Windows 8 and Windows Server 2012 compatibility cookbook

A developer's guide to Windows compatibility, reliability, and performance

September 11, 2012

Abstract

In the *Cookbook* we provide info about changes to and new features of the Windows® 8 and Windows Server® 2012 operating systems. We also provide guidelines for you to verify the compatibility of your existing and planned apps with the new operating systems. We assume that you are familiar with previous versions of Windows.

The *Cookbook* is for third party developers of apps designed to be used in the Microsoft Windows environment and is available for viewing at http://msdn.microsoft.com/library/hh848074(v=vs.85).aspx and for download at http://www.microsoft.com/downloads/details.aspx?FamilyID=2D6A4111-9F14-4DB8-A4C2-BE8C8C1414AD&displaylang=e&displaylang=en.

The content applies to:

- Windows 8
- Windows Server 2012
- Windows Server 2008 R2
- Windows 7
- Windows Server 2008
- Windows Vista®
- Windows Server 2003

Contents

Introduction

Windows 8 and Windows Server 2012 introduce the latest operating system technology and software development platforms for use by app developers and enterprises worldwide. As part of further enhancing the security, reliability, performance, and user experience of Windows, Microsoft has introduced many new features, improved existing features, and removed others.

While the goal of Windows 8 and Windows Server 2012 is to stay highly compatible with most of their respective apps written for previously released operating systems, some compatibility breaks are inevitable due to innovations, tightened security, and increased reliability. Overall, the compatibility of Windows 8 and Windows Server 2012 with existing apps is high.

This document builds on the concepts embodied in the *Windows Vista and Windows Server 2008 Application Compatibility Cookbook* (http://msdn.microsoft.com/library/bb757005.aspx) and the *Windows 7 and Windows Server 2008 R2 Application Quality Cookbook* (http://msdn.microsoft.com/library/dd371778(v=VS.85).aspx). Like them, this document shows you how to verify that your apps are compatible with the new operating system and provides an overview of the few known app incompatibility issues in Windows 8 and Windows Server 2012.

In addition, Microsoft is investing in several new and enhanced features to help you build higher quality apps and to troubleshoot when apps do not function properly on Windows 8 and Windows Server 2012.

This *Cookbook* contains more than three dozen topics divided into three major sections:

- Client and server compatibility
- New features and enhancements
- Tools and best practices

Four of these topics are new with this release of the *Cookbook*:

- RDS server components are removed from Windows 8
- East Asian type guidance
- Software input panel versus software keyboard
- Desktop gadgets removed

One topic has been updated:

- Switch text input changed from per-thread to per-user

We invite you to check out these topics to learn how to optimize your apps and take advantage of what this newest release of Windows has to offer.

Client and server compatibility

This section describes changes in the operating system that you should pay special attention to due to the potential impacts on existing apps and how new apps should be designed on clients, on servers, or on both clients and servers. Following is the list of topics covered in this section, grouped by feature area:

AppCompat

- Operating system versioning
- Security app detection rules update
- Determining shim status
- Server apps must be able to run without the Server Graphical Shell
- RDS server components are removed from Windows 8
- File type and protocol associations model
- Desktop Activity Moderator
- East Asian type guidance
- Software input panel versus software keyboard
- Switch text input changed from per-thread to per-user
- .NET Framework 4.5 is default and .NET Framework 3.5 is optional
- Desktop apps may not be visible after launching the default web browser or Windows 8 apps
- High-contrast mode
- App (executable) manifest
- Queued present model is being deprecated
- Program Compatibility Assistant scenarios for Windows 8
- Desktop gadgets removed

Storage and file system

- Advanced format (4K) disk compatibility update
- Thin provisioning of logical units
- Enhanced storage is now optional for WINPE and server SKU
- VDS is transitioning to WMIv2-based Windows Storage Management API
- Previous versions UI removed for local volumes
- StorAHCI replaces MSAHCI
- Windows 7 Backup and Restore deprecated
- Offloaded data transfers

Other

- Desktop Window Manager is always on
- Direct2D does not support rendering to “rich” metafiles in Internet Explorer 9
- Changes in DX9 legacy hardware support
- MSAA is not available to Windows Store apps
- Port 3 is deprecated for NDIS 6.30 drivers

Operating system versioning

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

The internal version number for Windows 8 and Windows Server 2012 is 6.2. All of the versioning APIs will return this version number (GetVersion, GetVersionEx).

Manifestation

The manifestation of this change is app-specific. This means that any app that specifically checks for the operating system version will get a higher version number, which can lead to one or more of these situations:

- App installers might be unable to install the app, and apps might be unable to start
- Apps might become unstable or crash
- Apps might generate error messages, but continue to function properly

Mitigation

Use these VersionLie shims for client apps that rely on a previous version:

- Win7RTMVersionLie (can also be used on server)
- VistaSP2VersionLie
- WinXPSP3VersionLie

To check for the version without blocking the user from installing or running the app, do a "greater than or equal to check." You can use this with the VerifyVersionInfo Win32 API (http://msdn.microsoft.com/library/ms725492(VS.85).aspx).

Solution

Generally, apps should not perform operating system version checks. If an app needs a specific feature, it is preferable to try to find the feature, and fail only if the needed feature is missing. At a minimum, apps should always accept version numbers greater than or equal to the lowest supported version of the operating system. Exceptions should occur only if there is a specific legal, business, or system-component requirement.

Resources

Application Compatibility Toolkit Download
http://go.microsoft.com/fwlink/?LinkID=205020

Known Compatibility Fixes, Compatibility Modes, and AppHelp Messages
http://go.microsoft.com/fwlink/?LinkID=205039

Security app detection rules update

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Windows Store apps being added to Windows 8 are all being installed to a common location under “Program Files” (%programfiles%) called \program files\WindowsApps.

Manifestation

This can cause collisions with existing configurations, and some antivirus/anti-malware detectors see this location as a potential location that is a cause for concern.

Mitigation

The current recommendations are:

- Move away from \program files\WindowsApps for storing any custom apps
- Antivirus/antimalware vendors should update their heuristics so they do not identify that location as a malware location

Determining shim status

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Windows 8 logs events when apps are being shimmed for various reasons. The easiest way to determine if your app is being shimmed is to check the event viewer. Go to Applications and Services Logs\Microsoft Windows Application-Experience Program\Telemetry.

Mitigation

The best way to get yourself out of shimming is to rename the executable or to increase the major version by 1 (recompile the executable).

Server apps must be able to run without the Server Graphical Shell

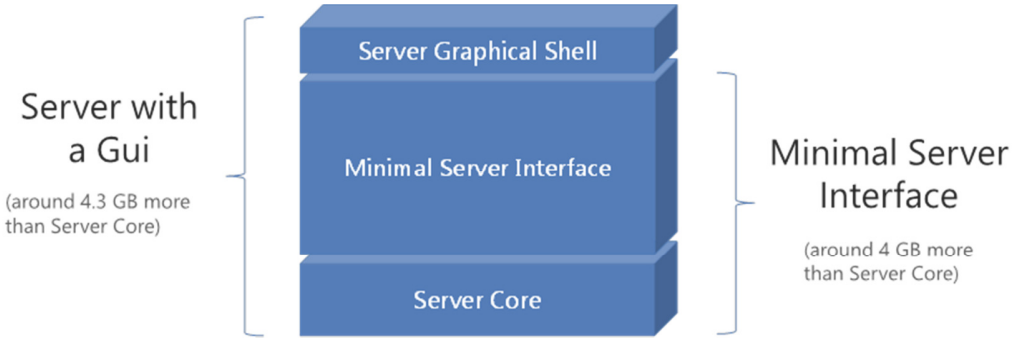
Platform

Servers – Windows Server 2012

Description

Server Graphical Shell, the feature that includes Windows Explorer and Internet Explorer, is installed by default on "Server with a GUI" installations of Windows Server 2012. The Server Graphical Shell feature can be uninstalled in order to reduce the potential servicing and performance footprint, thereby limiting the number of reboots that the server may incur, while still allowing management tools to be run locally on the server.

After an administrator uninstalls the Server Graphical Shell, the server is in the Minimal Server Interface configuration:



Administrators can then opt to run in the Minimal Server Interface configuration (which includes a set of local management tools), as the default, rather than in the Server with a GUI configuration. This allows local monitoring and management, while reducing both resource usage and frequency of servicing.

Administrators can reinstall Server Graphical Shell later if they need the functionality within it. (Administrators can also start from a Server Core install, and "build up" to the Minimal Server Interface configuration using the Features On Demand functionality.)

Server apps must be capable of running in the Minimal Server Interface configuration to take advantage of the reduced resource utilization and servicing footprint. This capability can be achieved by allowing the administrator to choose not to install parts of the app that needs the Server Graphical Shell, or by detecting the presence of Server Graphical Shell and disabling some aspects of the app.

The Minimal Server Interface has a reduced resource and servicing footprint because many APIs and binaries that are included in the Server Graphical Shell are not available in this configuration.

Where appropriate, server apps should also allow remote administration (preferably via Windows PowerShell Remoting) from another Windows Server or Windows Client installation. This enables better centralized administration of one or more machines

in the Minimal Server Interface configuration, or of machines in an even lower footprint configuration such as Server Core.

Manifestation

If an app requires any of the APIs or binaries that are not available in the Minimal Server Interface configuration, it may not display properly on the screen and/or be unusable.

Mitigation

Server app developers should identify those parts of their apps that require any of the removed APIs or binaries and include info for the server administrator identifying the parts of the app that won't run properly when using the Minimal Server Interface. If those parts of the app can be optionally installed or are not absolutely required for product functionality, the app can still be installed and run under the Minimal Server Interface configuration.

If the app cannot be used at all without the Server Graphical Shell, this limitation should be documented and the server administrator should be instructed to install the Server Graphical Shell. (If adding to a Server Core installation, this might require adding features using Features On Demand.) In addition, the app should check on startup that all required files are available, as the Server Graphical Shell can be uninstalled at any time before or after the installation of the app.

Solution

Rely on the smallest possible set of dependencies, and modularize apps so that the core app functionality can work without requiring more heavyweight user interface components installed. Develop apps that do not require any of the removed APIs or binaries, and rely instead on the functionality contained within the Minimal Server Interface or Server Core. This will reduce maintenance requirements while improving performance and user satisfaction.

Where there are parts of an app that might add significant functionality when the Server Graphical Shell is available, app developers can:

- Allow these additional features making use of Server Graphical Shell to be optionally installed, so they can be omitted from installations on a Minimal Server Interface configuration
- Detect the presence of Server Graphical Shell and adapt the app behavior

App developers should also ensure that server apps are remotely manageable where possible and appropriate.

Detecting Minimal Server Interface and Server Core

Windows Server will install a corresponding registry value for each server level installed. You can query for the existence of these keys to determine if the Server Graphical Shell or Minimal Server Interface features are installed and enabled.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Server\ServerLevels:

	Server Core	Minimal Server Interface	Server Graphical Shell
ServerCore=1	X	X	X
Server-Gui-Mgmt=1		X	X
Server-Gui-Shell=1			X

An “X” in the table above means that registry key will be present when the corresponding feature is installed.

Note that these server levels are additive; if the Server Graphical Shell is installed, so is the Minimal Server Interface and Server Core. In that case, *both* registry keys will be present.

Tests

Inspect your app code for requirements that use any of the removed APIs and binaries. After you have removed any instances of these from “core application” binaries, test your app in an environment that does not include the Server Graphical Shell. Tools such as the Process Monitor might help with this.

If you cannot completely discontinue use of these APIs and binaries, ensure that your app fails gracefully when run on Minimal Server Interface or Server Core.

Resources

Existing Server Core docs on MSDN
http://msdn.microsoft.com/library/ms723891(v=vs.85).aspx

Remote data service server components are removed from Windows 8

Platforms

Clients - Windows 8

Servers - Windows Server 2012

Description

The remote data service (RDS) server is not available in Windows 8:

- File msadcf.dll, which hosts the default "business object" RDSServer.DataFactory, is removed, and its associated files (msadcfr.dll, msadcfr.dll.mui, handler.reg and handsafe.reg) are removed
- File msdfmap.dll, which is the default Handler, is removed, and the file msdfmap.ini is removed
- File msadcs.dll, the ISAPI, is removed

Manifestation

Customers cannot host an RDS server on Windows 8.

Mitigation

Customers can keep their RDS server on a machine with Windows 7 or other down-level operating systems.

Solution

Customers can keep their RDS server on a machine with Windows 7 or other down level operating systems.

Resources

Data Access Technologies Roadmap
http://msdn.microsoft.com/en-us/library/ms810810.aspx

File type and protocol associations model

Platforms

Clients - Windows 8

Servers - Windows Server 2012

Description

The file type and protocol association model has changed in Windows 8. Apps are no longer able to programmatically set themselves as the default handler for a file type or protocol. Instead, now the user always controls what the default handler is for a file type or protocol.

Manifestation

How this change presents to the user depends upon how the app is designed, for example:

- Many apps check to see if they are the default every time they run and, if they are not, they prompt the user to set them as default. However, because apps can no longer accurately query to determine which app is the default handler for a file type or protocol, neither of these operations works.
- Many apps have a dialog box or menu built in or in their installer that specifies the file types for which the app should serve as the default. However, because apps can no longer programmatically set themselves as the default handler for a file type or protocol, this no longer works.

Mitigation

There are several things users can do to accommodate these changes:

- Users are prompted contextually to choose a default app to handle file types, protocols, or both when one has not been specified
- Users are offered the option to change their default handler after installing new apps that can handle a file type or protocol
- The default programs control panel allows users to set defaults for an app, or for a file type, protocol, or both; apps can link to the control panel
- Defaults can be changed from Windows Explorer

Solution

As a result of these changes, this API guidance is provided:

- The functionality of some method calls within the IApplicationAssociationRegistration API has changed, and ***should no longer*** be used.
    - ***Do not*** call QueryAppIsDefault/QueryAppIsDefaultAll to determine if an app is the default
    - ***Do not*** call QueryCurrentDefault to determine which app (if any) is the current default

  - ***Do not*** call SetAppIsDefault/SetAppIsDefaultAll to set the default app

- The guidance going forward is:
  - ***Do not*** query to see which app is the default handler for file types or protocols
  - ***Do not*** attempt to monitor changes in the default handler for file types or protocols
  - ***Do not*** attempt to set an app as the default handler for file types or protocols
  - ***Do not*** attempt to manage defaults for file types or protocols from within an app
  - ***Do*** integrate with the Set Default Programs control panel if you want to allow users of your app to access the default management UI (the management UI within the app is no longer supported)
    - Calling IApplicationAssociationRegistrationUI::LaunchAdvancedAssociationUI allows the user to access the 'Set Default Programs' control panel page for the specified app

Tests

- Test to verify that apps register properly in Set Default Programs control panel
- Test to verify that apps register properly to appear in the OpenWith list for the file types, protocols, or both, that they register to handle
- Test to verify that new app notifications appear after your app is installed and the user invokes a file type, protocol, or both, that your app has registered to handle

Resources

Best Practices for File Type and Protocol Associations in Windows Developer Preview Desktop Apps
http://go.microsoft.com/fwlink/?LinkId=228165

File Type Associations and AutoPlay Build Conference Presentation
http://channel9.msdn.com/events/BUILD/BUILD2011/PLAT-282T

Desktop Activity Moderator

Platform

Clients – Windows 8

NOTE: The DAM is present only on Windows 8 client machines that support connected standby. The DAM is not present on server SKUs.

NOTE: Windows Store apps are not impacted by the DAM.

Description

Our customers are shifting towards lighter, smaller, more mobile platforms to satisfy their computing needs. As part of the shift to mobile devices, users have become increasingly concerned about battery life of their devices. The Desktop Activity Moderator (DAM) is one of several new features in Windows 8 designed to ensure consistent, long battery life for devices that support connected-standby.

Connected standby occurs when the device is powered on, but the screen is turned off. In this power state, the system is technically always "on" (to support key scenarios like mail, VoIP, social networking, and instant messaging with Windows Store apps). It is analogous to the state a smart phone is in when the user presses the power button.

As such, software (including apps and operating system software) must be well-behaved during connected standby. The DAM was created to suppress desktop app execution in a manner similar to the Sleep state (S3 on ACPI devices). It does this by suspending or throttling desktop software processes across the system upon connected standby entry. This enables systems that support connected standby to deliver minimized resource usage and long, consistent battery life while enabling Windows Store apps to deliver the connected experiences they promise.

Details

The DAM is a kernel -mode driver that is loaded and initialized at system boot if the system supports connected standby. (This is determined by evaluating if the AOAC field in the `SYSTEM_POWER_CAPABILITIES` structure returned by `CallNtPowerInformation` is set to `TRUE`).

When the DAM is enabled and your desktop process is created, the DAM adds your process to system-managed job objects:

- If the process was created in session 0, DAM adds the process to a job object subject to **throttling**.
- If the process was created in an interactive session (session 1 or higher), DAM adds the process to a job object subject to **suspension**.

Note: For Windows 8, job objects can be nested. This means that the DAM's usage of job objects does not interfere with an app's existing usage of job objects.

When the screen is on, the DAM is disengaged and does not impact any processes on the system. When the system is in connected standby, depending on activity on the system, DAM might throttle or suspend processes.

- Processes that are subject to suspension have all their threads suspended (not allowed to run under any circumstances); app state (process memory) is maintained
- Processes that are subject to throttling cycle between suspended and unsuspended (a large majority of time is spent in the suspended state)
    - Be aware that Windows might also detect that critical activities are occurring and might unsuspend throttled services for longer periods of time during this activity.
    - Also, note that while in connected standby, sensors and networks might not be available, so throttled processes should be designed to be resilient to poor network conditions (for most processes, this doesn't require any change).

When DAM suspension is engaged or disengaged, DAM triggers delivery of a WM_POWERBROADCAST message to those processes subject to suspension that have opted-in to message delivery (via API call or compatibility shim, described later). After a few seconds delay, DAM suspends the process.

There are no notifications when DAM throttling is engaged or disengaged. Processes should not need modification; processes continue to function, albeit at a slower rate.

Manifestation

Processes are often suspended or throttled during the connected standby state. To the majority of suspended apps, this should be very similar to an S3 suspend / resume or S4 hibernate / resume transition. Manifestations might include but are not limited to inconsistencies in uptime / runtime vs. wall clock time, inconsistencies in timer behavior, or dramatic changes in operating system state before and after the suspend completes.

Suspension and throttling happens as a unit (all suspendable processes are suspended and unsuspended in unison, and all processes that can be throttled are throttled and unthrottled in unison), so communication between two suspended processes or two throttled processes does not pose a problem.

Software that relies on cross-process communication might need special consideration:

- **Communication between processes in session 0 (throttled) and session 1+ (suspended)** – examples include tray icons or UI components that represent current service state
- **Communication between components in user mode (session 0 or 1) and drivers (which are neither throttled nor suspended)** – examples include services that do work on behalf of a driver

In these cases, if cross-process communication is not handled correctly, apps can appear hung or unresponsive (though the user might not see this impact directly, as

the screen will be off when in connected standby). In most cases, however, services and drivers should already be developed to be robust against cross-process communication issues.

Vendors who create software for, or dependent on, the web should consider how process suspension affects connection lifetimes and handshakes. Additionally, because network connectivity might not be available when in Connected Standby mode, developers of processes created in session 0 should be especially aware of how intermittent network connections affect the process.

Solution

Windows Store apps are not impacted by the DAM. If your desktop app is impacted by the DAM, you can request notifications before suspension is engaged (for example, to save state or close network connections) using one of these methods:

- If your app has a window (HWND) and you want to handle these notifications through your window procedure, call RegisterSuspendResumeNotification to register for these messages (or UnregisterSuspendResumeNotification to unregister). You can use DEVICE_NOTIFY_WINDOW_HANDLE in the Flags parameter, and pass your window's HWND in as the Recipient parameter. The message received is the WM_POWERBROADCAST message.
- If your app does not have a window (HWND) or you want a direct callback, call PowerRegisterSuspendResumeNotification to register for these messages (or PowerUnregisterSuspendResumeNotification to unregister). You must use DEVICE_NOTIFY_CALLBACK in the Flags parameter and pass a value of type PDEVICE_NOTIFY_SUBSCRIBE_PARAMETERS in the Recipient parameter.
- If your app cannot be recompiled, you can opt in to receive these WM_POWERBROADCAST messages via the AppCompat toolkit (using the PromoteDAM shim).

The suspend message is `WM_POWERBROADCAST` with `wParam=`**`PBT_APMSUSPEND`**; this message is broadcast concurrently to all opted in processes on the system. Your app must perform any work on the suspend path quickly and efficiently. The timeout after the broadcast notification is global, not per process, so your process might be contending for system resources if the work required in this path is extensive.

The resume message is `WM_POWERBROADCAST` with `wParam=`**`PBT_APMRESUME`**; this message is broadcast concurrently to all opted in processes after a resume. Relative time of delivery to the system exit from connected standby is not guaranteed.

Tests

Test your software across connected standby transitions.

Resources

Mobile Battery Life Solutions for Windows 7
http://msdn.microsoft.com/en-us/windows/hardware/gg487547

SYSTEM_POWER_CAPABILITIES
http://msdn.microsoft.com/en-us/library/windows/desktop/aa373215(v=vs.85).aspx

CallNtPowerInformation function
http:/msdn.microsoft.com/library/windows/desktop/aa372675(v=vs.85).aspx

Job Objects
http://msdn.microsoft.com/en-us/library/windows/desktop/ms684161(v=vs.85).aspx

RegisterSuspendResumeNotification
http://go.microsoft.com/fwlink/?LinkId=239911

UnregisterSuspendResumeNotification
http://go.microsoft.com/fwlink/?LinkID=239912

PowerRegisterSuspendResumeNotification
http://go.microsoft.com/fwlink/?LinkId=239913

PowerUnregisterSuspendResumeNotification
http://go.microsoft.com/fwlink/?LinkID=239914

AppCompat toolkit
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=7352

East Asian type guidance

Platforms

Client – Windows 8

Servers – Windows Server 2012

Description

In Windows 8, ClearType font is enabled for most of the UI consistently, in Japanese, Chinese Simplified, Chinese Traditional, and Korean, in both Classic and Immersive styles.

Font Family

Font names will not be translated:

Japanese	Korean	Chinese Simplified	Chinese Traditional
Meiryo UI	Malgun Gothic	Microsoft YaHei UI	Microsoft JhengHei UI

Font metrics (9 pt.)

In Windows 8, Chinese Simplified and Chinese Traditional UI fonts conform more closely to Segoe UI metrics in order to provide the same UI look and feel and a consistent user experience with the English UI.

9 Point	Microsoft Yahei UI	Microsoft JhenHei UI	Malgun Tothic	Meiryo UI	Segoe UI
tmAscent	13	12	12	12	12
tmDescent	4	4	3	3	3
tmHeight	17	16	15	15	15
tmInternalLeading	5	4	3	3	3
Height (pixel value for 100DU)	213	200	188	188	188

Registry setting

Windows 8 has updated the FontLink chain for East Asian languages and defined at: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontLink\SystemLink to support the new fonts for Chinese Simplified and Chinese Traditional:

- Japanese SKU: Segoe UI links to Meiryo UI
- Korean SKU: Segoe UI links to Malgun Gothic
- Chinese Simplified SKU: Segoe UI links to Microsoft YaHei UI
- Chinese Traditional SKU (including HongKong SKU): Segoe UI links to Microsoft JhengHei UI

Also, all UI fonts link to each other.

Font weight mapping between Segoe UI and East Asian (EA) ClearType fonts:

Segoe UI	EA ClearType Fonts
Light	Regular
Semi-light	Regular
Regular	Regular
Semi-bold	Bold
Bold	Bold

Immersive reading

The UI font is also the default font for immersive reading scenarios such as for eBooks. Immersive reading assumes an extended reading time and implies the possibility of breaking up a long text into chapters.

eReading apps may allow the reader to change the font. We also expect more eBook content to be formatted in a designer-specified, often embedded font.

Interactive text

The UI font is used in applications such as mail or chat where the user is composing the text. These fonts belong to the sans serif font family designed for reading long texts onscreen. It is the default font in Microsoft Outlook, Word, and PowerPoint.

East Asian language fonts for reading and interactive in Modern CSS implementation:

Default

@authoringFontFamily: Calibri, Verdana, Arial, Helvetica, sans-serif;

@readingFontFamily: Cambria, Times New Roman, Times, serif;

@fontSizeAuthoring: 14.66667px;

@fontSizeReading: 14.66667px;

JA

@authoringFontFamily: Meiryo, Calibri, Verdana, Arial, Helvetica, sans-serif;

@readingFontFamily: MS Gothic, Cambria, Times New Roman, Times, serif;

@fontSizeAuthoring: 15.33px;

@fontSizeReading: 14.66667px;

KO

@authoringFontFamily: Malgun Gothic, Calibri, Verdana, Arial, Helvetica, sans-serif;

@readingFontFamily: Batang, Cambria, Times New Roman, Times, serif;

@fontSizeAuthoring: 15.66px;

@fontSizeReading: 14.66667px;

zh-Hans

@authoringFontFamily: Microsoft YaHei, Calibri, Verdana, Arial, Helvetica, sans-serif;

@readingFontFamily: SimSun, Cambria, Times New Roman, Times, serif;

@fontSizeAuthoring: 14.66667px;

@fontSizeReading: 15.5px;

zh-Hant

@authoringFontFamily: Microsoft JhengHei, Calibri, Verdana, Arial, Helvetica, sans-serif;

@readingFontFamily: MingLiu, Cambria, Times New Roman, Times, serif;

@fontSizeAuthoring: 14.66667px;

@fontSizeReading: 14.66667px;

Manifestation

This font change in Windows 8 may impact new and existing applications that call Common Controls and Common File Dialogs without specifying a font.

For example, if there is a CPropertySheet dialog in an MFC application that has no specific font set, then the property sheet will use the new ClearType font. On the other hand, the change doesn't impact WinForms Controls.

Tests

Test your application to see if any modifications such as specifying the font or resizing controls are needed.

Software input panel versus software keyboard

Platform

Clients - Windows 8

Description

In previous versions of Windows, Software Keyboard is used to input characters without a physical keyboard. Software Input Panel (SIP) is designed to improve the input experience on Windows 8 touch devices. The East Asian SIP sends the character codes instead of sending v-key events, which is different from how the Software Keyboard behaves. If the application or the website handles the key events only, it may not work as expected.

Manifestation

If the application is completely based on v_Key events, users will not be able to perform the operation on the application. There are some sites where v_Key events are used in their JavaScript code to get the input text before users send them. For instance, when users input the postal code, the site can display the address. In cases where the site cannot show the address candidate, users might be able to input the address themselves.

Mitigation

Users can use the standard layout instead of optimized layout. The other option is to use English SIP.

Solution

Developers can change the application to handle character codes instead of v_Key events. Websites that use the JavaScript event handlers onKeyUp() and onKeyDown() can use oninput() instead. The alternative option is to disable TSF.

Switch text input changed from per-thread to per-user

Platforms

Clients - Windows 8
Servers - Windows Server 2012

Description

Previously, the input language keyboard layout and IME switch, including IME modes, were thread-specific. For instance, if users opened a Notepad file and changed the input language from English to Japanese, and then opened a WordPad file and changed the input language from English to German, the Notepad file would retain Japanese as its input language and the WordPad file would retain German as its input language. In Windows 8 and Windows Server 2012, we are changing from this per-thread approach to a per-user approach. This means that when the input language or IME mode is changed in one app, it applies on other apps when the focus is on the app. In the example above, where users changed the Notepad file input language to Japanese and then opened a WordPad file, the WordPad input language would be Japanese. If users then changed the WordPad input language to German, when they return to the Notepad file, its input language would be in German instead of Japanese. Another example would be if users change the Korean IME mode to AlphaNumeric mode in a Notepad file, and then opens another Notepad file, it too will open in AlphaNumeric mode.

Because the input switch will now be per user instead of per thread by default, some Desktop apps may behave differently. The session starts with the default method and then the current settings follow the focus. The per-user input switch will use the current input method for the newly opened app.

These APIs used in Desktop apps are affected:

ImmSetConversionStatus
ActivateKeyboardLayout

The language or IME mode change in these APIs will follow the focus and apply across threads instead of only to the current thread. The per-user input switch is active when the focus is on the foreground. If the API is called after the focus is on the app, the change made by the API will apply to other apps later on when the focus moves to them. If the API is called before the focus is on the app, the language or IME mode change made by the API will be replaced by the per-user input switch with the current input language or IME mode to which the user last switched.

For instance, a user is using English while typing in App A. The user then starts App B, which calls the API ActivateKeyboardLayout to change the input language to German. If the API is called after the focus is on the App B, then the input language is German, not only in App B, but also in App A when the user returns to it. If the API is called before the focus is on App B, then the input language of App B will be English instead of German.

In addition to calling ImmSetConversionStatus, the IME mode could be changed by following the input scope set by the application. Some apps, including Windows Store

apps, may set the input scope on certain text fields, such as the Password field, URL field or email field. Certain IMEs react to the Input Scope and change the IME mode according to the input scope.

Japanese Microsoft IME behavior

Japanese IME keeps conversion modes (Hiragana, Halfwidth-AlphaNumeric, Fullwidth-Katakana, Halfwidth-Katakana, and Fullwidth-AlphaNumeric) per application meaning that each application retains conversion mode. This is because conversion mode is attached to application's particular input field. Microsoft IME will be handled as Hiragana mode after switching the application.

Thus, Japanese Microsoft IME ignores the change of conversion modes made by calling ImmSetConversionStatus when used in per user mode. This API is used in the IME mode property of .NET framework. InputScope is recommended in order to set the IME mode under per user mode in Windows 8.

Manifestation

Files opened in other apps after the input language is changed in the first app, will have the same input language as the initial app.

Mitigation

To reset the input switch from per user to per thread:

1. Go to the Control Panel.
2. Select **Clock** | **Language and Region** | **Language profile**.
3. Click **Advanced settings**.
4. In the **Switching input method** section, check the "Let me set a different input method in each app window" box.
5. Click **Save**.

Solution

Keep the per-user mode in mind when developing apps. With the per-user input switch, we do not recommend that the app reset the language or IME mode without the users' knowledge. If the app needs to change the IME mode, we prefer that it sets the input scope rather than call ImmSetConversionStatus. If an app must rely on a per-thread input switch in order to function, provide a message to the user instructing them to go to the Control Panel and make the change described above.

Resources

Get-WinLanguageBarOption
http://technet.microsoft.com/ library/hh852146

InputScopeNameValue Enumeration
http://msdn.microsoft.com/library/system.windows.input.inputscopenamevalue(v=vs.110).aspx

.NET Framework 4.5 is default and .NET Framework 3.5 is optional

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

.NET Framework 4.5 is enabled by default in Windows 8. Windows 8 does not include .NET 3.5 by default, but the files for .NET 3.5 are available on the Windows 8 installation media as an optional feature.

If the user is upgrading from Windows 7 to Windows 8, .NET Framework 3.5 is fully enabled to ensure that any apps on the computer continue to work correctly.

Manifestation

If the user performs a clean installation of Windows 8, and then installs apps that require .NET Framework 3.5 (or 2.0), they will trigger a request for the necessary .NET 3.5 files. Normally the missing files will be downloaded from Windows Update (after asking the user for permission), but if access to Windows Update is not possible, enabling .NET Framework 3.5 will fail unless an alternate source for the missing files has been specified.

Mitigation

To enable .NET Framework 3.5 on only test machines with clean installs of Windows 8:

1. Copy `\sources\sxs\` from the mounted operating system build ISO image to dotnet35 or similar folder. For example:

```
xcopy e:\sources\sxs\*.* c:\dotnet35 /s
```

2. Execute this command line using admin privileges:

```
Dism.exe /online /enable-feature /featurename:NetFX3 /All
/Source:c:\dotnet35 /LimitAccess
```

Note that the **sources\SxS** folder must not be used as a redistribution mechanism since this is not a supported mechanism.

Solution

For consumers:

Windows 8 includes a mechanism that automatically enables .NET Framework 3.5 when attempting to install the redistributable package or when an application installer that needs .NET 3.5 invokes the redistributable.

For App developers (and IT Administrators):

IT administrators can configure .NET 3.5 apps to run on either .NET 3.5 or .NET 4.5 (depending on what's already installed)**.** In order to run a managed app on either 3.5 or 4.5, just add a section in the application configuration file. This will ensure that if

.NET 3.5 is installed, app will run on .NET 3.5; otherwise the app will run on .NET 4.5. An example of the additional section in the configuration file is provided below:

```
<configuration>
   <startup>
      <supportedRuntime version="v2.0.50727"/>
      <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.5"/>
   </startup>
</configuration>
```

For enterprises or OEMs:

To enable .NET Framework 3.5 for EEAP builds and for applications that do not have access to Windows Update:

1. Copy \sources\sxs\ from the mounted OS build ISO image to the dotnet35 or similar folder. For example:

```
xcopy e:\sources\sxs\*.* c:\dotnet35 /s
```

2. Set the regkey:

```
[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Servicing]
“LocalSourcePath”=”c:\dotnet35”
```

For enterprises:

For machines that are configured to use WSUS for servicing, you can set a registry entry to allow the machine to use Windows Update for enabling .NET 3.5 instead of WSUS (servicing will still be done from WSUS if you do this).

1. Set the regkey:

```
[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Servicing]
“RepairContentServerSource”=DWORD(2)
```

This registry entry can also be set via Group Policy to bypass the WSUS server when enabling .NET 3.5. The policy is under Local Computer Policy -> Computer Configuration -> Administrative Templates -> System. Select the setting “Specify settings for optional component installation and component repair”.

If you select “Contact Windows Update directly to download repair content instead of Windows Server Update Services (WSUS)”, any attempts to add Windows features (for example, .NET Framework 3.5) or repair features will trigger file downloads from Windows Update. Target computers require Internet and WU access for this option. Normal servicing operations continue to use WSUS if it has been configured as a source.

A note regarding setting local source location via registry entries

IT administrators can set local source location(s) for .NET 3.5 files via a registry entry, so that users can use the Add/Remove Windows Features dialog to enable features with missing payload without having to specify a source location. The value of the registry entry can be controlled via group policy.

This registry entry is supported:

Entry	Type	Description
LocalSourcePath	REG_EXPAND_SZ	Local source path(s) to be used by default. Multiple paths can be specified; they should be separated by ";". Locations will be searched in the order they are specified. Local source location(s) that are specified on the DISM command line, take precedence over locations specified in this registry entry. Folder locations can be specified in this registry entry. WIMs can be used, but the path must be to the WIM file; there is no need to mount it, for example: wim:\\machine\share\file.wim:1 Notice the "1" at the end. You must specify the numerical index of the image you want to use in the WIM file. For a mounted WIM, the source path needs to refer to the windows directory of the mounted image, rather than to the mount point (for example: /source:<mount_point>\windows rather than /source:<mount_point>).

Resources

Implementing Registry-based Policy

http://msdn.microsoft.com/library/aa374292(v=VS.85).aspx

Desktop apps might not be visible after launching the default web browser or Windows Store apps

Platforms

Clients –Windows 8

Servers –Windows Server 2012

Description

The Windows Store apps user experience is focused on a single app at a time, with multitasking, app switching, and notifications provided by the Windows UI. Windows Store apps are sized to fill available space on screen, with the most common view being full screen. Therefore, when another app on the system is launched, you can no longer assume that the app will open in a desktop window alongside your app. This is always true of Windows Store apps and browsers that choose to participate in the new Windows experience. In certain scenarios, you may continue to see other apps at the same time, for example, if you are on the Desktop and launch another desktop application or you have applications in a snapped state.

Manifestation

When a desktop app uses common app activation techniques such as using the ShellExecute API on a file or protocol, Windows launches the app associated with that registration, which may be a Windows Store app and / or the user's default web browser (the default web browser might choose to participate in either the desktop or the new Windows UI). Windows Store apps are launched using the full screen, hiding the desktop and the desktop app from which it was launched.

NOTE: In Windows 8, Internet Explorer 10 is configured as the user's default browser, but the user may choose to install and set another browser as the default (no change from Windows 7). In Internet Explorer 10, when a link is opened from a desktop application, Internet Explorer will open on the desktop. But this setting can be changed by users so that Internet Explorer is opened as a Windows Store app. Browser vendors have been encouraged to adopt a similar "contextual launching" experience; however, developers should plan for the fact that not all browsers will behave similarly.

Mitigation

While there is no change that developers can make to their apps to mitigate this behavior, there are things the end user can do that you should communicate to them. Consider using the info gathered by the extended ShellExecuteEx() API to populate a contextually appropriate dialog. In that dialog, indicate to the user what app will be launched and whether that app is a Windows Store app or desktop app. The CLSID can be used to distinguish Windows Store apps from desktop apps. User options:

- Users who have a wide-screen monitor can snap the Windows Store app to the edge of the screen to expose the desktop and thereby view both the Windows Store app and the desktop app at the same time.

- If Internet Explorer is configured as the user's default browser, users can change its behavior by changing the Internet Options in the Control Panel. On the Programs tab, users can change how Internet Explorer handles links. Other browsers may offer a similar setting.

High-contrast mode

Platforms

Clients - Windows 8

Servers - Windows Server 2012

Description

In previous Windows operating systems, high-contrast mode was limited to themes running under classic themes, which were not visually styled. In Windows 8 and Windows Server 2012, classic mode has been removed and replaced with visually styled high contrast themes. One of the main benefits for you of this change is the removal of a separate code path for apps running in classic mode.

Developers still need to be educated in how high-contrast mode can affect their app and how to develop an app that is truly style-agnostic. This is important because while the incorrect use or assumption of theme colors can cause apps to behave correctly under a visual style such as Aero, those same apps respond incorrectly under high contrast. For instance, in Aero, text is always black and the highlight color is a light blue. In high-contrast black, however, the highlight color is black. If you assume black text, as has been the case in many in-box apps prior to Windows 8, and use the system default for highlighting, the user will see black text on a black background. It's necessary in these situations to understand how to use themes and system metrics correctly so the app looks correct across styles.

Manifestations

- Theming is not enabled in the client area of apps that do not contain a Windows 8 <supportedOS> tag in their app manifest. Therefore, the apps must render the client area, using the code path required to render in high-contrast mode of the classic theme.
- Theming is not enabled in both the non-client and client areas of apps in high-contrast themes. It is also not enabled in apps that do not contain a Windows 8 <supportedOS> tag in their app manifest and that draw in the non-client area of a window using the DwnIsCompositionEnabled() API. The entire app renders in the high-contrast mode of the classic theme.
- Apps that add support for Windows 8 in their manifest, but do not use visual styles for rendering, that is, they hardcode colors or images in their apps, might not render correctly in high-contrast themes. Text might be difficult to read or images might not appear as they should in high contrast mode.

Mitigation

The text colors in the high-contrast themes have been authored to be compliant with the Microsoft Accessibility guidelines. We maintain a 14:1 high-contrast ratio between foreground and background. If the colors that are enabled by default are not suitable to a particular end user, the end user can easily customize them through the control panel settings for 'Window Color' in those high-contrast themes.

These UI components are customizable in high-contrast themes:

- Window background color
- Text color
- Hyperlinks color
- Disabled text
- Selected text foreground and background colors
- Active window title foreground and background colors
- Inactive window title foreground and background colors
- Button foreground and background colors

Solution

If unexpected behavior is seen in apps in high-contrast themes, one of these solutions might help:

- **Manifesting an app for Windows 8:**
  Apps that don't contain the Windows 8 <supportedOS> tag in the app manifest will have their client areas rendered without a theme. In-box apps should all contain this entry in the app manifest. Add the 4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38 GUID value for Windows 8.
- **Using visual styles with owner-drawn UIs:**
  Owner-drawn controls should follow the instructions on MSDN for correctly rendering control parts and states, including text. Developers should not rely on the text or background color specified in a device context in order to use non-UxTheme methods for rendering. In the case where there is no theme part for the control in question, use GetThemeSysColor with the appropriate metric and draw the text using standard GDI methods. If none of the UxTheme calls are appropriate, use the GetSysColor method to get the appropriate metric.
- **Selecting text color:**
  Do not use a hardcoded text color, even if it's assumed to look fine in all common scenarios. The shipping themes are created in a way that supports high visibility with associated metrics. For instance, COLOR_HIGHLIGHTTEXT is meant to be used with COLOR_HIGHLIGHT as a background and COLOR_WINDOWTEXT is meant to be used with COLOR_WINDOW as a background. If there are exceptions to these associations, work with them in the theme parts and state definitions themselves and not in code. When designing high-contrast UIs, it is crucial that the UI be agnostic to the currently applied high-contrast theme, as high-contrast users can customize their colors.
- **Responding to WM_ThemeChange Event:**
  If your app caches the colors retrieved from the theme or applies colors in a nonstandard fashion, add a message handler for WM_THEMECHANGE that recalculates the stored color values and repaints the UI.

- **Writing a high-contrast WWA app:**
  Web apps do not have access to the UxTheme APIs, but should still be written with the current system metrics as the basis for the UI. There are a few resources for WWA developers to leverage to ensure a high-contrast compliant app:
  - The W3C CSS Color specification specifies syntax for using system metrics instead of specific colors
  - Support for high-contrast media queries is being added to Internet Explorer 10
  - WWAs can leverage the IAccessibilityCapabilities::get_HighContrast() method to check the state of high contrast

  Windows Store apps don't have many of the same issues with theme parts that are present in classic apps, but you still need to ensure high-contrast compliance. By default, Internet Explorer ignores certain user-defined styles and replaces them with high-contrast compliant values. For example, **background-image**, **background**, and **color** CSS properties are ignored.

  If you don't want Internet Explorer to ignore any properties you set and you have made sure that the UI is high-contrast compliant, you can set the new M3 CSS property **–ms-high-contrast: off** on a parent element.
- **Writing a high-contrast Windows Store app:**
  Windows Store apps should use the SystemColors class for determining proper UI element coloring, keeping in mind that certain system metric colors are designed to be used in conjunction, such as SystemColors.WindowColor and SystemColors.WindowTextColor. This facilitates a superior high-contrast experience.
- **Properly detecting high contrast in previous versions of Windows:**
  Apps running on previous versions of Windows do not have access to the new high-contrast themes even if the manifest specifies compatibility with the version of Windows in question. As such, it might be necessary to insert additional code paths to handle rendering in the classic environment used in previous versions of Windows. The presence of high contrast in this case should be checked by calling the SystemParametersInfo function with the SPI_GETHIGHCONTRAST flag. This is the only supported way of checking the presence of high contrast.

Tests

While testing an app, make sure that it renders correctly in all the in-box themes provided by Windows 8: Aero, Basic, High Contrast 1, High Contrast 2, High Contrast Black, and High Contrast White. Make sure that the text is clearly visible and easy to read in the high-contrast themes.

Resources

Aero Style Classes, Parts, and States (the new basic theme and high-contrast themes also use these states)
http://msdn.microsoft.com/en-us/library/ee453680(VS.85).aspx

Parts and States common to all visual styles
http://msdn.microsoft.com/en-us/library/bb773210(v=VS.85).aspx

Using Visual Styles with Custom and Owner-Drawn Controls
http://msdn.microsoft.com/en-us/library/dd373487(v=VS.85).aspx

GetSysColor function
http://msdn.microsoft.com/en-us/library/ms724371(v=VS.85).aspx

W3C CSS Color Module Level 3
http://www.w3.org/TR/css3-color/

SystemColors Class
http://msdn.microsoft.com/en-us/library/system.windows.systemcolors.aspx

SystemParametersInfo function
http://msdn.microsoft.com/en-us/library/ms724947(v=VS.85).aspx

Microsoft Accessibility
http://www.microsoft.com/enable/

App (executable) manifest

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

The compatibility section of the app (executable) manifest introduced in Windows helps the operating system determine the versions of Windows an app was designed to target. Additionally, the app manifest enables Windows to provide the behavior that the app expects based on the version of Windows that the app targeted.

The compatibility section of the manifest allows Windows to provide new behavior to newly created software while maintaining the compatibility for existing software. This section helps Windows deliver greater compatibility in future versions of Windows as well. For example, an app declaring support for only Windows 8 in the compatibility section will continue to receive Windows 8 behavior in future versions of Windows.

Manifestation

Apps without a compatibility section in their manifest will have Windows Vista behavior by default on Windows 7 and Windows 8 and future Windows versions. Be aware that Windows XP and Windows Vista ignore this manifest section and it has no impact on them.

These Windows components provide divergent behavior based on the compatibility section:

Remote procedure call (RPC) default thread pool

- Windows 8 and Windows 7: To improve scalability and reduce thread counts, RPC switched to the NT thread pool (default pool). For Windows Vista, RPC used a private thread pool:
    - For binaries compiled for Windows 7 and later versions of Windows, the default pool is used.
    - If I_RpcMgmtEnableDedicatedThreadPool is called before any RPC API is called, the private thread pool is used (Vista behavior).
    - If I_RpcMgmtEnableDedicatedThreadPool is called after an RPC call, the default pool is used, I_RpcMgmtEnableDedicatedThreadPool returns the error 1764, and the requested operation is not supported.
- Windows Vista (default): For binaries compiled for Windows Vista and earlier versions of Windows, the private pool is used.

DirectDraw lock

- Windows 8 and Windows 7: Apps manifested for Windows 7 and later versions of the operating system cannot call Lock API in DDRAW to lock the primary desktop video buffer; doing so will result in an error, and a NULL pointer for the primary is returned.

This behavior is enforced even if Desktop Window Manager Composition is not turned on. Apps with compatibility declared for Windows 7 and later must not lock the primary video buffer to render.
- Windows Vista (default): Apps can acquire a lock on the primary video buffer, as legacy apps depend on this behavior; running the app turns off Desktop Window Manager.

DirectDraw bit-block transfer (bitblt) to primary without clipping window

- Windows 8 and Windows 7: Apps manifested for Windows 7 and later versions of Windows are prevented from performing a bitblt to the primary Desktop video buffer without a clipping window; doing so results in an error and the bitblt area will not be rendered.
  Windows enforces this behavior even if you do not turn on Desktop Window Manager Composition. Apps with compatibility declared for Windows 7 and later must perform a bitblt to a clipping window.
- Windows Vista (default): Apps must be able to perform a bitblt to the primary without a clipping window, as legacy apps depend on this behavior; running this app turns off the Desktop Window Manager.

GetOverlappedResult API

- Windows 8 and Windows 7: Resolves a race condition where a multithreaded app using **GetOverlappedResult** can return without resetting the event in the overlapped structure, causing the next call to this function to return prematurely.
- Windows Vista (default): Provides the behavior with the race condition that apps might have a dependency on.
  Apps that must avoid this race prior to the Windows 7 behavior should wait on the overlapped event and, when signaled, call GetOverlappedResult with bWait == FALSE.

Shell themes status in high contrast mode

- Windows 8: Returns the real theming status for when in high contrast mode.
- Windows 7: Returns theming as unavailable when in high contrast mode because DWM is still on.
- Windows Vista (default): Returns theming as unavailable when in high contrast mode because DWM is still on.

Shell iPersistFile::Save method

- Windows 8: CShellLink::Save now determines if the IPersistFile handler is called with a relative path argument and fails the call if it is.
  The public documentation describing this behavior indicates that path argument has to be an absolute path:
  http://go.microsoft.com/fwlink/?LinkId=228369
- Windows 7 and earlier (default): CShellLink::Save does not determine if the iPersistFile handler sends a relative path check and allows apps to continue working with absolute or relative paths.

Program Compatibility Assistant (PCA)

- Windows 8: Apps with the compatibility section do not get the PCA mitigation.
- Windows 7: Apps with the compatibility section are tracked for potential compatibility issues for Windows 8 changes (described in this document).
- Windows Vista (default): Apps that fail to install properly or crash during runtime under some specific circumstances get the PCA mitigation.
  For more info, see the Resources section.

Leveraging feature capabilities

Update the app manifest with the latest compatibility info for operating system support. This section describes the additions to the manifest:

Namespace: Compatibility.v1 (xmlns="urn:schemas-microsoft-com:compatibility.v1">)

Section name: Compatibility (new section)

SupportedOS: GUID of supported operating system - The GUIDs that map to the supported operating systems are:

- {e2011457-1546-43c5-a5fe-008deee3d3f0}
  for **Windows Vista**: This is the default value for the switchback context
- {35138b9a-5d96-4fbd-8e2d-a2440225f93a}
  for **Windows 7**: Apps that set this value in the app manifest get the Windows 7 behavior
- {4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}
  for **Windows 8**: Apps that set this value in the app manifest get the Windows 8 behavior

Microsoft will generate and post GUIDs for future Windows versions as needed.

An XML example of an updated manifest:

Note: The attribute and tag names in the app manifest are case sensitive.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">
<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
   <application>
          <!--The ID below indicates app support for Windows Vista
-->
          <supportedOS Id="{e2011457-1546-43c5-a5fe-
008deee3d3f0}"/>
          <!--The ID below indicates app support for Windows 7 -->
          <supportedOS Id="{35138b9a-5d96-4fbd-8e2d-
a2440225f93a}"/>
          <!--The ID below indicates app support for Windows
Developer Preview -->
          <supportedOS Id="{4a2f28e3-53b9-4441-ba9c-
d69d4a4a6e38}"/>
   </application>
</compatibility>
```

```
</assembly>
```

The GUIDs for all the operating systems in the previous example provide down-level support. Apps that support multiple platforms do not need separate manifests for each platform.

Tests

An app can specify multiple supported operating system IDs. You should add a supported operating system ID if you have tested, or are in the process of testing, the app on that operating system. Windows Vista and prior operating system versions do not pay attention to these entries. Starting with Windows 7, Windows will choose the highest version GUID in the manifest up to the running Windows version, and give the app support at that level.

To verify that the app works with the new app manifest compatibility section:

1. Test the app with the new compatibility section and SupportedOS ID = { 4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38} to ensure that the app works properly using the latest Windows 8 behavior.
2. Test the app with the new compatibility section and SupportedOS ID = {35138b9a-5d96-4fbd-8e2d-a2440225f93a} to ensure that the app works properly using the Windows 7 behavior.
3. Test the app with the new compatibility section and SupportedOS ID = {e2011457-1546-43c5-a5fe-008deee3d3f0} to ensure that the app works properly using the Windows Vista behavior.

Resources

QueryActCtxW Function
http://msdn.microsoft.com/library/aa375704(v=VS.85).aspx

UAC manifest
http://msdn.microsoft.com/library/bb756929.aspx

Application Manifests for Windows Applications
http://msdn.microsoft.com/library/aa374191(v=VS.85).aspx

Desktop Window Manager (DWM)
http://msdn.microsoft.com/library/aa969540(v=VS.85).aspx

Context Mismatch Update
http://go.microsoft.com/fwlink/?LinkId=205035

Program Compatibility Assistant
http://msdn.microsoft.com/library/bb756937.aspx

Queued present model is being deprecated

Platform(s)

Clients – After Windows 8

Servers –After Windows Server 2012

Description

In the Windows release after Windows 8, these APIs will return E_NOTIMPL:

- DwmSetPresentParameters
- DwmSetDxFrameDuration
- DwmModifyPreviousDxFrameDuration

In addition, this API will only accept the NULL value for the hwnd parameter:

- DwmGetCompositionTimingInfo

We will stop supporting DwmGetCompositionTimingInfo with hwnd != NULL and remove associated functionality. If non-NULL value for hwnd is specified, this API will return E_INVALIDARG.

We also discourage the use of DwmGetCompositionTimingInfo API and suggest that developers switch to the DXGI flip model and associated DX present statistics API.

Manifestation

Applications that use the queued present model will not function correctly. The exact manifestation depends on the particular application, but can range from incorrect presentation timing to the application exiting unexpectedly). In practice, we do not expect to see many (if any) such applications. This model was used by Vista media player, which will not be used on Windows 9 (and possibly the old Zune player). At this time we don't have info about any other applications actually using this model.

Solution

Developers need to use the DXGI flip presentation mode instead of queued present (available in DX9 runtime since Windows 7, and on DX10 and DX11 runtimes in Windows 8).

Tests

Run general tests to ensure that inbox Windows components and major products continue to work on the next version of Windows.

Program Compatibility Assistant scenarios for Windows 8

Platforms

Clients - Windows XP | Windows Vista | Windows 7 | Windows 8

Description

Program Compatibility Assistant (PCA) is a feature in Windows 8 that helps end users to run desktop apps designed for earlier Windows versions. Windows 8 has great built in app compatibility that enables apps designed for Windows 7 or earlier Windows versions to work great on Windows 8 automatically. However, there are a small number of apps that can have trouble running without intervention.

When a user runs an app, PCA tracks the app and identifies any symptoms of certain known compatibility issues in Windows 8. When it detects any issue symptoms, it provides the user an opportunity to apply a recommended fix that will help run the app better on Windows 8.

Scenarios

PCA tracks apps for a set of known compatibility issues in Windows 8. PCA tracks the issues, identifies the fixes, and provides a dialog to the user with instructions to apply a recommended fix. The user can decide to apply the recommended fixes, or choose to do nothing and cancel out of the recommendation. If the user cancels out, PCA will no longer track that app.

PCA generally applies one of three Windows compatibility modes – Windows XP SP3, Windows Vista SP2, or Windows 7, depending on when the program (or its setup) was authored. PCA uses the LINK_DATE and SUBSYSTEM VERSION attributes of the program and the executable file manifest's TRUSTINFO and COMPATIBILITY sections to determine which of the modes is relevant and applies Windows XP SP3 (includes administrative privilege), Windows Vista SP2, or Windows 7 respectively. A glossary at the end of the document lists each of the compatibility modes that PCA applies and its description.

For all scenarios listed below, PCA tracks apps for a second time after a fix is applied. If the app continues to fail in the same way even after a compatibility fix is applied, PCA revert the fix. PCA will then permanently stop tracking the specific app that failed.

While PCA tracks many potential issues, not all of the issues will actually cause app failures. PCA recommends fixes only in situations where there is a high probability that the app failure is due to Windows compatibility reasons. The sections below expand on each of the PCA scenarios developed in Windows 8. Each section describes the problem scenario and the recommendations that PCA provides to allow the app to continue working properly on Windows 8.

To learn more about compatibility changes in Windows 8, refer to the other topics in the Windows 8 Compatibility Cookbook.

The scenarios that PCA tracks and recommends fixes to are:

- App fails to Install or Uninstall
- App fails to run with a Windows version check message
- App fails to launch due to administrative privilege
- App crashes due to specific memory problems
- App fails due to mismatched system files
- App fails due to Unhandled Errors on 64-bit Windows
- App fails while attempting to delete protected non-Windows files
- App fails while attempting to modify Windows files
- App fails due to using 8- or 16-bit color modes
- App fails due to graphics and display issues
- App fails to declare DPI awareness
- App fails due to missing Windows features
- App fails due to unsigned drivers on 64-bit Windows 8
- Tracking apps installed through compatibility settings
- App fails to launch installers or updaters
- App installers that need to run with administrative privilege
- Legacy Control Panel applets that need to run with administrative privilege

Each of these scenarios is expanded in below:

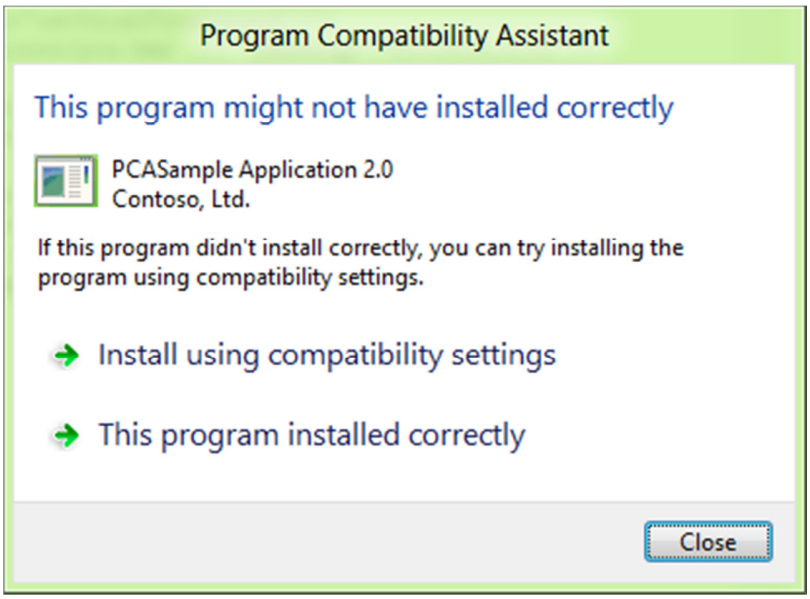
App fails to Install or Uninstall

One of the most common types of app failures occurs during the installation of the app. Older Setup programs most commonly fail in two ways:

- The setup program is not aware of the User Account Control (UAC) features in Windows 8, so, it may not run with the full privileges needed to make system changes to the protected areas of Windows 8
- The setup program checks for the Windows version and blocks itself from running if the version is higher than what it expects

These failure conditions are two of the most common types of compatibility failures in setup. PCA, with the help of various other Windows components such as UAC, detects Setup programs at launch and tracks them to the end of the install. If the Setup program either fails to add files or to add a valid entry in the 'Add Remove Programs' part of the Windows control panel, then PCA considers the setup to have failed.

In this case, PCA recommends a compatibility mode appropriate for the app. The compatibility mode allows the setup program to run in the Windows mode it was designed for and also ensures that the app runs with administrative privileges. PCA applies the RUNASADMIN compatibility mode along with the appropriate Windows XP, Windows Vista, or Windows 7 compatibility mode. The user, at the end of the failed install, will see a dialog with the PCA recommendation:



The user can then choose to:

- Run the program using the compatibility settings (recommended option), after which PCA will apply the recommended setting (compatibility mode), restart the setup program, and track it till the setup completes successfully
- Indicate that the program installed correctly, in which case PCA will not add any settings and will stop tracking the setup
- Click Close, in which case PCA will not add any settings and will stop tracking this setup

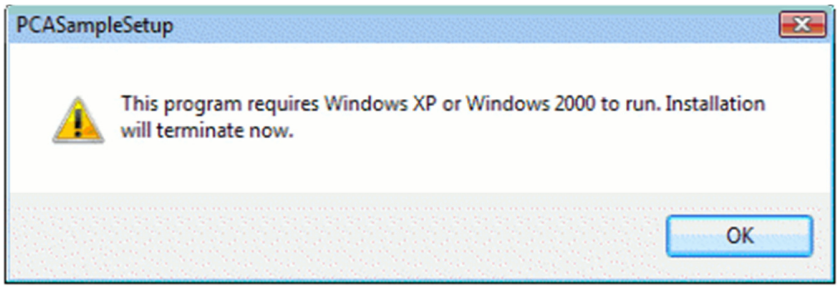
The same mechanism is used to help the app's uninstallation when a user tries to uninstall the app either from the 'Add Remove Programs' section in Windows, or from the app's uninstaller shortcut.

App fails to run with a Windows version check message

One of the more common compatibility failures in app runtime is due to the Windows version check. Many apps, upon launch check the Windows version; if they do not recognize the version, they block themselves even if the app could have run without issues.

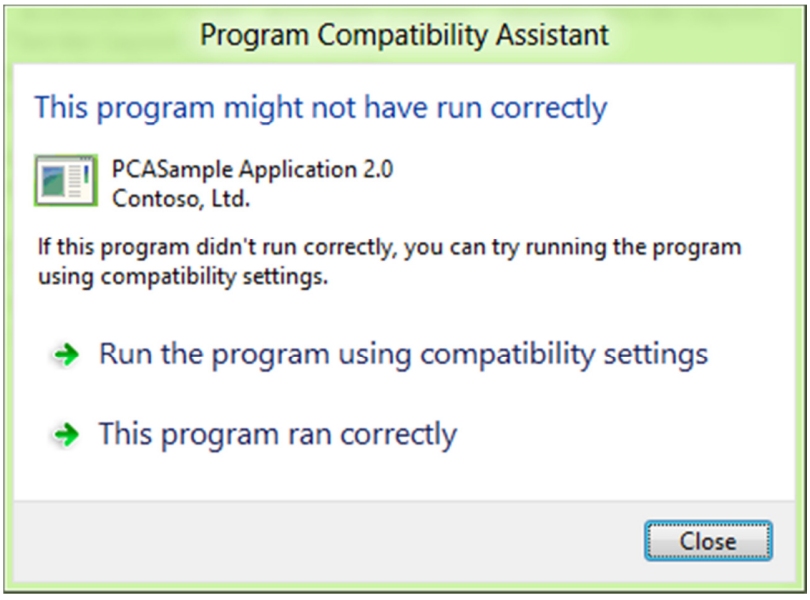
Generally, such checks are associated with two conditions that the PCA tracks:

- The app displays a message box that warns the user. An example is below:



- The app terminates immediately or crashes

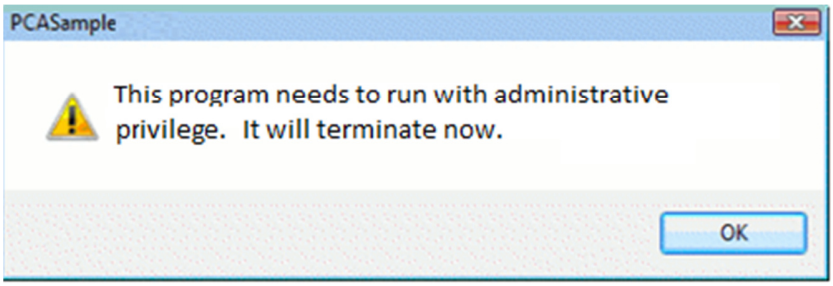
If PCA identifies both of these conditions for an app, it will provide a recommendation to the user. PCA will allow the user to re-run the app with compatibility settings. PCA will apply the appropriate Windows XP, Windows Vista, or Windows 7 compatibility mode based on the app. As in any of the scenarios, the user can tell PCA that the app ran correctly, or opt out of the recommended settings by clicking the close button. An example dialog is provided as below:



App fails to launch or run due to administrative privilege

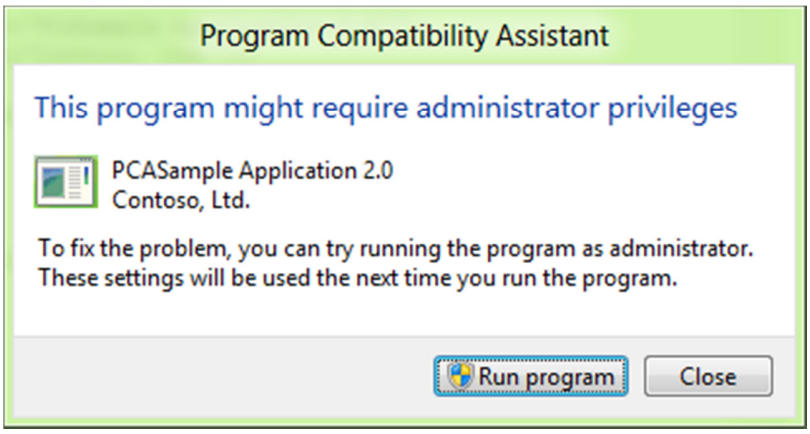
Some apps need administrative privilege to run and execute their functionality. However, in Windows 8, similar to Windows 7 and Windows Vista, apps run in lower privilege levels by default due to UAC. Newer apps designed for Windows Vista and above will generally declare the privilege level they need to run at using the EXE manifest's TRUSTINFO section. However, older apps generally fail in two ways:

- App displays a message to the user that it requires administrative privilege, as below example:



- App either terminates immediately or crashes

If PCA identifies both of these conditions for an app, it will provide a recommendation to the user. PCA will allow the user to re-run the app with administrative privileges (PCA applies the RUNASHIGHEST compatibility mode). The user will get a UAC prompt when the app re-runs. As in any of the scenarios, the user can choose to re-run with the recommended setting, or opt out of the recommended settings by clicking Close. An example dialog is provided as below:



App crashes due to specific memory problems

Some apps crash due to a well-known memory problem. The app dereferences a DLL from memory, and then calls a function to execute code in the same DLL. This causes an immediate crash of the app. While this problem is not due to Windows 8 compatibility changes, it is a relatively common problem seen in a wide variety of apps. PCA tracks this issue to give users a chance to run their app more reliably.

For these apps, PCA automatically applies the PINDLL compatibility mode silently. The compatibility mode invoked by PCA prevents the app from freeing the DLL from memory. So, the function call into the DLL by the app will work, preventing the app from crashing and allowing it to continue to function properly.

App fails due to mismatched system files

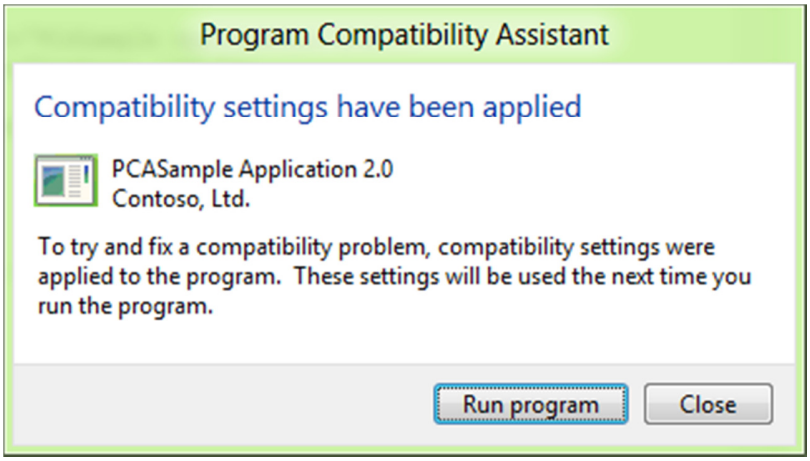
Some apps designed for Windows XP and prior include copies of Windows system DLLs along with their installers. When such apps are installed, the app has both an

older copy of the DLL in its own folder as well as the latest version of the DLL that is in the Windows system folders.

On Windows Vista and later, this condition can cause the app to fail when it tries to load the local DLL, since this DLL will not work well with the rest of the current Windows system DLLs. Since the app is generally not aware of the newer versions of this DLL, it fails to work properly.

When PCA detects that the DLL failed to load properly, it will applies a compatibility setting that allows Windows to load the latest version of the DLL from the Windows system folder so the app can run properly.

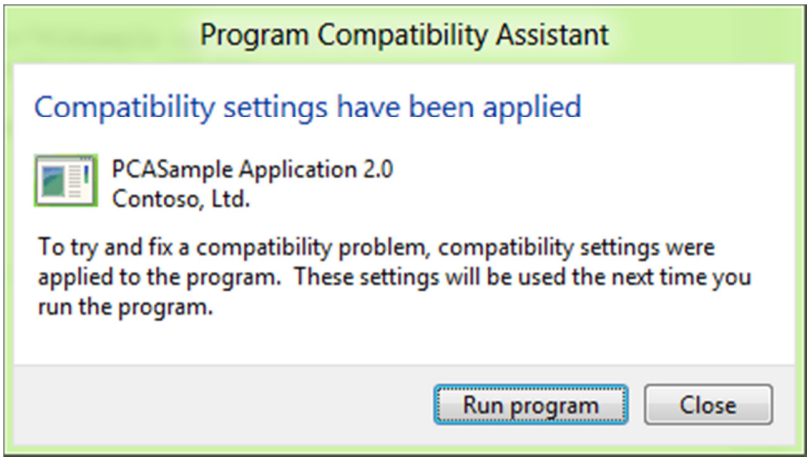
At the end of the first failed run of the app, users will see the PCA dialog that notifies them of the applied setting as below:



App fails due to Unhandled Errors on 64-bit Windows

On 64-bit version of Windows 8, a new exception was enabled to the message loop callback mechanism. While this exception was first introduced in Windows 7, it was not mandatory to handle this error. In Windows 8, apps that use message loops must handle this new exception. If they do not, they will crash. Apps designed for older Windows versions may not be aware of this exception, and hence may not handle this error (exception) properly.

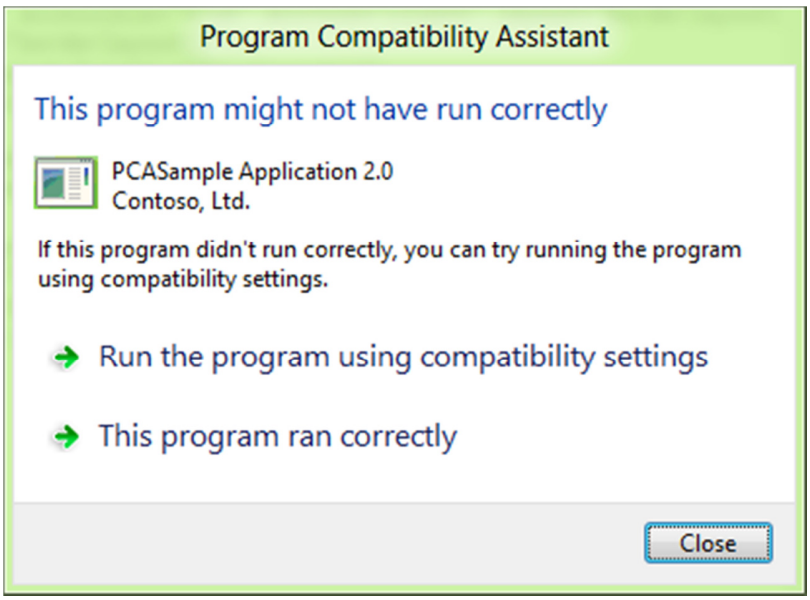
PCA detects apps that fail due to this unhandled error, and automatically applies the DISABLEUSERCALLBACKEXCEPTION compatibility mode for the app. After the setting is applied at the end of the run, the user is notified as below. The app will get the mode on the next run, and will be able to avoid this error.



App fails while attempting to delete protected non-Windows files

Some apps designed for Windows XP and prior assume that they usually run with full administrative privileges. As a course of normal app behavior, they may try to delete protected non-Windows files (either in program files or Windows folders). When the delete operation fails many such apps can crash.

PCA detects these apps that fail to delete protected files and crash, and provides a recommendation to the user. PCA will allow the user to re-run the app with compatibility settings. As in any of the scenarios, the user can tell PCA that the app ran correctly, or opt out of the recommended settings by clicking the close button. In this case PCA applies the VIRTUALIZEDELETE compatibility mode. An example dialog is provided as below:

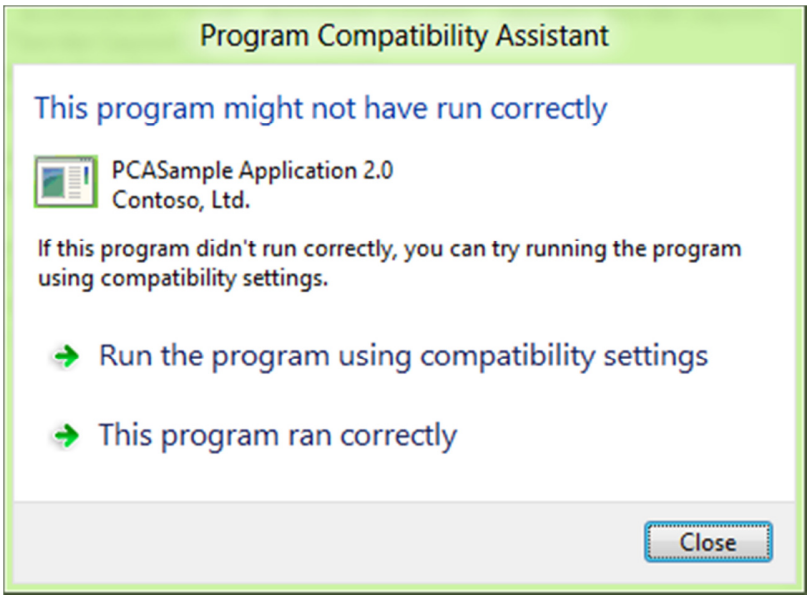


App fails while attempting to modify Windows files or registry keys

Some apps designed for Windows XP and prior assume that they usually run with full administrative privileges. As a course of normal app behavior, they may try to modify, delete or write Windows protected files (either in program files or Windows folders)

or Registry keys owned by Windows. When any of the write, delete or modify operation for a file or a registry key fails many such apps can crash or fail badly.

PCA detects these apps that fail to write to protected Windows files or registry keys, and provides a recommendation to the user when the app quits. PCA will allow the user to re-run the app with compatibility settings. As in any of the scenarios, the user can tell PCA that the app ran correctly, or opt out of the recommended settings by clicking the Close button. In this case PCA applies the WRPMITIGATION compatibility mode. An example dialog is provided as below:



App fails due to using 8- or 16-bit color modes

As part of reimagining Windows 8 for Windows Store apps, one of the key changes is that the Desktop Window Manager (DWM) will now support only 32-bit colors in Windows 8. Lower color modes are now simulated.

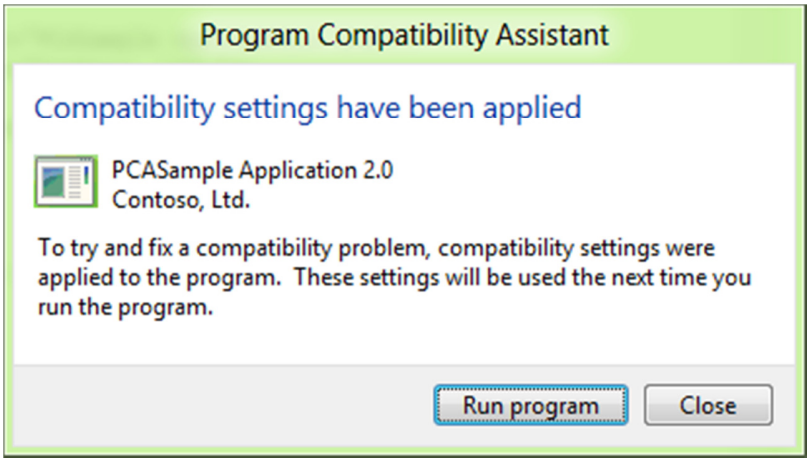
Many older apps and games designed for Windows XP or before use 8-bit or 16-bit color modes. With no mitigations, these apps could fail to execute on Windows 8. However, when these apps enumerate or try to use any of the 8-bit or 16-bit color modes for display, PCA immediately identifies the issue and with the help of DWM, ensures that the app will work properly with the simulated color mode.

Note that this happens as soon as the app requests the low color modes and is transparent to the user. The user does not have to restart the app to get this mitigation because this fix is always needed to ensure that the app works properly.

Application fails due to graphics and display issues

Since Desktop Window Manager (DWM) is always on in Windows 8, some older Windows XP era apps can fail if the app uses mixed mode graphics APIs, as in using both GDI and DirectX APIs to draw to the screen (mostly older games), and tries to use full screen mode:

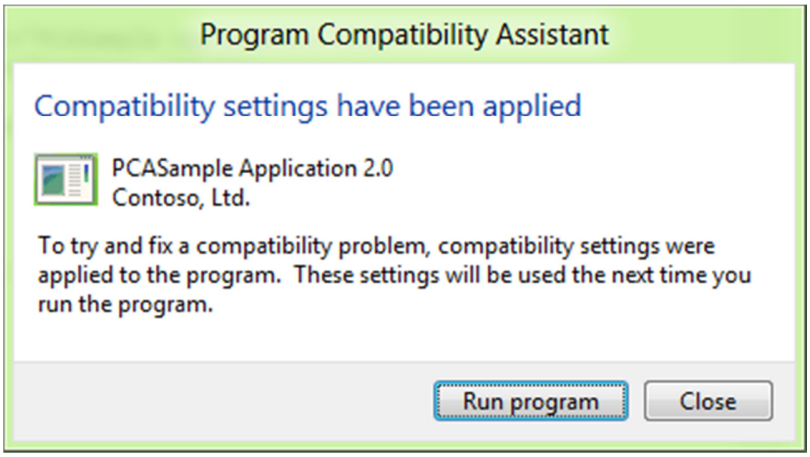
- DWM will prevent painting directly to the desktop and the game or app will either fail, or draw a black screen on to the desktop and none of the graphics will be visible
- In such cases, when the app quits, Windows detects that the app or game has a problem with full screen mode, and applies the DXMAXIMIZEDWINDOWEDMODE compatibility mode that allows the app or game to run in a maximized windowed mode instead of a full screen mode
- After the setting is applied at the end of the run, the user is notified by PCA as shown below; the app will get the compatibility mode on the next run, and will be able run properly



App fails to declare DPI awareness

Another typical display problem with many older apps happens when Windows and the app run in high DPI mode, but the app does not declare its awareness of High DPI through its EXE manifest. Among the common problems that can occur due to this mismatch in settings are clipped UI elements or text and incorrect font size. For more details on the issues, see this link here.

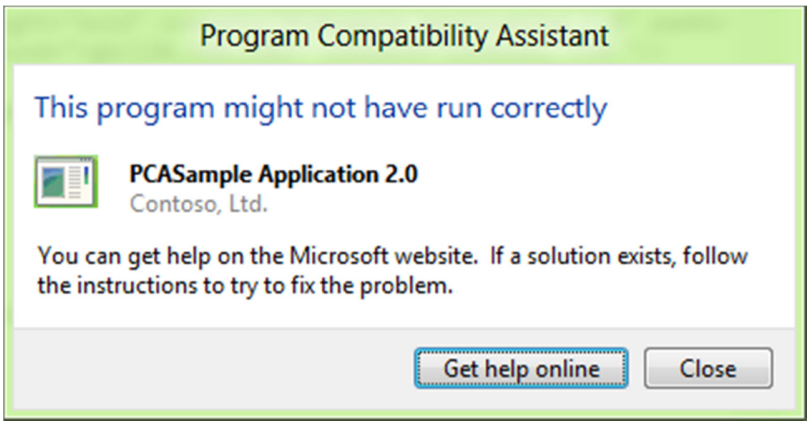
In such cases, Windows detects that the app is high DPI aware, and applies the HIGHDPIAWARE compatibility mode to the app at the end of the first run. PCA will then inform the user about this as shown below:



Application fails due to missing Windows features

Some apps depend on Windows features that have been removed since Windows Vista. When these apps try to load the missing DLLs or COM components, they fail to work.

PCA detects apps when they try to load the missing Windows features, and provides a recommendation to download these components and install them after the app terminates. The user can click on 'get Help Online' to find either an alternative or to download the feature and install it. If needed, the user can choose to do nothing by clicking Close.

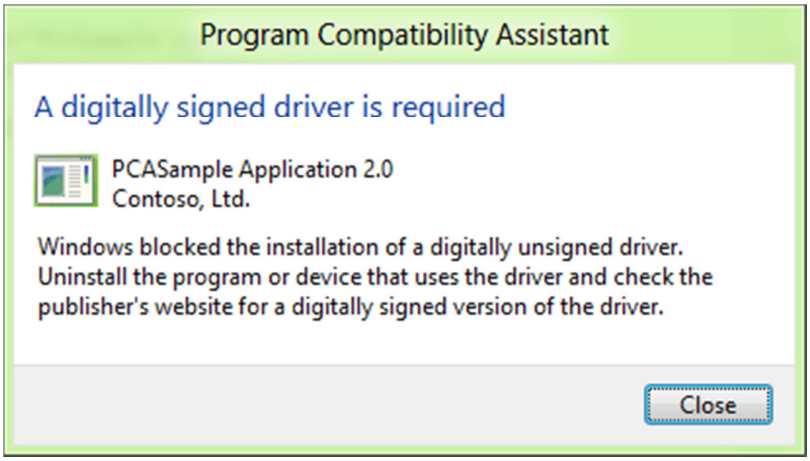


App fails due to unsigned drivers on 64-bit Windows 8

64-bit Windows has required digitally signed drivers (SYS files) since Windows Vista. However, older apps designed prior to the release of Windows Vista shipped drivers that were not digitally signed. If such an unsigned driver is installed, Windows will not load them. In rare cases, it is possible that Windows will not start if such drivers are marked as boot-time drivers.

Some older apps install drivers that are not signed on 64-bit Windows. Any device or app that tries to use this driver may fail or result in a system crash. To prevent such a scenario, PCA detects apps when they install unsigned drivers, and disables the driver it is marked as a boot-time driver.

It also instructs the user to acquire a digitally signed driver for the app to work properly. The message is shown as a result of the installation of the driver, and as a result of the installation of the app. If another app installs the same driver, that app will get the same message as well.



Tracking apps installed through compatibility settings

When an installer fails, PCA helps the installer with various compatibility modes depending on the type of failure. Once the installer succeeds with compatibility settings, PCA will track the shortcuts that the installer added. This is done to track if the apps that were installed may also need the compatibility settings applied to their installer.

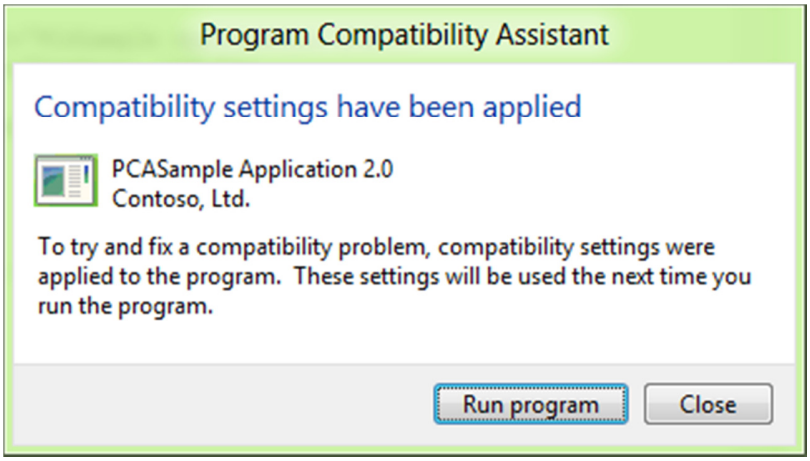
When a user launches such an app, PCA prompts the user to ask if the app worked properly. If the user answers, 'Yes,' the PCA stops tracking the app. If the user answers 'No,' then it applies the same compatibility mode that was applied to the app's installer, and re-runs the app with the compatibility mode applied.

App fails to launch installers or updaters

Apps sometimes launch child programs that need to run as administrators. This is typically the case when an app tries to launch its updater software to check and install new updates to the app. When apps directly run such child programs, the child program can fail to launch because the app itself did not have administrative privileges, or because the child program was not properly marked for elevation with the UAC manifest.

PCA tracks these errors and when the primary app closes, it automatically applies the ELEVATECREATEPROCESS compatibility mode that will help the child programs run correctly. When the app launches the child app on subsequent runs, the user will see a UAC dialog for the child program.

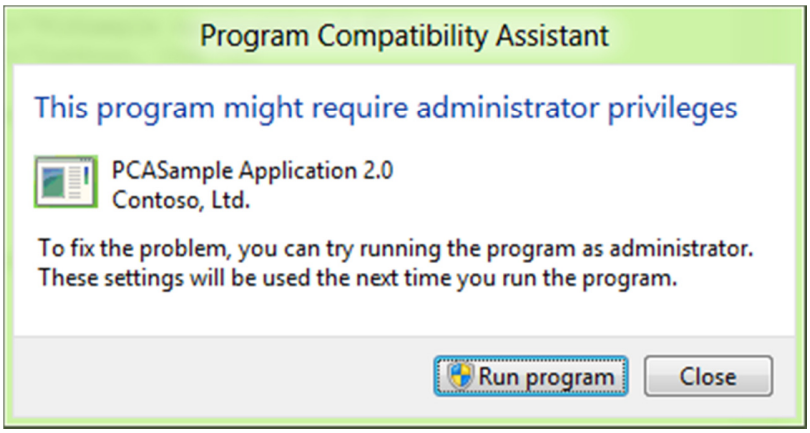
An Example of the PCA dialog is shown below:



App installers that need to run with administrative privilege

Installers of Windows desktop apps require administrative privileges since they write files, folders, and registry entries to protected system areas. Windows (UAC) has detection logic to identify when an installer is run, and immediately prompts the user to provide administrative privileges through the UAC dialog. However, in some cases, this logic will not be able to determine that an app was indeed an installer, and may not get administrative privileges. These are generally custom made installers that do not use well known install technologies such as Windows Installer or Install Shield.

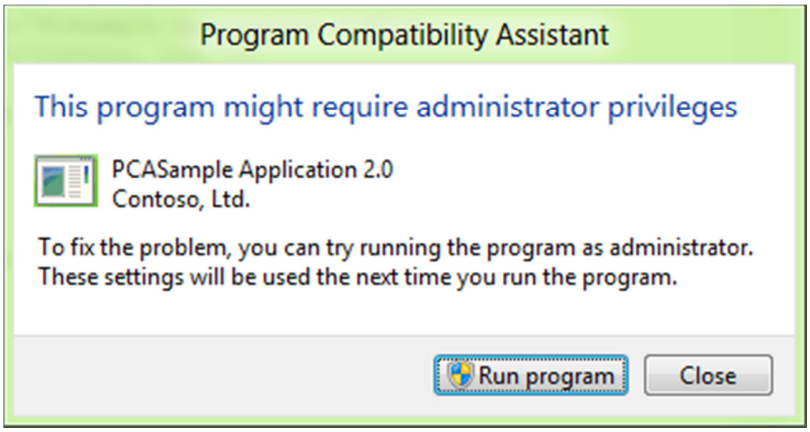
In such cases, PCA detects that the installer failed to write its files. At the end if the failed install, PCA will provide a recommendation to apply compatibility settings. If the user chooses to click 'Run Program,' PCA will apply the RUNASADMIN compatibility mode, and re-run the installer. If the user chooses to close, then no setting will be applied. An example PCA dialog is shown below:



Legacy Control Panel applets that need to run with administrative privilege

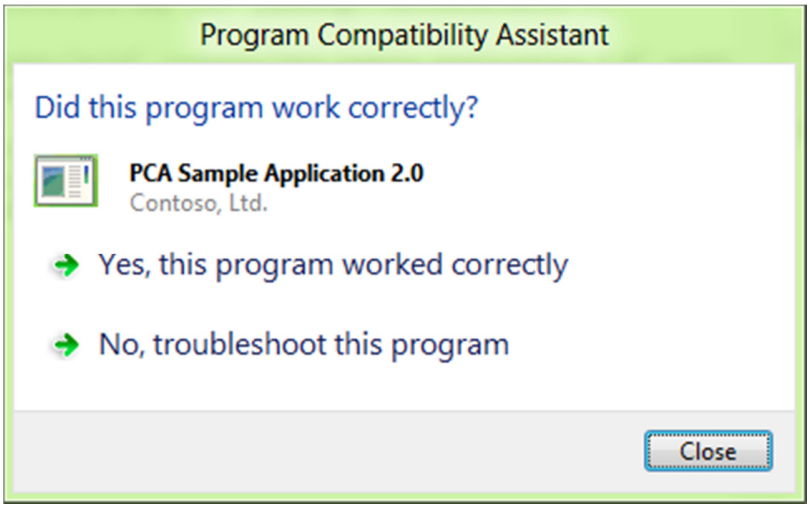
Control panel applets generally change system settings and need the ability to run ad administrator. However, those written before Windows Vista either do not have an EXE manifest or do not have the TRUSTINFO section that declares the privilege level they require.

When such applets are run, PCA detects them, and at the end of the first run, provides a recommendation to run with administrative settings. If the user chooses to click 'Run Program,' PCA applies the RUNASADMIN compatibility mode, and re-runs the installer. If the user chooses to close, then no settings will be applied. An example PCA dialog is shown below:



Verifying the recommended settings through user feedback

At the end of each of the scenarios (after the app is run with recommended compatibility settings), PCA will ask the user a simple question:



The user can provide feedback if the app worked or failed with the compatibility setting. This data will be sent anonymously to Microsoft. This helps to ensure that such fixes can be built into Windows 8 through Windows update process, so that future users of Windows 8 will no longer encounter the app failure, and PCA will no longer need to track the app for the failure.

Tracking issues that have no recommendations

Apps may fail in many different ways for compatibility reasons. PCA tracks many more compatibility issues than what is listed in the above scenarios. In these cases, many times, the issue manifestation depends on the app. This means that some apps handle such issues gracefully and recover from it, while others may not. So, for such

issues, while PCA still tracks the app, it does not provide a direct recommendation for a fix.

The issues that PCA tracks that do not have a recommended setting or a dialog include apps that:

- Have a very short runtime – Apps run for no more than three seconds
- Create global memory objects without administrative privileges
- Have an error (Win32 exception) on launch
- Check for administrative privilege (but may not fail)
- Use Indeo codecs (deprecated from Windows Vista)
- Try to write or delete keys from protected registry locations such as HKLM
- Crash on launch

Applying fixes through the compatibility tab and compatibility troubleshooter

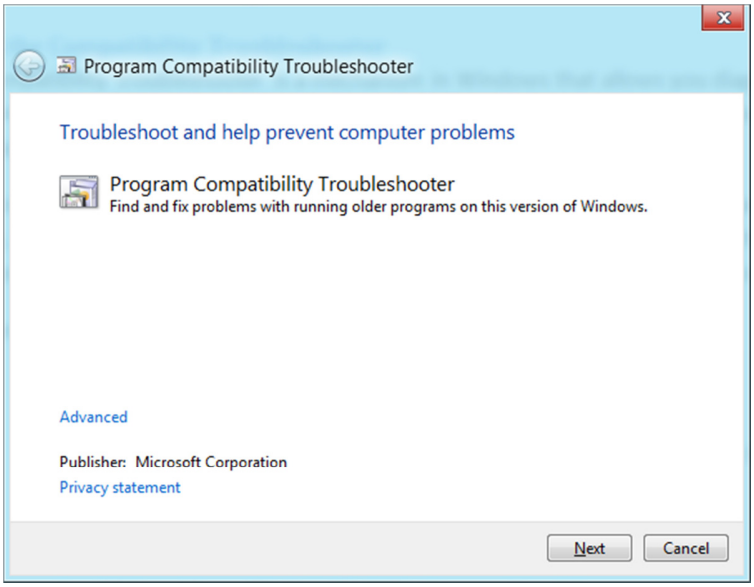
As mentioned above, apps can fail for a variety of compatibility reasons. Not all of these have clear PCA recommendation since the settings are app dependent. However, users can go to the Compatibility Troubleshooter or the Compatibility Tab to apply certain common fixes to try to get their failing app to work properly on Windows 8. In such cases, PCA will still track the app for compatibility issues, before and after the fix is applied. After the app is run with the fix applied, PCA will ask the user if the fix worked. Once the user answers the question, the data is sent anonymously through telemetry to Microsoft. This data is collected from many users and analyzed, and the qualifying fixes are then broadly distributed to all Windows 8 users through Windows update.

Using the Compatibility Troubleshooter

The Compatibility Troubleshooter is a mechanism in Windows that allows you diagnose problems with apps and apply recommended fixes to get them working properly. This is needed only when PCA does not to provide any recommendation for the app.

The troubleshooter allows users to walk through and answer a set of questions, and based on the replies, it will apply a set of fixes and allow the users to test their apps and verify the fixes. Once verified, the fixes will be applied permanently to the apps to make them work better on Windows 8.

The Troubleshooter UI is shown below for reference:



You can start the Compatibility Troubleshooter in two ways:

- From the start screen:
    1. Type: compatibility troubleshooter
    2. Under the settings section, click the 'Run programs made for previous version of Windows' tile
- From the app tile:
    1. From the Start screen, right click the app tile
    2. Click 'Open File Location' (Desktop apps only)
    3. From the Explorer ribbon, click the 'App tab'
    4. Choose 'Troubleshoot Compatibility'

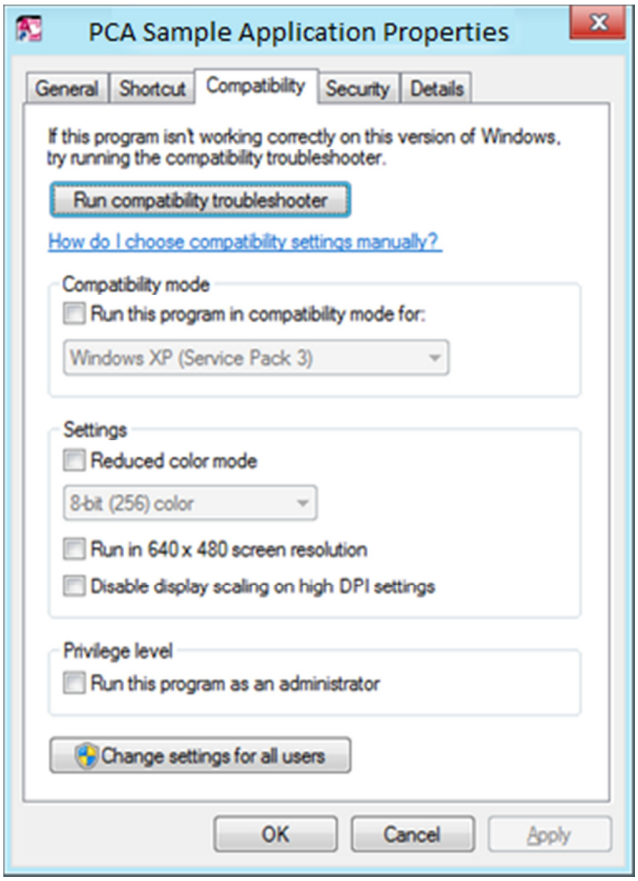
Using the Compatibility Tab

Note that this is recommended only for users who are experts in trying different compatibility settings. This method does not provide any recommendation of the type of fix to apply to apps. Here the user is expected to know what fixes can be applied to make the app work. If you are unsure of the fixes, please use the Compatibility Troubleshooter to find a fix for the app.

To access the Compatibility Tab:

- From the start screen:
    1. Right click the app tile
    2. Open file location (desktop apps only)

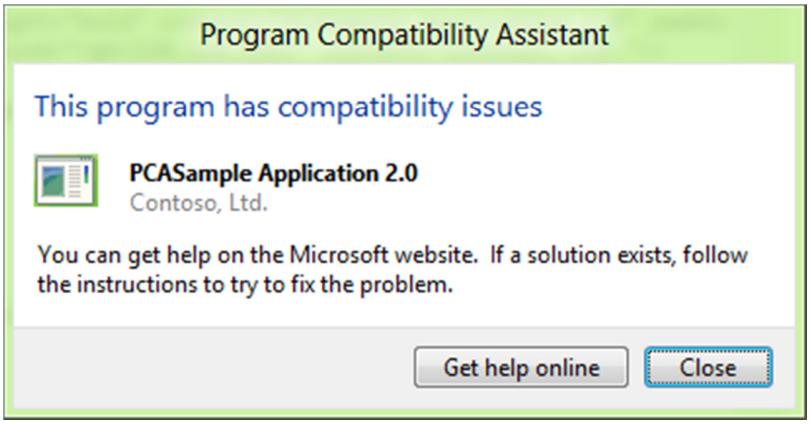
- From the Explorer ribbon:
  1. Click Properties
  2. Navigate to the Compatibility Tab
  3. Select the compatibility fixes
  4. Re-run the app
     Note that you can come back to the same place again to change or remove the fixes as well. You can also apply the fixes to all users on the machine using the button provided in the tab.



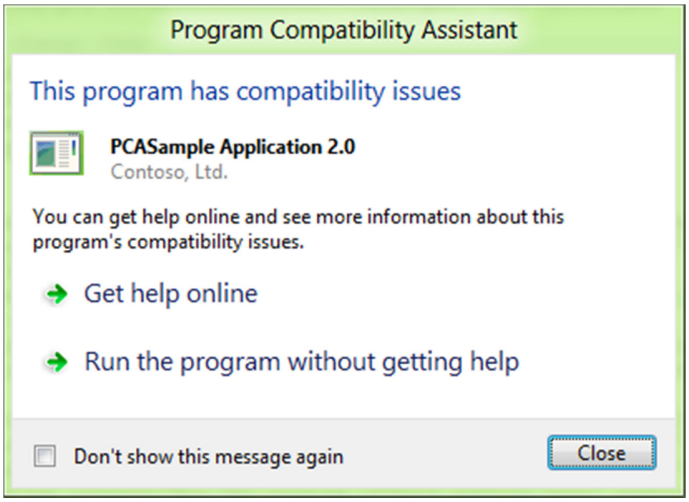
Apps with known compatibility issues

Apart from the runtime issues detection scenarios listed above, PCA also informs users at app startup if the app has known compatibility issues. The list is stored in the System app compatibility database. There are two types of these messages:

- **Hard Block Messages**—if the app is known to be incompatible and if allowing the app to run will result in severe impact to the system (for example, a Windows crash or being unable to boot after the installation), a blocking message as shown below will be displayed.



- **Soft Block Messages**— If the app has a known compatibility issue and may not work properly, then this message is shown:



In both cases, the 'Get help Online' option sends a Windows Error Report to get an online response from Microsoft and display it to the user. Typically the responses will point the user to one of three types of resources:

- An update from the app vendor
- An app vendor's website for more info
- A Microsoft Knowledge base article for more info

Telemetry for PCA

After PCA addresses any app issues on a Windows 8 machine and gets all the user feedback, it collects anonymous data about the app, the installer, the issues detected, and the compatibility settings applied to the app, and send it back to Microsoft. This data is collected from any user who is willing to provide such anonymous data (through the Customer Experience Improvement Program - CEIP). Once this data is collected, the app failures and fixes are analyzed, and the fixes are then distributed to the entire Windows ecosystem through the Windows Update mechanism so that any user of the app in the future benefits from the fix automatically.

Administrative controls and managing PCA settings

IT administrators can control PCA behavior in two ways:

- **Turn off PCA** – this setting allows IT administrators to turn off the dialogs that PCA shows to the users; PCA will still track and detect issues and send back telemetry
- **Turn off App telemetry** – this setting will turn off any collection and sending of telemetry data by PCA
  Note that if CEIP is turned off, this setting has no impact.

Designing apps to work with PCA

Developers need to ensure that their apps will work well across all of the compatibility scenarios described above. Developers must test and validate their apps for each of the above scenarios and ensure that there are no compatibility issues. If compatibility issues are identified, developers should make the fixes to their apps necessary to ensure that the compatibility issue is resolved. Some of the common fixes that developers should make include:

- Eliminate Windows operating system version checks at install and runtime
- Eliminate privilege check (checking for administrator access); use the EXE manifest to declare the right level of privilege needed
- Ensure that Windows binaries are not shipped within the app installer
- Eliminate writing to protected areas (registry, folders) or writing over protected files
- Digitally sign all binaries (EXE, DLL, SYS files)
- For installers, ensure that proper 'Add/Remove programs' entry is added: at a minimum, this app metadata entry should include the app name, publisher, Version string, and supported language. This will indicate to PCA that the installer completed successfully and will also provide a convenient way for users to uninstall the app

Ensuring that the TRUSTINFO and COMPATIBILITY section of the app (executable) manifest is updated as listed in the Windows 8 Compatibility Cookbook will let PCA know that the app was designed for Windows 8, and will also ensure that the app always runs natively without any compatibility modes applied to it.

To ensure that PCA considers the app to be designed for Windows 8:

- The all EXEs (installer or runtime) must be manifested for TRUSTINFO and COMPATIBILITY sections for Windows 8
- The installer should add an 'Add/Remove programs' entry

Glossary

The compatibility modes used by PCA are listed below with a brief description of what the mode enables.

Mode	Description
Windows7RTM	This mode emulates common Windows 7 behavior including the operating system version number 6.1
WindowsVistaSP2	This mode emulates common Windows Vista SP2 behavior including the operating system version number 6.0
WindowsXPSp3	This mode emulates common Windows XP SP3 behavior including the operating system version number 5.1. This also includes the RUNASHIGHEST mode
RUNASHIGHEST	This mode prompts the user to run the app with the highest available privilege. Users with administrative privileges will see a UAC elevation prompt for the app
RUNASADMIN	This mode always prompts the user to run the app with administrative privileges; apps with this mode will always get the UAC elevation prompt
ELEVATECREATEPROCESS	This mode makes child processes of the main app run with administrative privileges; the child processes will get a UAC elevation dialog
PINDLL	This mode forces a DLL to be in memory for an app even if the app unloads the DLL
DISABLEUSERCALLBACKEXCEPTION	This mode intercepts user call back exceptions and allows the app to continue on without having to handle the exception
VIRTUALIZEDELETE	This mode intercepts delete operations on protected files and prevents apps from failing due to unhandled exceptions from the delete operation
WRPMITIGATION	This mode returns success when an app tries to write, modify, or delete Windows protected files or registry entries (without actually completing the operation)
DXMAXIMIZEDWINDOWEDMODE	This mode identifies apps that go into full screen mode and redirects them into a maximized Window mode
HIGHDPIAWARE	This mode lets the rest of Windows know that the app is High DPI aware, and helps proper rendering of UI elements, text, font, etc.

Desktop gadgets removed

Platforms

- **Clients -** Windows 8
- **Servers -** Windows Server 2012

Description

From a functionality standpoint, Gadgets have already been deprecated and are outclassed by new live tiles and apps. Gadgets also present a security risk to users. As a platform, vulnerabilities exist such that even benign Gadgets could be exploited. Therefore, in order to uphold Windows 8 as our most modern and secure operating system, we have chosen to remove Gadgets from the operating system entirely. Windows 7 users with Gadgets on their Desktop will be notified and Gadgets will be uninstalled.

Manifestation

Desktop Gadgets will not be available on the Windows Desktop.

Mitigation

The Desktop Gadgets API and folder structure will remain in place for Windows 8. Applications which programmatically install and run Gadgets using this API will continue to run without fail (although the Gadgets themselves will not).

Solution

Desktop app developers should not package Gadgets in their installers. These Gadgets will just take up space for the user and will not be able to be run in the system.

Compatibility, performance, reliability, or usability tests

Developers are able to test compatibility of this change by disabling the optional component for Desktop Gadgets in Windows 7. To disable Gadgets on a Windows 7 PC, follow the steps below.

1. Navigate to Turn Windows Features on or off from the Control Panel.
2. Uncheck the box next to “Windows Gadget Platform”.
3. Click OK and follow any additional on-screen instructions.

Resources

Vulnerabilities in Gadgets Could Allow Remote Code Execution
http://support.microsoft.com/kb/2719662

List end

Advanced format (4K) disk compatibility update

Platforms

Clients – Windows XP | Windows Vista | Windows 7 | Windows 7 SP1 | Windows 8

Servers – Windows Server 2003 | Windows Server 2008 | Windows Server 2008 R2 | Windows Server 2008 R2 SP1 | Windows Server 2012

Description

This article is an updated version of the article titled "512-byte Emulation (512e) Disk Compatibility Update" which was released for Windows 7 SP1 and Windows Server 2008 R2 SP1. This update contains much new info, some of which is applicable only to Windows 8 and Windows Server 2012.

Areal densities are increasing yearly, and with the recent advent of 3 TB disks, the error correction mechanisms used to deal with the decreasing signal-to-noise-ratio (SNR) are becoming space inefficient; that is, an increased amount of overhead is required to ensure the media is usable. One of the storage industry solutions for improving this error correction mechanism is to introduce a different physical media format that includes a larger physical sector size. This new physical media format is called *Advanced Format*. Therefore, it is no longer safe to make any assumptions regarding the sector size of modern storage devices, and developers will need to study the assumptions underlying their code to determine if there is an impact.

This topic introduces the effect of Advanced Format storage devices on software, discusses what apps can do to help support this type of media, and discusses the infrastructure that Microsoft introduced with Windows Vista, Windows 7, and Windows 8 to enable developers to support these types of devices. While the material presented in this topic provides guidelines for improving compatibility with Advanced Format disks, the info applies generally to all systems with Advanced Format disks running Windows Vista, Windows 7, and Windows 8.

Summary of new large sector related features

The below list summarizes the new features delivered as part of Windows 8 and Windows Server 2012 to help improve customer and developer experience with large sector disks. More detailed description for each item follow.

- Builds upon the Windows 7 SP1 support for 4K disks with emulation (512e), and provides full inbox support for disks with 4K sector size without emulation (4K Native). Some supported apps and scenarios include:
    - Ability to install Windows to and boot from a 4K sector disk without emulation (4K Native Disk)
    - New VHDx file format
    - Full HyperV support
    - Windows backup
    - Full support in the NT file system (NTFS)
    - Full support with *new* Storage Spaces and Pools (SSP)
    - Full support with Windows Defender

- Provides a new API to query for physical sector size (FileFsSectorSizeInformation):
    - Available for network volumes
    - Can be issued to any file handle
    - Available for unprivileged apps
    - Friendlier usage model
- Includes enhanced "fsutil" command line utility to query for logical and physical sector size of volume with alignment info (basic version of utility without alignment info is available for Windows 7 with Microsoft KB 982018 and Windows Server 2008 R2 with Microsoft KB 982018)

Introduction to advanced format (4K) disks

One of the problems of introducing this change in the media format is the potential for introducing compatibility issues with existing software and hardware. As a temporary compatibility solution, the storage industry is initially introducing disks that emulate a regular 512-byte sector disk, but make available info about the true sector size through standard ATA and SCSI commands. As a result of this emulation, there are, in essence, two sector sizes:

Logical sector: The unit that is used for logical block addressing for the media. We can also think of it as the smallest unit of write that the storage can accept. This is the "emulation."

Physical sector: The unit for which read and write operations to the device are completed in a single operation. This is the unit of atomic write.

Most current Windows APIs, such as IOCTL_DISK_GET_DRIVE_GEOMETRY will return the logical sector size, but the physical sector size can be retrieved through the IOCTL_STORAGE_QUERY_PROPERTY control code, with the relevant info contained in the BytesPerPhysicalSector field in the STORAGE_ACCESS_ALIGNMENT_DESCRIPTOR structure. This is discussed in more detail later in the article.

Initial types of large sector media

The storage industry is quickly ramping up efforts to transition to this new Advanced Format type of storage for media having a 4 KB physical sector size. Two types of media will be released to the market:

4 KB native: This media has no emulation layer and directly exposes 4 KB as its logical and physical sector size. The overall issue with this new type of media is that the majority of apps and operating systems do not query for and align I/Os to the physical sector size, which can result in unexpected failed I/Os.

512-byte emulation (512e): This media has an emulation layer as discussed in the previous section and exposes 512-bytes as its logical sector size (similar to a regular disk today), but makes its physical sector size info (4 KB) available. The overall issue with this new type of media is that the majority of app and operating systems do not understand the existence of the physical sector size, which can result in a number of issues as will be discussed below.

Overall Windows support for large sector media

This table documents the official Microsoft support policy for various media and their resulting reported sector sizes. See this KB article for details.

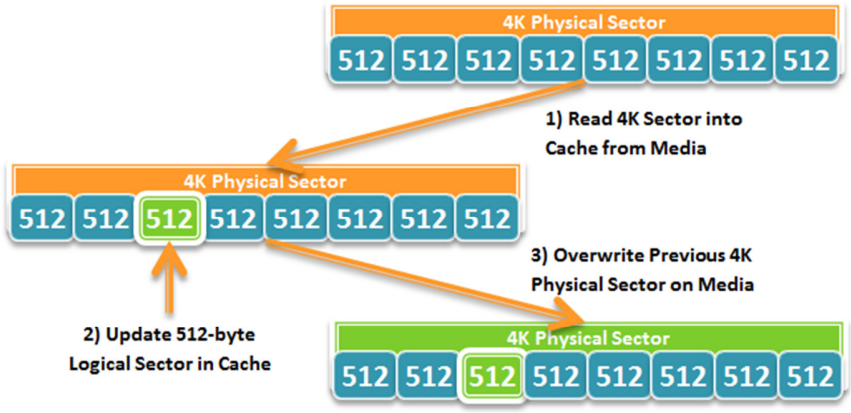
Common Names	Reported Logical Sector Size	Reported Physical Sector Size	Windows Version with Support
512-byte Native, 512n	512 bytes	512 bytes	All versions of Windows
Advanced Format, 512e, AF, 512-byte Emulation	512 bytes	4 KB	Windows 8 Windows Server 2012 Windows 7 w/ MS KB 982018 Windows 7 SP1 Windows Server 2008 R2 w/ MS KB 982018 Windows Server 2008 R2 SP1 Windows Vista w/ MS KB 2553708 Windows Server 2008 w/ MS KB 2553708
Advance Format, AF, 4K Native, 4Kn	4 KB	4 KB	Windows 8 Windows Server 2012
Other	Not 4 KB or 512 bytes	Not 4 KB or 512 bytes	Not supported

NOTES: While not stressed in the preceding table, Windows XP, Windows Server 2003, and Windows Server 2003 R2 do not support 512e or 4Kn media. While the system may boot up and be able to operate minimally, there may be unknown scenarios of functionality issues, data loss, or sub-optimal performance. Thus, Microsoft strongly cautions against using 512e media with Windows XP or other products based on the Windows XP codebase (such as Windows Home Server 1.0, Windows Server 2003, Windows Server 2003 R2, Windows XP 64-bit Edition, Windows XP Embedded, Windows Small Business Server 2003, and Windows Small Business Server 2003 R2).

How emulation works: read-modify-write (RMW)

A storage medium has a certain unit within which the physical medium can be modified. That is, the media can only be written, or rewritten, in units of the physical sector size. Thus, writes that are not performed at this unit level would require additional steps, which we will walk through in the example below.

In this scenario, an app needs to update the contents of a Datastor record located within a 512-byte logical sector. This diagram illustrates the steps necessary for the storage device to complete the write:



As illustrated above, this process involves some work by the storage device that can result in a performance loss. To avoid this additional work, apps must be updated to:

- Query for the physical sector size
- Ensure writes are aligned to that reported physical sector size

While this may initially appear to be only a performance issue, there can be more serious issue. Let's discuss this in the next section.

Resiliency: the hidden cost of read-modify-write

Resiliency speaks of the ability of an app to recover state between sessions. We have seen what is necessary for a 512e storage device to perform a 512-byte sector write – the Read-Modify-Write cycle. Let's look at what would happen if the process of overwriting the previous physical sector on the media was interrupted. What would be the consequences?

- Because most hard disk drives update in place, the physical sector – that is, the portion of the media where the physical sector was located – could have been corrupted with incomplete info due to a partial overwrite. Put another way, you can think of it as potentially having lost all 8 logical sectors (which the physical sector logically contains).
- While most apps with a data store are designed with the capability to recover from media errors, the loss of eight sectors, or put another way, the loss of eight commit records, can potentially make it impossible for the data store to recover gracefully. An administrator may need to manually restore the database from a backup or may even need to perform a lengthy rebuild.
- One more important impact is that the act of another app causing a Read-Modify-Write cycle can potentially cause your data to be lost – even if your app is not running! This is simply because your data and the other app's data could be located within the same physical sector.

With this in mind, it is important that app software reevaluate any assumptions taken in the code, and be aware of the logical-physical sector size distinction, along with some interesting customer scenarios discussed later in this article.

Doing the right thing (avoiding read-modify-write)

While some storage vendors may be introducing some levels of mitigation within certain 512e storage devices to try to ease the performance and resiliency issues of the Read-Modify-Write cycle, there is only so much any mitigation can handle in terms of workload. As such, apps should not rely on this mitigation as a long-term solution. Moreover, there is no guarantee that all classes of disks will have this mitigation in place, nor is there a guarantee that the mitigation is well-designed.

The solution to this is not in-drive mitigation, but to design apps to do the right set of things to help support this type of media. This section discusses common scenarios where apps may have issues with large sector disks, and suggests an avenue of investigation to try and resolve each issue.

Issue 1: the partition is not aligned to a physical sector boundary

When the administrator/user partitions the disk, the first partition may not have been created on an aligned boundary. This may cause all subsequent writes to become unaligned to physical sector boundaries. As of Windows Vista SP1 and Windows Server 2008, the first partition is placed at the first 1024 KB of the disk (for disks 4GB or larger, otherwise the alignment is 64 KB) that is aligned to a 4 KB physical sector boundary. However, given the default partitioning in Windows XP, a 3rd party partitioning utility or incorrect usage of Windows APIs, created partitions may not be aligned to a physical sector boundary. Developers will need to ensure that the correct APIs are used to help ensure alignment. The recommended APIs to help ensure partition alignment are outlined below.

The `IVdsPack::CreateVolume` and `IVdsPack2::CreateVolume2` APIs do not use the specified alignment parameter when a new volume is created, but rather use the alignment value default for the operating system (Pre-Windows Vista SP1 will use 63 bytes, and post Windows Vista SP1 will use the defaults stated above). Instead, use the `IVdsCreatePartitionEx::CreatePartitionEx` or `IVdsAdvancedDisk::CreatePartition` APIs that use the specified alignment parameter for those apps that need to create partitions.

The best way to help ensure that alignment is correct is to do it right when initially creating the partition. Otherwise your app will need to take alignment into account when performing writes or at initialization – which can be a very complex process.

Issue 2: unbuffered writes not aligned to physical sector size

The simplest issue is that unbuffered writes are not aligned to the reported physical sector size of the storage media. Buffered writes, on the other hand, are aligned to the page size – 4 KB – which coincidently is the physical sector size of the first generation of large sector media. However, most apps with a data store perform unbuffered writes, and thus will need to ensure these writes are performed in units of the physical sector size.

Some examples of scenarios where the resulting app I/O is unaligned:

> **Commit records are padded to 512-byte sectors**: Apps with a data store typically have some form of commit record that either maintains info about metadata changes or maintains the structure of the data store. In order to ensure that the

loss of a sector does not affect multiple records, this commit record is typically padded out to a sector size. With a disk with a larger physical sector size, the app will need to query for the physical sector size as shown in the prior section, and ensure each commit record is padded to that size. With a 4K disk, this ensures I/Os do not fail. With a 512e disk, not only does this avoid the Read-Modify-Write cycle, it helps ensure that if a physical sector was lost, only one Commit Record would be lost.

Log files are written to in unaligned chunks: Unbuffered I/O is typically used when updating or appending to a log file. Apps can either switch to buffered I/O, or internally buffer the log updates to units of the physical sector size to avoid failed I/Os or triggering a Read-Modify-Write.

To help determine if your app issues unbuffered I/O, make sure to include the FILE_FLAG_NO_BUFFERING flag in the *dwFlagsAndAttributes* parameter when you are call the CreateFile function.

Moreover, if you are currently aligning the writes to the sector size, this sector size is most likely just the *logical* sector size, as most existing APIs that query for the sector size of the media just query the unit of addressing – that is, the logical sector size. The sector size of interest here is the physical sector size, which is the real unit of atomicity. Some examples of APIs that retrieve the logical sector size are:

- GetDiskFreeSpace, GetDiskFreeSpaceEx
- FileFsVolumeInformation
- IOCTL_DISK_GET_DRIVE_GEOMETRY, IOCTL_DISK_GET_DRIVE_GEOMETRY_EX
- IVdsDisk::GetProperties, IVdsDisk3::GetProperties2

Here's how you can query for the physical sector size:

Preferred method for Windows 8

With Windows 8, Microsoft has introduced a new API that enables developers to easily integrate 4K support within their apps. This new API supports even greater numbers of scenarios than the legacy method for Windows Vista and Windows 7 discussed below. This API enables these calling scenarios:

- Calling from an unprivileged app
- Calling to any valid file handle
- Calling to a file handle on a remote volume over SMB2
- Simplified programming model

The API is in the form of a new info class, FileFsSectorSizeInformation, with associated structure FILE_FS_SECTOR_SIZE_INFORMATION, defined as follows:

```
typedef struct _FILE_FS_SECTOR_SIZE_INFORMATION {
    ULONG LogicalBytesPerSector;
    ULONG PhysicalBytesPerSectorForAtomicity;
    ULONG PhysicalBytesPerSectorForPerformance;
    ULONG FileSystemEffectivePhysicalBytesPerSectorForAtomicity;
    ULONG Flags;
    ULONG ByteOffsetForSectorAlignment;
```

```
    ULONG ByteOffsetForPartitionAlignment;
} FILE_FS_SECTOR_SIZE_INFORMATION,
*PFILE_FS_SECTOR_SIZE_INFORMATION;
```

Legacy method for Windows 7 and Windows Vista

Windows Vista and Windows Server 2008 introduced APIs to query for the physical sector size of the attached storage device for AHCI-based storage controllers. With Windows 7 and Windows Server 2008 R2, as of SP1 (or Microsoft Knowledge Base 982018), this support is extended to Storport-based storage controllers. Microsoft has provided a code sample on MSDN detailing how an app can query for the physical sector size of the volume. The code sample is located at http://msdn.microsoft.com/library/ff800831(v=VS.85).aspx.

While the code sample above allows you to get the physical sector size of the volume, you should do some basic sanity checking of the reported physical sector size before using it, as it has been observed that some drivers may not return correctly formatted data:

- Make sure that the reported physical sector size is >= the reported logical sector size; if it is not, your app should use a physical sector size equal to the reported logical sector size
- Make sure that the reported physical sector size is a power of two; if it is not, your app should use a physical sector size equal to the reported logical sector size
- If the physical sector size is a power-of-two value between 512-bytes and 4 KB, you should consider using a physical sector size rounded down to the reported logical sector size
- If the physical sector size is a power-of-two value greater than 4 KB, you should evaluate your app's ability to handle this scenario before using that value; otherwise, you should consider using a physical sector size rounded down to 4 KB

Using this IOCTL to get the physical sector size does have several limitations. It:

- Requires elevated privilege; if your app is not running with privilege, you may need to write a Windows Service Application as noted above
- Does not support SMB volumes; you may also need to write a Windows Service Application to support physical sector size querying on these volumes
- Cannot be issued to any file handle (the IOCTL must be issued to a Volume Handle)

Issue 3: file formats relying on 512-byte sectors

Some apps with standard file formats (such as VHD 1.0) may have these files hard-coded to assume a 512-byte sector size. Thus, updates and writes to this file would result in a Read-Modify-Write cycle on the device– which will potentially result in performance and resiliency issues for your customers. However, there are ways for an app to provide support for operating on this type of media, for example:

- Use buffering to ensure that writes are performed in units of the physical sector size
- Implement an internal Read-Modify-Write that can help ensure that updates are performed in units of the reported physical sector size
- If possible, pad records out to a physical sector, in such a way that the padding would be interpreted as empty space
- Consider redesigning a version of the app data structure with support for larger sectors

Issue 4: the reported physical sector size can change between sessions

There are many scenarios where the reported physical sector size of the underlying storage that hosts the Datastor may change. The most common of these is when you migrate the Datastor to another volume, or even across the network. A change in the reported physical sector size may be an unexpected event for many apps and potentially can result in some apps failing to re-initialize.

This is not the easiest scenario to support, and is mentioned here as an advisory. You should consider the mobility requirements of your customers and adjust your support accordingly to help ensure customers are not negatively impacted by using 4K native or 512e media.

How a user can retrieve the logical and physical sector size for a volume

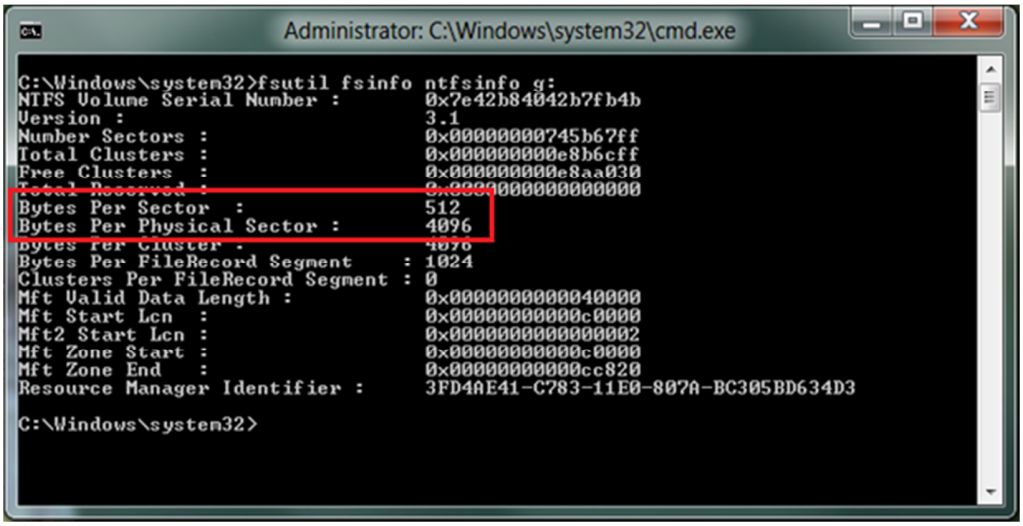
In-box with Windows is a utility to display the sector size info for a volume. Versions of Windows with supported “fsutil” are:

- Windows 8
- Windows Server 2012
- Windows 7 SP1 with Microsoft KB 982018
- Windows 7 with Microsoft KB 982018
- Windows Server 2008 R2 SP1 with Microsoft KB 982018 (v3)
- Windows Server 2008 R2 with Microsoft KB 982018 (v3)
- Windows Vista with Microsoft KB 2553708
- Windows Server 2008 with Microsoft KB 2553708

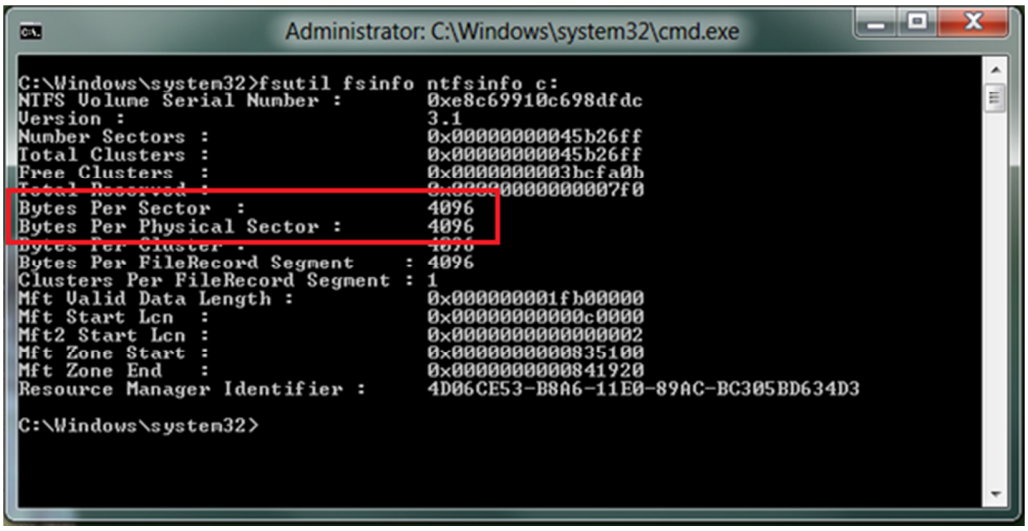
To get the sector size info, call the utility as follows from an elevated command prompt:

```
fsutil fsinfo ntfsinfo <drive letter>
```

A 4K Sector Disk with 512-byte Emulation has the “Bytes Per Sector” field set to 512 and the “Bytes Per Physical Sector” field set to 4096 as follows:



A 4K Native Disk has the “Bytes Per Sector” and “Bytes Per Physical Sector” fields both set to 4096 as follows:



A 512-byte Native Disk has the “Bytes Per Sector” and “Bytes Per Physical Sector” fields both set to 512.

Note: If the “Byte Per Physical Sector” field displays “Not Supported” then either the storage driver does not support IOCTL_STORAGE_QUERY_PROPERTY, or there was an error in retrieving the info.

Resources

Windows General Support Statement
http://support.microsoft.com/kb/2510009

Hotfix for Windows 7 and Windows Server 2008 R2
http://support.microsoft.com/kb/982018

Hotfix for Windows Vista and Windows Server 2008
http://support.microsoft.com/kb/2553708

HyperV Support Statement
http://support.microsoft.com/kb/2515143

General information about the IOCTL_STORAGE_QUERY_PROPERTY control code
http://msdn.microsoft.com/library/ff560590(VS.85).aspx

IOCTL_STORAGE_QUERY_PROPERTY Control Code:
http://msdn.microsoft.com/library/ff800830(VS.85).aspx

General information about the STORAGE_ACCESS_ALIGNMENT_DESCRIPTOR structure
http://msdn.microsoft.com/library/ff566344(VS.85).aspx

Description of the standard terminology used to describe Microsoft software updates
http://support.microsoft.com/kb/824684/

WDK sample code with details for how to extract the reported storage access alignment info from the STORAGE_ACCESS_ALIGNMENT_DESCRIPTOR structure when making a call to the IOCTL_STORAGE_QUERY_PROPERTY control code
http://msdn.microsoft.com/library/ff800831(v=VS.85).aspx

General information about ImageX Command-Line Options
http://technet.microsoft.com/library/dd799302(WS.10).aspx

Intel Chipset driver requirements to support 4 KB Sector Drives
http://www.intel.com/support/chipsets/imsm/sb/CS-031502.htm

Thin provisioning of logical units

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Windows Thin Provisioning is an interface to the end-to-end storage provisioning solution. To deliver a highly available, scalable and user-friendly storage provisioning service requires robust implementations from the server host to the storage target device. The Windows thin provisioning feature provides a synchronous view of the thin provisioning storage devices to the system administrator, IT manager, and storage administrator through the thinly provisioned logical unit's (LUN's) identification, threshold notification, resource exhaustion handling, space reclamation, and logical block addressing (LBA) state retrieval. The Windows thin provisioning feature presents a robust storage provisioning service model that can be applied to client-server storage systems, virtualization storage, and cloud storage services. Microsoft will ensure the high quality of the thin provisioning feature by enforcing the Thin Provisioning Logo program for storage array products.

The Windows Thin Provisioning storage solution:

- Allows storage administrators to create a larger LUN with fewer physical disk resources
- Adds or removes physical disk resource without interrupting the storage provisioning service
- Alerts users to the storage volume usage through the threshold setting
- Supports storage provisioning through the shared storage pool configuration
- Increases or decreases the size of the storage pool according to the demand and usage of storage space

Summary of the Windows Thin Provisioning features:

- Thin Provisioning LUN identification
- Handles for threshold notification, temporary resource exhaustion, and permanent resource exhaustion
- Storage space reclamation
- Mapping state queries of logical blocks

Manifestation

Without the proper understanding of the Windows handles for thin provisioning LUN, the app could fail with unexpected behavior or for an unknown cause.

Using thin provisioning LUNs

To use thinly provisioned LUNs in Windows 8 or Windows Server 2012, system and storage administrators should be aware of these matters:

- Thin provisioning LUN identification; system administrators or end users can use Defrag and the Storage Optimizer utility to identify the media type of the storage device
- When a threshold or a permanent resource exhaustion event is hit, Windows will log a system event to alert the system administrator
- Storage space reclamation can be performed manually or automatically by file delete notification or the scheduler of the storage optimizer
- Storage administrator should implement the threshold notification to alert system administrator or end user and avoid any unexpected storage service interruption

Tests

- Storage array must receive Windows 8 Thin Provisioning certificate to support Windows Thin Provisioning features
- Windows Thin Provisioning Hardware Certification Kit for storage array will be a useful test tool to verify storage array's capability for Windows Thin Provisioning feature support
- Windows expects the Thin Provisioning LUN and Full Provisioning LUN to work in the same system without any issue

Resources

T10 SCSI Block Command Spec (SBC3r27)
http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc3r27.pdf

Windows Thin Provisioning Logo Requirement
http://winqual.microsoft.com

Enhanced storage is now optional for WINPE and server SKU

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Enhanced storage was always available in Windows 7 USB devices. With the release of Windows 8, it is available for all storage devices. In client-based devices, it is enabled by default; in server devices it is optional and must be provisioned through WinPE.

Manifestation

The server device will fail if enhanced storage is not enabled.

Boot devices must be provisioned through WinPE with this enabled.

Solution

Because enhanced storage is optional for runtime server, developers must add it to WinPE for provisioning and activation of those drives. See the deployment guide for details.

Server administrators must then explicitly configure their server to use enhanced storage.

Virtual Disk Service is transitioning to Windows Storage Management API

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Beginning with Windows 8 and Windows Server 8, the Virtual Disk Service COM interface is superseded by the Storage Management API, a WMI-based programming interface. For managing storage subsystems, (Windows) disks, partitions, and volumes, we strongly recommend using the Storage Management API. For more info, see Windows Storage Management API.

For all usages except mirror boot volumes (using a mirror volume to host the operating system), dynamic disks are deprecated. For data that requires resiliency against drive failure, use Storage Spaces, a resilient storage virtualization solution. For more info, see Storage Spaces Technical Preview.

You can continue to use DiskPart, DiskRAID, and Disk Management during the deprecation period, but these tools will not work with Storage Spaces or with any other new WMIv2 based Windows Storage Management APIs or in-box storage management utilities or clients.

	APIs		Tools				
	VDS	**WMI**	**DiskPart**	**DiskRAID**	**Disk Mgmt GUI**	**PowerShell**	**Storage Spaces Control Panel**
Storage Subsystems	Yes	Yes	n/a	Yes	n/a	Yes	n/a
Basic Disks	Yes	Yes	Yes	n/a	Yes	Yes	No
Dynamic Disks	Yes	No	Yes	n/a	Yes	No	No
Storage Spaces	No	Yes	No	n/a	No	Yes	Yes

The result of this transition will be increased storage resiliency, availability, and scalability; a unified scripting and programming language, reduced storage management costs, and easier remote storage management.

Manifestation

The DiskPart and DiskRAID utilities used in the VDS environment do not support the new Storage Spaces. Similarly, the new Storage PowerShell utility does not support the deprecated Dynamic Disks.

Mitigation

Microsoft strongly recommends that you base any new storage management apps on the Windows Storage Management API, and that you plan to transition existing apps that are based on the VDS infrastructure to the Windows Storage Management API during your standard updating cycles.

Resources

Windows Storage Management API
http://msdn.microsoft.com/library/hh830613(v=vs.85).aspx

Storage Cmdlets in Windows PowerShell
http://technet.microsoft.com/library/hh848705.aspx

Windows Management Instrumentation
http://msdn.microsoft.com/library/aa394582(v=VS.85).aspx

Windows PowerShell
http://msdn.microsoft.com/library/dd835506(v=VS.85).aspx

Previous versions UI removed for local volumes

Platforms

Clients – Windows 8

Description

Earlier versions of Windows allowed users to restore previous versions of individual files from “shadow copies” of local volumes. These shadow copies are created by the “Previous Versions” feature through Volume Shadow Copy service (VSS). In Windows 7, users could configure and schedule shadow copies in System Protection UI available through System Properties.

However, Previous Versions were rarely used and negatively impacted the overall Windows performance; as a result the feature was removed. In Windows 8, these features are no longer available:

- The ability to browse, search or restore previous versions of files through Previous Versions UI
- The ability to configure or schedule previous versions of files through System Protection UI

However, users can still use the Previous Versions tab when accessing file shares on a Windows server.

Manifestation

The Previous Versions option is no longer available for local volumes in the Windows Explorer Properties menu.

Solution

Developers who need to create shadow copies of local volumes can still do so by calling VSS APIs in custom code.

StorAHCI replaces MSAHCI

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

StorAHCI, a Storport miniport, supports serial advanced technology attachment (SATA) advanced host controller interface (AHCI) controllers in Windows, and replaces MSAHCI, an ATAport miniport.

Manifestation

There should be no change in functionality or performance; this driver supports all the same devices that MSAHCI supports.

This change is transparent to the user.

Mitigation

Update any utilities and scripts that rely on ATAport bindings. Do not rely on the drive name. Instead use standard disk detection.

Windows 7 Backup and Restore deprecated

Platform

Clients – Windows 8

Description

While there is no behavioral change to Backup and Restore, this function is being deprecated and will not be updated. It was rarely used and its functionality has been replaced by the new File History feature. It will ship in Windows 8 and enthusiasts who upgraded from Windows 7 to Windows 8 or depend on Backup and Restore feature will be still able to use it. However, all access points to Backup and Restore, with the exception of the Control Panel, have been removed. The Control Panel applet used by Backup and Restore was renamed to Windows 7 File Recovery.

OEMs that were setting the registry key to disable backup notification in their images will no longer need to do that.

We do not recommend using both features at the same time. File History checks if Backup schedule exists and is active and if it finds one, it will not let users to turn it on. In this case, users who want to use File History will have to delete the Windows Backup schedule.

Manifestation

Workflows may be interrupted and documentation that refers to the previous method for accessing the Windows Backup and Restore feature will need to be updated to reflect these changes.

Mitigation

Apps that might trigger Backup and Restore should be rewritten to check if File History is on and let users choose their preferred method.

Offloaded data transfers

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

To advance the storage data movement, Microsoft has developed a new data transfer technology – offloaded data transfer (ODX). Instead of using buffered read and buffered write operations, Windows ODX starts the copy operation with an offload read and retrieves a token representing the data from the storage device, then uses an offload write command with the token to request data movement from the source disk to the destination disk. The copy manager of the storage devices performs the data movement according to the token. In the Windows 8, the IT manager and storage administrator are able to use the Windows ODX feature to interact with the storage device to move large files or data through the high-speed storage network. Windows ODX will significantly reduce client-server network traffic and CPU time usage during large data transfers because all the data movement is at the backend storage network. ODX can be used in virtual machine deployment, massive data migration, and tiered storage device support, and can lower the cost of physical hardware deployment through the ODX and thin provisioning storage features.

Note: This feature will only work on storage devices with SPC4 and SBC3 specification implementation.

Functional details

- The Windows ODX feature is embedded in the copy engine of the Windows operating system; during storage enumeration, Windows will query the ODX capability of the storage device
- Copy source storage device and copy destination storage device should be managed under the same copy manager for copy offload support
- If a copy offload operation fails, the storage device's copy manager must return the proper additional sense data for the apps' error handling
- The Windows copy engine will fall back to the traditional copy operation if the copy offload operation fails

Using ODX

- The data transfer app must ensure both copy source LUN and copy destination LUN are ODX capable before calling the ODX API routines
- In Windows explorer, users could use "drag and drop" or "copy and paste" to perform copy offload
- When the source LUN and destination LUN are mounted with the file system, the app must only call FSCTL_Offload_Read and FSCTL_Offload_Write to perform data transfer from the source LUN to the destination LUN

- If a copy offload operation fails, the storage device's copy manager must return the proper additional sense data for apps' error handling
- When the source LUN or destination LUN is not mounted with the file system and locked, the app must call IOCTL_STORAGE_MANAGE_DATA_SET_ATTRIBUTES with DeviceDsmAction_OffloadRead or DeviceDsmAction_OffloadWrite action to perform copy offload
- Storage management apps may use SCSI_PASS_THROUGH API to perform offloaded data transfers when both source and destination LUNs are not mounted with any file system and locked

Tests

- To ensure a robust user experience, verify the Windows ODX certification of the storage array
- The storage device must comply with Windows Offloaded Data Transfers certification (used to be Logo) requirements to support ODX feature
- Use the Windows Offloaded Data Transfers Hardware Certification Kit to verify the ODX feature support of the storage devices

Resources

T10 XCOPY Lite Spec (11-059r8)
http://www.t10.org/cgi-bin/ac.pl?t=f&f=spc4r35b.pdf
http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc3r30.pdf

Windows Offloaded Data Transfer Logo Requirement
http://winqual.microsoft.com

SCSI_PASS_THROUGH Structure
http://msdn.microsoft.com/library/ff565345(VS.85).aspx

Desktop Window Manager is always on

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

In Windows 8, Desktop Window Manager (DWM) is always ON and cannot be disabled by end users and apps. As in Windows 7, DWM is used to compose the desktop. In addition to experiences enabled in Windows 7, now DWM desktop composition enables desktop composition for all themes, support for Stereoscopic 3D, and management, separation, and protection of the experience with Windows Store apps.

Desktop composition for all themes

In Windows Vista and Windows 7, desktop composition is enabled only with the AERO Glass Theme. Hence users of Windows Classic and high contrast themes cannot use experiences enabled by desktop composition such as Windows Flip, automatic scaling for high resolution (DPI) scaling, thumbnail Preview and full screen magnifier. In addition, in these earlier versions of Windows, app developers must write and maintain multiple code paths – one where desktop composition is enabled and another where desktop composition is disabled.

With Windows 8, desktop composition is enabled for all themes. Users of Windows Classic and high contrast themes can use the experiences enabled by desktop composition such as Windows Flip, automatic scaling for high resolution (DPI) scaling, thumbnail previews, and full screen magnifier. In addition, developers don't need to write and maintain multiple code paths, thereby simplifying development.

Support for stereoscopic 3D

DWM desktop composition supports rendering and presentation of windowed and full-screen stereoscopic 3D app content.

Management, separation and protection of the experience with Windows Store apps

DWM desktop composition enables separation and protection of desktop app windows from the new Windows Store app windows by managing and separating the desktop app windows from the Windows Store app windows. Since desktop composition is responsible for managing all app windows, disabling desktop composition can result in unexpected behavior. In addition desktop composition is responsible for composing the new Start menu as well as additional window animations that form the core personality of the new Windows operating system.

Controlling desktop composition

In Windows Vista and Windows 7, desktop composition is disabled in a number of scenarios. In Windows 8, DWM desktop composition is a core operating system component and cannot be disabled. With a few exceptions, desktop composition is always on; it's started before the user logon and remains active for the duration of a

session. This section describes how Windows 8 treats the scenarios in Windows 7 where desktop composition is disabled.

Server SKU and certain client SKUs

In Windows 8, all server and client SKUs have desktop composition enabled. This ensures that server admins and users can benefit from the experiences enabled by desktop composition.

Fundamental requirements for desktop composition

Windows 8 ensures that graphics adapter and system color depth requirements are met through WDDM driver support and system color depth.

WDDM driver support

If a system does not have a WDDM-compliant graphics driver, Windows 8 uses Microsoft Basic Display Adapter as the default adapter. Since DWM always runs on the default adapter, it will choose Microsoft Basic Display Adapter to compose the desktop when a WDDM-compliant graphics driver is not available (whether not installed or disabled) on the system.

Microsoft Basic Display Adapter is a software rasterizer that uses the CPU rather than the GPU to perform all the drawing. Note that the performance of desktop composition on Microsoft Basic Display Adapter (especially animations) may not be as smooth as when running desktop composition on a GPU.

System color depth

Desktop Composition cannot run unless the color depth is set to 32 bits per pixel. In Windows 7 the color depth of the system can be changed in these scenarios:

- An end user uses the Windows Display control panel or a 3rd party control panel to change the system color
- An end user runs an app that changes the color depth of the system through a public API

Unlike Windows 7, Windows 8 does not support color depth other than 32 bits per pixel. The user can no longer change the color depth of the system by using the control panel.

In addition, app developers cannot use APIs to change the color depth of the system. Windows 8 will detect apps that try to change the color depth of the system to less than 32 bits per pixel, and inform the user that an app compatibility shim must be applied to run the apps. After confirmation from the user, the app compatibility shim is applied and the shim virtualizes the low color mode to the app while keeping the system running at 32 bits per pixel.

WinSAT

In Windows 8, desktop composition does not depend on WinSAT scores. Moreover, WinSAT no longer includes DWM assessment.

App compatibility and user action

In Windows 8:

- All of the options for disabling desktop composition that exist in Window 7 are removed
- Desktop composition is responsible for composing all themes
- Apps cannot use **DwmEnableComposition** to disable desktop composition. In order to maintain backward compatibility, a call to this API will return success; however, desktop composition is not disabled
- The "Disable desktop composition" shim is removed
- The option to "Disable desktop composition" from the compatibility tab of the Application Properties dialogue box is removed

An app uses a mirroring driver for remoting

In Windows 8:

- Does not support mirror drivers for remoting scenarios; while most of the existing apps that use mirror drivers should continue to work, due to the infrastructural change required to support existing mirror drivers in Windows 8 with DWM ON, some features or apps that use mirror drivers may not work
- Does support Desktop Duplication APIs for app developers who use mirror drivers for remoting scenarios.
- Does not support existing Accessibility Mirror Drivers
- Existing Mirror Drivers must be updated to ensure that they are compatible with Windows 8

Remote desktop connection

In Windows 8, desktop composition is always enabled for a remote desktop connection. A client computer connecting to a Windows 8 remote computer will always get desktop composition enabled for the remote desktop session irrespective of the Windows client version. Desktop composition is supported for multiple monitors on the client machine as well as for the remote app session.

In addition, when connecting to a Windows 8 remote computer, these settings in the Remote Desktop Connection client do not take effect:

- Color depth
- "Enable composition” checkbox

The color depth of the connection is always set to 32 bits per pixel and desktop composition is always enabled.

Direct2D rendering does not support rendering to "rich" metafiles in Internet Explorer 9

Platform

Clients – Windows Vista | Windows 7 | Windows 8

Servers – Windows Server 2008 | Windows Server 2008 R2 | Windows Server 2012

Description

Due to internal rendering changes in Internet Explorer 9, the IHTMLElementRender::DrawToDC and IViewObject::Draw APIs will now create a metafile containing a single bitmap that represents the web content instead of a rich metafile containing text and vector info. This change was due to the move from GDI rendering to hardware-accelerated Direct2D (D2D) rendering.

This change affects apps that use these APIs and rely on text or vector info in the metafile.

Manifestation

Depending on the app that is affected by this change, users might see broken or incorrect behavior in their apps.

Mitigation

Apps that only need to extract text info from a web document (without positioning info) can use the innerText property to extract text.

Apps that use IViewObject::Draw can use the FEATURE_IVIEWOBJECTDRAW_DMLT9_WITH_GDI feature control key to revert to GDI rendering if the document mode:

- Is less or equal to 8
- The FCK authorizes that host to use GDI
- And a metafile DC is passed to the API

Resources

IHTMLElementRender::DrawToDC
http://msdn.microsoft.com/library/aa752273(v=vs.85).aspx

IViewObject::Draw
http://msdn.microsoft.com/library/ms688655(v=vs.85).aspx

IHTMLElement::innerText Property
http://msdn.microsoft.com/library/aa752299(VS.85).aspx

Internet Feature Controls (I..L)
http://msdn.microsoft.com/library/ee330732(v=vs.85).aspx

Changes in DX9 legacy hardware support

Platform

Clients – Windows 8

Description

Intel and AMD/ATI no longer support their DX9 graphics parts and will not be updating drivers for these devices for Windows 8. Furthermore, these devices are not covered in-box for Windows 8. The last drivers for these devices are those available on WU and on the OEM/IHV's websites; many date from Vista, and many of these final-version drivers exhibit problems on Windows 8. In addition, nVidia has recently stated that they will not support their DX9 (Vista and older) mobile (notebook) parts for Win8. They continue to support their desktop DX9 parts.

All of these driver/part combinations are on the Internet Explorer 9 software fallback list. This means that for either performance or stability reasons, Internet Explorer 9 uses software rendering on these devices. This is a good indication that the experience with Windows Store apps will not be satisfactory on these drivers and parts.

Manifestation

As there is no in-box support for these devices, many users with these parts will wind up running on the Microsoft Basic Display Driver. This is a WARP-based WDDM 1.2 software GPU, and is actually faster than some of the parts in this class, for example, the Intel GMA500 series). It supports aero-glass and DWM, and can run Windows Store apps.

However, it has some important limitations:

- It doesn't support multiple monitors or projection
- It doesn't support sleep, though it does support hibernation
- It often will not allow changing screen resolution

Mitigation

If after testing, you find that the user experience is not acceptable, you may choose to set hardware requirements for their products that exclude these older DX9 Intel and AMD parts. You may also choose to alert your customers that they may have an unacceptable experience on these parts.

Tests

Evaluate the performance and experience on these devices:

- What is the experience like on the final driver version available?
- What is the experience like on the MSBDD?

MSAA is not available to Windows Store apps

Platform

Clients – Windows 8

Description

Windows 8 does not support MSAA (Microsoft Active Accessibility) for reading or automating accessible data from Windows Store apps. The UI Automation (UIA) API, available since Windows 7, should be used instead. Since there are no existing Windows Store app features, there are no existing implementations that are affected by this change. This affects only new apps and features written for Windows 8. Developers can still use MSAA to expose their accessibility data. If MSAA data is provided by a Windows Store app provider, UIA clients will still be able to read and automate the data.

To simplify the API for Windows 8 Windows Store apps, we decided to support only UI Automation as a client for these apps. UIA clients can read UIA providers natively, with no translation. UIA clients can read MSAA providers as translated through the UIA-to-MSAA Proxy. This will reduce the testing burden for Windows Store app developers.

Eliminating support for MSAA providers would further reduce the testing matrix, but there are major apps that are still MSAA-based and we do not want to render them inaccessible.

Manifestation

Windows Store app developers will not be able to access the details of MSAA within the app model.

Solution

No new automated cases should be authored in MSAA for features of Windows Store apps.

Switch any existing in-box tools that use MSAA to UIA if they need to interact with Windows Store apps. Windows Store app developers should use the UI Automation API.

Resources

UI Automation Fundamentals
http://msdn.microsoft.com/library/windows/desktop/ee684009(v=vs.85).aspx

Port 3 is deprecated for NDIS 6.30 drivers

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Windows 8 removes platform support for NDIS miniport WLAN drivers to create or start up the IHV extensibility port (also known as 3rd port). This feature was introduced in Windows 7 as a stopgap measure while the Wi-Fi Alliance (WFA) developed the Wi-Fi Direct specification. The specification is now finished, products using Wi-Fi Direct are being certified, and Windows 8 introduces a native Wi-Fi Direct stack.

Manifestation

Nothing, as 3rd port is a platform capability that WLAN miniport drivers start up if they need this functionality.

Solution

We are keeping the extensibility port feature available for existing drivers that do not report NDIS 6.30 to maintain device compatibility. However, new WLAN drivers that report NDIS 6.30 will not be able to start this port; instead, developers of these drivers should use Wi-Fi Direct.

New features and enhancements

This section of the document describes enhancements and new features that can improve both the users' and the developers' experience. A few of them, however, have impacts that developers need to be aware of regarding their existing and new products. Below is the list of new features and enhancements, grouped by feature area:

Security

- Early launch antimalware
- Secure boot feature signing requirements for kernel-mode drivers
- Measured boot

Performance

- Startup apps
- Automatic Maintenance

Usability

- Third-party input method editors

Storage and file system

- New API allows apps to send "TRIM and Unmap" hints to storage media
- Multipath I/O now supports extended storage request blocks
- Resilient file systems
- File server API support
- New File History feature
- Operating system now controls power to optical disk drives

USB

- Support for USB 3.0

Early launch antimalware

Platforms

Clients - Windows 8

Servers - Windows Server 2012

Description

As antimalware (AM) software has become better and better at detecting runtime malware, attackers are also becoming better at creating rootkits that can hide from detection. Detecting malware that starts early in the boot cycle is a challenge that most AM vendors address diligently. Typically, they create system hacks that are not supported by the host operating system and can actually result in placing the computer in an unstable state. Up to this point, Windows has not provided a good way for AM to detect and resolve these early boot threats.

Windows 8 introduces a new feature called Secure Boot, which protects the Windows boot configuration and components, and loads an Early Launch Anti-malware (ELAM) driver. This driver starts before other boot-start drivers and enables the evaluation of those drivers and helps the Windows kernel decide whether they should be initialized.

Manifestation

By being launched first by the kernel, ELAM is ensured to be launched before any third-party software, and is therefore able to detect malware in the boot process and prevent it from initializing.

Mitigation

Boot drivers are initialized based on the classification that is returned from the ELAM driver according to an initialization policy. By default, the policy initializes known good and unknown drivers, but will not initialize known bad drivers. A system administrator can specify a custom policy through Group Policy that can prevent unknown drivers from initializing or can enable drivers that are critical to the boot process, but have been tampered with, boot to be initialized.

Solution

An ELAM driver must register for kernel callbacks to get info about each boot-start driver as it is initializing. The ELAM driver can then return a classification for each driver. These functions are required:

IoRegisterBootDriverCallback() and IoUnRegisterBootDriverCallback()

An ELAM driver can also register for registry callbacks. Doing so enables the ELAM driver to inspect the configuration data that is used by each boot-start driver. The ELAM driver can then block or modify the data before it is used by the boot-start drivers, if necessary. These functions are required:

CmRegisterCallbackEx() and CmUnRegisterCallback()

A more detailed description of ELAM driver requirements and API usage is documented in a white paper on MSDN called Early Launch Antimalware.

Tests

ELAM drivers must be specially signed by Microsoft to ensure they are started by the Windows kernel early in the boot process. To get the signature, ELAM drivers must pass a set of certification tests to verify performance and other behavior. These tests are included in the Windows Hardware Certification Kit.

Resources

Early Launch Antimalware Whitepaper
http://msdn.microsoft.com/library/windows/hardware/br259096

CmRegisterCallbackEx()
http://msdn.microsoft.com/library/windows/hardware/ff541921(v=vs.85).aspx

CmUnRegisterCallback()
http://msdn.microsoft.com/library/windows/hardware/ff541928(v=vs.85).aspx

IoRegisterBootDriverCallback()
http://msdn.microsoft.com/library/windows/hardware/hh439379(v=VS.85)

IoUnRegisterBootDriverCallback()
http://msdn.microsoft.com/library/windows/hardware/hh439394(v=VS.85).aspx

Certifying hardware with the Windows Hardware Certification Kit Build Conference presentation
http://channel9.msdn.com/events/BUILD/BUILD2011/HW-659T

Windows Developer Preview Kits and Tools
http://msdn.microsoft.com/windows/hardware/br259105

Secure boot feature signing requirements for kernel-mode drivers

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

In Windows 8 and Windows Server 2012, including WinPE, the kernel has been locked down to prevent malware introduced by boot or root kits from circumventing Windows operating system security requirements for signed drivers.

This change affects all kernel-mode drivers for devices that support unified extensible firmware interface (UEFI) Secure Boot; UEFI Secure Boot is required for Windows 8 certification for client machines, and optional for servers. It does not affect user-mode drivers.

Standard development for kernel-mode drivers involves either the use of kernel mode debugging or the boot configuration data (BCD) <testsigning> setting. Both of these are disabled when Secured Boot is on.

Manifestation

Kernel-mode drivers will not run if they are not properly signed by a trusted certification authority (CA). The operating system will not allow untrusted drivers to run and standard mechanisms like kernel debugging and testsigning will not be permitted.

Mitigation

To test drivers, you must disable Secure Boot in BIOS as well as meet the other test-signing requirements (see below).

When distributing drivers more widely they must be properly signed by a trusted CA.

Resources

Signing Drivers
http://msdn.microsoft.com/en-us/windows/hardware/gg487317.aspx

Measured Boot

Platforms

Clients - Windows 8

Servers - Windows Server 2012

Description

As antimalware (AM) software has become better and better at detecting runtime malware, attackers are also becoming better at creating rootkits that can hide from detection. Detecting malware that starts early in the boot cycle is a challenge that most AM vendors address diligently. Typically, they create system hacks that are not supported by the host operating system and can actually result in placing the computer in an unstable state. Up to this point, Windows has not provided a good way for AM to detect and resolve these early boot threats.

Windows 8 introduces a new feature called Measured Boot, which measures each component, from firmware up through the boot start drivers, stores those measurements in the Trusted Platform Module (TPM) on the machine, and then makes available a log that can be tested remotely to verify the boot state of the client.

Manifestation

The Measured Boot feature provides AM software with a trusted (resistant to spoofing and tampering) log of all boot components that started before AM software. AM software can use the log to determine whether components that ran before it are trustworthy or if they are infected with malware. The AM software on the local machine can send the log to a remote sever for evaluation. The remote server may initiate remediation actions either by interacting with software on the client or through out-of-band mechanisms, as appropriate.

Mitigation

In enterprise scenarios, the system administrator has control of how Measured Boot info is used. In end-user scenarios for example, online banking), the consumer must opt in to use Measured Boot for the specific service.

Solution

A white paper is being prepared that details the APIs and function calls that must be made for various Measured Boot scenarios.

Startup apps

Platform

Clients – Windows 8

Description

One of the big bets with Windows is the ability to light up various form factors, from the traditional desktops and laptops to low-powered tablets. To ensure that our mutual customers have a great experience on any form factor they choose with Windows, two key success metrics that need to be addressed are increased battery life and excellent PC responsiveness. In order to achieve these, improvements have been made in multiple areas of Windows including process life cycle, sleep states and startup apps (apps with automated launch after machine boot). This topic highlights some of the impacts that startup apps have on a Windows device, and provides guidance to developers (ISV/IHV) and OEMs to rethink the usage patterns of startup apps in order to improve battery life and responsiveness for our mutual customers. It also describes the changes in Windows that put users in control of determining which of the startup apps actually get to execute.

Windows Store apps are designed to adhere to new standards of battery consumption and responsiveness, and Windows manages their life cycle by suspending and/or terminating them. However, desktop apps designed for prior Windows versions have not necessarily been designed to preserve battery life or be sensitive to user activity, and can affect system responsiveness (for example, when an app sends a regular 1-second heartbeat to check for updates, or pre-allocates memory upfront in case it needs it later). This can create a poor experience for the user who buys a Windows tablet PC with a long battery life expectancy and weeks of standby, but discovers the tablet's battery life does not achieve these goals. Also, since startup apps run in the background, the number of apps running on the system can be significantly more than what the user is aware of and affect system responsiveness. Startup apps are classified to include those leveraging these mechanisms to start:

- *Run* registry keys (HKLM, HKCU, wow64 nodes included)
- *RunOnce* registry keys
- Startup folders under the start menu for per user and public locations

New functionality has been added to Windows to ensure that end users are always in control of the apps that run on their systems. The Startup tab in Task Manager shows a list of startup apps, along with controls that allow users to disable startup apps.

To help users determine what to disable, Task Manager displays a measure of each startup app's impact. Impact is assessed based on an app's CPU and disk usage at startup. Impact values are determined by applying these criteria:

- **High impact** – Apps that use more than 1 second of CPU time or more than 3 MB of disk I/O at startup

- **Medium impact** – Apps that use 300 ms - 1000 ms of CPU time or 300 KB - 3 MB of disk I/O
- **Low impact** – Apps that use less than 300 ms of CPU time and less than 300 KB of disk I/O

Microsoft provides tools to help app developers assess, analyze and take steps to reduce their startup impact and improve the user experience. The Assessment and Deployment Kit provides the ability to run a boot performance assessment and measure the impact of apps that run at startup. The assessment results contain detailed analysis and remediation info where applicable, for the top-impacting components at Windows startup. Using the Windows Performance Analyzer, app developers can perform deep analysis to find the root cause of the performance impact and improve Windows startup performance. Install the Windows ADK from here.

Guidance

Startup apps span multiple categories as described in the table below. A set of recommendations for developers is mapped to the categories of startup apps to align with the Windows functional changes described above.

Startup Apps Categories		Description	Recommendation
Updaters		Monitor and notify users for online updates	**Maintenance Task** Note: All updates should be maintenance tasks, without any UI interaction requirements. Apps should just update themselves quietly, and rollback on failure
Hardware Assistance	Alternative Access Points	Provide access to Windows features and apps that are accessible via other access points in Windows	**Remove** Note: The key is to reduce duplicate features that exist in Windows
	Notifiers	Provide users with notifications regarding their devices	**Remove** Note: Windows provides Windows 8 notifications to users about their devices
Pre-launchers		Some of preliminary activities needed by apps is offloaded to a startup app during user login	**Remove** Note: Windows 8 is optimized for a fast experience for app launches
Utility	PC Sync	Provide sync functionality across multiple systems	**Startup** (Potential updates in Beta)
	Backup & Recovery	Entry point to save and restore the state of files, settings, or entire systems	**Windows Store app for interfacing with the users**
	Telemetry	Collect and send info about the user experience and environment	**Maintenance Task**
	PC Monitoring	Provide unsolicited system state monitoring and notifications that duplicate existing inbox functionality	**Remove** Note: The key is to reduce duplicate features that exist in Windows

Startup Apps Categories		Description	Recommendation
Security	Parental Control & Filters	Enforce rules and restrictions established for Internet access and usage	**Startup**
	Configuration & Management	Allow users to control diagnostic and remediation options for system security monitoring Notify users of findings and security actions	**Windows Store app for interfacing with the users**
Communication & Internet (IM & VoIP)		Send and receive messages and calls	**Windows Store app**
Music & MP3		Play, store, and manage music	**Windows Store app**
Photo & Video		Detect, record, render, store and manage photos and videos	**Windows Store app**
PC Gaming		Launch games across various domains	**Windows Store app**
Upsell & Advertisement		Draw attention to goods and services available for purchase	**Remove**

Note: Accessibility apps guidelines are covered by separate direct engagements with ISVs. See *Programming for Ease of Access* for details: http://msdn.microsoft.com/en-us/windows/bb879984.aspx.

Windows Store app

Windows Store apps enhance the user experience by introducing a Windows space with new coordinates: a new app model, a new user interface, and a Windows Store. These language and presentation framework options are available for developing Windows Store apps:

- HTML/JavaScript/CSS
- XAML/C#
- XAML/C++

Aggregated info for developing Windows Store apps is available at the Windows Dev Center: http://dev.windows.com and http://go.microsoft.com/fwlink/?LinkId=221445

Examples:

Developing Windows Store Games: http://go.microsoft.com/fwlink/?LinkId=228452

Developing Windows Store apps that use Media: http://go.microsoft.com/fwlink/?LinkId=228453

Automatic Maintenance tasks

Periodic background activity should be designed as Automatic Maintenance tasks. These are scheduled at system idle time to increase the responsiveness and energy efficiency of Windows PCs. Maintenance tasks can be created and configured by a desktop app at install time, using the desktop SDK. See the Automatic Maintenance topic that follows for details.

Resources

Accessibility Guidelines
http://msdn.microsoft.com/windows/bb879984.aspx

Windows ADK
http://go.microsoft.com/fwlink/?LinkId=234980

Windows Dev Center
http://go.microsoft.com/fwlink/?LinkId=221445

Automatic Maintenance

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

Windows depends on execution of inbox and third party maintenance activity for much of its value-add, including Windows Update, and automatic disk defragmentation, as well as antivirus updates and scans. Additionally, enterprises frequently use maintenance activity such as Network Access Protection (NAP) scanning to help enforce security standards on all enterprise workstations.

Maintenance activity in Windows is designed to run in the background with limited user interaction and minimal impact to performance and energy efficiency. However, in Windows 7 and earlier versions, performance and energy efficiency are still impacted due to the non-deterministic and widely varied schedule of the multiple maintenance activities in Windows. Responsiveness to users is reduced when maintenance activity runs while users are actively using the computer. Apps also frequently ask the user to update their software and run background maintenance, and direct users to multiple experiences, including Action Center, Control Panel, Windows Update, Task Scheduler MMC snap-in, and third-party controls.

The goal of Automatic Maintenance is to combine all background maintenance activity in Windows and help third-party developers add their maintenance activity to Windows without negatively impacting performance and energy efficiency. Additionally, Automatic Maintenance enables users as well as enterprises to be in control of maintenance activity scheduling and configuration.

Key problems

Automatic Maintenance is designed to address these problems with maintenance activity in Windows:

- Deadline scheduling
- Resource utilization conflicts
- Energy efficiency
- Transparency to the user

Functionality

Automatic Maintenance facilitates idle efficiency and permits all activity to run in a timely and prioritized fashion. It also helps enable unified visibility and control over maintenance activity, and allows third-party developers to add their maintenance activity to Windows without negatively impacting performance and energy efficiency. To do this, it provides a fully automatic mode, user-initiated mode, automatic stop, deadlines and notification, and enterprise control. These are each described below.

Fully automatic mode

This default mode enables intelligent scheduling during PC idle-time and at scheduled times—the execution and automatic pausing of maintenance activity without any user intervention. The user can set a weekly or daily schedule. All maintenance activity is non-interactive and executes silently.

The computer automatically is resumed from sleep when the system is not likely to be in use, respecting the Power Management policy, which in the case of laptops, defaults to allow wake-up only if it is on AC power. Full system resources at high power are used to complete the maintenance activity as quickly as possible. If the system was resumed from sleep for Automatic Maintenance, it is requested to return to sleep.

Any required user interactions related to activities such as configuration are performed outside of Automatic Maintenance execution.

User-initiated mode

If users need to prepare for travel, expect to be on battery power for a prolonged time, or wish to optimize for performance and responsiveness, they have the option of initiating Automatic Maintenance on demand. Users can configure Automatic Maintenance attributes, including the automatic run schedule. They can view the current status of Automatic Maintenance execution, and they can stop Automatic Maintenance if needed.

Automatic stop

Automatic Maintenance automatically stops currently running maintenance activities if the user starts interacting with the computer. Maintenance activity will resume when the system returns to idle status.

Note: All activities in Automatic Maintenance must support stopping in 2 seconds or less. The user should be notified that the activity has been stopped.

Deadlines and notification

Critical maintenance activity must execute within a pre-defined time window. If critical tasks have not been able to run within their designated time, Automatic Maintenance will automatically start executing at the next available system idle opportunity. However, if the task state remains behind deadline, Automatic Maintenance will notify the user about the activity and provide an option for a manual run of Automatic Maintenance. All tasks scheduled for maintenance will run, although the most starved tasks take precedence. This activity may impact system responsiveness and performance; therefore, Automatic Maintenance will notify the user that critical maintenance activity is executing.

Enterprise control

Enterprise IT professionals should be able to determine when Automatic Maintenance executes on their Windows systems, enforce that schedule via standardized management interfaces, and retrieve event data about the status of Automatic Maintenance execution attempts. Additionally, IT professionals should be able to invoke specific Automatic Maintenance activity remotely via standard management interfaces. Each time Automatic Maintenance executes, status reporting, including notifications when Automatic Maintenance could not be

executed because the user has manually paused the activity, runs. IT professionals should consider moving logon scripts to Automatic Maintenance to help make the user's logon experience quicker.

Creating an Automatic Maintenance task

This section details how developers can create a task using a task definition in XML or C language. Keep in mind that maintenance activity should not launch any user interface that requires user interaction, as Automatic Maintenance is completely silent and runs when the user is not present. Indeed, if the user interacts with the computer during Automatic Maintenance, any tasks in process will be ended until the next idle period.

Using XML

Task Scheduler includes a built-in command-line tool, schtasks.exe, that can import a task definition in XML format. The schema for task definition is documented at http://msdn.microsoft.com/library/aa383609(v=VS.85).aspx. Below is an example of an Automatic Maintenance task defined in XML.

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.4"
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2011-07-01T11:34:31</Date>
    <Author>IT Deptartment</Author>
  </RegistrationInfo>
  <Principals>
    <Principal id="Author">
      <RunLevel>LeastPrivilege</RunLevel>
      <GroupId>NT AUTHORITY\SYSTEM</GroupId>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>false</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <MaintenanceSettings>
      <Period>P2D</Period>
      <Deadline>P14D</Deadline>
    </MaintenanceSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>

<DisallowStartOnRemoteAppSession>false</DisallowStartOnRemoteAppSes
sion>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>P3D</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>cmd</Command>
      <Arguments>/c timeout -t 60</Arguments>
```

```
    </Exec>
  </Actions>
</Task>
```

To save the task on a Windows computer, save the above XML as a text file and use this command line:

```
Schtasks.exe /create /tn <task name> /xml <text file name>
```

Using C

An Automatic Maintenance task also can be created using C code. Below is a code sample that can be used to configure a task's Automatic Maintenance settings:

```
/*****************************************************************
**
This sample creates a maintenance task to start cmd window during
maintenance opportunities with periodicity of 2 days and deadline
Of 14 days.
*****************************************************************
*/

#define _WIN32_DCOM

#include <windows.h>
#include <iostream>
#include <stdio.h>
#include <comdef.h>
#include <wincred.h>
//  Include the task header file.
#include <taskschd.h>
//#pragma comment(lib, "taskschd.lib")
//#pragma comment(lib, "comsupp.lib")

int __cdecl
MainteanceTask( )
{
    //  ------------------------------------------------------
    //  Initialize COM.
    HRESULT hr;

    //  ------------------------------------------------------
    //  Create a name for the task.
    LPCWSTR wszTaskName = L"MaintenanceTask";

    ITaskService *pService = NULL;
    ITaskFolder *pRootFolder = NULL;
    ITaskDefinition *pTask = NULL;
    ITaskSettings *pSettings = NULL;
    IRegistrationInfo *pRegInfo= NULL;
    IPrincipal *pPrincipal = NULL;
    ITaskSettings3 *pSettings3 = NULL;
    IMaintenanceSettings* pMaintenanceSettings = NULL;
    IActionCollection *pActionCollection = NULL;
    IAction *pAction = NULL;
    IExecAction *pExecAction = NULL;
    IRegisteredTask *pRegisteredTask = NULL;

    wprintf(L"\nCreate Maintenance Task %ws", wszTaskName );

    hr = CoInitializeEx( NULL, COINIT_MULTITHREADED);
    if( FAILED(hr) )
    {
```

```
        wprintf(L"\nCoInitializeEx failed: %x", hr );
        return 1;
    }

    //  Set general COM security levels.
    hr = CoInitializeSecurity( NULL,
        -1,
        NULL,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        0,
        NULL);

    if( FAILED(hr) )
    {
        wprintf(L"\nCoInitializeSecurity failed: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Create an instance of the Task Service.
    hr = CoCreateInstance( CLSID_TaskScheduler,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_ITaskService,
                           (void**)&pService );
    if (FAILED(hr))
    {
        wprintf(L"\nFailed to create an instance of ITaskService:
%x", hr);
        goto CleanUp;
    }

    //  Connect to the task service.
    hr = pService->Connect(_variant_t(), _variant_t(),
_variant_t(), _variant_t());
    if( FAILED(hr) )
    {
        wprintf(L"\nITaskService::Connect failed: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Get the pointer to the root task folder.  This folder will
hold the
    //  new task that is registered.
    hr = pService->GetFolder( _bstr_t( L"\\") , &pRootFolder );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot get Root folder pointer: %x", hr );
        goto CleanUp;
    }

    //  If the same task exists, remove it.
    ( void ) pRootFolder->DeleteTask( _bstr_t(wszTaskName), 0  );

    //  Create the task definition object to create the task.
    hr = pService->NewTask( 0, &pTask );
    if (FAILED(hr))
    {
```

```
        wprintf(L"\nFailed to CoCreate an instance of the
TaskService class: %x", hr);
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Get the registration info for setting the identification.
    hr = pTask->get_RegistrationInfo( &pRegInfo );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot get identification pointer: %x", hr );
        goto CleanUp;
    }

    hr = pRegInfo->put_Author( _bstr_t(L"Author Name") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put identification info: %x", hr );
        goto CleanUp;
    }

    // The task needs to grant explicit FRFX to LOCAL SERVICE
(A;;FRFX;;;LS)
    hr = pRegInfo->put_SecurityDescriptor(
_variant_t(L"D:P(A;;FA;;;BA)(A;;FA;;;SY)(A;;FRFX;;;LS)") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put security descriptor: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Create the principal for the task - these credentials
    //  are overwritten with the credentials passed to
RegisterTaskDefinition
    hr = pTask->get_Principal( &pPrincipal );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot get principal pointer: %x", hr );
        goto CleanUp;
    }

    //  Set up principal logon type to interactive logon
    hr = pPrincipal->put_LogonType( TASK_LOGON_INTERACTIVE_TOKEN );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put principal info: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Create the settings for the task
    hr = pTask->get_Settings( &pSettings );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot get settings pointer: %x", hr );
        goto CleanUp;
    }

    hr = pSettings->QueryInterface( __uuidof(ITaskSettings3),
(void**) &pSettings3 );
    if( FAILED(hr) )
```

```
    {
        wprintf(L"\nCannot query ITaskSettings3 interface: %x", hr
);
        goto CleanUp;
    }

    hr = pSettings3->put_UseUnifiedSchedulingEngine( VARIANT_TRUE
);
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put_UseUnifiedSchedulingEngine: %x", hr
);
        goto CleanUp;
    }

    hr = pSettings3->CreateMaintenanceSettings(
&pMaintenanceSettings );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot CreateMaintenanceSettings: %x", hr );
        goto CleanUp;
    }

    hr = pMaintenanceSettings->put_Period ( _bstr_t(L"P2D") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put_Period: %x", hr );
        goto CleanUp;
    }

    hr = pMaintenanceSettings->put_Deadline ( _bstr_t(L"P14D") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put_Period: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Add an action to the task. This task will execute
notepad.exe.
    //  Get the task action collection pointer.
    hr = pTask->get_Actions( &pActionCollection );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot get Task collection pointer: %x", hr );
        goto CleanUp;
    }

    //  Create the action, specifying that it is an executable
action.
    hr = pActionCollection->Create( TASK_ACTION_EXEC, &pAction );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot create the action: %x", hr );
        goto CleanUp;
    }

    //  QI for the executable task pointer.
    hr = pAction->QueryInterface( IID_IExecAction, (void**)
&pExecAction );
    if( FAILED(hr) )
    {
```

```
        wprintf(L"\nQueryInterface call failed for IExecAction:
%x", hr );
        goto CleanUp;
    }

    //  Set the path of the executable to notepad.exe.
    hr = pExecAction->put_Path( _bstr_t(L"cmd") );
    if( FAILED(hr) )
    {
        wprintf(L"\nCannot put action path: %x", hr );
        goto CleanUp;
    }

    //  ------------------------------------------------------
    //  Save the task in the root folder.
    hr = pRootFolder->RegisterTaskDefinition(
            _bstr_t(wszTaskName),
            pTask,
            TASK_CREATE_OR_UPDATE,
            _variant_t(),
            _variant_t(),
            TASK_LOGON_INTERACTIVE_TOKEN,
            _variant_t(L""),
            &pRegisteredTask);
    if( FAILED(hr) )
    {
        wprintf(L"\nError saving the Task : %x", hr );
        goto CleanUp;
    }

    wprintf(L"\nSuccess!\n----------------------------------" );

CleanUp:

    if ( pService != NULL ) pService->Release();
    if ( pRootFolder != NULL ) pRootFolder->Release();
    if ( pTask != NULL ) pTask->Release();
    if ( pSettings != NULL ) pSettings->Release();
    if ( pRegInfo != NULL ) pRegInfo->Release();
    if ( pPrincipal != NULL ) pPrincipal->Release();
    if ( pSettings3 != NULL ) pSettings3->Release();
    if ( pMaintenanceSettings != NULL ) pMaintenanceSettings-
>Release();
    if ( pActionCollection != NULL ) pActionCollection->Release();
    if ( pAction != NULL ) pAction->Release();
    if ( pExecAction != NULL ) pExecAction->Release();
    if ( pRegisteredTask != NULL ) pRegisteredTask->Release();

    CoUninitialize();
    return SUCCEEDED ( hr ) ? 0 : 1;
}
```

Validating tasks

Validate that the task has been created successfully and runs as part of maintenance.

Validating task creation

Use this command line to export the task definition to a file and ensure that the task definition is as expected:

```
Schtasks.exe /Query /tn<task name> /xml <text file name>
```

Validating task execution

Run this command line to launch the task and validate that the Task Scheduler UI (taskschd.msc) shows the task has run:

```
Schtasks.exe /Run /tn<task name>
```

Resources

Task Schedule

http://msdn.microsoft.com/library/bb756979.aspx

Third-party input method editors

Platforms

Clients - Windows 8
Servers - Windows Server 2012

Description

Input method editors (IMEs) are software components that allow a user to type text in a language that has more characters than can be represented on a keyboard. (This is common, but not limited to, East Asian languages.) Instead of each single character appearing on a single key, users type combinations of keys that are then interpreted by the IME. The IME generates the character that matches the set of key strokes, sometimes presenting the user with a list of possible characters to pick from, and then inserts the character into the edit control window of the user's app.

In the past, Windows has allowed third-party IMEs to run in the Windows system, and this capability continues for Windows 8. Users can install a third-party IME and use it. In addition, we are hardening the system and processes to prevent malicious IMEs, improve security, and enhance the user experience.

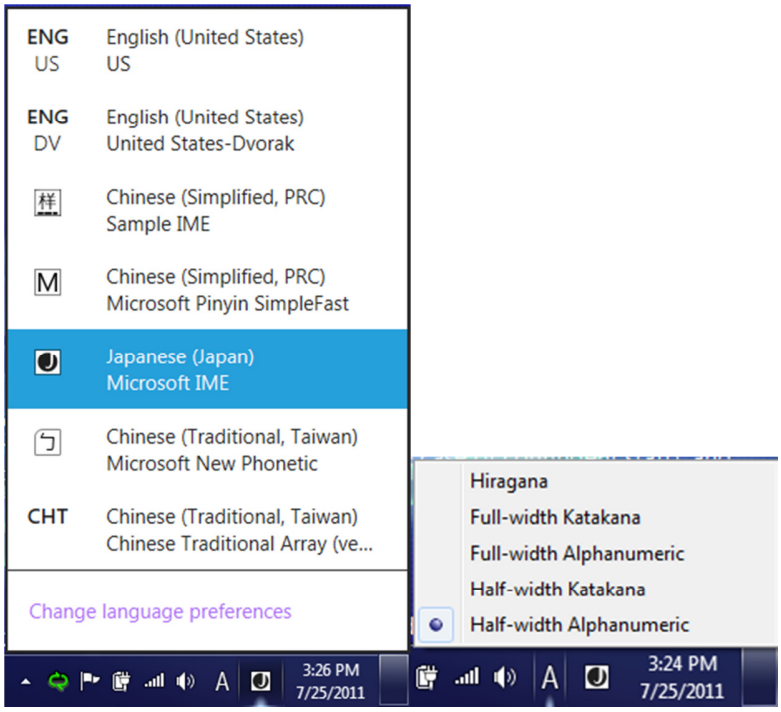
In Windows 8, you will find:

- Third-party IME support for both hardware keyboards and touch keyboards
- Third-party IME vendors must follow Microsoft guidelines to develop their IMEs for Windows 8
- Third-party IMEs must be digitally signed
- Third-party IMEs must be Text Services Framework (TSF) aware, and proper IME flags must be set to run correctly in Windows 8
- Legacy third-party IMEs will be able to run in desktop apps, but will be blocked in Windows Store apps
- Third-party IMEs can use the touch keyboard layout provided by Windows to link their IME, so that users can use their IME with touch keyboards. However, certain functions of in-box IMEs for touch keyboards will not be available to third-party IMEs
- Windows Defender will remove malicious IMEs from the Windows system

Manifestation

Input language and input method switch changes

Instead of showing all the IME mode icons along with the IME branding icon, only one IME mode icon along with the IME branding icon is shown. If you click the IME **branding** icon, you can switch input methods. If you click the IME **mode** icon, you can switch to a different IME mode.



Windows 8 input flyout and the IME flyout, showing Japanese IME as the current input method

If an IME relies on the language bar to show its mode icons in Windows 7, the IME must be changed in order to show its branding icon and mode icon in the input indicator in Windows 8.

Note: The details about how an IME can show its branding icon and mode icon in the SysTray on the desktop Taskbar will be documented and posted publicly in the Windows 8 IME Guidelines.

New Windows environment

The environment in Windows 8 changes the landscape for IMEs. The concepts of Windows Store apps, local context app containers, and API restrictions on IMEs were not present in Windows 7. Some existing Windows 7 IMEs stop responding when run inside a Windows Store app and therefore don't allow legacy IMEs to run inside Windows Store apps. Additionally, make sure that new versions of IMEs are validated to ensure that they are compatible with the new UI environment before they are run inside Windows Store apps.

Mitigation

You can use a desktop-compatible IME on the system. This might be your best option if you mainly use desktop apps, and you want to continue to use a preferred legacy IME for input. We recommend that you use a Windows 8 IME, and stop using legacy/non-certified IMEs. Notifications are provided by both the Language CPL and the Input Switch to warn you of the effects of using a desktop-compatible IME.

You will see either of the below behaviors if a desktop IME doesn't work across your system:

- The language CPL UI labels desktop-compatible IMEs, and displays a message that desktop-compatible IMEs work only in desktop apps.
- The input flyout greys out desktop-compatible IMEs when the user is inside Windows Store apps. This indicates that the IME does not work in this app. (In the desktop, desktop-compatible IMEs are not greyed out). If you switch to Window Store apps with a desktop-compatible IME and realize the IME is off, use the input indicator to change to an IME that is compatible with Windows Store apps.

Legacy or desktop-compatible IMEs are limited to these conditions:

- Upgrading from Windows 7 to Windows 8, with third-party IMEs on the system
- The vendor has not released a version compatible with Windows 8, and the user tries to use an existing Windows 7 version in the meantime

Solution

General

Use the existing text services framework (TSF) infrastructure to implement your IME logic and the Windows Store app common controls for your UIs. Create owned windows to host your UI.

New search APIs are being added to improve search prediction and provide a cleaner search experience in the UI.

APIs are also being added to notify third-party IMEs when a touch keyboard is invoked to protect the UI from being covered by the touch keyboard. A default classic touch keyboard layout is automatically loaded for third-party IMEs. No additional work is needed to integrate with this classic touch keyboard layout. However, third-party IMEs will be able to request an alternate touch layout.

Become familiar with the Windows 8 IME guidelines so that you can promote key Windows Store user experience principles in your IME. IMEs that adhere to the guidelines must set a flag to indicate that the IME is compatible with the Microsoft design. Windows 8 blocks all desktop IMEs from running in Windows Store apps.

Digital signing, in addition to revocation by Windows Defender, prevents malicious IMEs from being installed onto the Windows 8 system. Upon identity verification, a third-party IME's .dll is digitally signed. Only IMEs that have this digital signature can be installed onto the system without having a critical warning message appear to the user. Users can report malicious IMEs. After an IME has been determined to be malicious, Windows Defender removes it from the Windows system.

Text Services Framework

The IME must be TSF-aware in order to be able to run in Windows 8. Windows 8 blocks non-TSF-aware IMEs from running in Windows Store apps. When an app is started, TSF loads the IME .dll for the IME that the user has selected into the app process.

Note: To provide separate functionality or UIs between Windows Store apps and desktop apps, the .dll loaded by TSF can check which type of app it is being loaded into. The IME calls the ITfThreadMgrEx::GetActiveFlags method and checks the TF_TMF_IMMERSIVEMODE flag, and can trigger different app logic depending on the result.

When an IME is loaded into a Windows Store app, it is subject to the same app container restrictions as the app itself. This behavior ensures that IMEs are not able to violate Windows Store app security contracts, despite having access to the desktop SDK (because they are not distributed or certified by the Windows Store). Some functions that IMEs currently perform are affected inside an app container. Those functions include:

- Dictionary files
- Internet updating
- On-the-fly learning
- Sharing info between processes

See the Windows 8 IME Guidelines for more info.

Legacy IMEs do not work in Windows Store apps in order to avoid the potential for bad user experiences including system stoppages. IMEs that are compatible with the Windows Store apps must self-declare by setting a flag indicating this compatibility. This flag is provided by TSF in the TF_INPUTPROCESSORPROFILE structure. Details regarding how to use this flag to declare a third-party IME as Windows Store app-compatible will be documented and posted publicly in the Windows 8 IME Guidelines.

IMEs that are compatible with Windows Store apps are allowed to run in either desktop apps or Windows Store apps. IMEs that are not compatible can only run in desktop processes.

User interface

Although third-party IMEs have access to desktop windowing APIs, they must follow the same window API restrictions as the app they are running in. For example, an IME cannot draw on top of a Windows Store app while active in a desktop app. API restrictions are targeted to prevent these scenarios:

- Desktop apps taking focus from Windows Store apps
- Desktop apps drawing in Windows Store app
- Desktop apps interfering with Windows Store apps

Touch keyboard support

While touch keyboard (TKB) support is still available to third-party IME vendors, a fully customizable and integrated touch keyboard experience is not provided in Windows 8. However, third-party IMEs can map their IMEs with the keyboard layout optimized for touch. The Windows Soft Input Panel (SIP) provides a classic keyboard layout by default for third-party IMEs. Because the classic keyboard generates key

events similar to how a hardware keyboard does, there is currently no special implementation requirement for third-party IMEs to work with a touch keyboard. The input handling for hardware key events will also handle key events from classic touch layouts.

Note: IMEs might need to begin handling Unicode input events if TKB support is extended to include optimized keyboard layouts as well.

A third-party IME can choose to use the optimized keyboard layout for their IME. See the third-party IME Guideline for more info.

Make sure that your candidate pane's UI (and other UI elements) are not drawn underneath the touch keyboard. In most cases, the app should resize its window to account for the touch keyboard. However, if an app doesn't do this, IMEs can still use the InputPaneFramework API to learn the position of the touch keyboard. Third-party IMEs can use this API to get the screen space consumed by the touch keyboard prior to drawing candidate (or other) UIs, and reflow their UI to avoid drawing underneath the touch keyboard.

Searching

In Windows 8, Windows Store apps can easily provide their users with search features by implementing the search contract and integrating with the Search pane. The Search pane is a central location for users to perform searches across all their apps. Windows helps apps that use the Search pane get their users where they want to go as fast as possible. In particular, for IME users, it provides a unique search experience that lets compatible IMEs integrate with the Windows 8 for greater efficiency and usability.

An IME is compatible with the integrated search experience if it meets these criteria:

- Is compatible with the Windows Store app environment
- Implements TFS UILess mode APIs
- Implements TFS search integration APIs:
    - ItfSearchCandidateProvider
    - ItfSearchHardwareKeyboardBehaviors

When activated in the Search pane, the compatible IME is placed in UILess mode and cannot show its UI. Instead, it sends conversion candidates to Windows, which will then display them in the inline candidate list control.

The IME also sends Windows candidates that should be used to run the current search – these candidates could be the same as the conversion candidates, or could be tailored for search. Good search candidates meet these criteria:

- No prefix overlap
- No prediction candidate (only completion)

IMEs that do not meet the criteria and are not compatible with search are shown in the same way as in other Windows Store app controls and are not able to take

advantage of UI integration and search candidates. (Apps receive queries only after the user is done composing.)

When an app that supports the search contract receives a query, the query event will include a "queryTextAlternatives" array containing all known alternatives, ranked from the most relevant (likely) to least relevant (unlikely). Whenever alternatives are provided, the app should treat each alternative as a query, and return all results that match any of the alternatives (as if the user had issued multiple queries at the same time), essentially issuing an "or" query to the service providing the results. To enhance performance, apps will often limit matching to the 10 most relevant alternatives.

IME digital signature

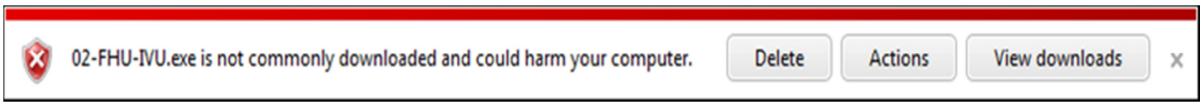
All third-party IMEs must be digitally signed in order to be installed onto the Windows 8 system as an IME. Using SmartScreen, users can see a warning message when downloading an unsigned IME from the web. To obtain a certificate and sign your files:

- **Use an Authenticode signature to digitally sign programs**
    - Obtain a valid Authenticode code signing certificate from one of the many certificate authorities supported by Windows
    - Use development tools (such as signtool.exe) to sign the apps prior to distribution
    - For more info and a step-by-step description of the code signing process, see the Everything you need to know about Authenticode code signing blog entry
- **Ensure downloads are not detected as malware**
    - Downloaded programs that are detected and confirmed as malware affect both the download's reputation and the reputation of the digital certificate used to sign that file
- **Apply for Windows certification**
    - Visit the Windows App Certification page on MSDN

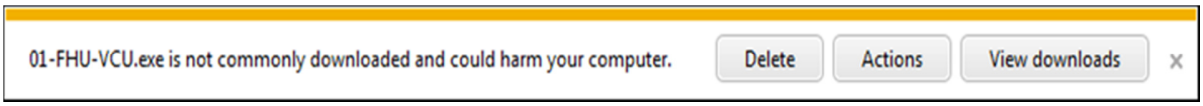
For more info, see these articles about digital signatures and code signing:

- Authenticode Overview
- Ensuring Integrity and Authenticity
- Code Signing Best Practices
- What are Digital Certificates?

If an IME **is not** signed, users receive this warning message when they try to download the IME:



If an IME **is** signed, users see this message instead:



Based on these notifications, users can choose whether to delete the file or ignore the warning and run the downloaded program.

IME revocation

IMEs that are malicious or that do not follow the Windows 8 IME Guidelines can be removed from the system by using Windows Defender. For more info about malicious IMEs, see *Third-party IMEs in Windows 8*.

Resources

ITfThreadMgrEx::Get Active Flags method
http://msdn.microsoft.com/en-us/library/windows/desktop/aa383154.aspx

SignTool
http://msdn.microsoft.com/en-us/library/aa387764.aspx

Everything you need to know about Authenticode Code Signing
http://blogs.msdn.com/b/ieinternals/archive/2011/03/22/authenticode-code-signing-for-developers-for-file-downloads-building-smartscreen-application-reputation.aspx

Windows app contracts and extensions
http://msdn.microsoft.com/en-us/library/windows/apps/hh464906.aspx

Certification requirements for Windows 8 desktop apps
http://msdn.microsoft.com/en-us/library/windows/desktop/hh749939.aspx

Certification requirements for Windows apps
http://msdn.microsoft.com/en-us/library/windows/apps/51A7C609-94AB-49ab-B8E0-F26FF776DDB4.aspx

Using the Windows App Certification Kit
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=27414

Authenticode Overview
http://msdn.microsoft.com/en-us/library/ms537359.aspx

Ensuring Integrity and Authenticity
http://msdn.microsoft.com/en-us/library/ms537361.aspx#Ensuring_Integrity_a

Code Signing Best Practices
http://msdn.microsoft.com/en-us/windows/hardware/gg487309.aspx

What are Digital Certificates?
http://msdn.microsoft.com/en-us/library/Aa190113

New API allows apps to send “TRIM and Unmap” hints to storage media

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

TRIM hints notify the drive that certain sectors that previously were allocated are no longer needed by the app and can be purged. This is typically used when an app makes large space allocations via a file and then self-manages the allocations to the file (for example, Virtual Hard Disk files).

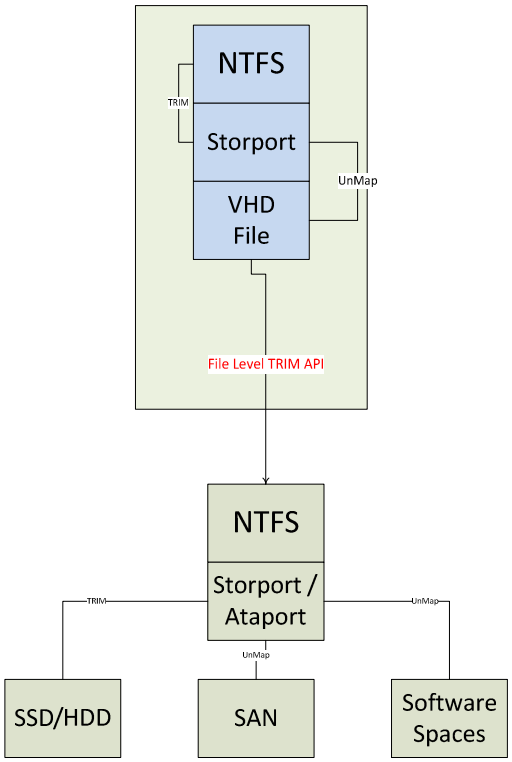
What is TRIM?

Solid state drives (SSDs) are typically flash memory based block-erased devices; this means that when data is written to the SSD, it cannot be over-written in place and must be written elsewhere until the block can be garbage collected. Since the SSD has no internal mechanism for determining that certain blocks are removed and others are needed. The only time the SSD can mark a sector ‘dirty’ is when it is over-written. In other cases, such as when a file is deleted, the SSD retains these sectors because the deletion is performed as a master file table (MFT) change only, and not as an operation to all the sectors of the file. In Windows 7, we introduced a standard way of communicating with SSDs about sectors that are not needed any more. This command is defined in the T13 specification (http://www.t13.org/Standards/Default.aspx?DocumentType=3) as the TRIM command; NTFS sends the TRIM command for some normal inline operations such as “deletefile.”

Other uses of TRIM in the storage world

Like SSDs, storage area networks (SANs) and the new Windows 8 feature Software Spaces implementations consume TRIM command hints to manage their spaces in thinly provisioned environments. SANs and Software Spaces allocate regions of storage in sizes that are greater than sectors or clusters (anywhere from 1MB to 1GB). When they receive TRIM hints for its allocation size (or greater than the allocation size), the SAN/SSD can de-allocate a region to free up the space for other files. They typically pass through all TRIM hints to the underlying media (SSD or HDD) so that they can consume the freed up space as appropriate. They do not typically move data around to de-allocate regions, nor do they keep track of TRIM areas to de-allocated regions (when the region is empty).

Thinly provisioned SANs use the TRIM hints that are passed to them to help reduce the overall physical storage footprint, hence reducing cost. The SCSI (T10: http://www.t10.org) specification defines the ‘Unmap’ command (similar to the TRIM command); here the command is applicable to all kinds of storage including HDDs, SSDs etc. The UnMap command helps to remove physical blocks from the SAN’s allocation.



How to use the new API

```
#define FSCTL_FILE_LEVEL_TRIM CTL_CODE(FILE_DEVICE_FILE_SYSTEM,
130, METHOD_BUFFERED, FILE_WRITE_DATA)

Where
typedef struct _EXTENT_PAIR {
    ULONGLONG Offset;
    ULONGLONG Length;
} EXTENT_PAIR, *PEXTENT_PAIR;

typedef struct _FILE_LEVEL_TRIM {
    //
    // A count of how many Offset:Length pairs are given
    //
    DWORD PairCount;
    //
    // All the pairs.
    //
    EXTENT_PAIR Pairs[1];
} FILE_LEVEL_TRIM, *PFILE_LEVEL_TRIM;
```

File TRIM is passed via the buffer if possible or asynchronously (non-buffered) to the Device IOCTL DSM command TRIM. This is mapped to the TRIM command for ATA devices and UnMap command for SCSI devices. The file TRIM code sends the regions one-by-one as specified by the extents above.

File TRIM does not wait for returns from the device, because the TRIM and Unmap commands are defined as hints to the underlying storage media and return codes are not expected.

End-to-end workflow:

1. Call File Trim
    a. File TRIM examines inputs for errors
    b. File TRIM breaks up the extents into LCN regions
    c. File TRIM rounds up and down for mis-aligned regions that are passed into TRIM
    d. File TRIM purges entries in the cache that relate to the TRIM areas
    e. File TRIM passes IOCTL_DSM (Trim) per region
2. File TRIM returns or errors
    a. Error checking is done on input validity
    b. If only some of the extents are valid, one error is returned for the complete API call

Use cases

Consumer virtual hard disk (VHD) mounted on an SSD:
The VHD is initially mounted on 'clean' unused media. As the VHD is used, the VHD consumes parts of the storage media for files, etc. When it deletes files in the storage media, these files are not removed from the SSD since the complete VHD is stored as one file on the SSD. The Hyper-V environment calls File TRIM for all regions that are deleted when delete-file occurs in the VHD environment. The File_TRIMs are translated to the SSD so that the SSD performance can be optimized.

Consumer VHD mounted on a thinly provisioned SAN:
The VHD is initially mounted on one minimum slab of a thinly provisioned environment. As files are stored in the VHD, the storage footprint of the VHD grows in multiples of slabs. When files are removed in the VHD, the Hyper-V calls File_TRIM to the underlying thinly provisioned SAN. If the TRIMs are bigger than the SLAB granularity, the SAN can now remove a SLAB and hence reduce the footprint of the VHD on that SAN.

If the VHD is resident on a Windows 8 based server, the Storage Optimizer will also send TRIMs to reduce the slab footprint of the VHD from within the mounted VHD.

Tests

There are no comparable APIs in earlier operating system releases. There should be no performance impact from the actual API itself, though the storage media (if implemented correctly) can show better write performance. The API should be used very carefully; only extents that are no longer needed should be passed down as those extents will be permanently removed from the storage media.

Resources

T13 specification
http://www.t13.org/Standards/Default.aspx?DocumentType=3

T10 SCSI specification
http://www.t10.org

Multipath I/O now supports extended storage request blocks

Platform

Servers – Windows Server 2012

Description

In Windows Server 2012, a new structure, the STORAGE_REQUEST_BLOCK (extended SRB) replaces the SCSI_REQUEST_BLOCK (legacy SRB) in the core storage stack. Extended SRBs replicate the functionality of the legacy SRBs, but are also extensible and scalable.

Multipath I/O (MPIO) supports extended SRBs and allows Device Specific Modules (DSMs) to specify extended SRB support as well. However, in order for a multipath device's storage stack to use extended SRBs, **all components in the stack must support extended SRBs, including the DSM.** Note that the Microsoft in-box DSM, MSDSM, does support extended SRBs.

The SCSI_PASS_THROUGH_EX structure is not part of the extended MPIO pass through structure since the extended SCSI pass through can be of a variable size, depending on the command line debugger (CDB). Instead, the extended MPIO pass through has a field that describes the offset from the beginning of the extended MPIO pass through structure to the SCSI_PASS_THROUGH_EX structure. The caller must make sure they allocate a buffer of the appropriate size and set the SCSI pass through offset correctly. As long as the caller follows the rules for extended SCSI pass-throughs, then there should not be any extra work for them to use the extended MPIO pass through.

Note: If the DSM does not support extended SRBs, MPIO will fail extended MPIO pass through requests that have the MPIO_IOCTL_FLAG_INVOLVE_DSM flag set.

Manifestation

If any part of the device's storage stack, including DSM, does not support extended SRBs, the storage stack will revert to using legacy SRBs.

Solution

There are only two MPIO requirements for a DSM to support STORAGE_REQUEST_BLOCKS:

- The DSM must report its type as **DsmType6**
- The DSM must provide the **DSM_ADDRESS_TYPE_SUPPORTED** function

If the DSM does not report DsmType6 or if it reports DsmType6 but does not provide the DSM_ADDRESS_TYPE_SUPPORTED function, MPIO will assume the DSM does not support extended SRBs.

The DSM_ADDRESS_TYPE_SUPPORTED function accepts two parameters, one of which is an address type. This address type must be one of the STORAGE_ADDRESS_TYPE_* values defined in srb.h. The DSM must return TRUE if it supports the given address type and FALSE if it does not. The definition of "support"

is ultimately up to the DSM. MPIO uses this function to ensure that the DSM will not misbehave if it is handed an extended SRB with a STOR_ADDRESS structure of the given type. As such, MPIO generally calls this function when a multipath device is being enumerated, but the function could be called at any time.

If a DSM never touches an extended SRB's STOR_ADDRESS, then it can return TRUE for any valid STORAGE_ADDRESS_TYPE_* value. Review the sample DSM in the WDK.

Other important notes

- DSMs that support extended SRBs must be able to handle both SCSI_REQUEST_BLOCK structures and STORAGE_REQUEST_BLOCK structures. Just because a DSM reports that it supports extended SRBs does not mean that it will exclusively receive extended SRBs. It may still receive any number of legacy SRBs in addition to extended SRBs, and therefore must be able to handle both.
- We have provided a header file in the WDK called srbhelper.h that contains inline functions to help drivers that must handle both extended and legacy SRBs. Our sample DSM uses this header so you can use it as an example of how to use these functions.
- A DSM that does not support extended SRBs will never have to handle extended SRBs.
- It's possible that a MPIO will cause the storage stack to fall back on legacy SRBs in the event that the DSM does not support a given STORAGE_ADDRESS_TYPE_* (but otherwise supports extended SRBs).
- "BTL8" is the only STORAGE_ADDRESS_TYPE_*currently defined for Windows 2012.

Resilient file system

Platform

Servers – Windows Server 2012

Description

Resilient File System (ReFS) is a new local file system. It maximizes data availability, despite errors that would historically cause data loss or downtime. Data integrity ensures that business critical data is protected from errors and available when needed. Its architecture is designed to provide scalability and performance in an era of constantly growing data set sizes and dynamic workloads.

The key features of ReFS are:

- **Integrity**: ReFS stores data so that it is protected from many of the common errors that can cause data loss. File system metadata is always protected. Optionally, user data can be protected on a per-volume, per-directory, or per-file basis. If corruption occurs, ReFS can detect and, when configured with Storage Spaces, automatically correct the corruption. In the event of a system error, ReFS is designed to recover from that error rapidly, with no loss of user data.
- **Availability**: ReFS is designed to prioritize the availability of data. With ReFS, if corruption occurs, and it cannot be repaired automatically, the online salvage process is localized to the area of corruption, requiring no volume down-time. In short, if corruption occurs, ReFS will stay online.
- **Scalability**: ReFS is designed for the data set sizes of today and the data set sizes of tomorrow; it's optimized for high scalability.
- **App Compatibility**: To maximize AppCompat, ReFS supports a subset of NTFS features plus Win32 APIs that are widely adopted.
- **Proactive Error Identification**: The integrity capabilities of ReFS are leveraged by a data integrity scanner (a "scrubber") that periodically scans the volume, attempts to identify latent corruption, and then proactively triggers a repair of that corrupt data.

Resources

Building Windows 8 Blog Post – Building the next generation file system for Windows: ReFS
http://blogs.msdn.com/b/b8/archive/2012/01/16/building-the-next-generation-file-system-for-windows-refs.aspx

Application Compatibility and ReFS
http://go.microsoft.com/?linkid=9797693

File server API support

Platform

Servers – Windows Server 2012

Description

Documentation regarding which APIs are supported for the Server Message Block 2.2 (SMB 2.2) protocol and Resilient/Cluster Shared Volumes (ReFS/CSVFS) file systems will be made available for developers at the resource listed below. The documentation is broken down into these categories:

- File Management Functions
- Directory Management Functions
- Volume Management Functions
- Security Functions
- File and Directory Support Codes
- Volume Control Codes
- Memory Mapped Files

Usage

Certain APIs are not supported with all protocols or in all scenarios while other APIs have been deprecated in Windows Server 2012. Review the list of APIs that are supported to ensure continued support for your apps.

Resources

API Support and Compatibility Matrix
http://go.microsoft.com/?linkid=9797692

New File History feature

Platform

Clients – Windows 8

Description

The new File History feature replaces the existing Backup and Restore function, and offers protection for user files stored in user libraries. A set of APIs is provided that allows developers to programmatically enable File History and select a specific storage target. Documentation for these APIs is available on MSDN.

It may affect workflows related to data protection and documentation that depend on the Windows Backup and Restore feature.

We do not recommend using both features at the same time. File History checks if a Backup schedule exists and is active and if it finds one, it will not let users to turn it on. In this case, users who want to use File History will have to delete the Windows Backup schedule.

Manifestation

Workflows may be interrupted and documentation for the previous method for accessing the Windows Backup and Restore feature will need to be updated to reflect these changes.

Solution

Revise apps that contain workflows that are adversely impacted by the new File History feature to remove any conflicts and incorporate the new set of APIs.

Tests

The API allows developers to determine if File History is turned on.

Operating system now controls power to optical disk drives

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

In previous versions of Windows, power to the optical drive was not managed when the optical drive was not in use. Now, if there is no media present in the optical disk drive (ODD), the operating system turns off the power to the optical drive. This feature is called zero power ODD (ZPODD). The feature is applicable only to optical drives that use a Slimline SATA (Serial Advanced Technology Attachment) connector.

Manifestation

We have not found any negative impacts from this new behavior; however, you should be aware of it as it may result in unexpected behavior of media-writing software.

Mitigation

To revert to always-on status, turn off this functionality in the Registry. The absolute path to the registry value is:

```
HKLM\SYSTEM\CurrentControlSet\Services\cdrom\Parameters\ZeroPowerODDEnabled
```

Its type is DWORD (32 bit), and if its value is 0, then ZPODD is disabled; if it's any other value, then ZPODD is enabled.

Tests

Test your media-writing software for any anomalies that occur due to this new feature. Please inform Microsoft (mailto:OptIssue@microsoft.com) of any issues you find with this new feature.

Support for USB 3.0

Platforms

Clients – Windows 8

Servers – Windows Server 2012

Description

In Windows 8 and Windows Server 2012, we added support for USB 3.0. We did this by adding a new software stack to power the USB 3.0 host controller, called an eXtensible Host Controller (XHC). We maintained interface parity, IRQL level, caller context, error status, retry frequency, and so on. There should be no difference for you when interacting with a device connected over USB 2.0 versus USB 3.0.

However, if you are writing a driver for a new, USB 3.0 device, definitely opt for new USB 3.0 capabilities.

Manifestation

Device transfers are faster and less power is consumed from a PC by USB devices overall. There is a risk that existing USB devices may not function correctly when connected to a USB 3.0 port. This should not happen, and is not expected, but, because the software that powers USB 3.0 is new, it is a risk.

Tools, best practices, and guidance

This section contains aids to help you confirm the continued compatibility of your existing apps or ensure optimal quality and compatibility for new apps you are designing. Descriptions of these tools, best practices, and guidance are included in this section:

- Windows Assessment Toolkit
- Windows App Certification Kit
- Windows Server App Certification Kit
- Windows Hardware Certification Kit

Windows Assessment Toolkit

Platforms

Clients - Windows 7 | Windows 8

Description

The Windows Assessment Toolkit and the Windows Performance Toolkit make up the Windows Assessment and Deployment Kit (ADK). Together they provide a complete solution for evaluating overall computer performance and automating the deployment of the Windows operating system to new computers.

This topic focuses on the **Assessment Toolkit**. The assessment results are used to diagnose potential problems, so that the hardware and software that you develop are both responsive and have a minimal impact on battery life, startup performance, and shutdown time. The same assessments are available for OEM partners, ISV/IHV partners, enthusiasts, and other members of the community, to establish a common framework to measure, compare, and review aspects of quality.

By using the Windows assessment tools, you can measure different aspects of performance across a variety of scenarios, from boot time to battery-life performance to high-definition video streaming. Assessments can identify potential issues, inconsistent behavior, and highlight areas to investigate.

In previous Windows releases, multiple tools were available to measure quality, including Velocity Test Suite (VTS/VOS), Fundamental Quality Tools Suite (FQTS), and System Power and Performance Tools Suite (SPPTS). The Assessment Toolkit combines the functionality from these performance-diagnostic tools into an integrated set of tools that is easier to use and provides broader, more meaningful results.

The Windows Assessment Toolkit maximizes consumer satisfaction by:

- Helping you to assess the overall experience of Windows, value-added software and drivers, and hardware configurations
- Providing detailed system data that can help you identify the root causes of quality issues
- Reducing your costs by revealing issues during development
- Generating comparisons of results from different computers or the same set of computers over time by creating comparison graphs from multiple result sets
- Generating feedback in a standardized format that you can use to improve the quality of your products

Important business objectives can be achieved by using the Assessments Toolkit:

- **Measure & compare** - You can use the data to compare components (software, drivers, or both) against other similar components to facilitate your decision making, recommendations, and competitive benchmarking

- **Improve quality** - You can work independently or with involved partners to build a component (software, driver, or both) as per pre-defined quality criteria
- **Track quality** – You can create a process for efficiently tracking quality of component versions and detect regressions after each iteration

Usage and best practices

Assessments are a combination of XML and binary files that induce a specific set of states on a computer, measure and record the activity, and preserve the recorded results. A job is a collection of one or more assessments and their settings, run at one time on a computer. The Assessment Platform provides the infrastructure for consistently running and displaying jobs, assessments, and results. The results often include diagnostics and remediation info that helps you determine areas that need additional investigation and corrective action. You can:

- Run assessments against a single computer or a small collection of computers with the **Windows Assessment Console** (Windows AC)
  This scenario is for users who want to view performance characteristics on one or several different computer configurations. Windows AC is a graphical user interface (GUI) that is used to group the assessments, create a job, package a job, run the job, and manage the job results. Results often include recommended actions.
- Run assessments against multiple computers in a lab environment with **Windows Assessment Services** (Windows AS)
  This scenario is primarily for users who want to run a suite of qualitative assessments against a complete line of desktops, laptops, or slate computers in a development environment.

The Assessment Toolkit offers these features:

- A simple graphical user interface (GUI) and assessments that can be used to assess a computer without any in-depth technical knowledge of the system
- Assessment results, viewed in the console or UI, which often include recommendations to help you improve the system
- The ability to run a preconfigured job with one click
- Predefined assessment settings in each preconfigured job assessment so that you can run a job on multiple computers and be confident that the results are comparable
- Jobs that can be customized to include the assessments you want to use and the settings you want to use
- Assessment Platform command-line syntax for scripting and automating jobs
- The ability to run a job, view the results, take remediation steps to improve the system, and then run the job again and compare the new results side-by-side with the old results to see the how the system has improved

The Assessment Toolkit is typically used in these scenarios:

Scenario	Description
"Black box"	Run a predefined job and examine the results for any unusual values or indications of issues with drivers, memory usage, or other areas that the assessments address.
Comparing results	1. Run a single assessment using the recommended settings on any computer that is running a supported operating system. 2. Use the Windows AC to package the job to run on another computer. 3. Save the results to a share so that you can compare the results. 4. Compare the results from any Windows computer with those of any other supported operating system to identify differences.
Clean computer	Run assessments on a clean computer that includes only the operating system to establish baseline system results.
Computer with added hardware or software components	Add new hardware or software to the clean computer system and then re-run the assessments to compare the results with clean computer results.
Creating assessments	Use public APIs to develop or extend an assessment, or integrate assessments with your tools and infrastructure.

These assessments can be used:

Assessment	Description
Driver Certification Pre-validation	The **Driver Certification Pre-validation** assessment verifies that the drivers on a running Windows operating system qualify for the Windows Certification Program. The results include recommendations that help you resolve issues that the assessment finds, such as unsigned drivers or expired signatures.
Driver Verification	The **Driver Verification** assessment verifies that an offline Windows image or a running Windows operating system contains the correct set of drivers. The results include recommendations to help you resolve any issues that the assessment finds. These issues might include missing, duplicate, older, or unnecessary drivers.
File Handling	The **File Handling** assessment provides an automated way to exercise common file operations and capture metrics. The metrics measure durations and throughput to help you understand how well a computer performs in end-user file-handling scenarios. The File Handling assessment uses a set of workloads to simulate a user who is copying, moving, compressing, uncompressing, and deleting files and folders on client systems.

Assessment	Description
Photo Handling	The **Photo Handling** assessment measures computer performance and battery life by simulating an end user who is viewing and manipulating photos.
Internet Explorer launch/tab create	The **Internet Explorer Startup Performance** assessment helps identify components that can affect the time needed to start Internet Explorer. The assessment measures the time to fully render a blank page, including the load time of the IExplore.exe process and the frame-creation and tab-creation intervals. It also measures the effect of all extensions, add-ins, and toolbars that are installed on the system. It does not measure any network or browsing performance.
On/Off	The **On/Off Transition** assessment measures the performance of Windows 8 boot, standby, and hibernate performance scenarios.
Internet Explorer Browsing Performance	The **Internet Explorer Browsing Performance** assessment measures the quality of the browsing experience in Internet Explorer, and evaluates CPU and graphic hardware capabilities. Three separate browsing workloads are provided to stress the computer in various ways.
Media Transcoding Performance	The **Transcode Video** assessment measures the process of changing a video file to a different format or bitrate. This assessment runs a series of transcode operations with common input and output file formats and resolutions
Windows **UI Performance**	The **Windows UI Performance** assessment evaluates the performance of some basic experiences within the Windows 8 user interface. The assessment measures responsiveness and rendering quality while the assessment exercises workloads that simulate user activities, such as using search and transitioning from Windows Store app to the classic desktop. Responsiveness results are measured in milliseconds. Low numbers mean that the computer is faster and more responsive. For rendering, the results show the frame rate and the number of glitches that occur.
Memory Footprint	You can use the **Memory Footprint** assessment to quantitatively compare a baseline operating system image against another operating system image. You can then identify the specific components that affect the memory footprint of the physical system. These components can include drivers, add-in apps, preloaded software packages, and antivirus programs.
First Boot Performance	The **First Boot Performance** assessment identifies issues that affect how long Windows takes to boot and display the Start screen the first time the computer is started. The results help OEMs diagnose what causes delays and provide recommendations for improving the experience.
Media Streaming	The **Streaming Media Performance** assessment helps you assess a computer configuration's performance when you stream media using Internet Explorer. You can use the assessment results to understand, compare, and improve the streaming media experience.

Assessment	Description
WinSAT Comprehensive	The **Windows System Assessment (WinSAT)** is used to rate and improve a computer's performance in several system components, including CPU, memory, disk, and graphics. WinSAT Comprehensive assessment results express the capability of a computer's hardware configuration in numbers. Higher scores generally mean that the assessed computer performs better and faster than a computer that has a lower score.
Energy Efficiency	The **Energy Efficiency** job provides an automated way for you to assess the battery life of a computer. Using workloads, the Energy Efficiency job also performs diagnostics that assess whether system components are using power when they should be idle.
MiniFilter Diagnostic Settings	The **MiniFilter Diagnostic** option runs within the Internet Explorer Startup Performance assessment, the File Handling assessment, and the Boot Performance (Windows 8) assessment. Selecting this option within the assessments that offer the MiniFilter diagnostic option produces metrics that help you evaluate the impact of MiniFilter operations on various assessment scenarios.
Windows Media Player Performance and Quality	The **Windows Media Player Performance and Quality** assessment launches WMP and plays multiple media clips one after another to capture performance and quality metrics related to media playback.

Other tools, such as the Windows Performance Toolkit, are included in the ADK. These tools provide in-depth info enabling you to analyze and trace system and app performance. See the Resources section below for more info.

Resources

Video:

Channel9 – BUILD ADK Videos
http://channel9.msdn.com/Events/BUILD/BUILD2011?sort=status&direction=asc&term=&t=assessment+and+deployment+kit

MSDN documentation:

Assessment and Deployment Kit (Always refer to the ADK_TechnicalReference.chm file in the ADK. It is the main source of documentation.)
http://msdn.microsoft.com/library/windows/hardware/br259106.aspx

Assessment Execution Engine
http://msdn.microsoft.com/en-us/library/windows/desktop/hh437709(v=VS.85).aspx

Dev Center:

Windows Performance Analysis
http://go.microsoft.com/fwlink/?LinkId=147307

Windows App Certification Kit

Platforms

Clients - Windows 7 | Windows 8

Description

The Windows App Certification Kit (ACK) is used to validate compliance with certification requirements, and replaces the Windows Software Logo Kit (WSLK) used for validation in the Windows 7 Software Logo program. Desktop, desktop device, and Windows Store apps can be certified; however, plug-in, ActiveX, and other web apps cannot be certified. The Windows ACK is included in the Windows Software Development Kit (SDK) and the Windows SDK for Windows Store apps.

Certification, while highly recommended, is not a requirement for running a desktop app under Windows 8, but it is required in order to list a desktop app in the Windows Store. Both Windows 7 and Windows 8 apps must use the new Windows App Certification Kit to become certified and eligible for sale in the Windows Store. Certification is embedded in the Windows Store onboarding process.

Requirements

There are 12 requirements for desktop certification and 9 requirements for Windows Store app certification. The online self-test and submission of results for desktop apps takes less than 45 minutes, and for Windows Store apps takes less than 20 minutes.

Desktop apps [Legend: ✓ = included ↑ = enhanced -- = skipped if not applicable]

Requirement	Win7	Win8 app	Device app	Highlights
Clean, reversible installation	✓	↑	↑	Must include Version & Publisher in ARP No leftovers on uninstall
Install to correct location	✓	✓	✓	
Digitally sign files & drivers	✓	↑	✓	Must sign main MSI and main EXEs drivers must have HW Certification except for Device Apps (timing)
x64 support	✓	✓	✓	
OS version check	✓	✓	✓	
UAC Compliance	✓	↑	--	Main EXEs must be marked *asInvoker* Those manifested to *requireAdministrator* must be Authenticode signed
Restart manager	✓	✓	--	
Loading in safe mode	✓	✓	--	
Multiuser support	✓	✓	--	Simplified Kit User Experience
App Reliability		✓	✓	No crashes or hangs during test No App Compatibility Modes Avoid loading arbitrary binaries using *AppInit_DLLs*
Support Windows Security features		✓	✓	BinScope Binary Analyzer Test
Do not disable Windows Security features		✓	✓	Attack Surface Analyzer Test

Windows Store apps

Requirement	Windows Store app	Highlights
Windows Store app test	✓	Number of tiles Number of apps per package App type *ResourceID* not listed *OSVersionHighestTested*
Use of supported APIs only	✓	Tests whether the app uses only the allowed APIs
Performance	✓	Tests how quickly your app launches and suspends
Package resource	✓	Checks the languages declared in the manifest to make sure the app has resources for those languages
Security	✓	Tests for security vulnerabilities
Crashes and hangs	✓	Monitors the app throughout the test run to detect if the app crashes or hangs

Usage

To certify your apps:

1. Install the latest Windows 8 build.
2. Review the certification requirements.
3. On a clean system, run the Windows App Certification Kit that is included in the Windows SDK and follow the instructions on the screen.
4. Review certification report and fix issues.
5. Complete the process:
    a. If this is a desktop app, submit the report at the Partner Portal.
    b. If this is a Windows Store app, start the Windows Store onboarding process.

Report any bugs you discover in the Windows App Certification Kit via the Partner Feedback tool at https://sysdev.microsoft.com (replaces WinQual).

Resources

Certification requirements for Windows 8 desktop apps
http://msdn.microsoft.com/library/windows/desktop/hh749939.aspx

Certification requirements for Windows apps
http://msdn.microsoft.com/library/windows/apps/51A7C609-94AB-49ab-B8E0-F26FF776DDB4.aspx

Using the Windows App Certification Kit
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=27414

Windows Server App Certification Kit

Platform

Servers – Windows Server 2012

Description

The Windows Server App Certification Kit (SCK) is used to validate compliance with certification requirements for Windows Server 2012 applications.

- Server apps primarily designed to run on Windows Server Operating Systems are eligible to participate in the program.
- Hosted apps/services, browser plug-ins, ActiveX controls, add-ons, zipped executables, and macros are ineligible.

Requirements

SCK includes automated tests for:

- Windows Installer
- Setup and Primary Functionality
- Drivers and Security
- Best Practices

Self-testing and submission of results for most Server apps should take 2 hours; more complex apps can take around 4 hours.

All drivers must separately pass Windows Hardware Certification testing and be signed by Microsoft for Windows Server 2012 when the driver signing programs become available.

Usage

To test your apps:

1. Install the latest Windows Server 2012 minimum configuration (Full Server GUI, MinShell, or Server Core) as required for your app.
2. Review the certification requirements.
3. On a clean system, run the SCK in either full UI mode or command line mode.
4. Complete the self-guided testing process.
5. Review results and fix your app as required.

Note: More info on how to submit results for your app will be published later.

Report any bugs you discover in the SCK by sending your feedback to WSLogoFB@Microsoft.com

Resources

Requirements for Windows Server App Certification Program
http://go.microsoft.com/fwlink/?LinkID=225718

Windows Server App Certification Kit (SCK)
http://go.microsoft.com/fwlink/?LinkID=241644

Windows Certification Program for Server Applications
http://go.microsoft.com/fwlink/?LinkID=241671

Windows Server Certification Feedback
WSLogoFB@microsoft.com

Windows Hardware Certification Kit

Platforms

Clients – Windows 7 | Windows 8

Servers – Windows Server 2008 R2 | Windows Server 2012

Server/Controller - Windows Server 2008 R2

Description

The Windows Hardware Certification Kit (Windows HCK) enables developers, ISVs, IHVs, and OEMs to certify their hardware devices, systems, and filter drivers for the latest Windows operating systems. It contains all the tools and documentation that you need to certify your hardware and filter drivers for these operating systems:

- Windows 8
- Windows 7
- Windows Server 2012
- Windows Server 2008 R2

The Windows HCK replaces the Windows Logo Kit and is a part of the Windows Certification Program.

Usage or best practices

Before you can test any hardware or filter driver, you must set up the proper test environment based on the hardware or filter driver you are certifying. This includes the controller, client, and potentially additional hardware (for example, multi-function devices) or software. After your environment is set up, you can test your hardware using the new HCK's Studio tool. These steps outline the certification test process.

1. Set up test environment.
2. Create a project.
3. Create one or more machine pools.
4. Select features to validate.
5. Select and run tests against those features.
6. View test results.
7. Submit your package.

Resources

Windows Hardware Development
http://msdn.microsoft.com/en-us/windows/hardware/

Windows Hardware Certification Program
http://msdn.microsoft.com/library/windows/hardware/gg463010

Windows Hardware Development Downloads
http://msdn.microsoft.com/en-us/windows/hardware/gg454513/

Start Developing Hardware for Windows
http://msdn.microsoft.com/library/windows/hardware/gg507680