

# Windows Platform Design Notes

---

*Designing Hardware for the Microsoft® Windows® Family of Operating Systems*

## Kernel Enhancements for Windows Whistler

**Abstract:** Microsoft has made many enhancements to the kernel of the next release of the Microsoft® Windows® operating system, code named "Whistler." This paper provides an overview of the new features and changes in the kernel for Windows Whistler, intended for system and peripheral designers, driver developers, and firmware developers who are creating products that run Windows Whistler.

This paper assumes that the reader is familiar with related concepts and issues for Windows 2000. For information, see the Windows Driver Development Kit (DDK; <http://www.microsoft.com/ddk/>) and the Windows 2000 Resource Kit (available through MSDN Professional subscription or through Microsoft Press).

### **Beta 2 Preview Draft - March 19, 2001**

*The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.*

*Microsoft Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to the patents, trademarks, copyrights, or other intellectual property rights except as expressly provided in any written license agreement from Microsoft Corporation.*

*Microsoft does not make any representation or warranty regarding specifications in this document or any product or item developed based on these specifications. Microsoft disclaims all express and implied warranties, including but not limited to the implied warranties of merchantability, fitness for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Microsoft does not make any warranty of any kind that any item developed based on these specifications, or any portion of a specification, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such intellectual property rights where appropriate. Microsoft shall not be liable for any damages arising out of or in connection with the use of these specifications, including liability for lost profit, business interruption, or any other damages whatsoever. Some states do not allow the exclusion or limitation of liability or consequential or incidental damages; the above limitation may not apply to you.*

*Microsoft, MSN, Win32, Win64, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.*

*© 2001 Microsoft Corporation. All rights reserved.*

**Contents**

Introduction .....	3
Registry Enhancements .....	3
Larger Registries .....	4
Faster Queries .....	4
Windows Whistler Support Enhancements .....	4
Kernel Changes for Improved Debugging .....	5
Built-in User Heap Leak Detection .....	6
Additional Heap Counters .....	8
I/O Subsystem .....	8
New Cancel Queue .....	9
File System Filter Driver APIs .....	9
Improved Low-Memory Performance .....	9
I/O Throttling .....	10
DMA Improvements .....	10
WebDAV Redirector .....	10
System Restore .....	10
Volume Snapshot Service .....	11
Changes in Existing I/O Features .....	11
Read Only Kernel and HAL Pages .....	12
New APIs .....	13
Memory Management .....	14
Logical Prefetcher for Faster Boot and Application Launch .....	14
Scalability Improvements Due to Reduced Lock Contention .....	15
Improved Caching and Backup Due to Dynamic Paged Pool Usage .....	16
Improved Server Scaling Due to Individual Page Charging .....	16
Improved Terminal Server and Network Server Scaling Due to Increased Number of System PTEs .....	16
Support of Giant Drivers Due to Increased Number of System PTEs .....	17
Support of Direct Execution from ROM .....	17
Power Management .....	17
Improved Boot and Logon Performance .....	18
Boot Loader Improvements .....	18
Operating System Boot Improvements .....	18
Operating System Hibernation Improvements .....	19
Operating System Standby Resume Improvements .....	19
Windows Whistler Boot and Resume Tools .....	20
Headless Support .....	20
Providing Out-of-Band Management .....	20
Implementing Headless Support in Windows Whistler .....	21
Designing Hardware and Firmware .....	21
Hot-Add Memory and Memory Mirroring Support .....	22
ccNUMA Support .....	22
Per Node Memory Allocation .....	23
Per Node Scheduling .....	23
OEM Support for ccNUMA .....	23
Static Resource Affinity Table .....	24
How the Operating System Can Use SRAT .....	25
New Support for Hardware .....	25
Intel Itanium Support .....	26
Benefiting from 64-bit Architecture .....	26
Designing for 64-Bit Compatible Interfaces .....	27
Components Removed .....	27
"Designed for Windows Whistler" Logo Program Requirements .....	27
References .....	28
Acronyms .....	28

## Introduction

Microsoft has made substantial enhancements to the kernel at the core of Windows Whistler. Kernel improvements are significant because the kernel provides low-level operating system functions, including thread scheduling, interrupt and exception dispatching, multiprocessor synchronization, and a set of routines and basic objects used by the rest of the operating system to implement higher-level constructs.

This paper describes the kernel improvements for Windows Whistler, which include:

- **Registry** – Larger registries, limited only by available system disk space. Improved algorithms for faster queries.
- **Support Enhancements** – Cross-session debugging, new quit and detach command for debugging without killing the application, and built-in user mode heap-leak detection.
- **I/O Subsystem** – New input/output (I/O) interfaces for performance enhancement, while retaining compatibility with Windows 2000 drivers. File System Filter driver application programming interface (API) improvements. Support for performance measurements in retail code, and improved low-memory performance.
- **Memory Management** – Broad range of improvements, including logical prefetch to improve boot and logon performance, reduced paged pool usage, enhanced terminal server support, support of giant drivers, and Windows Whistler execution from ROM.
- **Power Management** – Native support for processor performance control such including Intel SpeedStep Technology, AMD PowerNow!, and Transmeta LongRun for longer mobile PC battery life. Hibernate, standby, and resume performance have been greatly improved.
- **Improved Boot and Logon Performance** – When a Windows Whistler system is first booted, data is saved about all logical disk read operations. On later boots, this information is used to pre-fetch these files in parallel with other boot operations.
- **Headless Support** – For ‘lights-out’ datacenter deployment and remote administration.
- **ccNUMA Support** – Provides better performance for Cache Coherent–Non Uniform Memory Architecture (ccNUMA) computers, as well as an interface to let applications tailor their execution characteristics in the ccNUMA environment.

The Windows Whistler kernel improvements provide new opportunities for independent software vendors (ISVs) and independent hardware vendors (IHVs), and other value-added providers working with Windows 2000. Windows Whistler provides compatibility with Windows 2000 devices and drivers, while providing new APIs, enhancements, and other features that can be leveraged into future products and services.

---

## Registry Enhancements

As with Windows 2000, the registry plays a key role in the configuration and control of Windows Whistler. The registry, which resides on the disk as multiple files called hives, was originally designed as a repository for system configuration data. Although most people think of the registry as static data stored on the hard disk, it is also a window into various in-memory structures maintained by the Windows Whistler executive and kernel.

The registry code is redesigned for Windows Whistler, providing enhanced performance while remaining transparent to applications by using existing registry programming interfaces. Windows Whistler registry enhancements provide performance improvements, including the following areas:

- Converting a delayed close table to a Least Recently Used (LRU) list

- Reducing Kernel Control Block (KCB) lock contention with do not lock registry exclusive and do not touch volatile info
- Providing a security cache to eliminates duplicate security descriptors

The new registry implementation delivers two key benefits:

- Larger registries
- Faster queries

## Larger Registries

Windows Whistler supports larger registries than previous versions of the kernel, which were effectively limited to about 80 percent of the total size of paged pool. The new implementation is limited only by available system disk space.

A tendency to use the registry more like a database developed among registry consumers, which increased demands on registry size. The original design of the registry kept all of the registry files in the paged-pool, which, in the 32-bit kernel, is effectively limited at approximately 160 MB because of the layout of the kernel virtual address space. A problem arose because, as larger registry consumers such as Terminal Services and COM appeared, a considerable amount of paged-pool was used for the registry alone, potentially leaving too little memory for other kernel-mode components.

Windows Whistler solves this problem by moving the registry out of paged pool and using the cache manager to do an in-house management of mapped views of the registry files. The mapped views are mapped in 256K chunks into system cache space instead of paged pool.

## Faster Queries

Another issue that affected registry performance in earlier versions is the *locality problem*. Related cells are spread through the entire registry files. Accessing certain information, such as attributes of a key, could degenerate into page-faults, which lowers performance.

The Windows Whistler registry uses an improved algorithm for allocating new cells that keeps related cells in closer proximity—such as keeping cells on the same page or nearby pages, which solves the locality problem and reduces the page faults incurred when accessing related cells. A new hive structure member tracks freed cells instead of relying on linked freed cells. When future cells are allocated, the freed cell list and a vicinity argument are used to ensure the allocation is in the same bin as the hive.

Windows Whistler improves the way the registry handles big data. In versions before Windows Whistler, if an inefficient application constantly increased a value with a small increment, it created a sparse and wasteful registry file. Windows Whistler solves this problem with a big cell implementation where cells larger than 16K are split into increments of 16K chunks. This reduces fragmentation when the data length of a value is increased within a certain threshold.

---

## Windows Whistler Support Enhancements

Numerous product support enhancements have been built into Windows Whistler, including enhancements to the kernel that improve the debugger shipped with Windows Whistler and the DDK. These enhancements include:

- Kernel changes for improved debugging
- Built-in heap leak detection

- New heap performance counters

## Kernel Changes for Improved Debugging

The debuggers for Windows Whistler have been redesigned and include tools such as WinDBG, Kd, and Cdb. Although the new debugger will also work with Windows NT® 4 and Windows 2000, some features are only available when debugging Windows Whistler. There is also a 64-bit version of all of the debuggers to debug Intel Itanium-based servers running Windows Whistler.

Kernel enhancements available only for debugging under Windows Whistler:

- Cross-session debugging
- Quit and detach
- Debugging over an IEEE 1394 port
- Dynamic control over debug-child flag
- Improved kd serial bandwidth usage
- Loading updated driver files through kd
- Control over whether terminating a debugger also terminates the debuggee

The Windows Whistler debugger and debugger documentation are available at <http://www.microsoft.com/ddk/debugging/>

## Cross Session Debugging

Windows Whistler supports cross-session debugging. Previously, all debugging was done against the Microsoft Win32® environment subsystem (Csrss.exe) running on the computer. This caused potential problems, because on Terminal Services each client has its own Csrss process, so that debugging could not be done across terminal sessions. For example, if User1 logged in running Notepad, a second user, such as a system administrator, couldn't perform a terminal session into the same server and debug User1's Notepad process. The debugger code for Windows Whistler, along with accompanying kernel changes, supports such cross session debugging scenario.

## Quit and Detach

Windows Whistler makes it possible to debug a user-mode application and then detach the debugger, without killing the application. This functionality is supported by a new debugger command, **qd** (quit and detach), which supports detaching the debugger from an application. The debugger command **q** has the default behavior of killing the application.

### Example: Using Quit and Detach

Using Windows Whistler, it is possible to investigate an application with a non-crashing problem without killing the application. To investigate the application, follow these steps:

1. Attach to the process using the debugger.
2. Debug the application.
3. Detach the debugger using **qd**

### Example: Using QD and the Dump Command

When debugging a production machine or other computer you don't want to block with debugging, you can take advantage of **qd** and **.dump** command; these commands let you

generate a dump file for the file. The **.dump** command works on any version of Windows NT 4, Windows 2000, or Windows Whistler if you use the new Windows Whistler debugger.

To use **qd** and **.dump**:

1. Attach to the process using the debugger.
2. Generate a dump file for the application, using the **.dump** command.
3. Detach the debugger using **qd**
4. Analyze the dump file using the debugger (**WinDBG -z <dumpfile>**) without disturbing the original application.

## Debugging over an IEEE 1394 port

The debugger and debuggee machines running Whistler can be connected via IEEE 1394 ports. IEEE 1394 provides much better debugger performance than serial port debugging. For details about how to use IEEE 1394 debugging and other debugger features, see <http://microsoft.com/ddk/debugging/>

## Dynamic Control over Debug-child Flag

When a process is created for debugging, either the `DEBUG_PROCESS` or `DEBUG_ONLY_THIS_PROCESS` flag is used, indicating whether child processes should also be debugged. `DebugActiveProcess` always debugs with the `DEBUG_ONLY_THIS_PROCESS` flag. Prior to Whistler, these flag settings were permanent, but system changes in Whistler now allow the state of the flag to be changed. This allows debuggers more flexibility in whether child processes are debugged, because the flag can be changed at any time.

## Improved kd Serial Bandwidth Usage

Improvements in the kernel debugger protocol reduce the amount of data sent back and forth between the target machine and the kernel debugger, improving debugging performance over slow kernel debugger links.

## Loading Updated Driver Files through kd

Enhancements to the kernel debugger protocol and the kernel now allow the kernel to demand-load driver files from a properly configured kernel debugger. When the system loads a driver, it now queries the kernel debugger to see if the kernel debugger wants to provide the file for the driver in question. If so, the file content is downloaded from the kernel debugger and used. This avoids extra reboots just to copy driver files.

## Control over Whether Terminating a Debugger Also Terminates the Process Being Debugged

Prior to Whistler, terminating a debugger would always kill all processes being debugged by the debugger. The new `DebugSetProcessKillOnExit` API allows a debugger to control this behavior and select either a detach or the default kill.

## Built-in User Heap Leak Detection

Windows Whistler provides built-in user mode heap-leak detection. Poorly written or miscoded applications can “leak” heap memory. In earlier versions before Windows Whistler, when this

situation arose, special tools were needed on the server to help identify the cause of the memory leak. User mode component heap allocation leaks can be seen in two ways:

- Leak detection when the process is exiting
- Using a debugger extension to investigate leaks

## Leak Detection when the Process Is Exiting

Leak detection is made every time a process is cleanly exiting. It doesn't work if the process is terminated with `TerminateProcess()` or `TerminateThread()` / `ExitThread()` for the last thread in the process; but for most applications this is not a problem.

To enable leak detection when the process is exiting, set the registry key as follows:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\ImageName]
"ShutdownFlags"="3"
```

## Using a Debugger Extension to Investigate Leaks

Windows Whistler allows you to use a debugger extension to investigate leaks. This is useful because using leak detection when the process is exiting doesn't work for processes such as `Explorer.exe`, `Services.exe`, and others that are not cleanly shut down. In such cases, you can attach an `Nttd` to the process and use the new debugger extension **!heap -l** to create a similar list. If the size or contents of the block aren't enough to determine whether the program is leaking, you can enable the debug options to get stack traces for these blocks.

### !heap -l usage sample:

```
0:011>!heap -l
Scanning VM ...
Entry  User  Heap  Segment  Size PrevSize Flags
-----
00178908 00178910 00170000 00000000 258 258 busy extra
..... << other 20 blocks >>
005278a8 005278b0 00520000 00000000 40 80 busy extra
22 leaks detected.

0:011> dc 178908 + 20 l1
00178928 002787bc

0:011> dds 002787bc
002787bc abcdaaaa
002787c0 00000001
002787c4 0000000b
002787c8 00000001
002787cc 0000021a
002787d0 00071000
002787d4 00000000
002787d8 002787dc
002787dc 6a252a35 ntdll!RtlAllocateHeapSlowly+0x5b
002787e0 6a2505a4 ntdll!RtlAllocateHeap+0xa26
002787e4 6a206b0f ntdll!RtlDosPathNameToNtPathName_U+0xd9
002787e8 6a20bc48 ntdll!hOpenFile+0x2e
```

```

002787ec 6a20bfff ntdll!bOpenAndMapDB+0x17
002787f0 6a20c1ab ntdll!pdbOpenDatabase+0x2d
002787f4 6a20e4e2 ntdll!trGetMatchingExe+0x1ec
002787f8 6a20b145 ntdll!LdrpInstallAppcompatBackend+0xa0
002787fc 6a2117c6 ntdll!LdrpInitializeProcess+0x9f6
00278800 6a2041a0 ntdll!LdrpInitialize+0x256
00278804 6a2180db ntdll!KiUserApcDispatcher+0x7

```

In this example, the bug is in `hOpenFile`, which didn't free the string returned by `RtlDosPathNameToNtPathName_U`. In all, 22 heap allocations were not freed when the application was exited. The heap address that was allocated is in the entry column, and the size is in the size column. If the `GlobalFlag` for Stack Backtrace was enabled, then analysis could show the actual functions that allocated the heap memory. This can be done several ways

- Use `!heap -k <address>`, using the address of the heap that is shown to be leaking.
- Or-
- As shown in the example, go to the end of the leaking heap allocation, find the stack number that allocated the heap, and dump the structure `ntdll!RtlpStackTraceDataBase`. This structure holds all of the stack back traces. The stack back traces will be in numerical order, so dump the structure until you get to the stack number you are looking for and then `dds <address>`, where `<address>` is the address in the `RtlpStackTraceDataBase` where your stack number resides.

**Note:** In some cases false positives are seen, such as applications that keep encoded pointers and blocks passed as contexts to local procedure call (LPC) ports. In addition, this doesn't work for an application that has its own heap, such as Microsoft Office applications, or for paged heap enabled with the `/full` option.

## Additional Heap Counters

Another important new feature in Windows Whistler is heap performance monitoring. Performance Monitor (Perfmon) can display about 20 heap-related counters: amount of memory reserved and committed per heap, number of blocks allocated and freed for three class sizes, average allocated and free time, lock contention, and others. Perfmon will display these counters when the following registry key is set:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PerfProc\Performance]
"DisplayHeapPerfObject"=dword:00000001
```

Only the process heap and the heaps with higher usage are monitored.

---

## I/O Subsystem

The I/O subsystem consists of kernel components that provide an interface to hardware devices for applications and other mandatory system components. Windows Whistler enhances the I/O subsystem while retaining complete compatibility with drivers written for Windows 2000. This compatibility was essential because the I/O subsystem provides the interface to all devices, and too many changes to process I/O can break existing applications and drivers.

Enhancements were made by adding new APIs, available to drivers written to take advantage of the new Windows Whistler functionality. For this reason, while existing Windows 2000 drivers will work with Windows Whistler, they must be rewritten to take advantage of the new I/O improvements, including the following:

- New cancel queue
- File System Filter Driver APIs
- Improved low-memory performance
- I/O throttling
- Direct Memory Access (DMA) improvements
- WebDAV Redirector
- System Restore
- Volume Snapshot Service

## New Cancel Queue

Rather than having drivers perform device queuing and handling the I/O request packet (IRP) cancellation race, Windows Whistler I/O automates this process. In Windows Whistler, drivers handle IRP queuing and do not have to handle IRP cancellations. Intelligence in the queuing process lets the I/O APIs handle requests rather than drivers in cases where the I/O is cancelled. A common problem with cancellation of IRPs in a driver is synchronization between the cancel lock or the InterlockedExchange in the I/O Manager with the driver's queue lock.

Windows Whistler abstracts the cancel logic in the APIs while allowing the driver to implement the queue and associated synchronization. The driver provides routines to insert and remove IRPs from a queue, and it provides a lock to be held while calling these routines. The driver ensures that the memory for the queue comes from the correct pool. When the driver actually wants to insert something into the queue, it does not call its insertion routine, but instead calls `IoCsqInsertIrp()`.

To remove an IRP from the queue, the driver can either specify an IRP to be retrieved, or pass `NULL`, and the first IRP in the queue will be retrieved. Once the IRP has been retrieved, it cannot be canceled; it is expected that the driver will process the IRP and complete it quickly.

## File System Filter Driver APIs

Several new APIs provide greater all-around reliability. Microsoft worked with third-party developers to test their filter drivers. If a driver crashed attempting to perform illegal functions, together we determined the functionality required, and provided APIs to let them accomplish what needed to be done without harming the rest of the system. These APIs will be included in the Windows Installable File System Development (IFS) Kit for Windows Whistler Beta 2 release.

## Improved Low-Memory Performance

Windows Whistler is more resilient during periods of low memory because “must succeed” allocations are no longer permitted. Earlier versions of the kernel and drivers contained memory allocation requests that had to succeed even when the memory pool was low. These allocations would crash the system if no memory were available. Two important I/O allocation routines used “must succeed,” with the first being for IRP allocation, and the other for Memory Descriptor List (MDL) allocations. If memory couldn't be allocated, the system would blue screen if these routines were used. For Windows Whistler, kernel components and drivers are no longer allowed to request “must succeed” allocations; memory allocation routines will not allocate memory if the pool is too low. These changes allow drivers and other components to take appropriate error actions, rather than an extreme approach such as bug checking a machine.

## I/O Throttling

Another improvement for low-memory conditions is I/O throttling. If the system can't allocate memory, it throttles down to process one page at a time, if necessary, using freely allocated resources. This allows the system to continue at a slower pace until more resources are available.

## DMA Improvements

Three new entries are added to the end of the DMA\_OPERATIONS structure. These three entries will be accessible to any driver, which uses IoGetDmaAdapter(). To safely check whether the new functionality exists, the driver should set the version field of the DEVICE\_DESCRIPTION structure provided to IoGetDmaAdapter() to DEVICE\_DESCRIPTION\_VERSION2.

The current Hardware Abstraction Layers (HALs), which don't support the new interface, will fail the operation because of the version number. HALs that support this feature will understand the new version and will succeed the request, assuming all the other parameters are in order. The driver should try to access these new function pointers only when the driver successfully gets the adapter using DEVICE\_DESCRIPTION\_VERSION2.

## WebDAV Redirector

Windows Whistler includes a new component—the WebDAV redirector. The WebDAV redirector allows applications on Windows Whistler to connect to the Internet, and to natively read and write data on the Internet. The WebDAV protocol is an extension to Hypertext Transfer Protocol (HTTP) that allows data to be written to HTTP targets such as the Microsoft MSN® Web Communities. The WebDAV redirector provides file system-level access to these servers in the same that the existing redirector provides access to SMB/CIFS servers.

One way to access a WebDAV share is to use the **net use** command, for example:

**NET USE \* http://webserver/davscratch**

To connect to an MSN Community, use **http://www.msnusers.com/yourcommunityname/files/** as the target. The credentials you need in this case are your Passport credentials; enter these details in the Connect Using Different User Name dialog if you are using mapped network drive, or use the **/u:** switch with the **net use** command. For example:

**net use http://www.msnusers.com/foaname/files /u:yourpassportaccount@hotmail.com**

The simplest ways to create a WebDAV share are:

- Use Microsoft Internet Information Server (IIS). In IIS, you only need to make a directory browsable to access it through WebDAV and allow writes, and you can also save to it.
- Use MSN Communities. File Cabinets in MSN Communities are WebDAV shares. For details about how to use this MSN file cabinet, see <http://communities.msn.com/filecabinets>.

## System Restore

System Restore is a combination of a file system filter driver and user-mode services that provide a way for user to unwind configuration operations and restore a system to an earlier configuration. System Restore includes a file system filter driver called Sr.sys, which helps to implement a copy-on-write process. System Restore is a feature only of Windows Whistler Personal and the 32-bit version of Whistler Professional; and it is not a feature of the Server versions of Windows Whistler.

## Volume Snapshot Service

A volume snapshot is a point-in-time copy of that volume. The snapshot is typically used by a backup application so that it may backup files in a consistent manner, even though the files may be changing during the backup. Windows Whistler includes a framework for orchestrating the timing for a snapshot, as well as a storage filter driver, not a file system filter driver, that uses a copy-on-write technique in order to create a snapshot.

One important new snapshot-related I/O Control (IOCTL) that affects file systems is `IOCTL_VOLSnap_FLUSH_AND_HOLD_WRITES`. This is actually intended for interpretation by file systems, even though it is an IOCTL. This is because all file systems should pass the IOCTL down to a lower-level driver that is waiting to process the IOCTL after the file system. The choice of an IOCTL instead of an FSCTL ensures that even legacy file system drivers will pass the IOCTL down.

This IOCTL is sent by the Volume Snapshot Service. When a file system such as NTFS receives the IOCTL, it should flush the volume and hold all file resources to make sure that nothing more gets dirty. When the IRP completes or is cancelled, the file system then releases the resources and returns.

## Changes in Existing I/O Features

Windows Whistler includes several changes in existing I/O features, including:

- **FAT32 on DVD-RAM.** DVD-RAM disks can appear as both CD/DVD devices and as rewriteable disks. Windows Whistler will allow DVD-RAM media in DVD-RAM drives to be formatted and used with the FAT32 file system.
- **Defragmentation APIs.** Since the release of Windows NT 4.0, the NTFS file system has exposed APIs that allow a user-mode application to query the allocated ranges of files on disk, and optimize file arrangements in order to defragment (or carefully fragment) files in order to minimize seeks while processing file I/O. In Windows 2000, these APIs have a number of limitations; for example, they do not function on the master file table (MFT), the PageFile, or NTFS attributes. The feature set in Windows Whistler changes the behavior on NTFS as follows:
  - The defragmentation APIs will no longer defragment data by using the system cache. This means Encrypted files will no longer need to be opened with read access.
  - NTFS will now defragment at the cluster boundary for non-compressed files. In Windows 2000, this was limited to the page granularity for non-compressed files.
  - NTFS will now defragment the MFT. This was not allowed in Windows 2000. This is through the regular code path, so there is no limit to how much at once can be moved, and any part of it can be moved other than the first 0x10 clusters. If there is no available space in the MFT to describe the change, then it will be rejected. The API can move an MFT segment even if a file with its File Entry in that section is currently open.
  - NTFS will now defragment for cluster sizes greater than 4K. NTFS will now defragment Reparse points, bitmaps, and attribute\_lists. These can now be opened for file read attributes and synchronize. The files are named using the regular syntax (*file:name:type*); for example:

```
foo:$i30:$INDEX_ALLOCATION
foo::$DATA
foo::$REPARSE_POINT
foo::$ATTRIBUTE_LIST
```

- NTFS's QueryBitmap FSCTL will now return results on a byte boundary rather than page boundary.
- NTFS will now defrag all parts of a stream, up to and including the allocation size. In Windows 2000, it was not possible to defragment the file tail between valid data length (VDL) and end of file (EOF).
- You can now defrag into or out of the MFT Zone. The MFT Zone is now just an NTFS-internal hint for the NTFS allocation engine.
- To defragment a file, the Win32 open mode needs only have `FILE_READ_ATTRIBUTES | SYNCHRONIZE`.
- It is possible to Pin an NTFS file so that it may not be defragged using `FSCTL_MOVE_FILE`. This is done by calling `FSCTL_MARK_HANDLE` and passing `MARK_HANDLE_PROTECT_CLUSTERS` as an argument. This stays in effect until the handle is closed.
- **Large Files.** Windows Whistler and Windows 2000 Service Pack 2 are able to create sections on arbitrarily large mapped files. A constraint that had existed in earlier versions of the memory manager (creating Prototype Page Table entries for all pages in the section) does not apply, because the Windows Whistler memory manager can reuse prototype page table entries (PPTEs) for any parts of a section that do not have a mapped view. In fact, it only creates PPTEs for active views based on the view size (not the section size).
- **Verifiers.** There are new Verifier levels in addition to a new deadlock verifier.
- **Read-only NTFS.** NTFS will now mount read-only on an underlying read-only volume. If the volume requires a log restart or a Chkdsk, the mount will fail.
- **New flag: `FILE_READ_ONLY_VOLUME`.** `GetVolumeInformation()` now returns a `FILE_READ_ONLY_VOLUME` for Read-only volumes.
- **Remote Storage Service (RSS) on MO media.**
- **Encrypting File System (EFS).** The Client Side Caching database can now be encrypted.
- **Default NTFS ACL.** The default access control list (ACL) on NTFS volumes has been strengthened.
- **Read-only flag on directories.** The Read-only attribute has no defined effect on folders. However, in Windows Whistler, Windows Explorer is using this attribute on a folder to indicate that there is extra metadata in the folder that the shell should look at. This is a performance optimization.
- **Write-through mode.** On hot-plug media, the FAT file system will work in WriteThrough mode. This is to eliminate corruption that could occur on media such as CompactFlash when it is unplugged from the system without using the “Safely Remove Hardware” user interface

## Read Only Kernel and HAL Pages

On many Windows Whistler-based systems, the kernel and HAL pages will be marked read-only. This has affected drivers that were attempting to patch system code, dispatch tables, or data structures. The change to read-only kernel and HAL does not happen on all systems:

- On systems with less than 256 MB RAM, the read-only restriction is used.
- On systems with 256 MB or more RAM, the read-only restriction isn't used because Windows Whistler uses large pages to map the kernel and HAL.
- On all systems, the read-only restriction is used for all driver code because drivers are never mapped with large pages.

- Driver Verifier disables large pages, so you can enable this on any machine of any size in order to test your code.

## New APIs

Windows Whistler includes several new filter driver APIs, including:

- **SetShortName()**. This is a new Win32 API to set the short name of a file on NTFS.
- **GetVolumePathNamesForVolumeName()**. New API allows you to list all VolumePaths that a VolumeName may be mounted on.

```

BOOL
GetVolumePathNamesForVolumeName(
    LPCWSTR lpszVolumeName,
    LPWSTR lpszVolumePathNames,
    DWORD cchBufferLength,
    PDWORD lpcchReturnLength
)

```

This routine returns a Multi-Sz list of volume path names for the given volume name. The returned 'lpcchReturnLength' will include the extra trailing null characteristic of a Multi-Sz, unless ERROR\_MORE\_DATA is returned. In such a case, the list returned is as long as possible and may contain a part of a volume path.

### Arguments:

lpszVolumeName	- Supplies the volume name.
lpszVolumePathNames	- Returns the volume path names.
cchBufferLength	- Supplies the size of the return buffer.
lpcchReturnLength	- Returns the number of characters copied back to the return buffer on success or the total number of characters necessary for the buffer on ERROR_MORE_DATA.

### Return Value:

FALSE	- Failure
TRUE	- Success

- **FileIdBothDirectoryInformation()** and **FileIdFullDirectoryInformation()**. These two new FileInfo changes have been added to the file information class enumeration. The new file information classes can be passed as FileInformationClass parameter values to ZwQueryInformationFile, ZwSetInformationFile, and IRP\_MN\_QUERY\_DIRECTORY.
- **SetValid()**. NTFS has the concept of Valid Data Length on a file stream. This is a way to preserve the C2 'Object Reuse' requirement but not force file systems to write zeroes into file-tails. Definitions:
  - VDL = 'Valid Data Length'. Each stream has such a value.
  - EOF = Allocated file length. Each stream has such a value.
  - File Tail = the region from VDL to EOF. Clearly, each stream has such a region and may be zero length.

By definition, VDL must be less than 0 and less than or equal to the EOF. Any reads from the file tail are implicitly returned as zeroes by NTFS. Any writes into the file tail cause VDL to be

increased to equal the end of this write, and any data between the previous VDL and the start of this write are written to be zeroes. In Windows Whistler, we have added an NTFS-only API call to set the Valid Data Length on a file, available to administrative users with `SeManageVolumePrivilege` (described later in this section).

Expected users include:

- A 'Restore' application that has the ability to pour the raw clusters directly onto the disk through hardware channel. This provides a method for informing the file system that the range contains valid user data and it can be returned to the user.
- Multimedia/database tools that need to create large files, but not pay the zero-filling cost during the following times:
  - (a) file extend time (the cost here is to make the extend a synchronous operation)
  - (b) create time (the cost here is filling the file with zeroes)
- Served-metadata cluster file systems that need to remotely extend the file, then 'pump in' the data directly to the disk device.
- **SeManageVolumePrivilege.** The `SeManageVolume` privilege will let non-administrators and remote users do administrative disk tasks on a machine. With Windows Whistler, this privilege is only used to allow non-administrators and remote users make the `SetValidData` call. In the future, it will allow validated users to perform actions on the disk currently restricted to administrators.
- **IoAllocateWorkItem() and IoQueueWorkItem().** These APIs supersede `ExInitializeWorkItem()` and `ExQueueWorkItem()`, and are essential to support driver unloading.
- **IoGetDiskDeviceObject().** Returns the disk device object associated with a file system volume device object. The disk device object need not be an actual disk but in general associated with storage.

---

## Memory Management

Windows Whistler provides improved memory management. The memory manager provides the system services to allocate and free virtual memory, share memory between processes, map files into memory, flush virtual pages to disk, retrieve information about a range of virtual pages, change the protection of virtual pages, and lock the virtual pages into memory. The memory manager also provides a number of services, such as allocating and de-allocating physical memory and locking pages in physical memory for DMA transfers, to other kernel-mode components inside the executive as well as to device drivers.

Memory management enhancements include the following:

- Logical prefetcher for faster boot and application launch
- Enhanced memory management for better scalability
- Reduced paged pool usage
- Increased number of system Page Table Entries (PTEs)
- Support of giant drivers

### Logical Prefetcher for Faster Boot and Application Launch

When a Windows Whistler system is booted, data is saved about all logical disk read operations. On later boots, this information is used to pre-fetch these files in parallel with other boot operations. During boot and application launch, a Windows system demands and pages a sizable amount of data in small chunks (4K to 64K), seeking between files, directories, and metadata.

The Logical Prefetcher, which is new for Windows Whistler, brings much of this data into the system cache with efficient asynchronous disk I/Os that minimize seeks. During boot, the logical prefetcher finishes most of the disk I/Os that need to be done for starting the system in parallel to device initialization delays, providing faster boot and logon performance.

Logical prefetching is accomplished by tracing frequently accessed pages in supported scenarios and efficiently bringing them into memory when the scenario is launched again. When a supported scenario is started, the transition page faults from mapped files are traced, recording which page of a file is accessed. When the scenario has completed (either the machine has booted or the application started), the trace is picked up by a user-mode maintenance service, the Task Scheduler. The information in the trace is used to update or create a prefetch-instructions file that specifies which pages from which files should be prefetched at the next launch.

The user-mode service determines which pages to prefetch by looking at how successful prefetching has been for that scenario in the past, and which pages were accessed in the last several launches of the scenario. When the scenario is run again, the kernel opens the prefetch instructions file and asynchronously queues paging I/O for all of the frequently accessed pages. The actual disk I/Os are sorted by the disk drivers to go up the disk once to load all pages that are not already in memory. This minimizes seeks, cuts down on disk time, and increases performance. The kernel also prefetches the file system metadata for the scenario, for example, MFT entries and directory files. Because prefetching is useful only when the required data is not in memory, the applications that are launched frequently are not traced and prefetched each time.

## Settings for Logical Prefetch

Registry setting:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\Memory Management\PrefetchParameters
RootDirPath - Key Name
Reg_SZ      _ Data Type
Prefetch    - Value <default>
```

Value Names:

```
EnablePrefetcher (DWORD)
0x00000001= application launch prefetching
0x00000002= boot prefetching
```

Parameters are ANDed, so if all were enabled, the setting would be 0x00000003. The setting takes effect immediately. In Windows Whistler Server editions and later versions, only the boot prefetch is enabled by default. Application prefetch can be enabled by the registry setting cited here. The system boot prefetch file is in the %systemroot%\Prefetch directory. Although these prefetch-readable files can be opened using Notepad, they contain binary data that will not be recognized by Notepad. If you are going to view these file, make them read only or copy them to a different location before opening.

## Scalability Improvements Due to Reduced Lock Contention

After significant analysis to identify the resource synchronizations incurring the highest contention, the memory management subsystem has gone through numerous changes for Windows Whistler to reduce Page Frame Number (PFN), address windowing, system PTE, and dispatcher lock contention. The changes removed numerous unnecessary locks and in many cases redesigned the type of locking done, improving both scalability and performance.

## Improved Caching and Backup Due to Dynamic Paged Pool Usage

A major redesign of some internal Memory Manager structures causes substantially less paged pool to be consumed, allowing for greater caching capacity and faster response.

Paged pool is now only allocated while a view is active and even then, only for an amount proportional to the actual view size. When a view is unmapped, that pool is then immediately available for reclaiming if the system detects that overall pool usage is high. This can be done because the pool can be dynamically reallocated and repopulated later if the caller requests the view again. Paged pool used to be allocated for an amount proportional to the section (file) size, regardless of the actual views that were ever used.

- File backup of any size can now be accomplished, from a machine of any size. File back up of any size is possible because prototype PTEs are dynamically constructed and trimmed. It is possible to back up a 500-GB file from a 32-MB client, whereas previously you couldn't back it up at all.
- Much greater availability of paged pool for any component that needs it, since the major amount of it (prototype PTEs) can now automatically grow and shrink dynamically in response to how the system is being used.

## Improved Server Scaling Due to Individual Page Charging

- Greater scalability by reducing memory requirements for some operations. For example, a Web server that serves the same 4K page to 10,000 clients would require 40 MB, so that a lack of space could result in refusing requests. With Windows Whistler, only one page is charged, because the system knows that it's the same page, not 10,000 separate pages. This provides a substantial boost in scalability.

## Improved Terminal Server and Network Server Scaling Due to Increased Number of System PTEs

The number of system Page Table Entries has been increased to a maximum of approximately 1.3 GB, of which just under 1 GB is virtually contiguous. This is approximately twice as many PTEs as were available in Windows 2000 and six times more than Windows NT 4.0. This is subject to system configuration, such as RAM, registry switches, and other factors. Currently, a system must:

- Be booted *without* the **/3GB** switch.
- Have 128 MB or more of RAM.
- Not have the registry key set that allows this RAM to be used for system cache (as it must be used for system PTEs instead).
- Not have registry keys set to make session space or system mapped views larger than the default 48 MB.

On an x86-based systems with 256 MB, the changes made it possible to allocate a 960-MB contiguous mapping. The new mapping is made to keep the PTEs contiguous; by not freeing this second huge chunk of PTEs into the generally available pool until the initial (smaller) PTE pool cannot satisfy a request. Thus, the system will not fragment a big PTE before a driver can access it. Drivers should only experience contention for this resource from another driver that wants a huge chunk, instead of from small random system or driver allocations that happen to occur first.

These extra PTEs also greatly enhance any configuration where PTE usage is critical, for example Terminal Server scenarios, network servers with many concurrent I/Os, and so on.

## Support of Giant Drivers Due to Increased Number of System PTEs

Windows Whistler supports giant driver mappings. Although video drivers are the most obvious benefactors, this also enables other specialized drivers that support large amounts of dedicated RAM. Windows Whistler supports nearly a gigabyte of virtual continuous space for a driver. This compares to support of about 220K for Windows 2000 and about 100K for Windows NT 4.0.

## Support of Direct Execution from ROM

Windows Whistler supports executing applications directly from ROM. This enables products like Windows NT Embedded to ship on ROMs, allowing manufacturers to ship systems without disk drives for specific markets.

---

## Power Management

Windows Whistler brings improvements to the Power Manager, while continuing to support legacy drivers. The Power Manager, which is responsible for managing power usage for the system, administers the system-wide power policy and tracks the path of power IRPs through the system. As with Windows 2000, the Power Manager requests power operations by sending IRP\_MJ\_POWER requests to drivers. A request can specify a new power state or can query whether a change in power state is feasible. When sleep, hibernation, or shut down is required, the Power Manager sends an IRP\_MJ\_POWER request to each leaf node in the device tree requesting the appropriate power action. The Power Manager considers the following in determining whether the system should sleep, hibernate, or shut down:

- System activity level
- System battery level
- Shutdown, hibernate, or sleep requests from applications
- User actions, such as pressing the power button
- Control panel settings

Power management works on two levels, one applying to individual devices and the other to the system as a whole. The Power Manager, part of the operating system kernel, manages the power level of the entire system. If all drivers in the system support power management, the Power Manager can manage power consumption on a system-wide basis, utilizing not only the fully-on and fully-off states, but also various intermediate system sleep states.

Legacy drivers that were written before the operating system included power management support will continue to work as they did previously. However, systems that include legacy drivers cannot enter any of the intermediate system sleep states; they can operate only in the fully on or fully off states as before. Device power management applies to individual devices. A driver that supports power management can turn its device on when it is needed, and off when it is not in use. Devices that have the hardware capability can enter intermediate device power states. The presence of legacy drivers in the system does not affect the ability of newer drivers to manage power for their devices.

## Improved Boot and Logon Performance

Customer research has shown that one of the most frequently requested features that users want from their PCs is fast system startup, whether from cold boot or when resuming from standby or hibernation. When a Windows Whistler system is first booted, data is saved about all logical disk read operations. On later boots, this information is used to pre-fetch these files in parallel with other boot operations. The Windows Whistler operating system has improved PC startup times, which provides opportunities for system manufacturers who want to improve boot and resume times for new PCs. Windows Whistler has several improvements in the boot and resume processes, including:

- Boot loader improvements
- Operating system boot improvements
- Operating system hibernation improvements
- Operating system standby resume Improvements

### Boot Loader Improvements

The key boost to boot loader performance is through optimizing the disk reads. The Windows Whistler boot loader (Ntldr) caches file and directory metadata in large chunks in a most-recently-used manner, which reduces disk seeking. Each system file is now read with a single I/O operation. The resulting improvement in Windows Whistler is that the boot loader is approximately four to five times faster than in Windows 2000. Additionally, redundant PS/2-compatible mouse checks were removed from Ntldr.

Boot loader enhancements also provide similar improvements in hibernation resume times, mainly by streamlining the I/O paths used by Ntldr to read the hibernation image. The hibernation file is compressed as it is written, and for efficiency, the compression algorithm overlaps with the file I/O. However, when resuming from hibernation, Ntldr is using the BIOS to perform the I/O; therefore, it is not feasible to overlap the disk I/O reads with the decompression.

### Operating System Boot Improvements

Optimizing operating system load in Windows Whistler is achieved by overlapping device initialization with the required disk I/Os, and by removing or delaying loading all other processes and services from boot that are unnecessary at boot time. When tuning a system for fast booting, it is crucial to look at both the efficiency of device initialization and the disk I/Os. Windows Whistler initializes device drivers in parallel to improve boot time. Instead of waiting for each device sequentially, many can now be brought up in parallel. The slowest device has the greatest effect on boot. Overlapped device initialization can be viewed with the Bootvis.exe tool.

For more information, see <http://www.microsoft.com/hwdev/fastboot/>

For optimizing boot time, device drivers should only do what is required to initialize the device during boot, and defer all else to post-boot. The goal is that the device is usable after boot, but does not delay boot unnecessarily. Device drivers should never hold spin locks except for very short durations, particularly during boot, as this defeats any parallelism.

Examples of gains made by overlapping device initialization with disk I/O include:

- For PCs not in a domain, network initialization is now done in parallel to boot. Winlogon will not wait for network initialization for PCs in a workgroup; however, those PCs in a domain will still wait.

- Serial Plug and Play is now overlapped. Serial Plug and Play detects attached serial devices. This used to take 2.5 to 3 seconds for normal PCs, and could take several minutes in large servers with lots of ports.
  - Protocol binding is now done in parallel to boot. Previously, NDIS caused boot delays while binding protocols to adapters, due to the adapter negotiating link speed with hubs and switches. This also affects PCs having network adapters without network cables attached.
  - ATAPI boot disk enumeration has been improved from approximately 6 seconds to 2 seconds. Boot disk enumeration stalls all disk I/O during boot.
- In addition, several other services are now initialized in parallel to boot.

## Operating System Hibernation Improvements

During hibernation, all devices are powered off, and the system's physical memory is written to disk in the system hibernation file, \Hiberfil.sys. Before Windows Whistler writes to the hibernation file, all memory pages on the zero, free, and standby lists are freed; these pages do not need to be written to disk. Memory pages are also compressed before being written.

To optimize the hibernation process in Windows Whistler, several improvements have been implemented. The compression algorithm has been optimized to compress and decompress large blocks (64K) of data. In addition, the compression is overlapped with the disk write. As the current data block is being transferred to the disk, the next block of data is being compressed. Overlapping the compression with disk writes makes the compression time virtually free. Also, the hibernation file is written using IDE DMA instead of PIO mode. Most modern IDE controllers and disks achieve their best performance only in DMA mode.

## Operating System Standby Resume Improvements

During resume from standby, the operating system sends S0 IRPs to devices to indicate the change in system power state. As noted in the Power Management section of this paper, device drivers typically request D0 IRPs to change their device power state. The operating system is responsible for notifying each device in the correct order. There are two key ordering rules that must be followed to prevent deadlocks:

- A device cannot be turned on until its parent is turned on.
- All non-paged devices must be turned on before any paged device is turned on.

Because many devices may take significant time to go from D3 to D0, the key to good resume performance is to overlap device initialization as much as possible. The ordering chosen by the operating system is important in maximizing parallelism.

The following changes have been implemented to optimize the resume performance in Windows Whistler:

- The Sx IRP dispatching engine has been rewritten to maximize potential parallelism.
- USER has been changed to dispatch power events asynchronously, instead of tying up a worker thread waiting for them to complete.
- NDIS.SYS has been changed to complete S0 IRPs immediately, instead of waiting for D0 to complete and then initializing on a lower priority thread.
- Worker thread stacks are locked in memory before starting power operations, and unlocked when resume is complete.
- The kernel Plug and Play manager has been changed so it does not tie up a worker thread with an enumerate operation while waiting for power operations to complete.

- Pcmcia.sys, Kbdclass.sys, and Mouclass.sys have been changed to initialize in the background.

## Windows Whistler Boot and Resume Tools

Windows Whistler has the ability to trace boot and resume metrics, and to dump the resulting information to a binary file viewable through Bootvis.exe and other tools. Bootvis.exe displays various time-interlocked graphs showing such things as CPU Usage, Disk I/O, Driver Delays and resume activity. Bootvis.exe can show many types of useful details; the best way to start is by dragging an area on the graph and either double-clicking it or right-clicking to use the context menu to see what options are available. The operating system instrumentation starts about one second after the boot loader loads. Overall boot time shown in Bootvis.exe should add BIOS power-on self test (POST) time + one second. Taking boot traces with Driver Delays will lengthen boot by two to three seconds. The resulting binary file will be several MBs in size.

Information on using Bootvis.exe and improving boot and resume can be found at: <http://www.microsoft.com/hwdev/fastboot/>

---

## Headless Support

Windows Whistler provides native support for "headless server" operation on server platforms—that is, support for operating without a local display or input devices. Microsoft and Intel worked with the computer industry to define firmware and hardware requirements for server operations that include requirements for headless operation under Windows Whistler as part of *Hardware Design Guide Version 3.0 for Microsoft Windows 2000 Server*, which is available for download at <http://www.microsoft.com/hwdev/serverdg.htm>.

Additionally, Microsoft provides detailed documentation on how to design hardware and firmware that integrates well with headless server support included in Windows Whistler. For additional technical information about Windows support for headless operation, see <http://www.microsoft.com/hwdev/headless/>.

## Providing Out-of-Band Management

Windows provides many mechanisms for remotely managing a system when the operating system is loaded, fully initialized, and the system is functioning. This type of system management is called "in-band" management, and it typically occurs over the network. However, a different solution is required for managing a system when Windows network software is unavailable. For example, when the system has failed, when Windows is not loaded, when the system is in the process of loading, or when Windows is fully loaded, but the Windows network software is not available. Remote management that does not rely on Windows network software is called "out of band" management.

The goal for out-of-band management is to always return the system to a fully functioning state where in-band management is available. Making both in-band and out-of-band management available remotely are key components of headless functionality. With the exception of hardware replacement, all administrative functions that can be accomplished locally should be available remotely. For example, the following capabilities should be available for remote management:

- Power on a system that is currently off
- Change BIOS settings or view POST results
- Choose the operating system and set operating system options
- Interact with specific diagnostic and recovery tools

- View system blue-screens
- Reset a system

To accommodate the many possible system operation states, careful thought must be given to hardware, firmware, and operating system functionality to ensure a comprehensive and complementary implementation of out-of-band management capabilities. Each system component must contribute to a coherent, complementary solution in order to avoid expensive, confusing administration.

## Implementing Headless Support in Windows Whistler

A headless solution requires that local console I/O dependencies be removed from the operating system. Windows Whistler supports operating without a keyboard, mouse, or monitor attached to the system. On an ACPI-enabled system, Windows Whistler supports operating without a legacy 8042 keyboard controller. Windows Whistler also includes support for systems without VGA/display hardware.

Based on built-in Windows support, a USB-capable system running Windows Whistler can support hot plugging the mouse and keyboard while the operating system is running, and a system with a VGA card can support hot plugging a monitor.

In Windows Whistler, out-of-band console I/O support is provided for all Windows kernel components—the loader, setup, recovery console, operating system kernel, and blue-screens—and for a text-mode management console available after Windows Whistler has initialized, which is called the Special Administration Console. This functionality is called the Emergency Management Services (EMS). This I/O conforms to the following by default:

- 9600 baud
- 1 stop bit
- No parity
- 8 data bits
- Hardware flow control

- Output conventions are defined in “Standardizing Out-of-Band Management Console Output and Terminal Emulation (VT-UTF8 and VT100+),” available at <http://www.microsoft.com/hwdev/headless/>.

**Note:** These settings were chosen because they are the current “standard” configuration in UNIX administration and allow interoperability. However, additional settings are supported and may be passed via the Serial Port Console Redirection Table; documentation is available on the web at <http://www.microsoft.com/hwdev/headless/>.

## Designing Hardware and Firmware

There are three key areas that must be addressed to provide a high-quality headless platform that is cleanly integrated with Windows Whistler EMS:

- **Management port hardware:** Windows Whistler will support a standard legacy serial port.

The headless functionality in Windows Whistler supports a standard 16550 serial port at the legacy address of COM1, COM2, COM3, or COM4, 3F8, 2F8, 3E8, and 2E8, respectively, or a UART at well-known non-legacy address. A server must provide at least one UART interface capable of being used by Windows Whistler EMS. See the item "Passing of the out-of-band communication channel configuration information from the firmware to Windows Whistler" below for more information on making the location of the UART well known. A separate debug port must also be available and may not be the same port as used for EMS.

When using a serial port connection to access Windows EMS support, all serial cables must be null modem cables that support the Carrier Detect (CD) signal. Windows will not provide console I/O if Carrier Detect is not active.

**Note:** Cables with Carrier Detect shorted to Ready to Send (RTS) will work.

- **Standardized firmware and service processor management console user interfaces:** It is important the system firmware and out-of-band management service processor user interface be available via the same out-of-band communication channel as the EMS. The firmware must cleanly hand off the communication channel to the EMS during the boot process. The firmware must use the same terminal definition as Windows Whistler EMS, which includes using the same keystrokes to represent not standardized keystrokes, such as F12.
- **Passing of the out-of-band communication channel configuration information from the firmware to Windows Whistler:** This is achieved via the Serial Port Console Redirection Table or optionally, on 64-bit systems, the EFI Console Device Path.

For more information, see: <http://www.microsoft.com/hwdev/headless/>.

---

## Hot-Add Memory and Memory Mirroring Support

Windows Whistler introduces hot-add memory support, which will allow memory devices to be added to a machine and be made available to the operating system and applications as part of the normal memory pool—without turning the machine off or rebooting.

Hot-add memory support will be especially appreciated by enterprise system administrators who need to add memory resources to keep up with production demands, but are working with systems that require continuous (24x7) availability. This action would typically require shutting down the system and interrupting service. With the hot-add feature, service performance can be increased without service interruption.

Removal of memory regions will not be supported in Windows Whistler. A memory upgrade, which requires removing and then adding memory, will also not be supported in Windows Whistler.

Memory mirroring (different from hot-add) is also introduced in Whistler, providing increased availability for extremely high-end systems (for example, Stratus, and so on).

**CAUTION:** This feature will operate only on servers that have hardware support for adding memory while the server is operating. Most existing servers do not have this type of hardware support and could be damaged if memory is installed while the power is on. Consult the server operator's manual for more information. A small number of new servers will offer the ability to add memory while the server is operating. For these servers, the act of installing memory will automatically invoke the hot-add memory feature in the operating system.

---

## ccNUMA Support

Windows Whistler supports ccNUMA and NUMA-"lite" designs. This provides for near:far memory access time ratios of 1:3 or less.

**Note:** For optimal performance with Windows 2000 and later operating systems, it is recommended that system designers who are building platforms that present memories with different access times keep the ratio for access to "near" versus "far" memories relative to a given microprocessor at a 1:3 ratio or less, as seen by the operating system.

As processor speeds relative to bus speeds and the number of processors in symmetric multi-processor (SMP) systems have increased, the system bus and main memory have become a significant bottleneck. The trend in High Performance Computing (HPC) over the last decade has been to reduce dependence on a single system bus or system memory by building smaller systems (nodes), each with their own processors and memory, then providing a high-speed cache coherent interconnect network to form a larger system. Access from a processor to node local memory is fast, as is the case with small SMP systems. Access to memory in another node is much slower. Earlier systems using this approach could result in as much as a 50x *decrease* in performance for remote versus local access.

Today, the difference has been reduced significantly, but is still (usually) in the range of two to three times as long to access remote memory (versus local). Operating systems such as Windows 2000, which have been designed for SMP systems, can work acceptably on these newer systems; but significant performance gains are possible if the operating system is aware of underlying architecture. Windows Whistler contains code to take advantage of the memory available in both the current processor node as well as memory assigned to other processor nodes.

## Per Node Memory Allocation

An effective method for improving performance on a ccNUMA machine is to ensure that the processors use the memory closest to them. This guideline includes continuously running threads on the same processor node. Page coloring is used in Windows 2000 to ensure that page allocation is spread as much as possible throughout the physical address space of the system. To achieve good page placement in a ccNUMA system, each node will have its own palette of colors. Page allocation is round-robin within the palette of the node containing the threads ideal processor. If no pages are available in the palette, then we will round-robin from all colors in all palettes.

## Per Node Scheduling

The thread scheduler has been changed to examine and favor the following processor configurations:

- A thread's ideal processor
- Any real processor that is in the same node as the threads ideal processor
- Any logical processor that is in the same node as the threads ideal processor
- The highest numbered processor in the same node as the threads ideal processor

If this prioritization fails, then Windows Whistler will only consider other processor nodes if the average processor utilization of the other nodes is lower than the average utilization of the ideal node.

## OEM Support for ccNUMA

For Windows Whistler to utilize these additions for proper ccNUMA support, original equipment manufacturers (OEMs) must supply a HAL to identify and talk to the hardware in a way that Windows Whistler can understand. There is a BIOS feature that, if used on the OEM hardware,

can function properly out of the box with the Itanium-based HAL that ships with Windows Whistler. This is the Static Resource Affinity Table (SRAT).

## Static Resource Affinity Table

With the current Enterprise market trend to build large-scale systems out of “nodes” of smaller scale systems, the best performance is typically achieved when processors use memory that is physically located in the same smaller system node as the processor, rather than using memory located in other nodes.

SRAT, as described in the ACPI specification, can be used to describe the physical location of processors and memory in large-scale systems (such as ccNUMA) to Windows, allowing threads and memory to be grouped in an optimal manner.

The SRAT contains topology information for all the processors and memory present in a system at system boot, including memory that can be hot removed. The topology information identifies the sets of processors and physical memory ranges in the system, enabling a tight coupling between the processors and memory ranges in a node.

The ACPI 2.0 specification introduces the concept of proximity domains within a system. Devices in a system that belong to a proximity domain are tightly coupled, or “closer,” to each other than to other devices in the system. For example, in a ccNUMA machine consisting of several nodes interconnected through a switch, the latency of memory accesses from a processor to memory on the same node would typically be shorter than the latency of memory accesses on other nodes. The operating system should use this affinity information to determine the allocation of memory resources and the scheduling of software threads, therefore improving performance on large systems.

Including the methods that provide affinity information in the ACPI Namespace enables a generic mechanism that provides topology information. This information can be used to resolve scenarios in which components such as processors and memory can be hot added and hot removed from the system.

The Proximity method (`_PXM`) included in the ACPI 2.0 specification enables the specification of affinity between various devices in the ACPI Namespace. The `_PXM` method associated with a device in the ACPI Namespace returns a domain number. All devices that return the same domain number belong to the same proximity domain. In addition, specifying memory, processor devices, and the methods for identifying and removing these devices, enables the infrastructure for supporting hot-plug memory. However, defining the proximity and hot-plug information in the ACPI Namespace (instead of static ACPI tables) makes it unavailable to the operating system until the ACPI components of the operating system load.

Depending on the load order of the ACPI components with respect to the operating system construction of memory pools and processor startup, affinity and memory hot-plug information in the ACPI Namespace might not be available at the desired system startup phase. The SRAT addresses this problem and makes the affinity and hot-remove memory information available to the operating system in a static fashion before the ACPI components load. This information is needed only at boot time; only processors and memory present at boot time are represented. Unpopulated “slots” for memory and processors that can be hot added do not need to be represented.

The SRAT uses the concept of proximity domains to provide resource affinity information, similar to the domains used in the ACPI 2.0 specification. However, the domain numbers provided by the SRAT do not need to be identical to those used in the ACPI 2.0 Namespace. The SRAT is intended to enable operating system optimizations for the Windows Datacenter Server class of

machines with multiple processors and ccNUMA characteristics. These machines must also support the APIC/SAPIC interrupt model.

## How the Operating System Can Use SRAT

The SRAT is provided as an intermediate step to enable features such as ccNUMA optimizations on Windows Datacenter Server class machines until the capabilities of accessing the ACPI 2.0 Namespace are available at the desired startup phase. Until these capabilities are available, the operating system will scan and use the information provided in the SRAT at boot time. However, after these capabilities are integrated in the operating system, the operating system will no longer use the SRAT or the information provided in the ACPI namespace.

The SRAT provides data for the performance optimization of ccNUMA systems. The use of fine-grained interleaving, in which the memory is interleaved across the nodes at a small granularity, is mutually exclusive with respect to ccNUMA optimizations. The BIOS should not provide an SRAT if fine-grained interleaving is enabled. Fine-grained interleaving should be disabled if the SRAT is provided. It is expected that ccNUMA optimization will provide superior performance over fine-grained interleaving. Furthermore, the operating system will not support hot-plugging memory in the presence of interleaving.

The operating system will scan the SRAT only at boot time. The BIOS/system is expected to retain the proximity information used by the operating system at boot time across the lifetime of the operating system instance (until the next boot). This specifically implies that the BIOS/system retains the system topology across a system sleep state transition in which processor or memory context is lost (S2–S5), such that the proximity information provided by the SRAT at operating system boot time is valid across the transition. If the system topology is not retained, memory viewed as “local” to a set of processors could potentially become “non-local” memory on return from the sleep state and have an adverse impact on system performance.

The BIOS can use the Prepare to Sleep (`_PTS`) method to initiate the process of saving all information necessary to replicate the system topology. The BIOS can also use the sleep state argument that is provided by the operating system to the `_PTS` method to determine if the state save is required. If memory or processor hot-plug were not supported, providing a deterministic selection of boot processor/processor order in the Multiple APIC Description Table (MADT) and programming of the memory regions would be sufficient to ensure identical system topology on return from the sleep state.

The BIOS must reconstruct the SRAT on system reboot similar to the MADT.

---

## New Support for Hardware

Windows Whistler supports the Extensible Firmware Interface (EFI), a new standard for the interface provided by the firmware that boots PCs. Microsoft will support EFI as the only firmware interface to boot the 64-bit version of Windows for Itanium-based systems. Because the 64-bit version of Windows will not boot with BIOS or with the system abstraction layer (SAL) alone, EFI is a requirement for all Itanium-based systems to boot Windows.

Windows Whistler also supports the Globally Unique Identifier (GUID) Partition Table (GPT), which was introduced as part of the EFI initiative. GPT complements the older Master Boot Record (MBR) partitioning scheme that has been common to PCs. GPT allows use of very large disks. The number of partitions on a GPT disk is not constrained by temporary schemes such as container partitions, as defined by the MBR Extended Boot Record (EBR).

The GPT disk partition format is well defined and fully self-identifying. Data critical to platform operation is located in partitions and not in unpartitioned or hidden sectors. GPT disks use

primary and backup partition tables for redundancy and CRC32 fields for improved partition data structure integrity. The GPT partition format uses version number and size fields for future expansion. Each GPT partition has a GUID and a partition content type, so no coordination is necessary to prevent partition identifier collision. Each GPT partition has a 36-character Unicode name, which means that any software can present a human-readable name for the partition without any additional understanding of the partition.

For information about the EFI standard, see <http://www.microsoft.com/hwdev/efi/>.

---

## Intel Itanium Support

Windows Whistler for Intel Itanium-based systems is a fully featured 64-bit operating system that is compatible with most existing 32-bit applications. The 64-bit Windows operating system will provide high availability, advanced scalability, and large memory support based on the Intel Itanium chip with its extensive multiprocessing features, powerful floating-point arithmetic extensions, and multimedia-specific instructions. 64-bit Windows and the Itanium microprocessor are designed to address the most demanding business needs of today's Internet-based world including e-commerce, data mining, online transaction-processing, memory-intensive high-end graphics, complex mathematics, and high-performance multimedia applications.

The Microsoft vision is to make a broad portfolio of applications available, including leading Microsoft applications on 64-bit Windows. To achieve this goal, Microsoft will provide a rich set of development tools that will make it easy to write new applications and port existing ones. The 64-bit Windows platform will bring the following benefits to developers and end users:

- The full advantage of Intel Itanium-based architecture for reliability, high performance, and high availability
- Compatibility with Windows 2000-compatible applications and existing Win32-based applications
- API-level compatibility between the Microsoft Win64® API and the Win32 API
- Scalability of virtual memory up to 16 terabytes (TB)
- Interoperability with systems based on existing 32-bit architectures

## Benefiting from 64-bit Architecture

A 64-bit operating system supports far more virtual memory than a 32-bit operating system. For example, 32-bit Windows Whistler supports 4 GB of virtual memory, while 64-bit Windows supports 16 TB of virtual memory. Non-paged pool increases substantially, up to 128 GB for the 64-bit platform compared to 256 MB maximum for the 32-bit platform. With these new higher limits, the scalability that the 64-bit platform offers is enormous in terms of terminal server clients, page pools, network connections, and so on.

- Each application can support more users. All or part of each application must be replicated for each user, which requires additional memory.
- Each application has access to larger amounts of memory, which can increase performance in some scenarios.
- Each application has more memory for data storage and manipulation. Databases can store more of their data in the physical memory of the system. Data access is faster because disk reads are not necessary.
- Applications can manipulate large amounts of data easily and more reliably. Video composition for motion picture work requires 64-bit Windows for this reason. Modeling for

scientific and financial applications benefits greatly from memory-resident data structures that are not possible on 32-bit versions of Windows.

There are also important benefits for businesses:

- **Increased productivity.** Knowledge workers can spend their time thinking and producing, rather than waiting for the software to finish its tasks.
- **Lower cost of ownership.** Each server can support larger numbers of users and applications, so your business will require fewer servers. This translates directly into less management overhead—one of the highest costs in any computing environment.
- **New application opportunities.** New applications can be designed without the barriers imposed by 32-bit Windows. Data-intensive tasks that are impossible today can be done with 64-bit Windows.

## Designing for 64-Bit Compatible Interfaces

Porting drivers or software created for 32-bit Windows to 64-bit Windows should not create any problems for distributed applications, whether they use Remote Procedure Calls (RPC) or DCOM. The RPC programming model specifies well-defined data sizes and integer types that are the same size on each end of the connection. In addition, in the LLP64 abstract data model developed for 64-bit Windows, only the pointers expand to 64 bits—all other integer data types remain 32 bit. Because pointers are local to each side of the client/server connection and are usually transmitted as NULL or non-NULL markers, the marshaling engine can handle different pointer sizes on either end of a connection transparently.

However, backward compatibility issues arise when you add new data types or methods to an interface, change old data types, or use data types inappropriately.

For more information on developing for the 64-bit system, see:

[http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/buildapp/64bitwin\\_410z.htm](http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/buildapp/64bitwin_410z.htm)

---

## Components Removed

Several Windows 2000 components are removed in Windows Whistler:

- Posix subsystem
- OS/2 subsystem
- NEC PC98 support
- SGI 320 and SGI 540 support

---

## "Designed for Windows Whistler" Logo Program Requirements

Requirements for the Windows Logo Program for hardware that apply specifically for systems or peripherals that will receive the "Designed for Windows Whistler" are defined in *Microsoft Windows Logo Program System and Device Requirements, Version 2.0*, available on the web site at <http://www.microsoft.com/hwdev/winlogo/>.

Windows Hardware Quality Labs (WHQL) is providing the Hardware Compatibility Tests (HCTs) for Windows Whistler. For information, see <http://www.microsoft.com/hwtest/>.

## References

For the Windows Whistler debugger, see:  
<http://www.microsoft.com/ddk/debugging/>

For more information on improving boot and resume, see:  
<http://www.microsoft.com/hwdev/fastboot/>

For more information on headless support, see:  
<http://www.microsoft.com/hwdev/headless/>

For more information on the PCI Hot-Plug Specification, see:  
<http://www.pcisig.com>

For more information on the ACPI specification, see:  
<http://www.microsoft.com/hwdev/onnow>

For more information on developing for the 64-bit system, see:  
[http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/buildapp/64bitwin\\_410z.htm](http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/buildapp/64bitwin_410z.htm)

For information about memory and I/O resource allocations for PCI-to-PCI bridges, see “PCI-to-PCI Bridges and CardBus Controllers on Windows 2000 and Windows Whistler,” to be published at <http://www.microsoft.com/hwdev/PCI>.

---

## Acronyms

ACL – access control list

ACPI – Advanced Configuration and Power Interface

API – application programming interface

APM – Advanced Power Management

BIOS – basic input/output system

ccNUMA – Cache Coherent – Non Uniform Memory Architecture

CD – Carrier Detect

DDK – Driver Development Kit

DMA – direct memory access

DVD-RAM – Digital Video Disk Random Access Memory

EBR – Extended Boot Record

EFI – Extensible Firmware Interface

EFS – Encrypting File System

EMS – Emergency Management Services

EOF – end of file

FAT – file allocation table

GPT – GUID partition table

GUID – globally unique identifier

HAL – hardware abstraction layer  
HCT – Hardware Compatibility Test  
HPC – high performance computing  
HTTP – Hypertext Transfer Protocol  
IDE – Integrated Device Electronics  
IFS – Installable File System  
IHV – independent hardware vendors  
IIS – Microsoft Internet Information Server  
I/O – input/output  
IOCTL – I/O Control  
IRP – I/O request packet  
ISV – independent software vendor  
KCB – kernel control block  
LED – light-emitting diode  
LPC – local procedure call  
LRU – least recently used  
MADT – multiple APIC description table  
MBR – Master Boot Record  
MDL – Memory Descriptor List  
MFT – master file table  
MSN – Microsoft Network  
NDIS – Network Driver Interface Specification  
NTFS – NT File System  
Ntldr – NT Loader  
OEM – original equipment manufacturer  
PCI – Peripheral Component Interconnect  
PFN – page frame number  
PIO – programmed I/O  
POST – power-on self test  
PPTE – prototype page table entries  
PTE – page table entries  
RAM – random access memory

RPC – remote procedure call

RSS – Remote Storage Service

RTS – Ready to Send

SAL – system abstraction layer

SMP – symmetric multi-processor

SRAT – static resource affinity table

TB – Terabyte

USB – universal serial bus

VDL – valid data length

WDM – Windows Driver Model

WHQL – Windows Hardware Quality Labs