Windows Display Driver Model Enhancements in Windows Developer Preview

October 11, 2011

Abstract

This paper provides information about the various enhancements to the Windows Display Driver Model (WDDM) for the next version of Windows; this version of WDDM is referred to as “WDDM v1.2”. This document provides an overview of the features that are only enabled by a WDDM v1.2 driver. To take advantage of these features and provide the best experience, WDDM v1.2 is required by all systems shipped with the next version of Windows. WDDM v1.2 is a superset of WDDM v1.1, and WDDM 1.0. It is assumed the reader is familiar with WDDM v1.1 and v1.0.


This information applies to the following operating systems:

- Windows Developer Preview
- Windows Server Developer Preview

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the web at: Windows Display Driver Model Enhancements in Windows Developer Preview

Disclaimer: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. 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.
Contents

INTRODUCTION.................................................................................................................. 4
PART 1: WDDM in Windows 8.......................................................................................... 6
Introduction......................................................................................................................... 6
WDDM v1.2 Features........................................................................................................... 9
Performance Improvements.............................................................................................. 11
  Reduced system memory footprint.............................................................................. 11
  Video memory offer and reclaim................................................................................ 11
  Standby hibernate optimizations................................................................................ 12
Reliability improvements with increased GPU fault tolerance....................................... 13
  GPU preemption.......................................................................................................... 14
  TDR improvements...................................................................................................... 15
Improved ability to diagnose performance problems....................................................... 16
  User-mode driver logging............................................................................................ 16
Disabling Frame Pointer Omission (FPO) optimization.................................................... 17
XPS rasterization on the GPU......................................................................................... 17
Improvements to enhance end-user experience................................................................ 17
  Stereoscopic 3D........................................................................................................... 18
  D3D11 Video Playback Improvements......................................................................... 19
  Optimized Screen Rotation Support............................................................................ 19
Container ID support for displays (visually representing devices that are embedded within a device)........................................................................................................ 20
Seamless handoff of the display control between Windows and the WDDM graphics driver......................................................................................................................... 22
Driver optimizations that can help reduce power consumption...................................... 23
  DirectFlip................................................................................................................... 24
  GPU power management of idle and active power...................................................... 24
Advances to the display Infrastructure............................................................................ 24
  Microsoft Basic Display Driver.................................................................................. 25
  Desktop Duplication API............................................................................................. 26
  Support for headless systems..................................................................................... 27
PART 2: Direct3D Features & Requirements.................................................................... 29
Introduction....................................................................................................................... 30
DirectX feature improvements on Windows 8.................................................................. 30
  Pixel formats (5551, 565, 4444)............................................................................... 31
  Double-precision shader functionality......................................................................... 32
  Target-independent rasterization (TIR)......................................................................... 33
  No overwrite and discard......................................................................................... 34
  UAVs at every stage.................................................................................................... 34
Cross-process sharing of texture arrays (for supporting Stereoscopic 3D).................... 35
Unordered access views with multi-sample anti-alias sample access (UAV-MSAA)........ 35
Logic ops......................................................................................................................... 36
Improved control of constant buffers............................................................................ 36
Clearview.......................................................................................................................... 37
Tileable copy flag............................................................................................................ 37
Same-surface blits........................................................................................................... 37
Software requirements..................................................................................................... 38
Hardware requirements ................................................................. 39
PART 3: Graphics INF Requirements ............................................. 42
Introduction .................................................................................. 43
  Updated Feature score directive .................................................. 43
  Updated friendly name ................................................................. 45
  SKU differentiation directive ...................................................... 46
  SKU differentiation for device drivers ......................................... 46
  General UNICODE requirement ............................................... 47
  Installed display drivers directive ............................................. 48
  Copy flags to support PNP stop directive .................................... 48
  Driver\services start type directive ............................................ 49
  Capability override settings to disable OpenGL ......................... 49
  [Version] section directives ..................................................... 49
  [SourceDiskNames] section directives ....................................... 50
  General x64 directives ............................................................. 50
  General install section directives .............................................. 51
  [String] section changes for localized strings ......................... 51
  Driver DLL for display adapter or chipset has properly formatted file version 52
PART 4: Installation Scenarios ....................................................... 53
  Install ..................................................................................... 54
  Upgrade ................................................................................. 54
  Down level .............................................................................. 54
PART 5: WDDM v1.2 Driver Enforcement Guidelines ..................... 55
  WDDM v1.2 Driver Enforcement .............................................. 56
  User experience when a driver fails the dxgkrnl validation .......... 56
  WDDM driver and feature caps ................................................ 56
  Best practices ......................................................................... 57
  Resources ............................................................................... 58
  Glossary ................................................................................. 58
INTRODUCTION

The next version of Windows (code-named Windows 8) provides a Metro style user experience and supports key areas of innovation in graphics hardware and software that benefit end users, developers and system manufacturers in the following ways:

End User
- Rich graphical composition model that allows for more flexibility to support video playback and stereoscopic 3D scenarios
- Improved system and UI responsiveness by reducing memory footprint
- Improved reliability with increased GPU fault tolerance
- Seamless boot, resume, and driver upgrade experience
- Flicker free screen rotation experience
- High fidelity graphics experience even when windows bug checks

Developers
- Direct3D 11 video simplifies the development experience by providing a single API for graphical operations
- Consistent API/DDI platform to enable Stereoscopic 3D applications
- Optimizations that enable efficient usage of memory resources
- Improved DirectX application performance on lower power hardware configurations
- WDDM provides a consistent developer experience by supporting all the new DirectX technologies such as DirectX11.1, D2D, DWrite, DImage
- Higher performance anti-aliasing path for D2D applications
- High Level Shader model performance improvements that can enable developers to do more on the GPU without involving the CPU
- Higher performance for Direct3D 11.1 applications on small form factor platforms and power constrained devices that use tile-based rendering techniques
- Added capabilities to enable shader debugging at all shader stages on DirectX 11.1 hardware
- Enabling Direct3D 11 applications to implement high quality rendering algorithms without needing to allocate memory for large numbers of samples
- Improvements to deferred shading techniques
- Efficient buffer management for game developers

October 11, 2011
© 2011 Microsoft. All rights reserved.
• Developers can benefit from GPU Preemption by not having to optimize the size of the application workload
• With TDR improvements, non-interactive applications, such as Compute, will be allowed to use as much of the GPU as they need as long as they don’t interfere with other applications which need to share the GPU
• Improved ability to diagnose memory related issues with a better view into memory usage
• Improved ability to diagnose performance issues

**System Manufacturers**
• Optimizations to reduce system memory footprint to improve system responsiveness
• Improved reliability with increased GPU fault tolerance
• Ability to boot on headless configurations which have no GPUs, display hardware or stub drivers
• Driver optimizations to reduce system power consumption
• Rich graphical composition model that allows for more flexibility to support video playback and stereoscopic 3D scenarios
• Server platforms will benefit from the higher functionality that WDDM 1.2 provides
  o Compatible with UEFI GOP
  o Reboot-less driver upgrades
  o Compatible with newer versions of DirectX technologies

To deliver the best experience in Windows 8, Windows takes advantage of the graphics hardware paired with an optimized WDDM v1.2 driver.

This document provides information about the various enhancements to the Windows Display Driver Model (WDDM) for Windows® 8 operating systems. WDDM for Windows 8 is referred to as “WDDM v1.2”. This document provides an overview of the Windows 8 features that are only enabled by a WDDM v1.2 driver. To take advantage of these features and provide the best experience, WDDM v1.2 is required by all systems shipped with Windows 8.

The paper is divided into four parts:
• Part 1 provides details on WDDM in Windows 8
• Part 2 focuses on Direct3D feature requirements
• Part 3 covers Graphics INF requirements
• Part 4 covers the Installation experience
• Part 5 covers the WDDM v1.2 driver enforcement guidelines
# PART 1: WDDM in Windows 8

## In this part:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>WDDM v1.2 Features</td>
<td>9</td>
</tr>
<tr>
<td>Performance Improvements</td>
<td>11</td>
</tr>
<tr>
<td>Reduced system memory footprint</td>
<td>11</td>
</tr>
<tr>
<td>Video memory offer and reclaim</td>
<td>11</td>
</tr>
<tr>
<td>Standby hibernate optimizations</td>
<td>12</td>
</tr>
<tr>
<td>Reliability improvements with increased GPU fault tolerance</td>
<td>13</td>
</tr>
<tr>
<td>GPU preemption</td>
<td>14</td>
</tr>
<tr>
<td>TDR improvements</td>
<td>15</td>
</tr>
<tr>
<td>Improved ability to diagnose performance problems</td>
<td>16</td>
</tr>
<tr>
<td>User-mode driver logging</td>
<td>16</td>
</tr>
<tr>
<td>Disabling Frame Pointer Omission (FPO) optimization</td>
<td>17</td>
</tr>
<tr>
<td>XPS rasterization on the GPU</td>
<td>17</td>
</tr>
<tr>
<td>Improvements to enhance end-user experience</td>
<td>17</td>
</tr>
<tr>
<td>Stereoscopic 3D</td>
<td>18</td>
</tr>
<tr>
<td>D3D11 Video Playback Improvements</td>
<td>19</td>
</tr>
<tr>
<td>Optimized Screen Rotation Support</td>
<td>19</td>
</tr>
<tr>
<td>Container ID support for displays (visually representing devices that are embedded within a device)</td>
<td>20</td>
</tr>
<tr>
<td>Seamless handoff of the display control between Windows and the WDDM graphics driver</td>
<td>22</td>
</tr>
<tr>
<td>From Firmware to Windows operating system</td>
<td>22</td>
</tr>
<tr>
<td>From Windows operating system to WDDM graphics driver</td>
<td>23</td>
</tr>
<tr>
<td>From WDDM graphics driver to Windows operating system</td>
<td>23</td>
</tr>
<tr>
<td>Taking over the display control without disabling the WDDM graphics driver</td>
<td>23</td>
</tr>
<tr>
<td>Driver optimizations that can help reduce power consumption</td>
<td>23</td>
</tr>
<tr>
<td>DirectFlip</td>
<td>24</td>
</tr>
<tr>
<td>GPU power management of idle and active power</td>
<td>24</td>
</tr>
<tr>
<td>Advances to the display infrastructure</td>
<td>24</td>
</tr>
<tr>
<td>Microsoft Basic Display Driver</td>
<td>25</td>
</tr>
<tr>
<td>Desktop Duplication API</td>
<td>26</td>
</tr>
<tr>
<td>Support for headless systems</td>
<td>27</td>
</tr>
</tbody>
</table>
Introduction

The Windows Display Driver Model (WDDM) was introduced with Windows Vista as a replacement to the Windows XP Display Driver Model (XDDM). With its introduction in Windows Vista, the WDDM architecture offered functionality needed to enable several new features such as Desktop Composition, enhanced fault tolerance, video memory manager, GPU scheduler, cross process sharing of Direct3D surfaces, and so on. WDDM was specifically designed for modern graphics devices that were Direct3D 9 with pixel shader 2.0 or better and had all the necessary hardware features to support the WDDM features. WDDM for Windows Vista was referred to as “WDDM v1.0”.

Windows 7 made incremental changes to the driver model for supporting Windows 7 features and capabilities and was referred to as “WDDM v1.1”. WDDM v1.1 is a strict superset of WDDM v1.0. WDDM v1.1 introduced support for Direct3D 11, GDI hardware acceleration, Connecting and Configuring Displays, DXVA HD, and many other features. For more details on these features, refer to the Graphics guide for Windows 7.

Windows 8 introduces an array of new features and capabilities that require graphics driver changes. These incremental changes benefit end users and developers, and improve system reliability. The WDDM driver model that enables these Windows 8 features is referred to as “WDDM v1.2”. WDDM v1.2 is a superset of WDDM v1.1 and WDDM v1.0.

These changes can be represented in a simplified form as shown in Table 1.1 below.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Driver Models Supported</th>
<th>Direct3D versions supported</th>
<th>Features enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Vista</td>
<td>WDDM v1.0</td>
<td>D3D9, D3D10</td>
<td>Scheduling, Memory Management, Fault tolerance, D3D9 &amp; 10</td>
</tr>
<tr>
<td></td>
<td>XDDM on Server and limited UMPC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Vista SP1 / Windows 7 client pack</td>
<td>WDDM v1.05</td>
<td>D3D9, D3D10, D3D10.1</td>
<td>+ BGRA support in D3D10, D3D 10.1</td>
</tr>
<tr>
<td></td>
<td>XDDM on Server 2008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows 7</td>
<td>WDDM v1.1</td>
<td>D3D9, D3D10, D3D10.1, D3D11</td>
<td>GDI Hardware acceleration, DXVA HD, D3D11</td>
</tr>
<tr>
<td></td>
<td>XDDM on Server 2008 R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows 8</td>
<td>WDDM v1.2</td>
<td>D3D9, D3D10, D3D10.1, D3D11, D3D11.1</td>
<td>Smooth Rotation, Stereoscopic 3D, D3D11 Video, D3D11.1, etc.</td>
</tr>
</tbody>
</table>

Note: With Windows 8 and WDDM v1.2, XDDM will no longer be supported and XDDM drivers will no longer load on Windows 8 client or server. For the scenarios
traditionally dependent on XDDM, Windows 8 allows migration to WDDM as illustrated in Table 1.2. IHVs and system builders should adopt the alternative WDDM solution that works best for their customers. This means a Windows 8 system will always have a WDDM-based driver.

### Table 1.2 Migrating from XDDM to WDDM

<table>
<thead>
<tr>
<th>Currently using</th>
<th>WDDM support for XDDM scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDDM VGA Driver</td>
<td>Microsoft Basic Display Driver</td>
</tr>
<tr>
<td>XDDM IHV Driver</td>
<td>System builders need to work with the IHV to get:</td>
</tr>
<tr>
<td></td>
<td>• Display Only WDDM Driver</td>
</tr>
<tr>
<td></td>
<td>• Full Graphics WDDM Driver</td>
</tr>
<tr>
<td></td>
<td>Alternately Microsoft Basic Display Driver</td>
</tr>
<tr>
<td>XDDM Virtualization Driver</td>
<td>System builders need to work with the IHV to get a new Display Only Virtualization Driver</td>
</tr>
<tr>
<td>CSM for Int10 support on UEFI</td>
<td>No longer needed with UEFI GOP support</td>
</tr>
<tr>
<td>Remote Desktop Access/Collab</td>
<td>Desktop Duplication API</td>
</tr>
<tr>
<td>Remote Session Driver</td>
<td>No Change, no support for &lt;32bpp modes</td>
</tr>
</tbody>
</table>

**Note:** Microsoft will be providing a WDDM-based Basic Display Driver that is a replacement for today’s in-box XDDM Standard VGA driver that will provide basic display functionality and software-based 2D and 3D rendering.

WDDM v1.2 introduces new types of graphics drivers, targeting specific scenarios as described below:

- **a)** WDDM Full Graphics Driver: This is the full version of the WDDM graphics driver that supports hardware accelerated 2D and 3D operations. This driver is fully capable of handling all the render, display, and video functions. WDDM v1.0 and WDDM v1.1 are full graphics drivers. All Windows 8 client systems must have a full graphics WDDM 1.2 device as the primary boot device.

- **b)** WDDM Display Only Driver: This driver is supported only as a WDDM 1.2 driver and enables IHVs to write a WDDM based kernel-mode driver that is capable of driving display only devices. The OS handles the 2D or 3D rendering using software-simulated GPU. Display only devices are not allowed as the primary graphics device on client systems.

- **c)** WDDM Render Only Driver: This driver is supported only as a WDDM 1.2 driver and enables IHVs to write a WDDM driver that supports only rendering functionality. Render only devices are not allowed as the primary graphics device on client systems.

Table 1.3 summarizes driver model versus the “supported” driver categories and Table 1.4 explains scenario usage for the new driver types.

### Table 1.3 Driver model mapping WDDM driver categories

<table>
<thead>
<tr>
<th>Driver Model</th>
<th>Driver Category</th>
<th>Full Graphics</th>
<th>Display Only</th>
<th>Render Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDDM v1.0 (Windows Vista)</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WDDM v1.1 (Windows 7)</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WDDM v1.2 (Windows 8)</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

October 11, 2011
© 2011 Microsoft. All rights reserved.
Table 1.4 Windows certification requirements for WDDM driver types

<table>
<thead>
<tr>
<th></th>
<th>Client</th>
<th>Server</th>
<th>Client running in a Virtual Environment</th>
<th>Server Virtual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Graphics</td>
<td>Required as boot device</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Display Only</td>
<td>Not allowed</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Render Only</td>
<td>Optional as non-primary adapter</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Headless</td>
<td>Not allowed</td>
<td>Optional</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

WDDM v1.2 is required for all systems shipped with Windows 8. WDDM v1.0 and WDDM v1.1 will continue to work on Windows 8. However, the best experience and Windows 8–specific features are enabled only by a WDDM v1.2 driver.

WDDM v1.2 Features

The WDDM v1.2 feature set includes several new enhancements that improve performance, reliability, and the overall end user experience. Each of these features requires special support from 3rd-party WDDM v1.2 drivers. This section elaborates on what constitutes the WDDM v1.2 feature set.

WDDM v1.2 has both mandatory and optional features. The driver must implement all the mandatory features to claim itself as a “WDDM v1.2 driver”, while the driver can implement any combination (or none) of the optional features. A non-WDDM v1.2 driver must report none of the WDDM v1.2 features.

Table 1.5 summarizes the WDDM v1.2 feature set. “M” indicates mandatory, “O” indicates optional, “NA” indicates not applicable.

Table 1.5 WDDM v1.2 feature set

<table>
<thead>
<tr>
<th>Windows 8 features enabled by WDDM v1.2</th>
<th>Feature Benefit</th>
<th>WDDM Driver Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video memory offer and reclaim</td>
<td>Enables more efficient usage of video memory</td>
<td><strong>M</strong></td>
</tr>
<tr>
<td>GPU Preemption</td>
<td>Improves desktop responsiveness</td>
<td><strong>M</strong></td>
</tr>
<tr>
<td>TDR Improvements</td>
<td>Improved resiliency to GPU hangs</td>
<td><strong>M</strong></td>
</tr>
<tr>
<td>Screen Rotation Support</td>
<td>Optimized screen rotation experience without screen flicker</td>
<td><strong>M</strong></td>
</tr>
<tr>
<td>Stereoscopic 3D</td>
<td>Provides a consistent API &amp; DDI platform to enable Stereoscopic 3D scenarios</td>
<td><strong>O</strong></td>
</tr>
<tr>
<td>Windows 8 features</td>
<td>Feature Benefit</td>
<td>WDDM Driver Type</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>D3D11 Video Playback Improvements</td>
<td>Simplified programming experience for video playback applications</td>
<td>M ** M NA</td>
</tr>
<tr>
<td>DirectFlip</td>
<td>Improvements in the video playback and composition stack to improve battery life</td>
<td>M NA NA</td>
</tr>
<tr>
<td>Bugcheck Support</td>
<td>High resolution maintained when Windows bug checks</td>
<td>M NA M</td>
</tr>
<tr>
<td>PnP Start and Stop</td>
<td>Maintain a high resolution high graphics mode as display ownership is transition between firmware, Windows and driver.</td>
<td>M NA M</td>
</tr>
<tr>
<td>Standby Hibernate Optimizations</td>
<td>Enables optimizations to the graphics stack to improve performance on sleep and resume</td>
<td>O O NA</td>
</tr>
<tr>
<td>GPU power management of idle and active power</td>
<td>Provides a standardized infrastructure for fine-grained device power management</td>
<td>O O O</td>
</tr>
<tr>
<td>XPS Rasterization on the GPU</td>
<td>Enables a quality printing experience on Windows 8 with 3rd party drivers</td>
<td>M ** M NA</td>
</tr>
<tr>
<td>Container ID support for Displays</td>
<td>Helps represent monitor device connectivity and associated state to the user in the user interface like the device hub</td>
<td>M NA M</td>
</tr>
<tr>
<td>Disabling Frame Pointer Omission (FPO) optimization</td>
<td>Improves debugging of performance problems related to FPO on the field</td>
<td>M M M</td>
</tr>
<tr>
<td>User Mode Driver Logging</td>
<td>Improves ability to diagnose in investigating memory related issues by providing better view into memory usage</td>
<td>M M NA</td>
</tr>
</tbody>
</table>

* This feature is “Mandatory” for all WDDM v1.2 drivers with Direct3D 10-, Direct3D 10.1-, Direct3D 11-, or Direct3D 11.1-capable hardware.

** No new DDI changes. IHVs will have to pass the XPS rasterization conformance tests to ensure a quality printing experience for the GPU accelerated XPS printing scenarios.
Note: There will be a new set of APIs available on Windows 8 for duplicating the desktop for collaboration scenarios. For more details please refer to the MSDN documentation on Desktop duplication API.

The following section provides an overview of the features and improvements that constitute the WDDM v1.2 feature set. If you need additional details on these features, please refer to the feature documentation on MSDN. Feature-related logo requirements are captured as part of the Windows Hardware Certification Kit (WHCK) documentation. These features are grouped in the following categories:

- Performance improvements
- Improvements to enhance the end-user experience
- Driver optimizations that can help reduce power consumption
- Advances to the display infrastructure

Performance Improvements

This section elaborates on the various Windows 8 features that improve performance:

- Reduced system memory footprint (Video memory offer and reclaim, Standby Hibernate Optimizations)
- Reliability improvements with increased GPU fault tolerance (GPU Preemption, TDR Improvements)
- Improved diagnosability of performance problems (User Mode Driver Logging, Disabling Frame Pointer Omission optimization)
- XPS Rasterization on the GPU

Reduced system memory footprint

An important goal for Windows 8 is to improve system responsiveness by reducing the system memory footprint. System memory usage significantly affects system responsiveness. Increased system memory usage leads to increased paging activity, which directly leads to reduced system responsiveness. Thus, for the best responsiveness, all applications, processes, and operating system components should use as little system memory as possible.

The following features on Windows 8 will help applications/drivers to efficiently utilize memory resources and take advantage of the hardware they have:

- Video memory offer and reclaim
- Optimizations for sleep and resume

Video memory offer and reclaim

As new classes of applications start using the GPU for hardware acceleration, there is increased sharing of GPU resources such as video memory. Availability of GPU resources is generally not a problem on systems that have high-end GPUs. Today’s modern mid to high-end GPUs have 1G or more video memory on board, which is plenty for the most common user scenarios in Windows. Windows 8, however,
introduces a number of new scenarios where GPU capabilities are exploited by new classes of applications such as Internet Explorer (IE) and those using D2D or Direct Image. These applications, along with existing ones will be vying for existing GPU resources. An important point to consider here is the recent proliferation of low-end PCs. The GPUs in these PCs are typically integrated into the chipset/CPU and use portions of main system memory as video memory. Thus video memory usage translates directly into higher system memory usage. Thus, the amount of GPU resources such as video memory is pretty scanty. Use of multiple applications at the same time often puts a heavy burden on these PCs, adversely impacting system performance and eventually the end-user experience.

This feature provides a technique in which graphics-dedicated and shared system memory could potentially be conserved when not in use, thus making it available to other applications if needed.

Below are some scenarios that result in efficient memory resource usage through this feature:

- On low-end systems, which are under memory pressure, more video memory will be available to other applications running on the system because the WDDM v1.2 driver will offer back memory resources when not needed.
- On systems with sufficient system memory with no memory pressure, the applications will be able to use generous amounts of memory through extra caches and temporary surfaces to speed up performance. Under memory pressure, the use of the offer and reclaim API will result in prudent use of memory, making it available to other applications when needed.
- Temporary video memory resources that are offered by the WDDM v1.2 driver will be deleted by the video memory manager when the system goes into hibernation. This will result in time savings in the event of system hibernation because these resources do not have to be written to the Hiberfile. Time for system resume will also be correspondingly less because the Hiberfile is smaller.

All WDDM v1.2 drivers must use the new UMD offer and reclaim DDI to reduce the overhead of memory resources created for temporary surfaces in local and system memory. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...OfferReclaim”.

Standby hibernate optimizations
Windows 8 offers optimizations to the graphics stack to improve performance on sleep and resume.

When a PC transitions to sleep or resumes from sleep, there are several operations done to ensure video memory content is preserved and restored properly. Some of these operations are unnecessary and can be avoided as explained with the following concepts:

- An integrated graphics adapter uses system memory as video memory. Because system memory is always refreshed when a computer goes to sleep, no eviction
is necessary. Thus the delays introduced by the graphics stack can be brought down to “zero” delay or to the order of few milliseconds.

- The total time to purge on discrete adapters is simply the quotient of the amount of memory purged divided by the rate of purge. This time can be reduced by reducing the amount of memory to purge.

The goal here is to ensure that only data that can be recreated will be discarded.

WDDM v1.2 drivers can take advantage of these optimizations by specifying which allocations should be preserved during power state transitions.

A new generation of discrete graphics adapters may be designed to refresh their memory when in standby (Self Refreshing VRAM). These adapters will also benefit from these optimizations.

Eviction will still be relevant for discrete graphics adapters that don’t have the self-refreshing VRAM feature. For these cases, the performance optimization is to minimize the amount of data that is preserved. For instance, unused data in video memory such as offered allocations, discarded allocations, and unused DMA buffers, can be discarded.

Following are the benefits of utilizing this feature:

- Doing no work: On Integrated and discrete graphics adapters (with self-refresh VRAM feature), the delay introduced by the graphics stack can be brought down to “zero” delay or to the order of few milliseconds.
- Doing less work: On Discrete graphics adapters, the performance improvement will mostly be dependent on how much unused data in video memory is discarded.
- Reduced Memory Trashing: The larger the amount of memory evicted, the greater the effect of memory trashing. This has a bigger impact on discrete graphics adapters because they require a large amount of system memory to evict to.

Every device that can support this feature should take advantage of these optimizations. WDDM v1.2 drivers must set one or more of the Standby Hibernate Flags when enumerating segment capabilities. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...Standby Hibernate Flags”.

Reliability improvements with increased GPU fault tolerance

A common stability problem in graphics occurs when the system appears completely “frozen” or “hung” while processing an end-user command or operation. Users generally wait a few seconds and then reboot the system by pressing the power button. Usually the GPU is busy processing intensive graphics operations, typically during game-play. No screen updates occur, and users assume that their system is frozen.
Both Windows Vista and Windows 7 try to detect these problematic hang situations and recover a responsive desktop dynamically. The system does not reboot, but in most cases the screen flickers as it is redrawn. However, some older DirectX applications render a black screen at the end of recovery, and users must restart these applications. These GPU hangs are referred to as time-out detection and recovery errors (TDRs). Figure 1-1 illustrates the timeout detection and recovery process. For more details on this process, see section on “Timeout Detection and Recovery of GPUs through WDDM” on the WHDC website.

![Timeout Detection and Recovery of GPUs through WDDM](image)

**Figure 1-1. Timeout Detection and Recovery of GPUs through WDDM**

TDRs happen when a GPU command has taken too long to complete or the hardware is hung. TDRs enable the operating system to detect that the UI is not responsive.

Windows 8 introduces the following enhancements to enhance GPU fault tolerance:

- GPU preemption
- TDR improvements

**GPU preemption**

The Windows desktop and application UI are increasingly dependent on the GPU. As more applications leverage the GPU (e.g., IE9, Windows Live and others) it becomes harder to guarantee low-latency access to the GPU to any one application. When a high-priority request to access the GPU comes from an application, the GPU might be busy working on previously submitted work and may take a very long time to complete this work, effectively delaying the high-priority request. As the Windows UI is becoming increasingly touch focused, it is critical for applications trying to respond to touch input to get access to the GPU with the lowest latency possible to give users immediate feedback on their interaction.
This is made worse by the fact that an increasingly larger number of applications are trying to broadly use GPUs. Windows 7 saw the introduction of the Direct Compute API targeted specifically at scenarios wanting to leverage the GPU for compute-intensive applications such as video encoding. Leveraging this API today is not without risk. It is very easy for an application to inadvertently submit a very complex and long workload to the GPU, which may take a significant amount of time to complete during which the GPU is unavailable to other applications, resulting in an unresponsive desktop.

The GPU is a shared resource that is used by many applications, including UI rendering of most applications. Today, applications have to cooperate to avoid this problem. Cooperating is accomplished by attempting to break down the work to be executed on the GPU into small batches to avoid using the GPU exclusively for extended periods of time. This task is much more difficult than it may sound as the performance characteristics of a GPU can change drastically from low end to high end and batch size is a combination of application and driver behavior. There is no strict guideline on how to achieve this either. It is essentially accomplished through trial and error one application/developer at a time.

GPU preemption offers a better solution to this problem by making it possible to preempt long-running workloads on the GPU. This should free developers from having to fine tune their application for every GPU, allowing them to fully leverage the power of the GPU while maintaining great desktop responsiveness and allowing scenarios such as touch to feel great no matter what other applications are doing.

This feature is “required” for WDDM v1.2 drivers (Full Graphics and Render Only). For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...GPU Preemption” and “Device.Graphics....FlipOnVSyncMmlO”.

TDR improvements

The goal of an improved mechanism for GPU Timeout Detection and Recovery (or TDR) is improved resiliency to GPU hangs that are triggered by long-running graphics workloads taking more than the allowed time for work completion. On previous Windows operating systems, repeated operations like these result in repeated TDRs that eventually crash the system. Applications that compute on the GPU and that submit work to the GPU can also take much longer to complete than the allowed timeout.

With the Windows 8 feature support for GPU preemption, these tasks can execute without interfering with other applications, including the desktop window manager (DWM). The TDR improvements consist of the following changes:

- Detection change: Allow applications to opt out of TDR if they want to (long-running compute scenarios). This class of applications won’t hit a TDR for running for extended periods of time as long as the application remains pre-emptible and allows other tasks to run.

- GPU reset: As shown below on figure 1-2, only the hung GPU engine is reset. Only applications running on that GPU engine are affected by the TDR. Applications running on other GPU engines keep running.
• Repeated TDRs: A misbehaving process that causes seven or more successive faults will not be allowed to touch the GPU for its lifetime.

Non-interactive applications such as Compute applications will be allowed to use as much of the GPU as they need as long as they don’t interfere with other applications that need to share the GPU. In order to keep the desktop responsive, applications that negatively interfere with DWM will be penalized by preventing them from accessing the GPU.

![TDR Diagram](image)

**Figure 1-2. TDR Improvements**

This feature is “required” for WDDM v1.2 drivers (Full Graphics and Render Only). This requirement applies for GPUs that support “Per-Engine Reset”. WDDM v1.2 drivers must continue supporting the pre-Windows 8 TDR behavior of adapter-wide reset and restart. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device Graphics...TDR Resiliency”.

**Improved ability to diagnose performance problems**

With an increasing number of graphics applications utilizing GPU resources, the ability to diagnose graphics performance and video memory-related issues has become critical.

**User-mode driver logging**

To get a more actionable breakdown of video memory, the WDDM driver needs to expose the relationship between Direct3D resources and video memory allocations. This will be possible on Windows 8 with the introduction of additional UMD logging interfaces. With this information added to ETW traces it will be possible to see the video memory allocations from the API perspective.

For developers, it will clarify memory costs that are currently very hard to see, like internal fragmentation or the impact of rapidly discarding surfaces. It will also enable...
Microsoft to work better with customers and partners who provide traces for analysis of performance problems. In particular it will help overcome a common blocking point in investigating memory-related performance issues: the application is using too large a working set, but cannot tell which API resources or calls are causing the problem.

This feature is “required” for WDDM v1.2 drivers (Full Graphics and Render Only). A WDDM v1.2 driver must expose the relationship between Direct3D resources and video memory allocations by implementing the UMD ETW interfaces. In addition to the logging events, the driver must be able to report all existing mappings between resources and allocations at any point in time. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...UMD Logging”.

Disabling Frame Pointer Omission (FPO) optimization
On Windows 7, WDDM v1.1 kernel-mode drivers were required to disable FPO optimizations. This is to improve the ability to diagnose performance problems. For Windows 8, the same requirement will be applicable for all WDDM v1.2 drivers (user mode and kernel mode), making it easier to debug performance issues related to FPO on the field.

XPS rasterization on the GPU
This feature does not require any IHV code or behavioral changes. However XPS rasterization is a new usage pattern that could potentially expose bugs or assumptions in driver code. This requirement will help ensure a quality printing experience on Windows 8 with third-party video drivers.

This feature is “required” for WDDM v1.2 drivers (Full Graphics and Render Only). WDDM v1.2 drivers must be able to pass XPS rasterization display conformance tests in order to ensure high-quality Windows printing. The XPS rasterization display conformance requirement tests whether a WDDM v1.2 GPU driver produces correct rasterization results when used by Direct2D in the context of the XPS rasterizer. The XPS rasterizer is a system component used heavily by Windows print drivers to rasterize an XML Paper Specification (XPS) Print Descriptor Language (PDL). To determine the correctness of rasterization results, a comparison is performed between the results obtained from the XPS rasterizer when executed on a system with the subject WDDM 1.2 GPU driver, and results obtain from baseline use of the XPS rasterizer.

Improvements to enhance end-user experience
This section elaborates on the various Windows 8 enhancements that improve the end-user experience:

- Stereoscopic 3D
- Direct3D 11 video playback improvements
- Optimized screen rotation support
- Container ID support for displays
• Seamless handoff of the display control between Windows and the WDDM graphics driver (bugcheck support and PnP Start/Stop in WDDM)

Stereoscopic 3D

Windows 8 provides the ideal platform for further innovations for partners to deliver a solid media experience. Windows 8 enables a rich graphical composition model that allows for more flexibility to support video playback and stereoscopic 3D scenarios.

Windows 8 will provide a consistent API and DDI platform for Stereoscopic 3D scenarios such as gaming and video playback.

Stereoscopic 3D will be enabled only on systems that have all the components that are stereoscopic 3D capable. These include 3D-capable display hardware, graphics hardware, peripherals, and software applications. The Stereo design in the graphics stack is such that the particular visualization or display technology used is agnostic to the operating system. The Graphics driver talks to the Display and has knowledge about the display capabilities through the standardized EDID structure. The driver will enumerate Stereo capabilities only when it recognizes such a display connected to the system.

Display Control Panel Setting: The Stereoscopic display setting will be part of the existing Screen Resolution tab as shown on Figure 1-3.

![Figure 1-3. Stereo 3D Display Setting](image)

The Stereo setting is a checkbox with the following states:

• Not available (either **grayed out** or **invisible**): On systems incapable of Stereo displays.

• Set to Enabled (**checked**): This is the default setting on systems capable of Stereo displays and implies “Stereo-On-Demand”. By default, the DWM will be Mono mode as mentioned in the above section. DWM will switch to Stereo mode only when a stereo application is launched by the user (on-demand). Note that the DWM may thus be in Mono or Stereo mode when this checkbox is checked.

• Set to Disabled (**unchecked**): DWM will be in Mono mode if the user has unchecked this setting. Stereo applications will present in Mono mode in this case.

System builders are encouraged to test their stereo driver packages with the above settings to ensure correct functionality.
Stereo 3D functionality can be enabled only on DirectX 10–capable hardware and higher. However, since Direct3D 11 APIs work on DirectX 9.x and 10.x hardware, all WDDM v1.2 drivers must support and be tested thoroughly to ensure Direct3D 11 APIs work on all Windows 8 hardware.

Although Stereoscopic 3D is an “optional” WDDM v1.2 feature, Direct3D 11 API support is required on all Windows 8 hardware. So WDDM v1.2 drivers (Full Graphics and Render devices) must support Direct3D 11 APIs by adding support for cross-process sharing of texture arrays. This is to ensure that Stereo applications don’t have failures in mono modes. For more information on this requirement, please refer to the WHCK documentation on “Device.Graphics...Stereoscopic 3D Array Support”. Other relevant WHCK requirements are “Device.Graphics...Processing Stereoscopic Video Content” and “Device.Display.Monitor.Stereoscopic 3D Modes”.

D3D11 Video Playback Improvements

With wider adoption of D3D10 technologies in mainstream applications, some application developers would like to treat all content the same, which is challenging with video on D3D9 API and all 2D and 3D content processed through the D3D10 or 11 APIs. Windows 8 will introduce video on D3D11, whereby applications can use a single API to do all graphical operations in one complete API. Following are the key benefits:

- D3D11 video simplifies interoperability between Media Foundation and DirectX technologies.
- Using multiple APIs is harder to program, so using video on D3D11 simplifies the programming experience and makes the program more efficient. The new API provides more flexibility in using decoded and processed video.
- The new D3D11 API for Stereoscopic 3D video unpacks stereo frames into left- and right-eye images.
- Parity with DXVA2.0 and DXVA-HD in decoding and video processing capabilities.
- Works in Session 0 for transcoding scenarios.

D3D11 API support is required on all Windows 8 hardware. This feature is “required” for WDDM v1.2. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...D3D11 Video Decode” and “Device.Graphics...D3D11 Video Processing” requirements.

Optimized Screen Rotation Support

Windows 8 ensures a flicker-free screen rotation experience by ensuring that the output from the graphics adapter stays enabled during rotational mode change. Traditionally on desktops and laptops systems, screen rotation is not a frequently utilized scenario. On Windows 8 there are new form factor devices where screen rotation is utilized as a mainstream scenario. Windows 8 enables optimizations to the display infrastructure and the WDDM driver to ensure the monitor synch stays enabled during this transition.

In addition to the changes in the OS, the following documented DDI (DxgkDdiUpdateActiveVidPnPresentPath) is utilized to set rotation. This DDI has been used to update the Gamma Ramp, but was always documented to allow updating
other aspects of the VidPn paths. This DDI already supports rotation and should work correctly on most drivers. Going forward, this documented DDI will be utilized this way for enabling smooth rotation.

End users can experience the smooth rotation transition when the following are true:

- The platform is running a WDDM 1.2.
- The desktop composition manager is on and actively composing.
- The mode change request is determined to be “compatible” for doing the smooth mode transition. Two modes are compatible if they have the same dimensions (width and height), topology, refresh rates, pixel formats, and stride, and differ only in screen orientation (that is, are rotated).

This feature is “required” on all WDDM v1.2 drivers that support rotated modes.

Container ID support for displays (visually representing devices that are embedded within a device)

In recent years, display manufacturers have started introducing various new capabilities in the monitor devices to provide a better user experience. In particular, USB hubs are popular connectors on monitors for connecting mouse and keyboard. Also, connectors like HDMI support audio and therefore often audio speakers are embedded in monitors as well. Many new display devices also support touch capabilities. This provides a great user experience by reducing wire clutter on the user desktops.

At the same time it’s important to visually represent the connectivity and state of these devices to the user in an intuitive way. During Windows 7, Microsoft introduced the concept of the “Devices and Printers” page. As illustrated in figure 1.4, the Devices and Printers folder shows the user the installed devices that are connected to the PC, making it a handy way to check on a printer, music player, camera, mouse, or digital picture frame (just to name a few). At the same time, it also groups those devices that are contained within the same piece of hardware to make it easier for users to discover all their drivers.
The devices are grouped if they contain the same container ID. During Windows 7 Microsoft introduced the concept of container ID for devices. Per MSDN, “A container ID is a system-supplied device identification string that uniquely groups the functional devices associated with a single-function or multifunction device installed in the computer.”

For the concept of container ID to be successful, all the device classes in Windows must support it and the entire ecosystem needs to implement it in their hardware. In Windows 7, if multiple monitors that support audio are plugged in, it is not easy to for the user to determine which display maps to which audio end points. The same exists for touch digitizers. In Windows 8, the display device class adds support for container ID. This will make it possible for all the functions of a display device to report the same container ID and get visually paired in the Windows UI and the APIs.

To demonstrate the work flow, consider a monitor with audio speakers embedded. The following is the work flow:

1. User connects the monitor using a HDMI cable.
2. WDDM driver reports the presence of display device to the Windows graphics stack.
3. Windows graphics stack queries WDDM driver for the Container ID, using the new DDIs introduced in Windows 8.
4. Graphics driver queries the monitor for the container ID and passes it back to Windows.
5. At the same time the audio driver must pass the exact same container ID to the Windows audio stack.
6. Now if viewed in the Devices and Printers, the display and speakers will get grouped together.
In some cases, it is possible that the display device does not contain a container ID. In such a case, Windows will automatically generate a unique container ID using the manufacturer ID, product ID and serial number obtained from the EDID. Because these values are unique, it will ensure that the container ID is also unique. Windows 8 provides a new DDI that passes the same information to the WDDM driver so that it can be passed to the audio driver to generate the same container ID.

This feature is “required” on WDDM v1.2 drivers (Full Graphics and Display Only devices). For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...Container ID Support”.

**Seamless handoff of the display control between Windows and the WDDM graphics driver**

There are a few scenarios where the ownership of driving the display is transitioned between Windows operating system, WDDM graphics driver and firmware. Currently, these transitions are associated with hardware or the software getting reset or reconfigured. This can cause screen flashes and flickers. The following are the scenarios where such glitches can be seen today:

1. From firmware to Windows operating system:
   a. Transitioning a system from a low power state (shutdown, sleep, hibernate) to an active state
2. From Windows operating system to WDDM graphics driver:
   a. Transitioning a system from a lower power state (shutdown, sleep, hibernate) to an active state
   b. Driver upgrade
   c. Manually/programmatically enable the driver
3. From WDDM graphics driver to Windows operating system:
   a. Driver upgrade
   b. Bugcheck
   c. Manually/programmatically disable the driver

The glitch is most noticeable in the “bug check” case that causes a transition from a high resolution to a low color and lower-resolution mode (resolution falls back to 640 x 480 at 4 bits per pixel). The interface goes into a text-only mode with a blue background.

The glitches caused during these scenarios are not an ideal user experience. However, with advances in the hardware it is now possible to eliminate some of these flashes and provide a fast and fluid transition. Windows 8 introduces updates to the WDDM to enable such a smooth transition.

**From Firmware to Windows operating system**

All Windows 8 systems targeted for client SKUs are required to support a graphics mode via UEFI GOP. The UEFI GOP is also required to set the native timing and native resolution on the integrated panel of the system. When Windows is ready to take over the ownership of the display, the UEFI GOP will hand over a frame buffer that can be used to scan out to the display. At this time Windows will not attempt to reset
the display timings or the resolution. Windows will simply use the provided frame buffer, eliminating one screen flash. For more information on this “system” WHCK requirement, please refer to the WHCK documentation for “System...Firmware UEFI.display”

**From Windows operating system to WDDM graphics driver**

When Windows hands over ownership of the display to the WDDM graphics driver, it is indicated by a PnP start of the device by calling the DxgkDdiStartDevice DDI. At this time, typically the screen is blanked as the WDDM graphics driver takes over the display control. Windows 8 introduces a new API that allows the driver to query the Windows operating system for the exact state of the current frame buffer. With this information, it is possible for the driver to keep the display controller active and not cause a re-sync of the monitor. Because the driver also has detailed information about the frame buffer, it is possible to perform a smoother transition.

**From WDDM graphics driver to Windows operating system**

When the WDDM graphics driver wants to hand over display control to the Windows operating system, it is indicated by a PnP stop of the device by calling the DxgkDdiStopDevice. At this time, typically the screen is blanked as the Windows operating system take over the display control. Windows 8 introduces a new DDI that requires the WDDM driver to setup a frame buffer configured for scan out. The Windows operating system can render into this frame buffer while it is in control of the display. This makes it possible to perform a smooth transition. For more technical information of this hand off, please refer to the relevant WHCK documentation on “Device.Graphics...PnP Stop Start Support”.

**Taking over the display control without disabling the WDDM graphics driver**

Sometimes, the operating system experiences an unrecoverable error and has to “bug check” the system. In such a case, there are certain cases where the Windows operating system has to take over the display control but does not have the ability to stop the WDDM graphics driver. This is particularly true when the operating system is experiencing an unrecoverable error and needs to “bug check” the system.

Windows 8 provides two new interfaces (DxgkDdiSystemDisplayEnable and DxgkDdiSystemDisplayWrite) that allow the system to seamlessly transition into a state where Windows is able to display the error screen while maintaining the graphical interface at a high resolution and color depth. This will eliminate the jarring user experience in this error case.

This feature is “required” on WDDM v1.2 drivers (Full Graphics and Display Only). For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device/Graphics...Display Output Control”.

**Driver optimizations that can help reduce power consumption**

Energy efficiency has become a key distinguishing feature for system builders today. The following two features emphasize the optimizations that can be made on WDDM v1.2 drivers to reduce the energy footprint on Windows 8 based platforms.

- DirectFlip
• GPU power management of idle and active power

DirectFlip
This feature allows for special optimizations to the composition model to reduce power consumption. The following scenarios benefit from the DirectFlip optimization:

• To ensure optimal power consumption for video playback and other full screen scenarios, providing DirectFlip enables a minimum amount of memory bandwidth for displaying full-screen content while ensuring smooth transitions between full-screen applications, other applications, and the desktop environment.

• The user wants to view a video or run an application that covers the entire screen. When the user enters or exits the application or notifications appear over the application, no mode change is required and the experience is smooth. Furthermore, the user enjoys extended battery life on mobile devices because memory bandwidth requirements are reduced for full screen applications like video.

This feature is “required” for all “Full Graphics” WDDM v1.2 drivers. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...DirectFlip” requirements.

GPU power management of idle and active power
Windows 8 will provide new GPU power management infrastructure that will allow WDDM v1.2 drivers to power manage individual devices or a set of devices. The new infrastructure will provide a standardized mechanism to support F-state and P-state power management in collaboration with the operating system.

The key scenarios for this are:

• Mobile form factor device is able to go into idle and save power due to individual system components shutting down if not in use.

• New Windows System-on-Chip (SoC)–based devices behave like consumer devices and mobile phones where they turn on immediately when needed, saving energy.

This feature is “optional” for all WDDM v1.2 drivers. For more information on the feature requirements, please refer to the relevant WHCK documentation on “Device.Graphics...Runtime Power Management” requirements.

Advances to the display Infrastructure
Windows 8 has further enhancements and optimizations to the display infrastructure to further improve the user experience. Key investments include the following:

• Microsoft Basic Display Driver
• Desktop Duplication API
• Support for headless systems
Microsoft Basic Display Driver

On Windows 8, The Microsoft Basic Display Driver will be the in-box display driver that will replace both the XDDM VGA Save and VGA PnP drivers. Following are the key benefits of this change:

- **MSBDD** helps enable a consistent end user and developer experience, because it is compatible with the newer versions of DirectX APIs and technologies such as the “Desktop Composition”.
- Server scenarios can benefit from the higher functionality (specifically, features like reboot-less updates, dynamic start and stop, and so on) provided by the WDDM driver model.
- **MSBDD** will support Unified Extensible Firmware Interface (UEFI) Graphics Output Protocol (GOP).
- **MSBDD** will work on both XDDM and WDDM hardware.

The Microsoft Basic Display Driver is the default in-box display driver that is loaded during setup, in safe mode, in the absence of an IHV graphics driver or when the inbox installed graphics IHV driver is not working or is disabled. The primary purpose of this driver is to enable Windows to write to the display controller’s linear frame buffer.

This driver can either use the video BIOS to manage modes/resolutions on a single monitor, or on UEFI platforms this driver inherits the linear frame buffer set during boot, and no mode/resolution changes are possible.

As represented in Figure 1-5, the following are the scenarios where MSBDD will get utilized:

- **Server**: The MSBDD will be usable on server configurations that lack WDDM-capable graphics hardware.
- **Windows setup**: In the early phases of OS setup just before the final boot, only the MSBDD is loaded:
  - User has an old platform that is currently in working condition though it has no in-box graphics driver support on Windows 8; user upgrades to Windows 8 and uses the MSBDD for the setup, installation; and retrieving an IHV driver if one is available.
- **Driver installation**:
  - When a user is installing a new WDDM IHV driver, the MSBDD is utilized during the transition (from the point when the old WDDM IHV driver is uninstalled to the point before the new IHV driver is installed).
  - When a user is facing problems installing the latest WDDM IHV driver, the user/system could choose to disable the current graphics driver and fallback to using the MSBDD.
- **Driver upgrade**: With MSBDD, there is no need to go through a system reboot when upgrading to the IHV-recommended driver.
• Safe mode: In this mode, only trusted drivers get loaded which is the Microsoft Basic Display Driver (MSBDD).

Figure 1-5. Scenarios supported by Microsoft Basic Display Driver

Desktop Duplication API

Windows 8 introduces a new DXGI-based API that will make it easier for ISVs to support desktop collaboration and remote desktop access scenarios. Such applications are widely used in enterprise and educational scenarios. These applications share a common requirement: access to the contents of a desktop with the ability to transport the contents to a remote location. The new Desktop Duplication APIs will provide access to the desktop contents.

Currently there is no Windows API that allows an application to implement this scenario seamlessly. Therefore, applications use mirror drivers, screen scraping, and other proprietary means to access the contents of the desktop. However, these methods have a set of limitations:
• It can be challenging to optimize the performance.
• It is possible that these solutions don’t support newer graphics-rendering APIs because they are released after the shipping of the product.
• The operating system does not always provide rich metadata to assist with the optimization.
• Not all solutions are compatible with the desktop composition introduced since Windows Vista.

Windows 8 introduces a new DXGI-based API called Desktop Duplication API. This API provides access to the contents of the desktop via bitmaps and associated metadata.
for optimizations. This API works with the Aero theme enabled and is not dependent on the graphics API used by the applications. If the user is able to view the application on the local console then it will be remotable. This means that even full screen DirectX applications can be duplicated. The API does provide protection against accessing protected video content.

The API allows an application to ask Windows to provide access to the contents of the desktop along monitor boundaries. The application may choose to duplicate one or more of the active displays.

When an application requests duplication:

- Windows renders the desktop and provides a copy to the application.
- Each rendered frame is placed in GPU memory.
- Each rendered frame comes with metadata:
  - Dirty region
  - Screen-to-screen moves
  - Mouse cursor information
- Application is provided access to frame and metadata.
- Application is responsible for processing each frame:
  - Application may choose to optimize based on dirty region.
  - Application may choose to use hardware acceleration to process move and mouse data.
  - Application may choose to use hardware acceleration for compression before streaming out.

For detailed documentation and samples, please refer to the Windows Software Development Kit.

**Support for headless systems**

Windows 8 will have support for booting without any graphics hardware, and this will be accomplished with a stub display output if no display devices are found. This stub display is implemented as part of the in-box Microsoft Basic Display Driver. The stub display is used when no PnP driver is available, so this doesn’t require any third-party drivers and it works for both normal operation and for system crashes, so no hardware or firmware support is required to fake a display device.

On architectures in which VGA has previously been the norm, the Basic Display Driver requires positive confirmation that VGA is not present; otherwise it assumes that VGA hardware is available and that the system is not headless. System firmware should set the VGA Not Present flag in the IAPC_BOOT_ARCH field of FADT and if there is any VBIOS, it should implement an empty mode list through the VESA BIOS Extensions (VBE). The intent is that these mechanisms should work to indicate that VGA is not present even if the system implements a VBIOS with int 10h mode 12h support for compatibility with previous versions of Windows. In the absence of VBE support, the Basic Display Driver will use a display initialized by the boot loader so a headless system should not represent a working display through UEFI GOP.
PART 2:
Direct3D Features & Requirements

In this part:
Introduction................................................................................................. 30
DirectX feature improvements on Windows 8........................................ 30
  Pixel formats (5551, 565, 4444).............................................................. 31
  Double-precision shader functionality................................................... 32
    Sum of absolute differences............................................................... 33
Target-independent rasterization (TIR).................................................... 33
  SampleCount =1 (Limited TIR on 10, 10.1 & 11).................................... 33
No overwrite and discard......................................................................... 34
  Rendering content on a tile-based deferred-rendering (TBDR) architecture...34
  Updating resources on a TBDR architecture........................................... 34
  UAVs at every stage............................................................................... 34
Cross-process sharing of texture arrays (for supporting Stereoscopic 3D)....35
  Unordered access views with multi-sample anti-alias sample access........35
Logic ops.................................................................................................. 36
  Improved control of constant buffers.................................................. 36
    Partial constant buffer updates......................................................... 36
    Offsetsetting constant buffer updates.............................................. 36
  Clearview.............................................................................................. 37
Tileable copy flag.................................................................................... 37
  Same-surface blits................................................................................ 37
Software requirements............................................................................. 38
Hardware requirements........................................................................... 39
Introduction

Direct3D offers a rich collection of 3D graphics APIs, widely used by software applications for complex visualization and game development. Depending on the capability of the graphics adapter, Direct3D allows applications to utilize hardware acceleration for the entire 3D rendering pipeline or for partial acceleration. Newer versions of the Direct3D APIs like Direct3D 9Ex and Direct3D 10 are available only from Windows Vista onwards, because WDDM provides the display driver interfaces needed for the functionality. Figure 2-1 illustrates the incremental versions of Direct3D APIs supported on the various versions of WDDM.

![Figure 2-1. Direct3D APIs supported on the various versions of WDDM](image)

The following sections elaborate on the feature improvements, and Windows 8 Direct3D software and hardware requirements.

DirectX feature improvements on Windows 8

Windows 8 includes DirectX feature improvements that benefit developers, end users and system manufacturers in the following areas:

- Higher performance for DirectX applications on lower-power hardware configurations: **Pixel formats (S551, 565, 4444)**
- High Level Shader model performance improvements that can enable developers to do more on the GPU without involving the CPU: **Double-precision shader functionality**
- Higher performance anti-aliasing path for D2D applications: **Target-independent rasterization**
- Higher performance for Direct3D11.1 applications on mobile platforms and power constraint devices that use tile-based renderers: **No overwrite and discard**
• Added capabilities to enable shader debugging at all shader stages on DirectX 11.1 hardware: **UAVs at every stage**
• Enabling DirectX 11 applications to implement high-quality rendering algorithms without needing to allocate memory for large numbers of samples: **Unordered access views with multi-sample anti-alias sample access**
• Improvements to deferred shading techniques: **Logic ops**
• Efficient buffer management for game developers: **Improved control of constant buffers**

**Pixel formats (5551, 565, 4444)**
To better support graphics in low-power configurations using DirectX, the following DirectX9 pixel formats must be supported in Direct3D for Windows 8:
• DXGI_FORMAT_B5G6R5_UNORM
• DXGI_FORMAT_B5G5R5A1_UNORM
• DXGI_FORMAT_B4G4R4A4_UNORM

These additional formats will provide increased performance on lower-power hardware in DirectX applications. These formats are supported on all GPUs to date. Table 2.1 describes the required support for these formats depending on the hardware feature level.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Feature Level 9.0</th>
<th>Feature Level 10.0</th>
<th>Feature Level 10.1</th>
<th>Feature Level 11+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typed Buffer</td>
<td>No</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Input Assembler Vertex Buffer</td>
<td>No</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>Texture1D</td>
<td>No</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Texture2D</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Texture3D</td>
<td>No</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>TextureCube</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Shader Id*</td>
<td>No</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Shader sample* (with filtering)</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Capability</td>
<td>Feature Level 9_x</td>
<td>Feature Level 10.0</td>
<td>Feature Level 10.1</td>
<td>Feature Level 11+</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Shader gather4</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>required</td>
</tr>
<tr>
<td>Mipmap</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Mipmap Auto-Generation</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
</tr>
<tr>
<td>RenderTarget</td>
<td>required for 565, no for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
</tr>
<tr>
<td>Blendable RenderTarget</td>
<td>required for 565, no for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
</tr>
<tr>
<td>UAV Typed Store</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>optional</td>
</tr>
<tr>
<td>CPU Lockable</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>4x MSAA</td>
<td>optional</td>
<td>optional</td>
<td>required for 565, optional for 4444, 5551</td>
<td>required for 565, optional for 4444, 5551</td>
</tr>
<tr>
<td>8x MSAA</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>required for 565, optional for 4444, 5551</td>
</tr>
</tbody>
</table>
## Double-precision shader functionality

WDDM 1.2 drivers that support double precision are now required to support additional double-precision floating-point instructions in HLSL shader model 5 in all shader stages. The new instructions are: Double-Precision Reciprocal, Double Precision Divide, and Double Precision Fused Multiply-Add. Now that the runtime can pass these instructions directly to the driver, the implementation is able to optimize their performance, or implement them as specialized single instructions in hardware.

**Note:** Developers must ensure that they are running with FEATURE_LEVEL_11 or higher with double-precision support (D3D11_FEATURE_DOUBLES), and on a WDDM v1.2 driver, in order to use these features.

## Sum of absolute differences

Image processing is a critical application in modern devices. A common operation is pattern matching or search. Video-encoding operations typically search for matching square tiles (typically 8x8 or 16x16), and image recognition algorithms search for more general shapes identified by a bit mask. To improve the performance of these scenarios, a new intrinsic has been added to HLSL for Shader Model 5.0 in all shader stages. This intrinsic msad4() corresponds to and generates a group of MSAD instructions in the shader IL. All WDDM 1.2 drivers must support this instruction either directly in hardware, or as a set of other instructions (emulated).
Notes:

- Ideally, the MSAD instruction should be implemented such that overflow results in saturation, not in a wrap behavior. However, for this release, developers are warned that overflow behavior is undefined.

- Developers must check to ensure that they are running with FEATURE_LEVEL_11 or higher on a WDDM v1.2 driver to use this feature. Developers must not rely on result accuracy for accumulation values that overflow (that is, go above 65535).

Target-independent rasterization (TIR)

This feature provides a high performance anti-aliasing path for Direct2D usage scenarios involving high-quality anti-aliasing of structured graphics. TIR enables Direct2D to move the rasterization step from the CPU to the GPU while still preserving the Direct2D anti-aliasing semantics and quality. Using this capability, the software layer can evaluate a large number of sub-pixel sample positions for coverage, yet only allocate the memory required for a smaller number of samples. This enables the performance advantage of using the GPU to render but retaining the image quality of a CPU-rendered implementation. This allows a single sample to be broadcast to multiple samples of a multi-sample anti-aliased render target.

**SampleCount =1 (Limited TIR on 10, 10.1 & 11)**

D3D10.0 - D3D11.0 hardware (and Feature Level 10_0 - 11_0) supports ForcedSampleCount set to 1 (and any sample count for Render Target View) along with the described limitations (for example, no depth/stencil).

For 10_0, 10_1 and 11_0 hardware, when ForcedSampleCount is set to 1, line rendering cannot be configured to 2-triangle (quadrilateral)–based mode (that is, the MultisampleEnable state cannot be set to true). This limitation isn't present for 11_1 hardware. Note the naming of the 'MultisampleEnable' state is misleading because it no longer has anything to do with enabling multisampling; instead it is now one of the controls along with AntiAliasedLineEnable for selecting line-rendering mode.

This limited form of target-independent rasterization, ForcedSampleCount = 1, closely matches a mode that was present in D3D10.0 but due to API changes became unavailable for D3D10.1 and D3D11 (and Feature Levels 10_1 and 11_0). In D3D10.0 this mode was the center-sampled rendering even on an MSAA surface that was available when MultisampleEnable was set to false (and this could be toggled by toggling MultisampleEnable). In D3D10.1+, MultisampleEnable no longer affects multisampling (despite the name) and only controls line-rendering behavior.

No overwrite and discard

**Rendering content on a tile-based deferred-rendering (TBDR) architecture**

Render targets in Direct3D 11.1 can now support a discard behavior via a new set of resource APIs. Developers will need to be aware of this capability and call an additional Discard() method to run more efficiently on TBDR architectures (with no penalty to traditional graphics hardware). This will improve performance on mobile platforms and other power-constrained devices that use tiled renderers.
Updating resources on a TBDR architecture

Because TBDR architectures complete multiple passes over the same command buffer, there needs to be special care to tell the driver when a portion of a sub-resource was not modified during a previous draw call. Having a NO_OVERWRITE usage on UpdateSubResource can help the driver in managing resources where no previous draw calls were made to a region of a texture. This simply requires that the developer inform the driver of the application’s intent of either discarding the existing data, or protecting it from overwrite. This will enable more efficient rendering on TBDR architectures while introducing no penalties when run on traditional desktop hardware.

New variants of the Direct3D 11 UpdateSubresource() and CopySubresourceRegions APIs (which both update a portion of a GPU surface) provide an addition Flags field where NO_OVERWRITE or DISCARD can be specified.

These APIs will drive not only the Direct3D 11.1 DDI but also Direct3D 9 DDIs. So new drivers for any DX9+ hardware are required to support revised BLT, BUFBLT, VOLBLT, and TEXBLT DDIs adding the flags discussed here.

These are also required to be supported for all Direct3D 10+ hardware with Direct3D 11.1 drivers.

UAVs at every stage

In Direct3D11, the number of UAVs was limited to eight at the Compute Shader and to eight combined (RTVs + UAVs) at the Pixel Shader. In DX11.1 the number that can be bound has been increased. For DirectCompute the limit is now 64, and for graphics the combined total bound at the output merger is 64 (that is, graphics can have 64 minus the up-to-eight potentially used by RTVs).

Unordered access views can now be accessed from any shader stage, but still come out of the total for the graphics pipeline.

Adding UAVs at every shader stage allows developers to add debugging information to the pipeline. This ease of development will make Windows a more desirable platform for writing GPU-accelerated applications.

This requires a DX11.1 feature level.

Cross-process sharing of texture arrays (for supporting Stereoscopic 3D)

While Stereoscopic 3D is an “optional” WDDM v1.2 system feature, there is underlying infrastructure that must be implemented by all WDDM v1.2 device drivers regardless of whether they support the Stereoscopic 3D system feature.

DirectX 10 (or greater)—capable graphics hardware must support cross-process sharing of texture arrays. This provides a basis for enabling Stereoscopic 3D. The WDDM v1.2 Direct3D DDIs require support of arrayed buffers as render targets independent of hardware feature level.

This ensures that Stereo applications won’t have failures in mono modes. For example: Even for cases when Stereo is not enabled on the system, applications should still be able to create Stereo swap chains or arrayed buffers as render targets...
and then call Present() – in that case, only the left view is displayed (or if the “prefer” right DXGI present flag is set, only the right view).

Hence WDDM v1.2 drivers (Full Graphics & Render devices) must support Direct3D 11 APIs by adding support for cross process sharing of texture arrays. In earlier versions cross-process shared resources could be only single-layer surfaces. In Windows 8, the maximum size of a shared array is two elements (as is sufficient for stereo). For more information on this requirement, please refer to the WHCK documentation on “Device.Graphics...Stereoscopic 3D Array Support”. Other relevant WHCK requirements are “Device.Graphics...Processing Stereoscopic Video Content” and “Device.Display.Monitor.Stereoscopic 3D Modes”.

Unordered access views with multi-sample anti-alias sample access
Direct3D 11 allowed rasterization to unordered access views (UAVs) with no RTVs/DSVs bound. Even though UAVs can have arbitrary sizes, the implementation is able to operate the rasterizer using the pixel dimensions of the viewport/scissor rectangle. The sample pattern for DirectX11 hardware is single sample only. The DirectX 11.1 hardware specification expands to allow multiple samples. This is a variation of target-independent rasterization where only UAVs are bound for output.

UAV-only rendering with multisampling at the rasterizer is now possible by keying off the ForcedSampleCount state described earlier, with the sample patterns limited to 0, 1, 4, and 8 (not 16, which TIR supports). (The UAVs themselves are not multi-sampled in terms of allocation.) A setting of 0 is equivalent to the setting 1 - single sample rasterization.

Shaders can request pixel-frequency invocation with UAV-only rendering. However requesting sample-frequency invocation is invalid (produces undefined shading results). The SampleMask rasterizer state does not affect rasterization behavior at all here.

Support for this feature is available on DirectX 11.0+ hardware including hardware that does not support full 11_1 level of target-independent rasterization with RTVs. The driver can report that it supports UAV-only multi-sample anti-alias sample access (MSAA) rendering (implying 4 and 8 samples are both supported). All DirectX 11+ hardware supports 1. If the hardware can do full 11_1 target-independent rasterization with RTVs (which requires 16-sample support), then UAV-only MSAA rasterization support is required (meaning 4 and 8 samples in the UAV-only case).

This feature enables applications to implement high quality rendering algorithms such as analytic AA without needing to allocate memory for large numbers of samples.

Logic ops
Allowing for logic operations at the output merger allows software developers to perform some operations on images that are currently not possible. For example, they can compute masks much more effectively and easily and also implement modern deferred-shading techniques for 3-D rendering.
Although this functionality exists in most 3D hardware, it is not currently as general as the color blending is. As a result, the configuration of logic ops is constrained in the following ways:

- When logic ops are used in the first RT blend desc, IndependentBlendEnable must be set to false, so that the same logic op applies to all RTs.
- When logic ops are used, all RenderTargets bound must have a UINT or SINT format (undefined rendering otherwise).

WDDM v1.2 drivers (Full Graphics & Render devices) must support logic ops.

**Improved control of constant buffers**

**Partial constant buffer updates**

Constant buffers today require monolithic copies from source to destination during updates that clobber the entire buffer. Where it is desired to update only a portion of the constant buffer, an offset for the writes is ideal. This ability to random-access write into a constant buffer has been requested by game developers and makes constant buffer management much more natural and efficient. These capabilities were already supported for other buffer types, and are now added to constant buffers in WDDM1.2 drivers.

This feature must be supported for all Direct3D 10+ hardware with Direct3D 11.1 drivers.

For the developer, this is emulated on DX9 hardware so it works on all feature levels.

**Note:** Either the NO_OVERWRITE or DISCARD flag must be specified.

**Offsetting constant buffer updates**

A common desire for high-performance game engines is to collect a large batch of constant buffer updates for constants to be referenced by separate Draw*() calls, each needing its own constants, all at once. This is facilitated by allowing the application to create a large buffer and then pointing individual shaders to regions within it (like a view, but without having to make a whole object to describe the view).

Constant buffers now can be created with a size larger than the maximum constant buffer size addressable by an individual shader (at most 4096 16-byte elements - 65kB, where each "element" is one four-component shader constant). The constant buffer resource size is now limited only by the size of memory allocation the system is capable of handling.

When a constant buffer larger than 4096 elements is bound to the pipeline via *SetShaderConstants() APIs such as VSSetShaderConstants(), it appears to the shader as if it is only 4096 elements in size.

A new variant of the *SetShaderConstants() APIs, *SetShaderConstants1() allows a "FirstConstant" and "ConstantCount" to be specified along with the binding. When the shader accesses a constant buffer bound this way it will appear as if it starts at the specified "FirstConstant" offset (where 1 means 16 bytes) and has a size defined by ConstantCount (number of 16-byte constants). This is basically a lightweight
"View" of a region of a larger constant buffer. (Both FirstConstant and ConstantCount must be a multiple of 16).

This feature must be supported by all WDDM v1.2 drivers for Direct3D 10+ hardware. The Direct3D 11 runtime emulates the appropriate behavior for Feature Level 9_x.

Clearview
This feature enables the implementation to perform an efficient clear operation on a video memory resource, clearing multiple rects in a single API/DDI call. The API includes support for rectangles defining a subset of the resource to be cleared. This capability was supported in the DX9 DDI, and is now required for Windows 8 drivers (WDDM 1.2). This approach results in improved performance for 2D operations such as those used in imaging and UI.

Tileable copy flag
Tileable copy allows an application to notify the implementation that the image source and destination are pixel aligned and will participate in no cross-pixel exchange of information in a subsequent rendering pass. This enables significant performance improvements on some implementations that benefit from caching subsets of the image data during the copy operation. This capability was supported in the DX9 DDI, and is now required for Windows 8 drivers (WDDM 1.2).

Same-surface blits
Many UI operations such as scrolling require transferring image data from one portion of an image to another. This feature adds support for a copy operation where both source rectangle and destination rectangle are in the same image/resource. In the case of overlapping source and destination rectangles, the situation must be handled correctly by the implementation/driver. This was already required by the DX9 DDI and is now required in WDDM v1.2 for all hardware. This approach results in significant performance improvements of key UI scenarios.

Software requirements
For Windows 8, independent hardware vendors must write a WDDM v1.2 driver capable of supporting the relevant Direct3D “feature level” user-mode driver (UMD) DDIs. For instance, Direct3D 9–capable hardware must at least support the Direct3D version 9 DDI. These software requirements vary based on the DirectX hardware level as specified on Table 2.2 below.

<table>
<thead>
<tr>
<th>DirectX Hardware</th>
<th>Software Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3D9</td>
<td>Required: WDDM v1.2</td>
</tr>
<tr>
<td></td>
<td>Required: D3D9 - UMD DDI</td>
</tr>
<tr>
<td>D3D10</td>
<td>Required: WDDM v1.2</td>
</tr>
<tr>
<td></td>
<td>Required: D3D9 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11.1 - UMD DDI</td>
</tr>
</tbody>
</table>
### DirectX Hardware

<table>
<thead>
<tr>
<th>DirectX Hardware</th>
<th>Software Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3D10.1</td>
<td>Required: WDDM v1.2</td>
</tr>
<tr>
<td></td>
<td>Required: D3D9 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10.1- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11.1 - UMD DDI</td>
</tr>
<tr>
<td>D3D11</td>
<td>Required: WDDM v1.2</td>
</tr>
<tr>
<td></td>
<td>Required: D3D9 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10.1- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11.1 - UMD DDI</td>
</tr>
<tr>
<td>D3D11.1</td>
<td>Required: WDDM v1.2</td>
</tr>
<tr>
<td></td>
<td>Required: D3D9 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D10.1- UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11 - UMD DDI</td>
</tr>
<tr>
<td></td>
<td>Required: D3D11.1 - UMD DDI</td>
</tr>
</tbody>
</table>

Table 2.3 elaborates on the new functionality that is exposed via User-mode driver (UMD) DDI changes on Windows 8.

### Table 2.3 New functionality exposed via UMD changes in Windows 8

#### Software Spec Changes

<table>
<thead>
<tr>
<th>D3D9 - UMD DDI exposes the following new features in Window 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D3D11.1 - UMD DDI exposes the following new features in Windows 8 across FL 10, 10.1, 11, and 11.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D3D11.1 - UMD DDI exposes the following new features for feature level 11 &amp; 11.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D3D11.1 - UMD DDI exposes the following new features for feature level 11.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
</tbody>
</table>
### Software Spec Changes

<table>
<thead>
<tr>
<th>Required</th>
<th>UAV-MSAA (at 16 samples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>TIR</td>
</tr>
</tbody>
</table>

### Hardware requirements

Independent hardware vendors must follow the Windows 8 Direct3D rendering requirements for hardware as specified on table 2.4 below and refer to the DirectX feature level hardware DDK documentation for specifics.

#### Table 2.4 Direct3D Rendering Requirements for Hardware

<table>
<thead>
<tr>
<th>DirectX Hardware Version</th>
<th>Required/Optional</th>
<th>Window 8 Rendering Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3D9</td>
<td>Required</td>
<td>D3D9 HW Spec</td>
</tr>
<tr>
<td>D3D10</td>
<td>Required</td>
<td>D3D9 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10 HW Spec</td>
</tr>
<tr>
<td>D3D10.1</td>
<td>Required</td>
<td>D3D9 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10.1 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D11 HW Spec</td>
</tr>
<tr>
<td>D3D11</td>
<td>Required</td>
<td>D3D9 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10.1 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D11 HW Spec</td>
</tr>
<tr>
<td>D3D11.1</td>
<td>Required</td>
<td>D3D9 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D10.1 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D11 HW Spec</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>D3D11.1 HW Spec</td>
</tr>
</tbody>
</table>

Table 2.5 provides guidance on the Direct3D hardware spec updates for Windows 8.

#### Table 2.5 Direct3D Rendering Requirements for Hardware

<table>
<thead>
<tr>
<th>Hardware Spec Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D3D10 HW Spec changes for Window 8</strong></td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If Implemented</td>
</tr>
<tr>
<td><strong>D3D10.1 HW Spec changes for Windows 8</strong></td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td><strong>Hardware Spec Changes</strong></td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>D3D11 HW Spec changes for Windows 8</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>D3D11.1 HW Spec for Windows 8</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Required</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
<tr>
<td>If implemented</td>
</tr>
</tbody>
</table>

* Already in the Direct3D 9 hardware specification but previously weren't exposed in Direct3D 10.
PART 3:
Graphics INF Requirements

In this part:
Introduction........................................................................................................43
Updated Feature score directive......................................................................43
Updated friendly name.....................................................................................45
SKU differentiation directive..........................................................................46
SKU differentiation for device drivers..........................................................46
General UNICODE requirement......................................................................47
Installed display drivers directive.................................................................48
Copy flags to support PNP stop directive......................................................48
Driver\services start type directive.................................................................49
Capability override settings to disable OpenGL...........................................49
[Version] section directives.........................................................................49
[SourceDiskNames] section directives...........................................................50
General x64 directives....................................................................................50
General install section directives.................................................................51
[String] section changes for localized strings..............................................51
Driver DLL for display adapter or chipset has properly formatted file version..52
Introduction

WDDM drivers in Windows 8 will require INF changes to the graphics driver. The most notable change is in the feature score. Newer WDDM v1.2 drivers will require a higher feature score than existing WDDM drivers.

Following is the list of all relevant INF requirements for Windows 8 graphics drivers. This section elaborates on each one of these:

- Updated feature score for all Windows 8 display drivers
- Updated friendly name for WDDM v1.2 display drivers
- SKU differentiation directive for all Windows 8 display drivers
- General requirement that the INF encoded as UNICODE for all Windows 8 display drivers
- InstalledDisplayDrivers directive
- Copy flag for the user-mode driver binary
- Driver/Services Start Type directive
- CapabilityOverride settings
- Version section directive
- SourceDiskNames directive
- x64 architecture directive
- General install section directives
- String section changes for localized strings
- Driver DLL for display adapter or chipset has properly formatted file version

Updated Feature score directive

This is a new general installation setting that is required for all Windows 8 Display Driver Model drivers. Table 3.1 shows the values that apply for Windows 8 (key changes highlighted in red).

<table>
<thead>
<tr>
<th>Driver Model</th>
<th>Feature Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8 WHQL</td>
<td>E0</td>
</tr>
<tr>
<td>Windows 8 Pre-Release Driver</td>
<td>E3</td>
</tr>
<tr>
<td>Windows 7 WHQL</td>
<td>E6</td>
</tr>
<tr>
<td>Windows 7 Inbox</td>
<td>EC</td>
</tr>
<tr>
<td>Windows Vista WHQL</td>
<td>F6</td>
</tr>
<tr>
<td>Windows Vista Inbox</td>
<td>F8</td>
</tr>
<tr>
<td>Microsoft Basic Display Driver</td>
<td>FB</td>
</tr>
<tr>
<td>XDDM 3rd party</td>
<td>FC</td>
</tr>
<tr>
<td>XDDM Inbox in Windows Vista</td>
<td>FD</td>
</tr>
</tbody>
</table>
| VGA                         | FE            | *(Not used in Windows 8)*
<table>
<thead>
<tr>
<th>Driver Model</th>
<th>Feature Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default or No Score</td>
<td>FF</td>
</tr>
<tr>
<td>Unsigned drivers</td>
<td>No feature score = FF</td>
</tr>
</tbody>
</table>

**Note:** IHVs should not insert the feature score for their XDDM drivers. Microsoft will apply the appropriate feature score through the Class Installer or in the in-box XDDM driver INFs.

Each operating system release introduces a new feature score “value”. For Windows 8 this is “E3” for in-box drivers and “E0” for WHQL drivers. The feature score is used by the operating system when deciding which driver to install when multiple possible drivers exist. The intent is that a driver with a higher ranked feature score is chosen because it is considered “better”.

All Windows 8 in-box driver devices have a higher ranked feature score than all existing Windows 7 drivers, because the in-box drivers have been tested on Windows 8 and that the existing Windows 7 drivers have not. This will result in the in-box Windows 8 driver replacing existing Windows 7 drivers. An IHV may use the “E0” feature score with a Windows 7 driver if:

- The driver has been tested on Windows 8.
- The driver has “fixes” that make it better than the Inbox driver.
- The driver is intended to be retained on upgrade to Windows 8.

Here are the elements used to choose the best match on a driver in order from most to least significant:

1. **Signature**
   a. Signed
   b. Unsigned
2. **Scope**
   a. Specific
   b. Basic - DNF_BASIC_DRIVER
3. **Signature score**
   a. Within signed
      i. #define SIGNERSCORE_LOGO_PREMIUM   0x0D000001
      ii. #define SIGNERSCORE_LOGO_STANDARD 0x0D000002
      iii. #define SIGNERSCORE_INBOX       0x0D000003
      iv.  #define SIGNERSCORE_UNCLASSIFIED 0x0D000004 // UNCLASSIFIED == INBOX == STANDARD == PREMIUM when the SIGNERSCORE_MASK filter is applied
      v.   #define SIGNERSCORE_WHQL        0x0D000005 // base WHQL.
      vi.  #define SIGNERSCORE_AUTHENTICODE 0x0F000000
   b. Within unsigned
      i. #define SIGNERSCORE_UNSIGNED      0x80000000
      ii. #define SIGNERSCORE_W9X_SUSPECT   0xC0000000
      iii. #define SIGNERSCORE_UNKNOWN     0xFF000000
4. **Feature Score**, for display
   a. Windows 8 WHQL   E0
   b. Windows 8 Pre-Release Driver  E3
5. Match type (INF matches are listed under the models section as
Description=Install Section, HWID, CompatID. With 0 or 1 HW IDs and 0 or
more CompatIDs)
   a. Device HardwareID == INF HardwareID
   b. Device HardwareID == INF CompatID
   c. Device CompatID == INF HardwareID
   d. Device CompatID == INF CompatID

6. Match rank
   a. Priority of match within list of matches from device

7. Driver date

8. Driver version number

Updated friendly name

This is a localizable string name requirement for all Windows 8 in-box display driver
INFs. All Windows 8 in-box drivers are to use the E3 feature score, regardless of
friendly name. The friendly name will reflect the driver model supported by the INF
per below.

For WDDM 1.2 drivers that were tested on Windows 8 and are being included in the
box in Windows 8, (Microsoft Corporation – WDDM v1.2) must be appended to the
device name, as shown in the following example.

; Localizable Strings

IHV_DeviceName.XXX = “Foo Device Name (Microsoft Corporation – WDDM v1.2)”

Note: To easily highlight drivers for testing only, that are going to enable Windows 8–
specific optional features, “optimized for Windows 8”, I strongly suggest the following
so that end users can easily see that it’s not a standard Windows 8 driver. (This
should make bugs easier to triage as well).

For example: WDDM v1.2 specific work
IHV_DeviceName.XXX = “Foo Device Name (Engineering Sample – WDDM v1.2)”

For WDDM 1.1 drivers that were tested on Windows 8 and are being included in the
box in Windows 8, (Microsoft Corporation – WDDM v1.1) must be appended to the
device name, as shown in the following example.
; Localizable Strings

IHV_DeviceName.XXX = "Foo Device Name (Microsoft Corporation – WDDM v1.1)"

For WDDM 1.0 drivers that were tested on Windows 8 and are being included in the box in Windows 8, *(Microsoft Corporation – WDDM v1.0)* must be appended to the device name, as shown in the following example.

; Localizable Strings

IHV_DeviceName.XXX = "Foo Device Name (Microsoft Corporation – WDDM v1.0)"

### SKU differentiation directive

As of Windows Server 2008 and Windows Vista SP1, the in-box display driver INFs were modified to include a new value that represented the drivers as “Client Only” and that they would not install on “Server SKUs”. This new directive is now required for all display drivers in Windows 8.

In Windows Vista RTM the following values were used.

X86:

[Manufacturer]

%ATI% = ATI.Mfg

[ATI.Mfg]

In Vista SP1\Server 2008 the following values were used:

X86:

[Manufacturer]

%ATI% = ATI.Mfg,NTx86...1

[ATI.Mfg,NTx86...1]

X64:

[Manufacturer]

%ATI% = ATI.Mfg,NTamd64...1

[ATI.Mfg,NTamd64...1]

For Windows 8, the same values used for Windows Vista SP1 and Windows Server 2008 are to be used.
SKU differentiation for device drivers

IHVs can use “ProductType” INF values to indicate that a given INF is valid only for server or client platforms. This works on Windows XP and later operating systems and the changes are relatively simple to implement.

This means that even if a client-only driver package is present in the driver store of a server system, that driver would not be installable. To do so, someone would have to hack the INF and re-import the driver package as unsigned.

**Details:** The “INF Manufacturer Section” description in the DDK shows add “TargetOSVersion” to filter device installations based on various criteria. One of these criteria is “ProductType”, which can be used to specify a category of SKUs that the package can be installed on. These are the values defined for ProductType:

```
0x00000001 (VER_NT_WORKSTATION)
0x00000002 (VER_NT_DOMAIN_CONTROLLER)
0x00000003 (VER_NT_SERVER)
```

For any given architecture, say x64, here is how a typical INF would be decorated to install on any SKU:

```
[Manufacturer]
%MSFT%=Models.amd64

[Models.NTam64]
<models entries>

In order to restrict this INF to install on client only, you need to add a ProductType of “1” to the decoration. The number may be expressed as decimal or hexadecimal... the documentation shows hexadecimal, but I will use decimal in the example for simplicity.

[Manufacturer]
%MSFT%=Models.amd64...1

; models section for workstation
[Models.NTam64...1]
<models entries>

For server, the syntax breaks it down to install on a client and a plain server. Each of these has its own product type... unfortunately the INF syntax needs you to specify both to cover both cases. Thus you need to duplicate the entire models section to really cover the server SKU:

[Manufacturer]
%MSFT%=Models.amd64...1amd64...3
; models section for client
[Models.NTamd64...1]
IHV_DeviceName.XXX = "Foo Generic Device Name (Microsoft Corporation – WDDM v1.2)"
IHV_DeviceName.YYY = "Foo Enthusiast Device Name (Microsoft Corporation – WDDM v1.2)"
<models entries>

; models section for Server
[Models.NTamd64...3]
IHV_DeviceName.XXX = "Foo Generic Name (Microsoft Corporation – WDDM v1.2)"
IHV_DeviceName.ZZZ = "Foo Datacenter Name (Microsoft Corporation – WDDM v1.2)"
<models entries>

General UNICODE requirement
INF files should be saved, encoded as UNICODE; they must not be ANSI.

To check for UNICODE in INF files
1. Use Notepad to open the INF file.
2. On the File menu, click Save As.
3. If “ANSI” appears in the Encoding field of the dialog box, change the encoding to “Unicode” and save the file under a new name.

Figure 3-1 shows the Save As dialog box for a file that has ANSI encoding.

![Figure 3-1. Save As Dialog Box that has ANSI encoding](image)

The proper default value is shown below on Figure 3-2.

![Figure 3-2. Save As Dialog Box that has Unicode encoding](image)

Installed display drivers directive
This is a new software device setting that gives the proper name for the UMD installed as part of this driver package.

HKR,, InstalledDisplayDrivers, %REG_MULTI_SZ%, UserModeDriverName1, UserModeDriverName2, UserModeDriverNameWow1, UserModeDriverNameWow2

For example:

X86:
```
HKR,, InstalledDisplayDrivers, %REG_MULTI_SZ%, r200umd
```

X64:
```
HKR,, InstalledDisplayDrivers, %REG_MULTI_SZ%, r200umd, r200umdva, r200umd64, r200umd64va
```

Copy flags to support PNP stop directive

This is a new file section flag that is required for Windows Vista Display Driver Model in order to support reboot-less driver upgrades.

**Note:** This is required only for the user-mode driver binaries, not the kernel-mode driver entry.

For example:

```
; File sections
;
[r200.Miniport]
   r200.sys

[r200.Display]
   r200umd.dll,,0x000004000  ; COPYFLG_IN_USE_TRY_RENAME
   r200umd2.dll,,0x000004000  ; COPYFLG_IN_USE_TRY_RENAME
```

Driver\services start type directive

This is a service installation setting requirement for all display drivers. WDDM drivers are PnP so must be demand started, StartType =3.

For example:

```
; Service Installation Section
;```
Capability override settings to disable OpenGL

This is a software device setting for all in-box display INFs to ensure that no in-box drivers are exposed to possible interoperability issues with out-of-box OpenGL ICDs. For example:

[R200_SoftwareDeviceSettings]
HKR,, CapabilityOverride,, %REG_DWORD%, 0x8

[Version] section directives

All inbox drivers must NOT reference the layout.inf file.

All inbox drivers must NOT reference any catalog files.

For example:

[Version]
Signature="$Windows NT$"
Provider=%MSFT%
ClassGUID={4D36E968-E325-11CE-BFC1-08002BE10318}
Class=Display
DriverVer=11/22/2004, 6.14.10.7000

Note:
no line item for LayoutFile=layout.inf
no line item for CatalogFile=delta.cat

WHQL Display Drivers must NOT reference the layout.inf file.

For example:

[Version]
Signature="$Windows NT$"
Provider=%HIV%
ClassGUID={4D36E968-E325-11CE-BFC1-08002BE10318}
Class=Display
DriverVer=11/22/2004, 6.14.10.7000

Note:
[SourceDiskNames] section directives

On Windows Vista and later versions of Windows, in-box INFs use the [SourceDisks] directives. However, the values of these sections were changed from what is typically noted in an IHV production driver package today.

For example, IHV production drivers:

[SourceDisksNames]
1 = %DiskID1%

[SourceDisksFiles]
r200.sys = 1
r200umd.dll = 1

The Windows Inbox INF requirement:

[SourceDisksNames]
3426=windows cd

[SourceDisksFiles]
IHKD. sys = 3426
IHVUMD. dll = 3426
IHVVID. dll = 3426

[SignatureAttributes] Section Directives:

On Vista and subsequent OS's, inbox INFs use the [SignatureAttributes] directives. There is no need to reference your miniport (.sys) file.

For example:

[SignatureAttributes]
IHVUMD1. dll=SignatureAttributes.PETrust
IHVUMD2. dll=SignatureAttributes.PETrust

[SignatureAttributes.PETrust]
PETrust=true

General x64 directives

This is a general reminder for what changes are needed to properly decorate the INF for use on 64-bit Windows.

For example:

[DestinationDirs]
DefaultDestDir = 11
R200:Miniport = 12 ; drivers
R200:Display = 11 ; system32
R200:DispWow = 10, SysWow64

[Manufacturer]
%ATI% = ATI.Mfg, NTamd64

[ATI.Mfg,NTamd64]

[R200_RV200]
FeatureScore=F8
CopyFiles=R200:Miniport, R200:Display, R200:DispWow
AddReg = R200:SoftwareDeviceSettings
AddReg = R200_RV200:SoftwareDeviceSettings
DelReg = R200:RemoveDeviceSettings

; File sections
;
[r200:Miniport]
r200.sys

[r200:Display]
r200umd.dll,,0x00004000 ; COPYFLG_IN_USE_TRY_RENAME

[R200:DispWow]
r2umd32.dll,,0x00004000 ; COPYFLG_IN_USE_TRY_RENAME

General install section directives
This is a general reminder that all references to out-of-box or production/retail binaries, services, regadd, or delreg sections that are normally part of your retail WHQL driver packages are not listed in the Windows in-box driver packages.

It’s hard to list examples for this because it varies so much per vendor, but the rule of thumb is to not refer to anything required by your OpenGL ICDs, OpenCL, Control Panel, Help files, out-of-box services, polling applications, and so on.

[String] section changes for localized strings
This INF requirement is in place to ensure that pseudo-localized builds work. The requirement is to delineate localizable versus non-localizable strings within the strings section using these specific examples given below.

For a **BAD** example of this (no preface of what is localized or not):
Driver DLL for display adapter or chipset has properly formatted file version

The file version of the display driver DLLs must be of the form A.BB.CC.DDDD:

- The A field must be set to 9 for WDDM 1.2 drivers on Windows 8.
- The A field must be set to 8 for WDDM 1.1 drivers on Windows 7.
- The A field must be set to 7 for WDDM 1.0 drivers on Windows Vista.
- The A field must be set to 6 for XDDM drivers on Windows Vista.

For Windows 7 and earlier (WDDM 1.1 and earlier) drivers the BB field must be set to the DDI version that the driver supports:

- DirectX 9 drivers (which expose any of the D3DDEVCAPS2_* caps) must set BB to 14.
- DirectX 10 drivers must set BB to 15.
- Direct3D 11-DDI driver on Direct3D 10 hardware must set BB to 16.
- Direct3D 11-DDI driver on Direct3D 11 hardware must set BB to 17.

For Windows 8 (WDDM 1.2) drivers the BB field must be set to the highest DirectX feature level supported by the driver on the graphics hardware covered by the driver:

- A Feature Level 9 driver must set BB to 14.
- A Feature Level 10 driver must set BB to 15.
- A Feature Level 11 driver must set BB to 17.
- A Feature Level 11_1 driver must set BB to 18.
Because for WDDM 1.2 drivers we are setting $BB$ to reflect feature level supported, irrespective of hardware DX level, 16 is not used, as it was specific to D3D11-DDI on DX10 hardware for WDDM 1.1 drivers.

The $CC$ field can be equal to any value between 01 and 9999.

The $DDDD$ field can be set to any numerical value between 0 and 9999.

For example:

- Windows Vista DirectX 9.0–compatible WDDM drivers can use the range 7.14.01.0000 to 7.14.9999.9999.
- Windows 7 DirectX 10.0–compatible WDDM 1.1 drivers can use the range 8.15.01.0000 to 8.15.9999.9999.
- Windows 8 WDDM 1.2 drivers on DX10 hardware would be 9.15.01.0000 to 9.15.9999.9999.

**Recommendation** (this will become a requirement in a future release): It is strongly recommend that the DriverVer in the display driver .INF file also conform to the above DLL version-numbering requirement, except that for Windows 8, WDDM 1.2 drivers, the $BB$ field in the INF DriverVer must be set for the highest DirectX feature level supported by the driver on the graphics hardware listed in the INF.
PART 4:
Installation Scenarios

In this part:
Install..........................................................................................................................54
Upgrade.........................................................................................................................54
Down level....................................................................................................................54
Install

Windows 8 maintains compatibility with older driver models with the exception of XDDM. Windows Vista— and Windows 7—certified WDDM v1 and v1.1 drivers will install on Windows 8 systems (for devices with no Windows 8 in-box WDDM IHV driver). XDDM hardware drivers will not load on Windows 8 systems. Graphics IHVs will be shipping WDDM v1.2 drivers that will fully exploit the graphics improvements made in Windows 8 and support all available features thereon.

Upgrade

Table 4.1 illustrates the upgrade experience from Windows 7 client or Windows Server 2008 R2 to Windows 8. If a WDDM driver exists, it is migrated to the new OS. However, an in-box IHV driver will outrank a migrated IHV driver unless the migrated driver is a WDDM WHQL driver with a higher-ranked feature score. XDDM drivers and VGA drivers will not be retained. The system will default to the Microsoft Basic Display Driver (MSBDD) on devices with no Windows 8 in-box WDDM IHV driver.

<table>
<thead>
<tr>
<th>Current OS</th>
<th>Target OS</th>
<th>Current Driver</th>
<th>Driver Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7 client or Windows Server 2008 R2</td>
<td>Windows 8</td>
<td>WDDM IHV Driver</td>
<td>Windows 8 in-box WDDM IHV driver OR migrated WDDM IHV driver (for devices with no Windows 8 driver)</td>
</tr>
<tr>
<td>XDDM IHV Driver</td>
<td>MSBDD (for devices with no in-box Windows 8 driver)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS VGA Driver</td>
<td>Windows 8 in-box WDDM IHV driver OR MSBDD (for devices with no in-box Windows 8 driver)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The INF for in-box drivers must have the same feature score for all devices. Feature score must be associated with the Windows OS version the driver was tested on, not the DDI version of the device.

Down level

An IHV may choose to create a unified driver package that is a WDDM 1.2 driver on Windows 8, but appears like a WDDM 1.1 or 1.0 driver on previous OS releases.
PART 5:
WDDM v1.2 Driver Enforcement Guidelines

In this part:
  WDDM v1.2 Driver Enforcement................................................................. 56
  User experience when a driver fails the dxgkrnl validation.................. 56
  WDDM driver and feature caps................................................................. 56
  Best practices........................................................................................... 57
  Resources.................................................................................................. 58
  Glossary.................................................................................................... 58
WDDM v1.2 Driver Enforcement

There will be dxgkrnl validation enforced in the Windows 8 operating system for checking whether the mandatory v1.2 features are supported by the WDDM v1.2 driver. WDDM v1.2 has both mandatory and optional features. The driver must set all the mandatory feature caps to claim itself as a “WDDM v1.2 driver”, while the driver can implement any combination (or none) of the optional features. A non-WDDM v1.2 driver must report none of the WDDM v1.2 features.

User experience when a driver fails the dxgkrnl validation

If a driver has wrongfully claimed itself as “WDDM v1.2” or has implemented partial features (only some of the mandatory features), then it will fail to create an adapter and the system will fall back to the Microsoft Basic Display Driver.

WDDM driver and feature caps

Table 5.1 lists the requirements for a driver to specify to Windows the “WDDM Driver Type” and version. Table 5-2 lists all the feature caps (visible to dxgkrnl) that WDDM v1.2 drivers are required to set.

<table>
<thead>
<tr>
<th>Table 5.1 WDDM Driver Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full Graphics</strong></td>
</tr>
<tr>
<td><strong>Display-Only</strong></td>
</tr>
<tr>
<td><strong>Render-Only</strong></td>
</tr>
<tr>
<td><strong>OR</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5.2 WDDM v1.2 Feature Caps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>WDDM version</td>
</tr>
<tr>
<td>Bugcheck and PnP Stop support for Non VGA</td>
</tr>
<tr>
<td>Optimized screen rotation Support</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Feature Table

<table>
<thead>
<tr>
<th>Feature</th>
<th>WDDM Driver Type</th>
<th>Feature Caps</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Preemption</td>
<td>M</td>
<td>DXGK_DRIVERCAPS::PreemptionCaps</td>
</tr>
<tr>
<td>FlipOnVSyncMmlo</td>
<td>M</td>
<td>DXGK_FLIPCAPS::FlipOnVSyncMmlo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlipOnVSyncMmlo is NOT a new feature; this is already documented and has been available since Windows Vista; the requirement here is to set FlipOnVSyncMmlo cap</td>
</tr>
<tr>
<td>TDR Improvements</td>
<td>M</td>
<td>DXGK_DRIVERCAPS::SupportPerEngineTDR</td>
</tr>
<tr>
<td>Optimizing the graphics stack to improve performance on sleep &amp; resume</td>
<td>O</td>
<td>DXGK_SEGMENTDESCRIPTOR3::Flags</td>
</tr>
<tr>
<td>Stereoscopic 3D: New infrastructure to process and present stereoscopic content</td>
<td>O</td>
<td>D3DKMDT_VIDPN_SOURCE_MODE_TYPE</td>
</tr>
<tr>
<td>DirectFlip</td>
<td>M</td>
<td>DXGK_DRIVERCAPS::SupportDirectFlip</td>
</tr>
<tr>
<td>GDI Hardware acceleration (This is a required WDDM v1.1 feature)</td>
<td>M</td>
<td>DXGK_PRESENTATIONCAPS::SupportKernelModeCommandBuffer</td>
</tr>
<tr>
<td>GPU power management of idle and active power</td>
<td>O</td>
<td>If this feature is supported the DDI functions must be supported (SetPowerComponentFState and PowerRuntimeControlRequest)</td>
</tr>
</tbody>
</table>

### Best practices

As explained in this document, to deliver the best experience on Windows 8, Windows takes advantage of the graphics hardware paired with a Windows 8–optimized WDDM v1.2 driver. This section summarizes the best practices.

**System manufacturers:**

- Ensure the following cases are fully tested and work well with your system configurations:
  - Compatible with Microsoft Basic Display Driver
• Reboot-less updates on servers

✓ Design new servers with WDDM hardware and adopt the relevant WDDM driver type that best suits your customer’s needs.

✓ Work with graphics hardware vendors to get certified WDDM v1.2 drivers for validation.

✓ For headless systems:

• System firmware should set the VGA Not Present flag in the IAPC_BOOT_ARCH field of FADT and if there is any VBIOS, it should implement an empty mode list through the VESA BIOS Extensions (VBE).

• In the absence of VBE support, the headless system should not represent a working display through UEFI GOP.

✓ Refer to the Windows 8 logo requirements kit for validation and testing information.

✓ Test a variety of hardware configurations on both desktops and mobile systems to ensure a solid end-user experience in Windows 8.

Graphics hardware vendors:

✓ Work with Microsoft to develop Windows 8 WDDM v1.2 drivers.

✓ Test prerelease Windows 8 WDDM v1.2 drivers on the prerelease versions of Windows 8.

✓ Provide updated WDDM v1.x drivers to Microsoft for deployment through Windows Update.

✓ In addition to the Windows certification test suite, validate graphics and gaming performance, application compatibility, and various self-host scenarios on each ASIC family.

✓ Test WDDM v1.0 and v1.1 drivers on prerelease versions of Windows 8.

✓ Make the full retail package for WDDM v1.2 drivers available as early as possible.

Independent software vendors (ISVs):

✓ Test existing and upcoming DirectX games with WDDM v1.2 drivers on prerelease versions of Windows 8.

✓ Test individual applications on prerelease versions of Windows 8.

✓ Take advantage of the Windows 8 DirectX feature improvements.

Resources

New Reference Topics for Windows Developer Preview

DirectX Developer Center

Timeout detection and recovery of GPUs through WDDM
Glossary

**Direct2D**
Microsoft’s 2D API that uses the rendering power of the GPU through DirectX graphics.

**DWM**
Desktop Window Manager. The Windows component that presents the contents of the desktop.

**DxgKmL.sys**
The Windows kernel portion of the WDDM graphics stack. It resides in the kernel and provides a variety of services, including communications with the kernel-mode WDDM video driver.

**DXVA**
DirectX Video Acceleration.

**Direct3D 10 Level 9 (also called Feature Level 9)**
The Microsoft Direct3D 10 driver that uses the DirectX 10 API on DirectX 9 hardware.

**headless system**
The term “headless” is utilized in this document to refer to a server system configuration that operates without the presence of locally attached keyboard, mouse, and video graphics devices. These systems can host applications that one or more remote client machines can access, so it really does not require user interaction at the local console.

**HLSL**
High Level Shading Language for DirectX. Using HLSL, C-like programmable shaders can be created for the Direct3D pipeline.

**TBDR**
Tile-Based Deferred Rendering.

**WDDM**
The Windows Display Driver Model (WDDM) was introduced beginning with Windows Vista as a replacement to the Windows XP Display Driver Model (XDDM).

**WDDM v1**
This is the first version of the Windows Display Driver Model. WDDM v1 drivers from various graphics hardware vendors shipped with Windows Vista.

**WDDM v1.1**
This is the optimized WDDM v1.1 driver version that shipped for Windows 7. It contains several enhancements to the first version of WDDM. WDDM v1.1 drivers support additional features on Windows 7 that WDDM v1 drivers do not support.

**WDDM v1.2**
WDDM driver version optimized for Windows 8. The WDDM driver model that enables Windows 8 features is referred to as WDDM v1.2. WDDM v1.2 is a superset of WDDM 1.1, and WDDM 1.0.
**WHCK**
Windows Hardware Certification Kit, previously known as Windows Logo Kit (WLK).

**XDDM**
The Windows 2000/XP display driver model.