   - In Solution Explorer, right-click the application subproject name and click **Rebuild**.
8. Update the OS design to include support for device connectivity, and rebuild the OS image as follows:
   a. In Catalog Items View, type **SYSGEN_SHELL** in the Search box, click the green arrow box, and then check the **Target Control Support (Shell.exe)** box.
   b. Repeat the previous step for any other catalog items required for device connectivity, as described in Tools Catalog Items and Sysgen Variables (http://go.microsoft.com/fwlink/?LinkId=232868).
9. Download the OS image to the hardware board. For more information, see Downloading an OS to a Device over Ethernet (http://go.microsoft.com/fwlink/?LinkId=232867).
10. Load the application using Target Control as follows:
    a. On the **Target** menu, click **Target Control**.
    b. At the command line, type `s <application_name>`.
11. Interact with the UI on the display screen attached to your hardware board. When you test the XAML UI, observe the following:
    - How long it takes to start the Silverlight for Windows Embedded application.
    - How long it takes to load individual controls in the application window.
    - How long it takes to load bitmap images in the application window.
    - How long it takes the UI to respond to gestures (if you implemented gestures).
    - Whether animations run efficiently or slower than expected.
    - Whether video runs efficiently or slower than expected.
    - How long it takes to scroll through a list box (if you implemented a list box).

© 2012 Microsoft. All rights reserved.

Use the performance data that you gather from interacting with the UI to decide which aspects of your Silverlight UI to tune for graphics performance. Also, use the Windows Embedded Compact Test Kit (CTK) Silverlight for Windows Embedded Performance test suite or the Performance Monitor (XRMon.exe) to gather performance metrics. Then, use the guidelines in Tuning Graphics Performance to improve graphics performance.

Testing the Shell and Applications to Obtain Performance Data

The Windows Embedded Compact Test Kit (CTK) includes the Silverlight for Windows Embedded Performance test suite, a set of test cases that you use to obtain UI performance data for applications based on Silverlight for Windows Embedded. These applications can include the sample home screen, Windows Embedded Compact applications such as Media Player and Photo Viewer, and your custom Silverlight applications.

You can use the Silverlight for Windows Embedded Performance test suite to measure performance metrics for Silverlight applications that run on the device hardware. Performance metrics include the following:

- Time to load XAML
- Amount of memory used to load XAML
- Time to render the first frame
- FPS for running an animation
- FPS for resizing an image
- FPS for list box interaction
- FPS for screen transitions

For more information about the CTK, see the Windows Embedded CTK User Guide (http://go.microsoft.com/fwlink/?linkid=197421) at MSDN.

Run performance tests to determine your key graphics performance issues and what types of changes to implement. Then, tune graphics performance by using the guidelines in the following section.

To use the Silverlight for Windows Embedded Performance test suite, you must include support for CEPerf in your OS image.

To add support for CEPerf

1. In Platform Builder, on the **Project** menu, click **<OSDesignName> Properties**.
2. Expand **Configuration Properties**, and then click **Build Options**.
3. Set **Include CEPerf** to **Yes (IMGCEPERF=1)**.
4. Click **OK**.
5. Rebuild the OS image.

© 2012 Microsoft. All rights reserved.

You can follow the instructions in the Windows Embedded CTK User Guide (http://go.microsoft.com/fwlink/?linkid=197421) at MSDN to run the test cases available in the Shell node of the CTK test catalog. To create a test case for your custom application, use the following instructions.

To create a test for your Silverlight for Windows Embedded application

1. In the Windows Embedded CTK application, click **File**, point to **New**, and then click **Test Case**.
2. Click the (+) icon to expand **Windows Embedded Test Catalog**.
3. Click the (+) icon to expand **Shell**.
4. Click the (+) icon to expand **Silverlight for Windows Embedded**.
5. Select **Applications Load – Browser**.
6. On the **Test Case Information** tab, click **New**.
7. In **Name**, type a name for your test. For example, type “Applications Load – Silverlight Application.”
8. In **Run type**, click **Fully Automated**.
9. In **Test harness**, click **Tux**.
10. In **Command line**, replace the text with the command-line argument for your test. For example, replace the text with the following:

```
-o -d XRAppLoadTests.dll -c "processname=<app_name> waittime=15
closewindow"
```

11. In the text, replace `<app_name>` with the name of the application .exe file.
12. In **Supported architecture**, double-click the microprocessor architecture for your hardware board.
13. Click **Browse**.
14. Browse to C:\Program Files\Windows Embedded Compact Test Kit\tests\target\<supported_architecture>.
15. Select the test to run on the application. Silverlight tests are prefixed with “xr.” For example, choose xrapploadtests.dll.
16. Click **Open**.
17. Click **OK**.
18. Click **Advanced**.
19. In **List command line**, replace the text with the list command line for your test. For example:

```
-o -d XRAppLoadTests.dll -c "processname=<app_name> waittime=15
closewindow" -l
```

© 2012 Microsoft. All rights reserved.

20. Click **OK**.
21. Click **Save**.

After you complete these steps, follow the instructions in "Viewing Performance Test Results Graphically" in the Windows Embedded CTK User Guide (http://go.microsoft.com/fwlink/?linkid=197421) at MSDN to run the test and gather performance data by using the Graph Tool.

For reference information about the tests available in the Silverlight for Windows Embedded Performance test suite, see Shell Tests (http://go.microsoft.com/fwlink/?linkid=197471) in the Windows Embedded Compact 7 documentation.

Use the performance data you gather from the tests to decide which aspects of your Silverlight UI should be tuned for graphics performance. Then, use the guidelines in the following section to tune graphics performance.

Silverlight for Windows Embedded Performance Tool (XRMon.exe)

You can also use the Silverlight for Windows Embedded Performance Tool (XRMon.exe) to collect graphics performance information for a Silverlight for Windows Embedded shell or application. You use XRMon.exe to test performance on devices that use the following microprocessor types:

- ARM v5
- ARM v6
- ARM v7
- MIPS II
- MIPS II Floating Point
- x86

To download XRMon.zip, which you use to extract XRMon.exe to a folder on your computer, see Silverlight for Windows Embedded Performance Tool: XRMon (http://go.microsoft.com/fwlink/?LinkId=221814).

After you download XRMon.zip and extract its contents, follow the instructions for running the tool in the XRMon.pdf white paper, located in the root directory of the folder to which you extracted the contents of the XRMon.zip file.

XRMon.exe provides performance counters that are designed to collect specific performance metrics, as described in the following section.

Performance Counters

You can specify performance counters to obtain performance data on specific Silverlight for Windows Embedded methods or graphics-rendering operations. By using performance counters with XRMon.exe, you can create CEPerf sessions for gathering specific types of performance data.

After you run XRMon.exe with performance counters, the CEPerf session output is included in the .txt file that you create by using the instructions in the XRMon.pdf white paper, which is installed with the tool.

© 2012 Microsoft. All rights reserved.

To use performance counters

1. Download a run-time image to your device. For more information, see Developing an OS Design for Windows Embedded (http://go.microsoft.com/fwlink/?LinkID=210187).
2. In Platform Builder, on the **Target** menu, click **Target Control**.
3. At the command prompt, run XRMon.exe by following the instructions in the XRMon.pdf white paper.
4. When you run XRMon.exe, use one or more **EnableSession** flags. **EnableSession** supports the following values:

EnableSession argument	Description
API_Application	Creates a CEPerf session for gathering performance data on the duration of time spent in calls to the following methods: **Initialize**, **LoadCore**, **CreateHostFromElementTree**, **CreateHost**, **CreateHostFromXaml**, **CreateObject**, **AddResourceModule**, **RemoveResourceModule**, **LoadImageFromFile**, **ResolveResourceToSurface**, **ParseXaml**, and **ParseXamlWithExistingRoot**.
API_Host	Creates a CEPerf session for gathering performance data on the duration of time spent in calls to the following methods: **GetRootElement**, **GetFocusedElement**, and **HandleGestureEvent**.
API_Factory	Creates a CEPerf session for gathering performance data on the duration of time spent in calls to class factory methods such as **RegisterCustomControlFactory**, **ResolveCustomPropertyName**, **RegisterObject**, and **ResolveAttachedProperty**.
API_Render	Creates a CEPerf session for gathering performance data on the duration of time spent in a method call for graphics rendering methods that include handling the WM_PAINT window message, advancing animations, and drawing on a surface.

© 2012 Microsoft. All rights reserved.

API_Render_FirstPaint	Creates a CEPerf session for gathering performance data on the duration of time prior to starting an application that occurs before the first WM_PAINT window message is handled.
API_MessageQueue	Creates a CEPerf session for gathering performance data for any XR_HOOKPROC (http://go.microsoft.com/fwlink/?LinkId=225740) procedures.
API_Render_FrameRate	Creates a CEPerf session that uses animation frame-rate performance counters for reporting statistics such as current frame rate, average frame rate, and short animation frame rate for animations less than 1,000 milliseconds.
Core_Layout	Creates a CEPerf session for gathering performance data on updating the layout of UI elements.
Core_Render	Creates a CEPerf session for gathering performance data on graphics-rendering operations including rasterizing, presenting, and composing graphics primitives on a display screen.

The following example command shows how to run XRMon.exe with the API_Application CEPerf session:

```
>s xrmon -process=TestApp -enablesession=API_Application @release\ttl.xrmon
```

Tuning Graphics Performance

This section provides guidelines for tuning the graphics performance of the Silverlight for Windows Embedded UI.

Optimizing Your BSP, Hardware, or Drivers for Graphics Performance

To help improve graphics performance, you can optimize the board support package (BSP), hardware, or device drivers for the Windows Embedded Compact powered device that the UI runs on.

© 2012 Microsoft. All rights reserved.

This section provides information about the following tasks to optimize graphics performance:

- Adding Minimum Graphics Hardware Components to the Hardware Board
- Adding Support for Binary Shaders to the BSP

Adding Minimum Graphics Hardware Components to the Hardware Board

When you work with a hardware vendor to create a hardware board for your Windows Embedded Compact powered device, make sure that the board includes the recommended requirements for graphics hardware and memory.

When you include graphics hardware that supports hardware acceleration, you can implement an OpenGL or DirectDraw graphics renderer and support bitmap caching, which can help improve load time and animation frame rate.

For information about the recommended requirements for Silverlight for Windows Embedded, see Silverlight for Windows Embedded Hardware Recommendations (http://go.microsoft.com/fwlink/?LinkId=227923)

Adding Support for Binary Shaders to the BSP

OpenGL Embedded Systems (ES) hardware acceleration uses OpenGL ES drivers and binary shaders, including a vertex shader and a fragment shader.

An OpenGL ES fragment shader is a program for the Graphics Processing Unit (GPU) that provides calculations required to render pixels for your Silverlight for Windows Embedded UI. An OpenGL ES vertex shader is a program for the GPU that processes vertexes of primitives, such as vertexes drawn by **DrawTriangleStrip**.

Note

Not all OpenGL ES drivers support binary shaders.

Windows Embedded Compact 7 provides default binary shaders in Shaders.dll, which a board support package (BSP) provider can customize for your hardware and add to your BSP. By customizing Shaders.dll, you can help improve load time and graphics performance, and you can tune the shaders code for the capabilities of the device hardware.

We recommend that you develop a custom shader.dll library that the OpenGLRendering engine can use to replace the default fragment shader. However, if this is not possible, you can still make modifications to the default fragment shader's source code in order to tune graphics performance.

Silverlight for Windows Embedded uses BGRA pixel format. If your GPU hardware currently supports BGRA pixel format, you will not have to use the color conversion algorithm implemented in the default fragment shader. The fragment shader is used for each pixel of each surface, and the additional color conversion algorithm negatively impacts graphics performance and is only required for GPU hardware that supports RGBA pixel format.

© 2012 Microsoft. All rights reserved.

The **COpenGLDevice::InitializeState()** method in %_WINCE700%\Public\COMMON\Oak\XamlRenderPlugin\OpenGLXaml\OpenGLDevice.cpp includes a call to the **InitializeShader** function, which compiles default source code that includes the color conversion algorithm. The default source code is provided as an input parameter in **InitializeShader** in %_WINCE700%\Public\COMMON\Oak\XamlRenderPlugin\OpenGLXaml\OpenGLDevice.cpp. The color conversion algorithm is beneath the following code comment:

```
"   // This is worth removing if you know your hardware does BGRA;\r\n
"   // even the if() test can be expensive.\r\n"
```

For GPU hardware that does not use BGRA pixel format and requires color conversion, modify the default source code for **InitializeShader** to remove the `if (expression)` portion of the `if` statement, because evaluating the `if (expression)` code for each pixel is unnecessary and negatively impacts performance.

For GPU hardware that supports BGRA pixel format and does not require color conversion, remove all the code beneath the comment, including the `if` statement.

To customize Shaders.dll for the device hardware

1. Obtain a command-line shader compiler from your GPU hardware manufacturer.
2. Copy files from %_WINCEROOT%\PUBLIC\COMMON\OAK\XamlRendererPlugin\OpenGL\shaders to your BSP directory.
3. (Optional) customize the source code for the fragment shader (defaulteffect.fs.glsl) and the vertex shader (defaulteffect.vs.glsl).
4. Link defaulteffect.fs.glsl and defaulteffect.vs.glsl to Shaders.dll.
5. Use the GPU hardware manufacturer's instructions for compiling Shaders.dll for the GPU by using the command-line shader compiler.

   Be sure to check the default settings of the shader compiler optimizer parameters, and adjust them if necessary.
6. Add the recompiled Shaders.dll to the BSP for your hardware board.

Note

At run time, Silverlight for Windows Embedded and Internet Explorer Embedded will both try to load Shaders.dll, extract defaulteffect.vs.bin and defaulteffect.fs.bin, and load the files onto the GPU.

If Shaders.dll is not found, or if the extracted files do not match the GPU type, Silverlight and Internet Explorer Embedded will instead try to compile their own default shader source code at run time. However, run-time compilation is not supported by all OpenGL ES drivers, and might result in less robust graphics performance than that of shaders that are compiled for the GPU.

© 2012 Microsoft. All rights reserved.

The source code in the vertex shader and pixel shader is based on OpenGL Shading Language. For more information about OpenGL programming, see the OpenGL API Documentation (http://go.microsoft.com/fwlink/?linkid=196793).

Optimizing Operating System Components for Graphics Performance

To improve graphics performance, you can optimize components in the Windows Embedded Compact operating system that affect UI load time and animation frame rate.

This section provides information about the following tasks for optimizing graphics performance:

- Adding Minimum Graphics Hardware Components to the Hardware Board
- Adding Support for Binary Shaders to the BSP

Adding Support for Hardware Acceleration to the OS

Hardware acceleration for Silverlight for Windows Embedded caches rendered UI elements in graphics memory, which consumes more memory but results in better graphics performance.

Hardware acceleration is a requirement for supporting the OpenGL ES or DirectDraw graphics renderer, and for supporting bitmap caching. When you implement hardware acceleration, you can help improve both load time and animation frame rate.

For more information about how to implement hardware acceleration, see Implement Hardware Acceleration for Graphics in Silverlight for Windows Embedded in the Silverlight for Windows Embedded Developer's Guide (http://go.microsoft.com/fwlink/?linkid=191039) at MSDN.

Customizing the Implementation of the Graphics Renderer (Optional)

By using cached composition and hardware-accelerated graphics, you can improve graphics processing speed, which helps improve graphics performance. When you add support for hardware acceleration to an OS, you must add a graphics renderer, either OpenGL or DirectDraw, to the OS image.

You can optionally customize the graphics renderer source code to optimize graphics performance for your device hardware.

For example, you can remove specific graphics operations from the code that occur at a specific stage in the graphics pipeline, and then test the UI to determine whether the performance metrics improve.

For steps on how to customize the graphics renderer, see Step 2 through Step 4 in the section Implement Hardware Acceleration for Silverlight for Windows Embedded in Silverlight for Windows Embedded Developer's Guide (http://go.microsoft.com/fwlink/?linkid=191039) at MSDN.

After you rebuild the C++ code for the graphics renderer and rebuild the OS image, you can rerun the Silverlight for Windows Embedded Performance tests and determine whether the performance metrics have improved. For more information, see Test the Shell and Applications to Gather Performance Data earlier in this document.

© 2012 Microsoft. All rights reserved.

For more information about tuning graphics performance in OpenGL code, see Tuning Your OpenGL Application (http://go.microsoft.com/fwlink/?linkid=196804).

Tuning Performance of the Window Compositor

The Window Compositor displays windows on a device screen by using alpha blending. To implement alpha blending, the Window Compositor first draws each window to a back-buffer surface in memory. Then, it aggregates those surfaces and composes them on a single back-buffer surface. Finally, it displays the windows on the screen.

By using this rendering process, the Window Compositor provides alpha blending, which improves the visual appearance of windows. However, alpha blending can impact graphics performance.

Developers should consider both the benefits of an improved visual appearance and the potential drawbacks of graphics performance when deciding whether to include the Window Compositor (SYSGEN_COMPOSITION). For example, if the UI cannot display several windows at the same time, you might not have to include the Windows Compositor.

When an OS design includes the Window Compositor, developers should follow the programming guidelines for improving its graphics performance. For more information, see Window Compositor Graphics Performance Tuning Guide (http://go.microsoft.com/fwlink/?linkid=199845) at MSDN.

Optimizing UI Elements and XAML Files for Graphics Performance

In Microsoft Expression Blend 3, the way that you design the UI in XAML files can affect the graphics performance when the UI elements are rendered on a display screen. For example, you can decide to use specific types of XAML elements that are optimized for hardware acceleration, or reorder XAML elements in a file and set z-order for optimal display buffer usage. Detailed information and a variety of recommended tasks for designing the UI in XAML files to optimize graphics performance are provided in the following paragraphs.

This section includes information about the following tasks for optimizing graphics performance:

- Using UI Elements in Your Application that are Optimized for Hardware Acceleration
- Setting Bitmap Caching on UI Elements (Optional)
- Reordering XAML Elements and Setting Z-Order for Optimal Display Buffer Usage
- Removing Unnecessary XAML Elements
- Using a VirtualizingStackPanel (Optional)
- Setting the TargetType for ControlTemplate and Style Elements

© 2012 Microsoft. All rights reserved.

Using UI Elements in Your Application that are Optimized for Hardware Acceleration

Silverlight for Windows Embedded includes UI elements that provide graphic effects that are optimized for hardware acceleration and that work better with the graphics renderer.

This section describes UI elements that are optimized for hardware acceleration.

Using Optimized UI Elements to Create Animations

Animating UI elements that are optimized for hardware acceleration can help improve animation frame rate.

To decide which UI elements to animate, use the UI elements in the following chart to achieve better graphics performance from animations.

Table 1: UI Elements Optimized for Animations

XAML element	Supporting graphics hardware
Transform	2-D graphics hardware support
RotateTransform (http://go.microsoft.com/fwlink/?linkid=138049)	2-D graphics hardware support
ScaleTransform (http://go.microsoft.com/fwlink/?linkid=151476)	2-D graphics hardware support
TranslateTransform (http://go.microsoft.com/fwlink/?linkid=151511)	2-D graphics hardware support
PlaneProjection (http://go.microsoft.com/fwlink/?linkid=178941)	2-D graphics hardware support

Effects based on the IXREffect (http://go.microsoft.com/fwlink/?linkid=197473) class, such as the Blur and Drop Shadow effects, are also optimized for hardware acceleration. Both require a CPU that supports a pixel shader.

When deciding which UI elements to animate, note that the following are not optimized for hardware acceleration:

- LinearGradientBrush
- RadialGradientBrush
- GradientStop
- FrameworkElement.Margin

If multiple UI elements use the same type of animation for the same property, create a panel, such as a StackPanel (http://go.microsoft.com/fwlink/?linkid=137376), and move the UI elements into the panel element. Then, set the animation storyboard to animate the property in the parent element instead of in

© 2012 Microsoft. All rights reserved.

the multiple UI elements. For example, if three buttons all animate their Opacity (http://go.microsoft.com/fwlink/?linkid=197475) property in the same way, you can move them into a StackPanel (http://go.microsoft.com/fwlink/?linkid=137376) element and animate the Opacity (http://go.microsoft.com/fwlink/?linkid=197475) property on the StackPanel (http://go.microsoft.com/fwlink/?linkid=137376) instead of each button. For more information, see the "Group XAML Elements into a Panel Element" section in this document.

By animating a common property for multiple UI elements in their panel element, you can reduce the amount of rasterization operations required to play the animation, which helps improve animation frame rate.

To set the animation in a parent element in XAML

1. In your XAML file, identify multiple Storyboard (http://go.microsoft.com/fwlink/?linkid=136802) elements that create the same animation, yet target different UI elements by using the Storyboard.TargetName (http://go.microsoft.com/fwlink/?linkid=197476) attached property.
2. Move the UI elements into a StackPanel (http://go.microsoft.com/fwlink/?linkid=137376) parent element.
3. Add the Storyboard (http://go.microsoft.com/fwlink/?linkid=136802) element to the StackPanel (http://go.microsoft.com/fwlink/?linkid=137376) parent element, and set its Storyboard.TargetProperty (http://go.microsoft.com/fwlink/?linkid=197477) to a property of the StackPanel (http://go.microsoft.com/fwlink/?linkid=137376).
4. Delete the duplicate Storyboard (http://go.microsoft.com/fwlink/?linkid=136802) elements.

Using Optimized UI Elements when you Display Images

A WriteableBitmap (http://go.microsoft.com/fwlink/?linkid=197479) element is also not optimized for hardware acceleration when it is rendered at load time. To help improve load time, replace WriteableBitmap (http://go.microsoft.com/fwlink/?linkid=197479) elements with BitmapSource (http://go.microsoft.com/fwlink/?linkid=197480) elements, although you won't be able to use the feature that modifies the bitmap's pixels at run time.

UI elements that represent shapes and images that you don't plan to animate, such as Path (http://go.microsoft.com/fwlink/?linkid=136997) or Polygon (http://go.microsoft.com/fwlink/?linkid=142358) elements, are also not optimized for hardware acceleration. To help improve load time, replace those elements with BitmapSource (http://go.microsoft.com/fwlink/?linkid=197480) elements that have .bmp image files which display the same graphic as the UI element.

© 2012 Microsoft. All rights reserved.

Using VisualState and ControlTemplate Elements

When you create animations for user controls, use VisualState (http://go.microsoft.com/fwlink/?linkid=143430) elements to animate user controls when users change their state.

When you define the appearance of your controls, use reusable ControlTemplate (http://go.microsoft.com/fwlink/?linkid=147145) elements instead of creating new XAML elements to define the appearance of each individual control.

Both of these elements represent on-demand parsed constructs, which can help improve graphics performance during load time.

Setting Bitmap Caching on UI Elements (Optional)

In Windows Embedded Compact 7, Silverlight for Windows Embedded provides default bitmap caching behavior by automatically setting bitmap caching for UI elements that should use hardware acceleration.

When UI elements that support bitmap caching are rendered on the screen, bitmaps for those elements are cached in memory so that they can be redrawn to the screen at a later time, which improves load time for that UI element.

However, if you have a Silverlight UI that has a significant number of graphic elements and a device with limited video memory, you should set bitmap caching explicitly on UI elements that should use hardware acceleration.

By setting bitmap caching explicitly, you can ensure that specific UI elements always use hardware acceleration, which helps improve graphics performance.

You should set the cache mode for all UI elements that implement animations or transformations.

To set bitmap caching for a XAML element in Silverlight for Windows Embedded, add a **CacheMode** attribute to the element and set its value to “BitmapCache.”

The following example code sets bitmap caching for a Canvas (http://go.microsoft.com/fwlink/?linkid=141459) element:

```
<Canvas Width="338" Height="400" Background="MidnightBlue" CacheMode="BitmapCache" >
```

Reordering XAML Elements and Setting Z-Order for Optimal Display Buffer Usage

The way that XAML elements are defined in XAML files can affect graphics performance of a Silverlight for Windows Embedded application when the elements are parsed and displayed on the device screen.

Changing the Order of XAML Elements

The default z-order, or the order in which elements are rendered on the screen, is based on the hierarchical order of elements in the XAML file.

© 2012 Microsoft. All rights reserved.

Therefore, the order that you define elements in a XAML file implicitly determines both the z-order and the display buffer on which the XAML elements are grouped.

UI elements that use bitmap caching, such as animations, should be rendered first in the z-order. When these UI elements are rendered first, it reduces the amount of back-buffer surfaces that the system must create for other elements during the rendering process, which reduces graphics memory consumption.

To change the order of XAML elements

- Move UI elements that use bitmap caching toward the beginning of the XAML file.

Grouping XAML Elements into a Panel Element

Identify logical groups of UI elements that should be rendered on the screen together and place them inside a Panel (http://go.microsoft.com/fwlink/?linkid=186933) element such as a Grid (http://go.microsoft.com/fwlink/?linkid=136581) or Canvas (http://go.microsoft.com/fwlink/?linkid=141459). This can help improve load time for the group of UI elements.

For example, if you created a set of button elements that form a calculator on the main Canvas (http://go.microsoft.com/fwlink/?linkid=141459) element, you can create a Canvas (http://go.microsoft.com/fwlink/?linkid=141459) sub-element and move the button elements into that sub-element.

To group XAML elements into a Canvas

1. In your XAML file, define a Panel (http://go.microsoft.com/fwlink/?linkid=186933) element, such as a Grid (http://go.microsoft.com/fwlink/?linkid=136581) or Canvas (http://go.microsoft.com/fwlink/?linkid=141459), as a sub-element of the layout root Panel (http://go.microsoft.com/fwlink/?linkid=186933) element.
2. Move the logical group of UI elements under that Panel (http://go.microsoft.com/fwlink/?linkid=186933) element, so that they are sub-elements of that Panel (http://go.microsoft.com/fwlink/?linkid=186933) element.

Grouping XAML Elements into User Controls

If you plan to display the logical group of elements in several windows, or in several different Silverlight for Windows Embedded applications, add the group of elements to a custom user control. This can help improve load time for the group of UI elements.

By adding the elements to a custom user control, you can reuse the elements and their functionality without writing additional code for each window or application.

© 2012 Microsoft. All rights reserved.

For more information, see Create a Custom User Control in Silverlight for Windows Embedded in Silverlight for Windows Embedded Developer's Guide (http://go.microsoft.com/fwlink/?linkid=191039) at MSDN.

Separating XAML into Several Smaller Files

At run time, when Silverlight for Windows Embedded periodically loads smaller XAML files instead of one large XAML file when the application starts, it reduces the workload of the XAML parser to parse and load each page or each control, which can help improve load time.

To separate XAML into several smaller files

1. Move XAML elements that are displayed in a single window in your application into separate .xaml files.
2. Move logical groups of XAML elements displayed in windows in your application into separate .xaml files.
3. In your C++ application code, call IXRApplication::ParseXaml (http://go.microsoft.com/fwlink/?linkid=197474) to load the individual .xaml files only when the application will display the elements in that XAML file.

Setting the Z-Order on UI Elements

To ensure that a specific XAML element is always rendered higher in the z-order than another XAML element, you can set the z-order directly for both elements. Note that UI elements that use bitmap caching, such as animations, should be rendered first in the z-order.

To explicitly set z-order on a XAML element

1. Add a Canvas.ZIndex (http://go.microsoft.com/fwlink/?linkid=197481) attached property to the UI element.
2. To render the UI element first, set the Canvas.ZIndex (http://go.microsoft.com/fwlink/?linkid=197481) value to a higher number. For example, set it to "5."
3. To render the UI element after another element, set the Canvas.ZIndex (http://go.microsoft.com/fwlink/?linkid=197481) value to a lower number. For example, set it to "1."

Removing Unnecessary XAML Elements

To help improve graphics performance, identify and remove unnecessary XAML elements. With fewer XAML elements in the visual tree, the application can load the visual tree more quickly.

© 2012 Microsoft. All rights reserved.

If you have a visual tree with a large number of XAML elements, and you are experiencing issues with graphics performance, reevaluate the UI elements in your XAML files, and identify XAML elements that are not essential to the functionality of the application (such as decorative UI elements that do not handle UI events and that do not support user interaction). Then, remove nonessential XAML elements (such as decorative background shapes or geometries).

During the design phase, the designer might replace XAML elements with other XAML elements and forget to delete the unused XAML elements, which are lower in the z-order and are obscured by elements higher in the z-order. Or, the designer might add and then decide not to use elements that are invisible, yet are still defined in the XAML file (such as shape elements with a transparent background color, or text boxes with no text). You should identify and delete any unused XAML elements in your Expression Blend project that are present in **XAML View**, yet are no longer displayed in **Design View**.

Using a VirtualizingStackPanel (Optional)

If you use ItemsControl (http://go.microsoft.com/fwlink/?linkid=141886) elements in your Silverlight UI, such as a ListBox (http://go.microsoft.com/fwlink/?linkid=141725) or ComboBox (http://go.microsoft.com/fwlink/?linkid=142209), you should also use a **VirtualizingStackPanel**.

With a **VirtualizingStackPanel**, Silverlight renders only the elements in the items control that are currently visible. By using a **VirtualizingStackPanel**, you can help improve load time for the items in the items control.

To use a VirtualizingStackPanel

1. Add the VirtualizingStackPanel.IsVirtualizing (http://go.microsoft.com/fwlink/?linkid=197483) attached property to the ItemsControl (http://go.microsoft.com/fwlink/?linkid=141886) element.
2. Set the attached property value to **True**.

To reuse item containers after they are no longer visible on the screen

1. Add the VirtualizingStackPanel.VirtualizationMode (http://go.microsoft.com/fwlink/?linkid=197484) attached property to the UI element.
2. Set the attached property value to **Recycling**.

The following example code shows a list box element that uses a **VirtualizingStackPanel**.

```
<ListBox x:Name="listbox" Width="200"  Height="150"
VirtualizingStackPanel.IsVirtualizing="True">
```

Setting the TargetType for ControlTemplate and Style Elements

If you use a ControlTemplate (http://go.microsoft.com/fwlink/?linkid=147145) or Style (http://go.microsoft.com/fwlink/?linkid=149151) element in the XAML for a Silverlight for Windows Embedded application, you must explicitly set the ControlTemplate.TargetType

© 2012 Microsoft. All rights reserved.

(http://go.microsoft.com/fwlink/?linkid=197486) or Style.TargetType (http://go.microsoft.com/fwlink/?linkid=197485) attribute value.

Silverlight for Windows Embedded internally compiles your XAML files into binary format, known as Binary XAML (BAML). By using BAML, applications don't have to parse XAML files each time that the application starts, which reduces load time and can help improve graphics performance.

Silverlight for Windows Embedded uses algorithms to compile XAML into BAML. When Silverlight compiles ControlTemplate (http://go.microsoft.com/fwlink/?linkid=147145) and Style (http://go.microsoft.com/fwlink/?linkid=149151) elements into BAML, if it cannot determine the type of control for the template or style, it does not compile the template or style into binary format. Therefore, you must always set the **TargetType** XAML attribute.

To set the TargetType XAML attribute

1. In your XAML files, add a **TargetType** attribute to every Style (http://go.microsoft.com/fwlink/?linkid=149151) and ControlTemplate (http://go.microsoft.com/fwlink/?linkid=147145) element.
2. Set the **TargetType** attribute value to the name of the element to which you want to apply the template or style.

   If the element is a custom user control, set the **TargetType** attribute value to the name of the custom user control prefixed with its XAML namespace.

The following example code shows a Style (http://go.microsoft.com/fwlink/?linkid=149151) element with the **TargetType** set to a Button (http://go.microsoft.com/fwlink/?linkid=134564) element.

```
<Style x:Key="ButtonStyle1" TargetType="Button">
```

Optimizing Images for Graphics Performance

When you use more than 20 image files in a UI, or if you use image files that contain a large number of bits and have an exceptionally large file size, images can take longer to load on the display screen. A typical image file is considered exceptionally large if it is greater than 2 KB, while a big image file, such as a background image, is considered exceptionally large if it is greater than 150 KB. To help images load quickly, you can optimize them for graphics performance.

This section provides information about the following tasks for optimizing graphics performance:

- Modifying the File Type of Graphic Images to Optimize Load Time
- Loading Images from Resource Binaries

Modifying the File Type of Graphic Images to Optimize Load Time

Use Windows Bitmap (.bmp) images as bitmap source files in your Silverlight for Windows Embedded UI, to help improve load time.

© 2012 Microsoft. All rights reserved.

For an image of the same height and width dimensions, choosing a file type for the image is a tradeoff between graphics performance and memory use.

Uncompressed image file formats such as .bmp files require more memory; however, they load much faster and can help improve graphics performance. Compressed file formats such as .jpeg are more difficult to decode and take longer to load.

Table 2: Graphic Images and Graphics Performance

Image type	Graphic performance impact	Memory impact	OS design requirements
Joint Photographic Experts Group (JPEG) (*.jpg, *.jpeg files)	Slowest to display at load time	Requires less memory	SYSGEN_IMAGING, SYSGEN_IMAGING_JPG_DECODE, SYSGEN_IMAGING_JPG_ENCODE
Portable Network Graphic (PNG) (*.png files)	Slow to display at load time	Requires less memory	SYSGEN_IMAGING, SYSGEN_IMAGING_PNG_DECODE, SYSGEN_IMAGING_PNG_ENCODE
Windows Bitmap (*.bmp files	Fastest to display at load time	Requires more memory	SYSGEN_IMAGING, SYSGEN_IMAGING_BMP_DECODE, SYSGEN_IMAGING_BMP_ENCODE

Loading Images from Resource Binaries

Images with exceptionally large file sizes can require a lot of system memory and can impact graphics performance. Exceptionally large file sizes for typical image files are usually greater than 2 KB, and exceptionally large file sizes for big image files, such as background images, are usually greater than 150 KB. To optimize graphics performance for images that require a lot of memory, use IXRApplication::LoadImageFromResource (http://go.microsoft.com/fwlink/?LinkId=222528) instead of IXRApplication::LoadImageFromFile (http://go.microsoft.com/fwlink/?LinkId=222529).

When you call **IXRApplication::LoadImageFromResource**, Silverlight for Windows Embedded loads graphic images from a binary resource file, such as a .dll file or .exe file, that you registered by calling IXRApplication::AddResourceModule (http://go.microsoft.com/fwlink/?LinkId=222411).

If you want to call **IXRApplication::LoadImageFromResource**, prepare image resources for your application by first running XRPack at the command line and setting the ImageNoCompress option, as shown in the following example. This option converts an image to an uncompressed bitmap instead of a compressed bitmap. By using uncompressed bitmaps, you can reduce the time that would be required to parse a compressed bitmap to an uncompressed bitmap at run time.

```
C:\WINCE700\public\COMMON\oak\bin\i386>xrpack /AppXaml=MainPage.xaml
/ImageNoCompress /ResFile=thumbnailCAMDMCM8.jpg
```

© 2012 Microsoft. All rights reserved.

```
/TargetHeader=TestAppLocalizingImagesGenerated.h
/TargetRes=TestAppLocalizingImages.rc

Microsoft (R) XRPack - XamlRuntime resource packaging utility version 7.00.437
Copyright (C) Microsoft Corporation.  All rights reserved.

TargetHeader =
C:\WINCE700\public\COMMON\oak\bin\i386\TestAppLocalizingImagesGenerated.h
TargetRes    = C:\WINCE700\public\COMMON\oak\bin\i386\TestAppLocalizingImages.rc

Targets up to date.
```

For more information, see XAML Resource Packager (http://go.microsoft.com/fwlink/?LinkId=204756).

Conclusion

This document describes the two major performance measurements of a UI. It provides guidelines for how to test the Silverlight for Windows Embedded shell and applications manually by verifying the XAML on the hardware board, and automatically by running Silverlight for Windows Embedded performance test cases.

In addition, this document provides techniques for tuning graphics performance of Silverlight for Windows Embedded, including selecting graphics hardware, customizing the graphics renderer, and modifying the Silverlight XAML.

Additional Resources

For more information, see the following resources:

- Optimizing Performance in Windows Embedded Compact 7 (http://go.microsoft.com/fwlink/?LinkId=222409)
- Windows Embedded CTK User Guide (http://go.microsoft.com/fwlink/?LinkId=197421)
- Silverlight for Windows Embedded Performance Tool: XRMon (http://go.microsoft.com/fwlink/?LinkId=221814)
- Windows Embedded website (http://go.microsoft.com/fwlink/?LinkID=203338)

© 2012 Microsoft. All rights reserved.

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.

© 2012 Microsoft. All rights reserved.