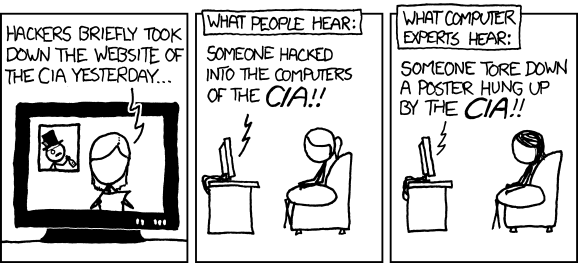
***91st Network Warfare Squadron*** 



**Offensive Cyberspace Operations**

**Interactive Operator Training Annex**

****

**18 March 2015**

**Table of Contents**

|  |  |
| --- | --- |
| **Packet Filtering Fundamentals** | **Page 3** |
| **Packet Analysis Tools** | **Page 31** |
| **Windows Operating System Fundamentals** | **Page 51** |
| **Unix/Linux Operating System Fundamentals** | **Page 61** |
| **OCO Mission Phases** | **Page 71** |
| **OCO Emergency Actions** | **Page 81** |
| **Appendix 1 – Common Ports/Protocols** | **Page 91** |

|  |  |
| --- | --- |
| **Appendix 2 – Packet Crafting** | **Page 10** |

|  |  |
| --- | --- |
| **Appendix 3 – NTFS Meta Files** | **Page 11** |
| **Appendix 4 – Common Windows CMD Commands** | **Page 12** |
| **Appendix 5 – Common Windows Binaries** | **Page 13** |
|  |  |

Packet Filtering Fundamentals

Berkeley Packet Filter

The Berkeley Packet Filter (BPF) provides on some Unix-like systems a raw interface to data link layers, permitting raw link-layer packets to be sent and received. In addition, if the driver for the network interface supports promiscuous mode, it allows the interface to be put into that mode so that all packets on the network can be received, even those destined to other hosts.

In addition, it supports "filtering" packets, so that only "interesting" packets will be supplied to the software using BPF; this can avoid copying "uninteresting" packets from the operating system kernel to software running in user mode, reducing the CPU requirement to capture packets and the buffer space required to avoid dropping packets.

Packet Analysis Techniques

On wired broadcast LANs, depending on the network structure (hub or switch), one can capture traffic on all or just parts of the network from a single machine within the network; however, there are some methods to avoid traffic narrowing by switches to gain access to traffic from other systems on the network (e.g., ARP spoofing). For network monitoring purposes, it may also be desirable to monitor all data packets in a LAN by using a network switch with a so-called monitoring port, whose purpose is to mirror all packets passing through all ports of the switch when systems (computers) are connected to a switch port. To use a network tap is an even more reliable solution than to use a monitoring port, since taps are less likely to drop packets during high traffic load.

On wireless LANs, one can capture traffic on a particular channel, or on several channels when using multiple adapters.

On wired broadcast and wireless LANs, to capture traffic other than unicast traffic sent to the machine running the sniffer software, multicast traffic sent to a multicast group to which that machine is listening, and broadcast traffic, the network adapter being used to capture the traffic must be put into promiscuous mode; some sniffers support this, others do not. On wireless LANs, even if the adapter is in promiscuous mode, packets not for the service set for which the adapter is configured will usually be ignored. To see those packets, the adapter must be in monitor mode.

When traffic is captured, either the entire contents of packets can be recorded, or the headers can be recorded without recording the total content of the packet. This can reduce storage requirements, and avoid legal problems, but yet have enough data to reveal the essential information required for problem diagnosis.

The captured information is decoded from raw digital form into a human-readable format that permits users of the protocol analyzer to easily review the exchanged information. Protocol analyzers vary in their abilities to display data in multiple views, automatically detect errors, determine the root causes of errors, generate timing diagrams, reconstruct TCP and UDP data streams, etc.

Some protocol analyzers can also generate traffic and thus act as the reference device; these can act as protocol testers. Such testers generate protocol-correct traffic for functional testing, and may also have the ability to deliberately introduce errors to test for the DUT's ability to deal with error conditions.

Protocol analyzers can also be hardware-based, either in probe format or, as is increasingly more common, combined with a disk array. These devices record packets (or a slice of the packet) to a disk array. This allows historical forensic analysis of packets without the users having to recreate any fault.

Packet Analysis Tools

TCPdump

Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight.

WINdump

WinDump is the Windows version of tcpdump, the command line network analyzer for UNIX. WinDump is fully compatible with tcpdump and can be used to watch, diagnose and save to disk network traffic according to various complex rules. It can run under Windows 95, 98, ME, NT, 2000, XP, 2003 and Vista.

WinDump captures using theWinPcap library and drivers, which are freely downloadable from the WinPcap.org website. WinDump supports 802.11b/g wireless capture and troubleshooting through the Riverbed AirPcap adapter.

WinDump is free and is released under a BSD-style license.

tShark

TShark is a network protocol analyzer. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark's native capture file format is pcap format, which is also the format used by tcpdump and various other tools.

Without any options set, TShark will work much like tcpdump. It will use the pcap library to capture traffic from the first available network interface and displays a summary line on stdout for each received packet.

TShark is able to detect, read and write the same capture files that are supported by Wireshark. The input file doesn't need a specific filename extension; the file format and an optional gzip compression will be automatically detected.

dumpcap

Dumpcap is a network traffic dump tool. It lets you capture packet data from a live network and write the packets to a file. Dumpcap's default capture file format is pcap-ng format.

Wireshark

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the GTK+ widget toolkit in current releases, and Qt in the development version, to implement its user interface, and using pcap to capture packets; it runs on Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

Windows

Boot Process

**Windows 3.x/9x**

In Windows 3.x and 95/98/ME, the boot loader phase is handled by MS-DOS. During the boot phase, CONFIG.SYS and AUTOEXEC.BAT are executed, along with the configuration settings files WIN.INI and SYSTEM.INI. Virtual device drivers are also loaded in the startup process: they are most commonly loaded from the registry (HKLM\System\CurrentControlSet\Services\VxD) or from the SYSTEM.INI file.

When all system configuration files and device drivers have been loaded, the 16-bit modules, KRNL386.EXE, GDI.EXE, and USER.EXE, are loaded, then the 32-bit DLLs (KERNEL32.DLL, GDI32.DLL, and USER32.DLL) are loaded. The 32-bit VxD message server (MSGSRV32) starts MPREXE.EXE, which is responsible for loading the network logon client (such as Client for Microsoft Networks, Microsoft Family Logon or Windows Logon).

When a user is logging on to Windows, the startup sound is played, the shell (usually EXPLORER.EXE) is loaded from the [boot] section of the SYSTEM.INI file, and startup items are loaded.

In all versions of Windows 9x except ME, it is also possible to load Windows by booting to a DOS prompt and typing "win". There are some command line switches that can be used with the WIN command: with the /D switch, Windows boots to safe mode, and with the /D:n switch, Windows boots to safe mode with networking. The latter switch only works properly with Windows 95. In Windows 3.1, additional options are available, such as /3, which starts Windows in 386 enhanced mode, and /S, which starts Windows in standard mode.

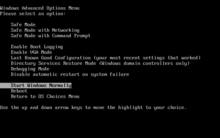
A startup sound was added in Windows 3.1.

**Windows NT**

**Boot Loader Phase**

Windows NT startup process starts when the computer finds a *Windows boot loader*, a portion of Windows operating system responsible for finding Microsoft Windows and starting it up. On IA-32or x64 systems, the boot loader is called Windows Boot Manager (BOOTMGR). Prior to Windows Vista however, the boot loader was NTLDR. Microsoft has also released operating systems for Intel processors which use IA-64 architecture. The boot loader of these editions of Windows is IA64ldr.efi (later referred as simply IA64ldr). It is an Extensible Firmware Interface(EFI) program.

**Operating system selection**

[](http://en.wikipedia.org/wiki/File:Windows_Advanced_Options_menu.png)

**NTLDR Bootloader's Advanced Option Menu**

The boot loader, once executed, searches for a Windows operating system. Windows Boot Manager does so by reading [Boot Configuration Data](http://en.wikipedia.org/wiki/Boot_Configuration_Data) (BCD), a complex firmware-independent database for boot-time configuration data. Its predecessor, NTLDR, does so by reading the simpler [boot.ini](http://en.wikipedia.org/wiki/Boot.ini). If the boot.ini file is missing, the boot loader will attempt to locate information from the standard installation directory. For Windows NT and 2000 machines, it will attempt to boot from C:\WINNT. For Windows XP and 2003 machines, it will boot from C:\WINDOWS.

Both databases may contain a list of installed Microsoft operating systems that may be loaded from the local hard disk drive or a remote computer on the [local network](http://en.wikipedia.org/wiki/Local_Area_Network). NTLDR supports operating systems installed on disks whose file system is [NTFS](http://en.wikipedia.org/wiki/NTFS) or [FAT](http://en.wikipedia.org/wiki/File_Allocation_Table) file systems, CDFS ([ISO 9660](http://en.wikipedia.org/wiki/ISO_9660)) or [UDFS](http://en.wikipedia.org/wiki/Universal_Disk_Format).[[12]](http://en.wikipedia.org/wiki/Windows_NT_startup_process#cite_note-12) Windows Boot Manager also supports operating systems installed inside a[VHD](http://en.wikipedia.org/wiki/VHD_(file_format)) file, stored on an NTFS disk drive.

In the Windows 2000 or in later versions of Windows which [hibernation](http://en.wikipedia.org/wiki/Hibernation_(computing)) is supported, the Windows boot loader starts the search for operating systems by searching for *hiberfil.sys*. NTLDR looks into the [root folder](http://en.wikipedia.org/wiki/Root_directory) of the default volume specified in boot.ini. Windows Boot Manager looks up the location of hiberfil.sys in BCD. If this file is found and an active memory set is found in it, the boot loader loads the contents of the file (which is a compressed version of a physical memory dump of the machine) into memory and restores the computer to the state that it was prior to hibernation.

Next, the boot loader looks for a list of installed operating system entries. If [more than one operating system is installed](http://en.wikipedia.org/wiki/Multi_boot), the boot loader shows a boot menu and allows the user to select an operating system. If a non NT-based operating system such as [Windows 98](http://en.wikipedia.org/wiki/Windows_98) is selected (specified by an [MS-DOS](http://en.wikipedia.org/wiki/MS-DOS) style of path, e.g. C:\), then the boot loader loads the associated "boot sector" file listed in *boot.ini* or BCD (by default, this is *bootsect.dos* if no file name is specified) and passes execution control to it. Otherwise, the boot process continues.

**Loading the Windows NT kernel**

The operating system starts when certain basic drivers flagged as "Boot" are loaded into memory. The appropriate file system driver for the partition type (NTFS, FAT, or FAT32) which the Windows installation resides are amongst them. At this point in the boot process, the boot loader clears the screen and displays a textual progress bar, (which is often not seen due to the initialization speed); Windows 2000 also displays the text "Starting Windows..." underneath. If the user presses F8 during this phase, the advanced boot menu is displayed, containing various special boot modes including [Safe mode](http://en.wikipedia.org/wiki/Safe_mode), with the Last Known Good Configuration, with debugging enabled, and (in the case of Server editions) Directory Services Restore Mode. Once a boot mode has been selected (or if F8 was never pressed) booting continues.

The following files are loaded sequentially.

1. ntoskrnl.exe (the kernel)
2. hal.dll (type of hardware abstraction layer)
3. kdcom.dll (Kernel Debugger HW Extension DLL)
4. bootvid.dll (for the windows logo and side-scrolling bar)
5. config\system (one of the registry hives)

Next, NTDETECT.COM and the Windows NT kernel ([*Ntoskrnl.exe*](http://en.wikipedia.org/wiki/Ntoskrnl.exe)) and the [Hardware Abstraction Layer](http://en.wikipedia.org/wiki/Hardware_Abstraction_Layer) ([*hal.dll*](http://en.wikipedia.org/wiki/Hal.dll)) are loaded into memory. If multiple hardware configurations are defined in the Windows, the user is prompted at this point to choose one.

With the kernel in memory, boot-time device drivers are loaded (but not yet initialized). The required information (along with information on all detected hardware and Windows Services) is stored in the HKEY\_LOCAL\_MACHINE\System portion of the registry, in a set of registry keys collectively called a *Control Set*. Multiple control sets (typically two) are kept, in the event that the settings contained in the currently-used one prohibit the system from booting. HKEY\_LOCAL\_MACHINE\System contains control sets labeled ControlSet001, ControlSet002, etc., as well as CurrentControlSet. During regular operation, Windows uses CurrentControlSet to read and write information. CurrentControlSet is a reference to one of the control sets stored in the registry. Windows picks the "real" control set being used based on the values set in the HKLM\SYSTEM\Select registry key:

* Default will be the boot loader's choice if nothing else overrides this
* If the value of the Failed key matches Default, then the boot loader displays an error message, indicating that the last boot failed, and gives the user the option to try booting anyway, or to use the "Last Known Good Configuration".
* If the user choose (or has chosen) Last Known Good Configuration, the control set indicated by the LastKnownGood key is used instead of Default.

When a control set is chosen, the Current key gets set accordingly. The Failed key is also set to the same as Current until the end of the boot process. LastKnownGood is also set toCurrent if the boot process completes successfully.

Which services are started and the order each group is started are provided by the following keys:

* HKLM\SYSTEM\CurrentControlSet\Services
* HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder

For the purposes of booting, a driver may be one of the following:

1. A "Boot" driver that is loaded by the boot loader prior to starting the kernel. "Boot" drivers are almost exclusively drivers for hard-disk controllers and file systems ([ATA](http://en.wikipedia.org/wiki/AT_Attachment), [SCSI](http://en.wikipedia.org/wiki/SCSI), file system filter manager, etc.); in other words, they are the absolute minimum that the kernel will need to get started with loading other drivers, and the rest of the operating system.
2. A "System" driver which is loaded and started by the kernel after the boot drivers. "System" drivers cover a wider range of core functionality, including the display driver, CD-ROM support, and the TCP/IP stack.
3. An "Automatic" driver which is loaded much later when the GUI already has been started.

With this finished, control is then passed from the boot loader to the kernel.

**Kernel Phase**

The initialization of the kernel subsystem and the Windows Executive subsystems is done in two phases.

During the first phase, basic internal memory structures are created, and each CPU's [interrupt controller](http://en.wikipedia.org/wiki/Interrupt_controller) is initialized. The memory manager is initialized, creating areas for the file system cache, [paged](http://en.wikipedia.org/wiki/Paging) and non-paged pools of memory. The [Object Manager](http://en.wikipedia.org/wiki/Object_Manager_(Windows)), initial [security token](http://en.wikipedia.org/wiki/Token_(Windows_NT_architecture)) for assignment to the first [process](http://en.wikipedia.org/wiki/Process_(computing)) on the system, and the Process Manager itself. The System processes are created at this point.

The second phase involves initializing the device drivers which were identified by [NTLDR](http://en.wikipedia.org/wiki/NTLDR) as being system drivers.

Through the process of loading device drivers, a "progress bar" is visible at the bottom of the display on Windows 2000 systems; in Windows XP and Windows Server 2003, this was replaced by an animated bar which does not represent actual progress. Prior to Windows XP, this part of the boot process took significantly longer; this is because the drivers would be initialized one at a time. On Windows XP and Server 2003, the drivers are all initialized asynchronously.

**Session Manager**

Once all the Boot and System drivers have been loaded, the kernel (system thread) starts the[Session Manager Subsystem](http://en.wikipedia.org/wiki/Session_Manager_Subsystem) (smss.exe).

Before any files are opened, Autochk is started by *smss.exe*. Autochk mounts all drives and checks them one at a time to see whether or not they were cleanly unmounted. If autochk determines one or more volumes are dirty, it will automatically run chkdsk and provides the user with a short window to abort the repair process by pressing a key within 10 seconds (introduced in Windows NT 4.0 Service Pack 4; earlier versions would not allow the user to abort chkdsk). Since Windows 2000, XP and 2003 show no text screen at that point (unlike NT, which displayed a blue text screen), the user will see a different background picture holding a mini-text-screen in the center of the screen and show the progress of chkdsk there.

At boot time, the Session Manager Subsystem:

* Creates environment variables (HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment)
* Starts the kernel-mode side of the Win32 subsystem (win32k.sys). This allows Windows to switch into graphical mode as there is now enough infrastructure in place.
* Starts the user-mode side of the Win32 subsystem, the [Client/Server Runtime Server Subsystem](http://en.wikipedia.org/wiki/Client/Server_Runtime_Server_Subsystem) (csrss.exe). This makes Win32 available to user-mode applications.
* Creates [virtual memory](http://en.wikipedia.org/wiki/Virtual_memory) paging files (HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management)
* Performs any rename operations that are queued up. This allows previously in-use files (e.g. drivers) to be replaced as part of a reboot.
* Executes any programs listed in HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\BootExecute such as autocheck and convert.
* Starts the [Windows Logon Manager](http://en.wikipedia.org/wiki/Winlogon) (winlogon.exe). Winlogon is responsible for handling interactive logons to a Windows system (local or remote). The [Graphical Identification aNd Authentication](http://en.wikipedia.org/wiki/Graphical_Identification_And_Authentication) (GINA) library is loaded inside the Winlogon process, and provides support for logging in as a local or [Windows domain](http://en.wikipedia.org/wiki/Windows_Server_domain) user.

The Session Manager stores its configuration atHKLM\SYSTEM\CurrentControlSet\Control\Session Manager. The exact operation of most of these items is based on the configuration set in the registry.

**Authentication**

[](http://en.wikipedia.org/wiki/File:XP_login.png)

GINA dialog box in [Windows XP](http://en.wikipedia.org/wiki/Windows_XP).

**Winlogon** starts the [Local Security Authority Subsystem Service](http://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service) (LSASS) and [Service Control Manager](http://en.wikipedia.org/wiki/Service_Control_Manager) (SCM), which in turn will start all the Windows services that are set to *Auto-Start*. It is also responsible for responding to the [secure attention sequence](http://en.wikipedia.org/wiki/Secure_attention_key) (SAS), loading the user profile on logon, and optionally locking the computer when a [screensaver](http://en.wikipedia.org/wiki/Screensaver) is running.

The login process is as follows:

* The Session Manager Subsystem starts Winlogon.
* Winlogon starts the Service Control Manager (services.exe).
  + Starts the *auto-start* services.
  + Updates the Control Sets; the LastKnownGood control set is updated to reflect the current control set.
* (Windows XP) Winlogon starts UIHost (logonui.exe), a full-screen graphical UI.
* Winlogon loads [GinaDll](http://en.wikipedia.org/wiki/Graphical_identification_and_authentication" \o "Graphical identification and authentication) (msgina.dll)
  + (Optional) Login prompt is displayed by GINA, and the user presses the Secure Attention Sequence (SAS) ([Control-Alt-Delete](http://en.wikipedia.org/wiki/Control-Alt-Delete)). Winlogon checks if the system is configured to login to a specific account automatically (AutoAdminLogon).
  + Login dialog is displayed by GINA
  + User enters credentials (username, password, and domain)
  + GINA passes credentials back to Winlogon
* Winlogon passes credentials to LSASS
* LSASS tries to use cached data in the LSA database (SYSTEM hive)
* If there is none, LSASS determines which account protocol is to be used by using the Security Packages listed in the key HKLM/SYSTEM/CurrentControlSet/Control/Lsa:
  + msv1\_0.dll implements the NT LAN Manager protocols. This package is used in stand-alone systems and domain-member systems for backward compatibility.
  + Kerberos.dll provides remote login by using [Active Directory](http://en.wikipedia.org/wiki/Active_Directory).
* LSASS enforces the local security policy (checking user permissions, creating audit trails, doling out security tokens, etc.).
* Control is passed back to Winlogon to prepare for passing the control to the user.
  + Create Windows Stations (WinSta0)
  + Create the desktops (Winlogon, Default and ScreenSaver)
  + It then starts the program specified in the Userinit value which defaults to userinit.exe. This value supports multiple executables.

If the user is trying to login to the local host then the HKLM/SAM key will be used as database. If the user is trying to login to another host then the NetLogon [service](http://en.wikipedia.org/wiki/Windows_service) is used to carry the data.

msv1\_0.dll<->netlogon<->remote netlogon<->remote msv1\_0.dll<->remote SAM

On Windows XP, GINA is only shown if the user presses the SAS.

Winlogon has support for plugins that get loaded and notified about specific events and LSASS also supports plugins (security packages). Some rootkits bundle Winlogon plugins because they are loaded before any user logs in. Some keys allow multiple comma-separated values to be supplied that allow a malicious program to be executed at the same time as a legit system file. The hashing algorithms stored in the [SAM](http://en.wikipedia.org/wiki/Security_Accounts_Manager) database are known to be broken.

Winlogon's responsibilities and the login process have changed significantly from the above in Windows Vista.

**Shell**

[](http://en.wikipedia.org/wiki/File:Windows_XP_SP3.png)

[Windows XP](http://en.wikipedia.org/wiki/Windows_XP) default shell.

Userinit is the first program that runs with the user credentials. It is responsible to start all the other programs that compose the user shell environment.

The shell program (typically Explorer.exe) is started from the registry entry Shell= pointed to by the same registry entry in keyHKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\IniFileMapping\system.ini\Boot; its default value isSYS:Microsoft\Windows NT\CurrentVersion\Winlogon, which evaluates toHKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon.

* Userinit loads the user profile. There are a few types of user profiles and it can be local or remote. This process can be very slow if the user profile is of the "roaming" type.
* User and Computer [Group Policy](http://en.wikipedia.org/wiki/Group_Policy) settings are applied.
  + Run user scripts
  + Run machine scrips
  + Run proquota.exe
* Runs the startup programs *before* the shell gets started.
* Starts the shell configured in registry, which defaults to explorer.exe.
* Userinit exits and the shell program continues running without a parent process.

Userinit runs startup programs from the following locations:

* HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
* HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
* HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
* HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows\Load
* HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows\Run
* HKCU\Software\Microsoft\Windows\CurrentVersion\Run
* HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
* %ALLUSERSPROFILE%\Start Menu\Programs\Startup\ (this path is localized on non-English versions of Windows before Vista)
* %USERPROFILE%\Start Menu\Programs\Startup\ (this path is localized on non-English versions of Windows before Vista)

## **Differences in NT and Windows Vista/7**

**Boot Configuration Data** (BCD) is a firmware-independent [database](http://en.wikipedia.org/wiki/Database) for [boot-time](http://en.wikipedia.org/wiki/Booting) configuration data. It is used by Microsoft's new Windows Boot Manager and replaces the [boot.ini](http://en.wikipedia.org/wiki/Boot.ini) that was used by [NTLDR](http://en.wikipedia.org/wiki/NTLDR).

Boot Configuration Data are stored in a data file that has the same format as the [Windows Registry](http://en.wikipedia.org/wiki/Windows_Registry) hives and is eventually loaded at registry key [HKEY\_LOCAL\_MACHINE\BCD00000]. The file is located either on the [EFI System Partition](http://en.wikipedia.org/wiki/EFI_System_Partition) (on machines that use [Extensible Firmware Interface](http://en.wikipedia.org/wiki/Extensible_Firmware_Interface) firmware) or in \Boot\Bcd on the [system volume](http://en.wikipedia.org/wiki/System_partition_and_boot_partition) (on machines that use IBM PC compatible firmware).

Boot Configuration Data may be altered using a command-line tool (bcdedit.exe), using Registry Editor (regedit.exe), using [Windows Management Instrumentation](http://en.wikipedia.org/wiki/Windows_Management_Instrumentation), or with third-party tools such as [EasyBCD](http://en.wikipedia.org/wiki/EasyBCD" \o "EasyBCD), BOOTICE, or Visual BCD Editor.

Boot Configuration Data contain the menu entries that are presented by the Windows Boot Manager, just as boot.ini contained the menu entries that were presented by NTLDR. These menu entries can include:

* Options to boot Windows Vista by invoking winload.exe.
* Options to resume Windows Vista from hibernation by invoking winresume.exe.
* Options to boot a prior version of the Windows NT family by invoking its NTLDR.
* Options to load and to execute a [volume boot record](http://en.wikipedia.org/wiki/Volume_boot_record).

Boot Configuration Data allows for third-party integration, so anyone can implement tools like diagnostics or recovery options.

## **Winload.exe**

The Windows Boot Manager invokes **winload.exe**—the operating system [boot loader](http://en.wikipedia.org/wiki/Booting)—to load the operating system kernel executive ([ntoskrnl.exe](http://en.wikipedia.org/wiki/Ntoskrnl.exe)) and core [device drivers](http://en.wikipedia.org/wiki/Device_drivers). In that respect, winload.exe is functionally equivalent to the operating system loader function of [NTLDR](http://en.wikipedia.org/wiki/NTLDR) in prior versions of Windows NT.

File System

Windows makes use of the [FAT](http://en.wikipedia.org/wiki/File_Allocation_Table),  [NTFS](http://en.wikipedia.org/wiki/NTFS), [exFAT](http://en.wikipedia.org/wiki/ExFAT) and [ReFS](http://en.wikipedia.org/wiki/ReFS" \o "ReFS) file systems (the last of these is only supported and usable in [Windows Server 2012](http://en.wikipedia.org/wiki/Windows_Server_2012); Windows cannot boot from it).

Windows uses a [*drive letter*](http://en.wikipedia.org/wiki/Drive_letter) abstraction at the user level to distinguish one disk or partition from another. For example, the [path](http://en.wikipedia.org/wiki/Path_(computing)) C:\WINDOWS represents a directory WINDOWS on the partition represented by the letter C. Drive C: is most commonly used for the primary hard disk partition, on which Windows is usually installed and from which it boots. This "tradition" has become so firmly ingrained that bugs exist in many applications which make assumptions that the drive that the operating system is installed on is C. The use of drive letters, and the tradition of using "C" as the drive letter for the primary hard disk partition, can be traced to [MS-DOS](http://en.wikipedia.org/wiki/MS-DOS), where the letters A and B were reserved for up to two floppy disk drives.

### File Allocation Table (FAT)

### FAT16

In November 1987, [Compaq MS-DOS 3.31](http://en.wikipedia.org/wiki/Compaq_MS-DOS_3.31) (a modified OEM version of MS-DOS 3.3 released by Compaq with their machines) introduced what today is simply known as *the FAT16*format, with the expansion of the 16-bit disk sector count to 32 bits in the BPB. Although the on-disk changes were minor, the entire DOS disk driver had to be converted to use 32-bit sector numbers, a task complicated by the fact that it was written in 16-bit [assembly language](http://en.wikipedia.org/wiki/X86_assembly_language). The result was initially called the *DOS 3.31 Large File System*. [Microsoft](http://en.wikipedia.org/wiki/Microsoft)'s DSKPROBE tool refers to type [0x06](http://en.wikipedia.org/wiki/Partition_type#PID_06h) as *BigFAT*, whereas some older versions of FDISK described it as *BIGDOS*. Technically, it is known as **FAT16B**.

Since older versions of DOS were not designed to cope with more than 65535 sectors, it was necessary to introduce a new partition type for this format in order to hide it from pre-3.31 issues of DOS. The original form of FAT16 (with less than 65536 sectors) had a [partition type](http://en.wikipedia.org/wiki/Partition_type)[0x04](http://en.wikipedia.org/wiki/Partition_type#PID_04h). To deal with disks larger than this, type[0x06](http://en.wikipedia.org/wiki/Partition_type#PID_06h) was introduced to indicate 65536 or more sectors. In addition to this, the disk driver was expanded to cope with more than 65535 sectors as well. The only other difference between the original FAT16 and the newer FAT16B format is the usage of a [newer BPB](http://en.wikipedia.org/wiki/DOS_3.31_BPB) format with 32-bit sector entry. Therefore, newer operating systems supporting the FAT16B format can cope also with the original FAT16 format without any necessary changes.

If partitions to be used by pre-DOS 3.31 issues of DOS need to be created by modern tools, the only criteria theoretically necessary to meet are a sector count of less than 65536, and the usage of the old partition ID ([0x04](http://en.wikipedia.org/wiki/Partition_type#PID_04h)). In practice however, type [0x01](http://en.wikipedia.org/wiki/Partition_type#PID_01h) and [0x04](http://en.wikipedia.org/wiki/Partition_type#PID_04h) primary partitions should not be physically located outside the first 32 MiB of the disk, due to other restrictions in MS-DOS 2.x, which could not cope with them otherwise.

In 1988, the FAT16B improvement became more generally available through [DR DOS](http://en.wikipedia.org/wiki/DR_DOS) 3.31, PC DOS 4.0, [OS/2](http://en.wikipedia.org/wiki/OS/2) 1.1, and MS-DOS 4.0. The limit on partition size was dictated by the 8-bit[signed](http://en.wikipedia.org/wiki/Signedness) count of sectors per cluster, which originally had a maximum power-of-two value of 64. With the standard hard disk sector size of 512 bytes, this gives a maximum of 32 KiB cluster size, thereby fixing the "definitive" limit for the FAT16 partition size at 2 GiB for sector size 512. On [magneto-optical](http://en.wikipedia.org/wiki/Magneto-optical) media, which can have 1 or 2 KiB sectors instead of 0.5 KiB, this size limit is proportionally larger.

Much later, [Windows NT](http://en.wikipedia.org/wiki/Windows_NT) increased the maximum cluster size to 64 KiB, by considering the sectors-per-cluster count as unsigned. However, the resulting format was not compatible with any other FAT implementation of the time, and it generated greater internal fragmentation. [Windows 98](http://en.wikipedia.org/wiki/Windows_98), SE, and ME also supported reading and writing this variant, but its disk utilities did not work with it and some FCB services are not available for such volumes. This contributes to a confusing compatibility situation.

Prior to 1995, versions of DOS accessed the disk via [CHS](http://en.wikipedia.org/wiki/Cylinder-head-sector) addressing only. When MS-DOS 7.0 / [Windows 95](http://en.wikipedia.org/wiki/Windows_95) introduced [LBA](http://en.wikipedia.org/wiki/Logical_block_addressing) disk access, partitions could start being physically located outside the first ca. 8 GiB of this disk and thereby out of the reach of the traditional CHS addressing scheme. Partitions partially or fully located beyond the CHS barrier therefore had to be hidden from non-LBA-enabled operating systems by using the new partition type [0x0E](http://en.wikipedia.org/wiki/Partition_type#PID_0Eh) in the partition table instead. FAT16 partitions using this partition type are also named **FAT16X**. The only difference, compared to previous FAT16 partitions, is the fact that some CHS-related geometry entries in the BPB record, namely the number of sectors per track and the number of heads, may contain no or misleading values and should not be used.

The number of root directory entries available for FAT12 and FAT16 is determined when the volume is formatted, and is stored in a 16-bit field. For a given number RDE and sector size SS, the number RDS of root directory sectors is RDS=ceil((RDE×32)/SS), and RDE is normally chosen to fill these sectors, i.e., RDE\*32=RDS\*SS. FAT12 and FAT16 media typically use 512 root directory entries on non-floppy media. Some third-party tools, like [mkdosfs](http://en.wikipedia.org/wiki/Mkdosfs" \o "Mkdosfs), allow the user to set this parameter.

### FAT32

In order to overcome the volume size limit of FAT16, while at the same time allowing DOS [real mode](http://en.wikipedia.org/wiki/Real_mode) code to handle the format, Microsoft designed a new version of the file system, **FAT32**, which supported an increased number of possible clusters, but could reuse most of the existing code, so that the available [conventional memory](http://en.wikipedia.org/wiki/Conventional_memory) footprint was reduced by less than 5 KiB under DOS. Cluster values are represented by [32-bit](http://en.wikipedia.org/wiki/32-bit) numbers, of which 28 bits are used to hold the cluster number. The boot sector uses a 32-bit field for the sector count, limiting the FAT32 volume size to 2 TiB for a sector size of 512 bytes and 16 TiB for a sector size of 4,096 bytes. FAT32 was introduced with MS-DOS 7.1 / Windows 95 OSR2 in 1996, although reformatting was needed to use it, and [DriveSpace 3](http://en.wikipedia.org/wiki/DriveSpace_3)(the version that came with Windows 95 OSR2 and Windows 98) never supported it. Windows 98 introduced a utility to convert existing hard disks from FAT16 to FAT32 without loss of data. In the Windows NT line, native support for FAT32 arrived in [Windows 2000](http://en.wikipedia.org/wiki/Windows_2000). A free FAT32 driver for [Windows NT 4.0](http://en.wikipedia.org/wiki/Windows_NT_4.0) was available from Winternals, a company later acquired by Microsoft. Since the acquisition the driver is no longer officially available. Since 1998, Caldera's dynamically loadable [DRFAT32](http://en.wikipedia.org/wiki/DRFAT32) driver could be used to enable FAT32 support in DR-DOS. The first version of DR-DOS to natively support FAT32 and LBA access was OEM DR-DOS 7.04 in 1999. That same year [IMS](http://en.wikipedia.org/wiki/Intelligent_Micro_Software) introduced native FAT32 support with [REAL/32](http://en.wikipedia.org/wiki/REAL/32) 7.90, and IBM 4690 OS added FAT32 support with version 2. [Ahead Software](http://en.wikipedia.org/wiki/Ahead_Software) provided another dynamically loadable FAT32.EXE driver for DR-DOS 7.03 with Nero Burning ROM in 2004. IBM PC DOS introduced native FAT32 support with OEM PC DOS 7.10 in 2003.

The maximum possible size for a file on a FAT32 volume is 4 GiB minus 1 byte or 4,294,967,295 (232 − 1) bytes. This limit is a consequence of the file length entry in the directory table and would also affect huge FAT16 partitions with a sufficient sector size. Large video files, DVD images, and databases easily exceed this limit.

The open FAT+ specification proposes how to store [larger files](http://en.wikipedia.org/wiki/Large_file_support) up to 256 GiB minus 1 byte or 274,877,906,943 (238 − 1) bytes on slightly modified and otherwise backward-compatible FAT32 volumes, but imposes a risk that disk tools or FAT32 implementations not aware of this extension may truncate or delete files exceeding the normal FAT32 file size limit. Also, support for **FAT32+** (and**FAT16+**) is limited to some versions of [DR-DOS](http://en.wikipedia.org/wiki/DR-DOS) and not available in mainstream operating systems so far. (This extension is critically incompatible with the/EAS option of the FAT32.IFS method to store[OS/2 extended attributes](http://en.wikipedia.org/wiki/FAT_extended_file_attributes) on FAT32 volumes.)

As with previous file systems, the design of the FAT32 file system does not include direct built-in support for long filenames, but FAT32 volumes can optionally hold [VFAT](http://en.wikipedia.org/wiki/File_Allocation_Table#VFAT) long filenames in addition to short filenames in exactly the same way as VFAT long filenames have been optionally implemented for FAT12 and FAT16 volumes.

Two partition types have been reserved for FAT32 partitions, [0x0B](http://en.wikipedia.org/wiki/Partition_type#PID_0Bh) and [0x0C](http://en.wikipedia.org/wiki/Partition_type#PID_0Ch). The latter type is also named **FAT32X** in order to indicate usage of LBA disk access instead of CHS. On such partitions, some CHS-related geometry entries in the EBPB record, namely the number of sectors per track and the number of heads, may contain no or misleading values and should not be used.

### Extended Attributes

[OS/2](http://en.wikipedia.org/wiki/OS/2) heavily depends on [extended attributes](http://en.wikipedia.org/wiki/Extended_attribute) (EAs) and stores them in a hidden file called "EA␠DATA.␠SF" in the [root directory](http://en.wikipedia.org/wiki/Root_directory) of the [FAT12](http://en.wikipedia.org/wiki/File_Allocation_Table#FAT12) or [FAT16](http://en.wikipedia.org/wiki/File_Allocation_Table#FAT16) volume. This file is indexed by two previously reserved bytes in the file’s (or directory's) [directory entry](http://en.wikipedia.org/wiki/FAT_directory_entry) at offset [0x14](http://en.wikipedia.org/wiki/Design_of_the_FAT_file_system#DIR_OFS_14h).  In the[FAT32](http://en.wikipedia.org/wiki/File_Allocation_Table#FAT32) format, these bytes hold the upper 16 bits of the starting cluster number of the file or directory, hence making it impossible to store [OS/2 EAs](http://en.wikipedia.org/wiki/OS/2_EA) on FAT32 using this method.

However, the third-party FAT32 [installable file system](http://en.wikipedia.org/wiki/Installable_file_system) (IFS) driver FAT32.IFS version 0.70 and higher by Henk Kelder & Netlabs for OS/2 and [eComStation](http://en.wikipedia.org/wiki/EComStation) stores extended attributes in extra files with filenames having the string "␠EA.␠SF" appended to the regular filename of the file to which they belong. The driver also utilizes the byte at offset [0x0C](http://en.wikipedia.org/wiki/Design_of_the_FAT_file_system#DIR_OFS_0Ch) in directory entries to store a special mark byte indicating the presence of extended attributes to help speed up things. (This extension is critically incompatible with the FAT32+ method to store files larger than 4 GiBminus 1 on FAT32 volumes.)

Extended attributes are accessible via the [Workplace Shell](http://en.wikipedia.org/wiki/Workplace_Shell) desktop, through [REXX](http://en.wikipedia.org/wiki/REXX) scripts, and many system [GUI](http://en.wikipedia.org/wiki/Graphical_user_interface) and [command-line](http://en.wikipedia.org/wiki/Command_line_interface) utilities (such as [4OS2](http://en.wikipedia.org/wiki/4OS2)).

To accommodate its [OS/2](http://en.wikipedia.org/wiki/OS/2) subsystem, [Windows NT](http://en.wikipedia.org/wiki/Windows_NT) supports the handling of extended attributes in [HPFS](http://en.wikipedia.org/wiki/High_Performance_File_System), [NTFS](http://en.wikipedia.org/wiki/NTFS), FAT12 and FAT16. It stores EAs on FAT12, FAT16 and HPFS using exactly the same scheme as OS/2, but does not support any other kind of [ADS](http://en.wikipedia.org/wiki/Alternate_Data_Streams) as held on NTFS volumes. Trying to copy a file with any ADS other than EAs from an NTFS volume to a FAT or HPFS volume gives a warning message with the names of the ADSs that will be lost. It does not support the FAT32.IFS method to store EAs on FAT32 volumes.

[Windows 2000](http://en.wikipedia.org/wiki/Windows_2000) onward acts exactly as Windows NT, except that it ignores EAs when copying to FAT32 without any warning (but shows the warning for other ADSs, like "Macintosh Finder Info" and "Macintosh Resource Fork").

[Cygwin](http://en.wikipedia.org/wiki/Cygwin) uses "EA␠DATA.␠SF" files as well.

### Long file names

One of the user experience goals for the designers of [Windows 95](http://en.wikipedia.org/wiki/Windows_95) was the ability to use [long filenames](http://en.wikipedia.org/wiki/Long_filename) (LFNs—up to 255 [UCS-2](http://en.wikipedia.org/wiki/UCS-2) [code units](http://en.wikipedia.org/wiki/Code_unit) long), in addition to classic [8.3 filenames](http://en.wikipedia.org/wiki/8.3_filename) (SFNs). For [backward](http://en.wikipedia.org/wiki/Backward_compatibility) and [forward compatibility](http://en.wikipedia.org/wiki/Forward_compatibility) LFNs were implemented as an optional extension on top of the existing FAT file system structures using a [workaround](http://en.wikipedia.org/wiki/Workaround) in the way directory entries are laid out.

This transparent method to store long file names in the existing FAT file systems without altering their data structures is usually known as [**VFAT**](http://en.wikipedia.org/wiki/Design_of_the_FAT_file_system#VFAT) (for "Virtual FAT") after the Windows 95 virtual device driver.

Non VFAT-enabled operating systems can still access the files under their short file name alias without restrictions, however, the associated long file names may get lost, when files with long file names are copied under non VFAT-aware operating systems.

In Windows NT, support for VFAT long filenames started from version [3.5](http://en.wikipedia.org/wiki/Windows_NT_3.5).

Linux provides a VFAT filesystem driver to work with FAT volumes with VFAT long filenames. For some while, a [UVFAT](http://en.wikipedia.org/wiki/UVFAT) driver was available to provide combined support for [UMSDOS](http://en.wikipedia.org/wiki/File_Allocation_Table#UMSDOS)-style permissions with VFAT long filenames.

[OS/2](http://en.wikipedia.org/wiki/OS/2) added long filename support to FAT using [extended attributes](http://en.wikipedia.org/wiki/FAT_extended_file_attributes) (EA) before the introduction of VFAT; thus, VFAT long filenames are invisible to OS/2, and EA long filenames are invisible to Windows, therefore experienced users of both operating systems would have to manually rename the files.

In order to support [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) applications, the [FlexOS](http://en.wikipedia.org/wiki/FlexOS" \o "FlexOS)-based [IBM 4690 OS](http://en.wikipedia.org/wiki/IBM_4690_OS) version 2 introduced its own [virtual file system](http://en.wikipedia.org/wiki/Virtual_file_system) (VFS) architecture to store long filenames in the FAT file system in a backwards compatible fashion. If enabled, the virtual filenames (VFN) are available under separate logical drive letters, whereas the real filenames (RFN) remain available under the original drive letters.

### Forks and Alternate Data Streams

The FAT file system itself is not designed for supporting [Alternate Data Streams](http://en.wikipedia.org/wiki/Alternate_Data_Streams) ([ADS](http://en.wikipedia.org/wiki/Fork_(filesystem))), but some operating systems that heavily depend on them have devised various methods for handling them in FAT drives. Such methods either store the additional information in extra files and directories ([Mac OS](http://en.wikipedia.org/wiki/Mac_OS)), or give new semantics to previously unused fields of the FAT on-disk data structures ([OS/2](http://en.wikipedia.org/wiki/OS/2) and [Windows NT](http://en.wikipedia.org/wiki/Windows_NT)).

Mac OS using PC Exchange stores its various dates, file attributes and long filenames in a [hidden file](http://en.wikipedia.org/wiki/Hidden_file) called "FINDER.DAT", and resource forks (a common Mac OS ADS) in a subdirectory called "RESOURCE.FRK", in every directory where they are used. From PC Exchange 2.1 onwards, they store the Mac OS long filenames as standard FAT long filenames and convert FAT filenames longer than 31 characters to unique 31-character filenames, which can then be made visible to Macintosh applications.

[Mac OS X](http://en.wikipedia.org/wiki/Mac_OS_X) stores resource forks and metadata (file attributes, other ADS) in a hidden file with a name constructed from the owner filename prefixed with ".\_", and Finder stores some folder and file metadata in a hidden file called ".DS\_Store".

**NTFS**

### Scalability

In theory, the maximum NTFS volume size is 264−1 clusters. However, the maximum NTFS volume size as implemented in Windows XP Professional is 232−1 clusters partly due to partition table limitations. For example, using 64 kB clusters, the maximum Windows XP NTFS volume size is 256 [TBs](http://en.wikipedia.org/wiki/Tebibyte) minus 64 [KBs](http://en.wikipedia.org/wiki/Kibibyte). Using the default cluster size of 4 kB, the maximum NTFS volume size is 16 TB minus 4 kB. (Both of these are vastly higher than the 128 [GB](http://en.wikipedia.org/wiki/Gibibyte) limit lifted in [Windows XP SP1](http://en.wikipedia.org/wiki/Windows_XP#Service_Pack_1).) Because partition tables on master boot record (MBR) disks only support partition sizes up to 2 TB, dynamic or [GPT](http://en.wikipedia.org/wiki/GUID_Partition_Table) volumes must be used to create NTFS volumes over 2 TB. Booting from a GPT volume to a Windows environment requires a system with [UEFI](http://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface) and 64-bit support.

The maximum theoretical file size on NTFS is 16 [EB](http://en.wikipedia.org/wiki/Exabyte) (16 × 10246 or 264 bytes) minus 1 kB or 18,446,744,073,709,550,592 bytes. With [Windows 8](http://en.wikipedia.org/wiki/Windows_8) and [Windows Server 2012](http://en.wikipedia.org/wiki/Windows_Server_2012), the maximum file size implemented is 256 TB minus 64 KB or 281,474,976,645,120 bytes.

NTFS supports a maximum cluster size of 64 kB.[[16]](http://en.wikipedia.org/wiki/NTFS#cite_note-16)

### Journaling

NTFS is a [journaling file system](http://en.wikipedia.org/wiki/Journaling_file_system) and uses the NTFS Log ($LogFile) to record metadata changes to the volume. It is a critical functionality of NTFS (a feature that FAT/FAT32 does not provide) for ensuring that its internal complex data structures (notably the volume allocation bitmap), or data moves performed by the [defragmentation](http://en.wikipedia.org/wiki/Defragmentation) API, the modifications to [MFT](http://en.wikipedia.org/wiki/NTFS#Master_File_Table) records (such as moves of some variable-length attributes stored in MFT records and attribute lists), and indices (for directories and [security descriptors](http://en.wikipedia.org/wiki/Security_descriptor)) will remain consistent in case of system crashes, and allow easy rollback of uncommitted changes to these critical data structures when the volume is remounted.

The [USN Journal](http://en.wikipedia.org/wiki/USN_Journal) (Update Sequence Number Journal) is a system management feature that records (in $Extend$UsnJrnl) changes to files, streams and directories on the volume, as well as their various attributes and security settings. The journal is made available for applications to track changes to the volume. This journal can be enabled or disabled on non-system volumes and is not enabled by default for a newly added drive.

### Hard links

Hard links allows different file names to refer to the same file contents.

[Hard links](http://en.wikipedia.org/wiki/Hard_link) are similar to [directory junctions](http://en.wikipedia.org/wiki/NTFS#Directory_junctions), but refer to files instead. Hard links may link to files in the same volume only because each volume has its own [MFT](http://en.wikipedia.org/wiki/NTFS#Master_File_Table). Hard links have their own file metadata, so a change in file size or attributes under one hard link may not update the others until they are opened.

Hard links were originally included to support the [POSIX](http://en.wikipedia.org/wiki/POSIX) subsystem in Windows NT.

Windows uses hard links to support [Short (8.3) filenames](http://en.wikipedia.org/wiki/8.3_filename) in NTFS. Operating system support is needed because there are legacy applications that can work only with 8.3 filenames. In this case, an additional filename record and directory entry is added, but both 8.3 and long file name are linked and updated together, unlike a regular hard link.

### Alternate data streams (ADS)

[Alternate data streams](http://en.wikipedia.org/wiki/Fork_(filesystem)) allow more than one data stream to be associated with a filename, using the format "filename:streamname" (e.g., "text.txt:extrastream").

NTFS Streams were introduced in [Windows NT 3.1](http://en.wikipedia.org/wiki/Windows_NT_3.1), to enable Services for Macintosh (SFM) to store [resource forks](http://en.wikipedia.org/wiki/Resource_fork). Although current versions of Windows Server no longer include SFM, third-party [Apple Filing Protocol](http://en.wikipedia.org/wiki/Apple_Filing_Protocol) (AFP) products (such as [GroupLogic](http://en.wikipedia.org/wiki/GroupLogic" \o "GroupLogic)'s [ExtremeZ-IP](http://en.wikipedia.org/wiki/ExtremeZ-IP" \o "ExtremeZ-IP)) still use this feature of the file system. Very small ADS (called Zone.Identifier) are added by [Internet Explorer](http://en.wikipedia.org/wiki/Internet_Explorer) and recently by other browsers to mark files downloaded from external sites as possibly unsafe to run; the local shell would then require user confirmation before opening them.[[21]](http://en.wikipedia.org/wiki/NTFS#cite_note-21) When the user indicates that they no longer want this confirmation dialog, this ADS is deleted.

Alternate streams are not listed in Windows Explorer, and their size is not included in the file's size. They are ignored when the file is copied or moved to another file system without ADS support, attached to an e-mail, or uploaded to a website. Thus, using alternate streams for critical data may cause problems. Microsoft provides a tool called Streams to view streams on a selected volume. Starting with [Windows PowerShell](http://en.wikipedia.org/wiki/Windows_PowerShell) 3.0, it is possible to manage ADS natively with seven cmdlets: Add-Content, Clear-Content, Get-Content, Get-Item, Out-String, Remove-Item, and Set-Content.

[Malware](http://en.wikipedia.org/wiki/Malware) has used alternate data streams to hide code. As a result, malware scanners and other special tools now check for alternate data streams.

### File compression

NTFS can [compress](http://en.wikipedia.org/wiki/Data_compression) files using LZNT1 algorithm (a variant of the [LZ77](http://en.wikipedia.org/wiki/LZ77)). Files are compressed in 16-cluster chunks. With 4 kB clusters, files are compressed in 64 kB chunks. The compression algorithms in NTFS are designed to support cluster sizes of up to 4 kB. When the cluster size is greater than 4 kB on an NTFS volume, NTFS compression is not available. If the compression reduces 64 kB of data to 60 kB or less, NTFS treats the unneeded 4 kB pages like empty [sparse file](http://en.wikipedia.org/wiki/Sparse_file) clusters—they are not written. This allows for reasonable random-access times as the OS just has to follow the chain of fragments. However, large compressible files become highly fragmented since every chunk < 64KB becomes a fragment. Single-user systems with limited hard disk space can benefit from NTFS compression for small files, from 4 kB to 64 kB or more, depending on compressibility. Files less than 900 bytes or so are stored within the directory entry at the[MFT](http://en.wikipedia.org/wiki/NTFS#Master_File_Table).

Flash memory, such as [SSD](http://en.wikipedia.org/wiki/SSD) drives do not have the head movement delays of [hard disk drives](http://en.wikipedia.org/wiki/Hard_disk_drives), so fragmentation has only small effects. Users of fast [multi-core processors](http://en.wikipedia.org/wiki/Multi-core_processor) will find improvements in application speed by compressing their applications and data as well as a reduction in space used. Note that SSDs with Sandforce controllers already compress data. However, since less data is transferred, there is a reduction in IOPS/s.

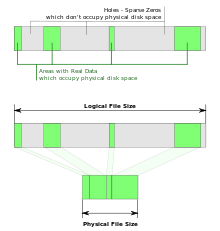
The best use of compression is for files that are repetitive, seldom written, usually accessed sequentially, and not themselves compressed. Log files are an ideal example.

Compressing system files needed at boot time, like drivers, NTLDR, winload.exe, or BOOTMGR may prevent the system from booting correctly, as compression filters are not available then. However, in later editions of Windows, compression of important system files is disallowed.

Files may be compressed or decompressed individually (via changing the advanced attributes) for a drive, directory, or directory tree, becoming a default for the files inside.

Although read–write access to compressed files is mostly [transparent](http://en.wikipedia.org/wiki/Network_transparency), Microsoft recommends avoiding compression on server systems and/or network shares holding roaming profiles because it puts a considerable load on the processor. Compression is not recommended by Microsoft for files exceeding 30 MB because of the performance hit. Since many fragments are created for compressible files, [defragmentation](http://en.wikipedia.org/wiki/Defragmentation) may take longer.

### Sparse files



A sparse file: Empty bytes don't need to be saved, thus they can be represented by [metadata](http://en.wikipedia.org/wiki/Metadata).

[Sparse files](http://en.wikipedia.org/wiki/Sparse_file) are files interspersed with empty segments for which no actual storage space is used. To the applications, the file looks like an ordinary file with empty regions seen as regions filled with zeros.

Database applications, for instance, may use sparse files. As with compressed files, the actual sizes of sparse files are not taken into account when determining quota limits.

### Volume Shadow Copy

The [Volume Shadow Copy Service](http://en.wikipedia.org/wiki/Shadow_Copy) (VSS) keeps historical versions of files and folders on NTFS volumes by copying old, newly overwritten data to shadow copy via [copy-on-write](http://en.wikipedia.org/wiki/Copy-on-write) technique. The user may later request an earlier version to be recovered. This also allows data backup programs to archive files currently in use by the file system. On heavily loaded systems, Microsoft recommends setting up a shadow copy volume on a separate disk.

### Transactions

As of Windows Vista, applications can use [Transactional NTFS](http://en.wikipedia.org/wiki/Transactional_NTFS) to group changes to files together into a transaction. The transaction will guarantee that all changes happen, or none of them do, and it will guarantee that applications outside the transaction will not see the changes until they are committed.

It uses similar techniques as those used for Volume Shadow Copies (i.e. copy-on-write) to ensure that overwritten data can be safely rolled back, and a [CLFS](http://en.wikipedia.org/wiki/Common_Log_File_System) log to mark the transactions that have still not been committed, or those that have been committed but still not fully applied (in case of system crash during a commit by one of the participants).

Transactional NTFS does not restrict transactions to just the local NTFS volume, but also includes other transactional data or operations in other locations such as data stored in separate volumes, the local registry, or SQL databases, or the current states of system services or remote services. These transactions are coordinated network-wide with all participants using a specific service, the [DTC](http://en.wikipedia.org/wiki/Distributed_Transaction_Coordinator), to ensure that all participants will receive same commit state, and to transport the changes that have been validated by any participant (so that the others can invalidate their local caches for old data or rollback their ongoing uncommitted changes). Transactional NTFS allows, for example, the creation of network-wide consistent distributed filesystems, including with their local live or offline caches.

### Encryption

[Encrypting File System](http://en.wikipedia.org/wiki/Encrypting_File_System) (EFS) provides strong and user-transparent encryption of any file or folder on an NTFS volume. EFS works in conjunction with the EFS service, Microsoft's CryptoAPI and the EFS File System Run-Time Library (FSRTL). EFS works by encrypting a file with a bulk [symmetric key](http://en.wikipedia.org/wiki/Symmetric_key_algorithm) (also known as the File Encryption Key, or FEK), which is used because it takes a relatively small amount of time to encrypt and decrypt large amounts of data than if an asymmetric key cipher is used. The symmetric key that is used to encrypt the file is then encrypted with a public key that is associated with the user who encrypted the file, and this encrypted data is stored in an alternate data stream of the encrypted file. To decrypt the file, the file system uses the private key of the user to decrypt the symmetric key that is stored in the file header. It then uses the symmetric key to decrypt the file. Because this is done at the file system level, it is transparent to the user.  Also, in case of a user losing access to their key, support for additional decryption keys has been built into the EFS system, so that a recovery agent can still access the files if needed. NTFS-provided encryption and NTFS-provided compression are mutually exclusive; however, NTFS can be used for one and a third-party tool for the other.

The support of EFS is not available in Basic, Home and Media Center versions of Windows, and must be activated after installation of Professional, Ultimate and Server versions of Windows or by using enterprise deployment tools within Windows domains.

### Quotas

[Disk quotas](http://en.wikipedia.org/wiki/Disk_quota) were introduced in NTFS v3. They allow the administrator of a computer that runs a version of Windows that supports NTFS to set a threshold of disk space that users may use. It also allows administrators to keep track of how much disk space each user is using. An administrator may specify a certain level of disk space that a user may use before they receive a warning, and then deny access to the user once they hit their upper limit of space. Disk quotas do not take into account NTFS's transparent file-compression, should this be enabled. Applications that query the amount of free space will also see the amount of free space left to the user who has a quota applied to them.

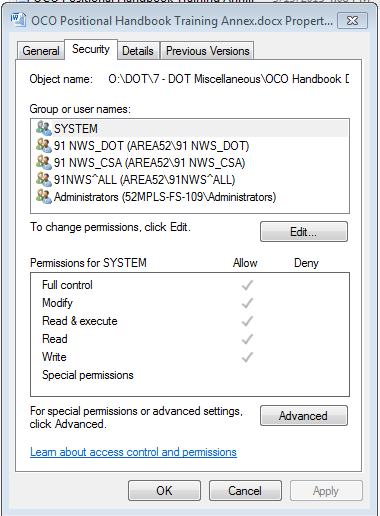
### Reparse points

[NTFS reparse points](http://en.wikipedia.org/wiki/NTFS_reparse_point), introduced in NTFS v3, are used by associating a reparse tag in the user space attribute of a file or directory. When the object manager parses a file system name lookup and encounters a reparse attribute, it will *reparse* the name lookup, passing the user controlled reparse data to every file system filter driver that is loaded into Windows. Each filter driver examines the reparse data to see whether it is associated with that reparse point, and if that filter driver determines a match, then it intercepts the file system call and executes its special functionality. [Reparse points](http://en.wikipedia.org/wiki/NTFS_reparse_point) are used to implement Volume Mount Points, Directory Junctions, Hierarchical Storage Management, Native Structured Storage, Single Instance Storage, and Symbolic Links.

### Resizing

Starting with [Windows Vista](http://en.wikipedia.org/wiki/Windows_Vista) Microsoft added the built-in ability to shrink or expand a partition, but this capability is limited because it will not relocate page file fragments or files that have been marked as unmovable. So shrinking will often require relocating or disabling any [page file](http://en.wikipedia.org/wiki/Pagefile#Windows_NT), the index of [Windows Search](http://en.wikipedia.org/wiki/Windows_Search), and any [Shadow Copy](http://en.wikipedia.org/wiki/NTFS#Volume_Shadow_Copy) used by [System Restore](http://en.wikipedia.org/wiki/System_Restore). Various third-party tools are capable of resizing NTFS partitions.

## **Internals**



NTFS filesystem permissions on a Windows/7 system

Internally, NTFS uses [B+ trees](http://en.wikipedia.org/wiki/B%2B_tree) to index file system data. Although complex to implement, this allows faster file look up times in most cases. A [file system journal](http://en.wikipedia.org/wiki/Journaling_file_system) is used to guarantee the integrity of the file system metadata but not individual files' content. Systems using NTFS are known to have improved reliability compared to FAT file systems.

NTFS allows any sequence of 16-bit values for name encoding (file names, stream names, index names, etc.) except 0x0000. This means [UTF-16](http://en.wikipedia.org/wiki/UTF-16) code units are supported, but the file system does not check whether a sequence is valid [UTF-16](http://en.wikipedia.org/wiki/UTF-16/UCS-2) (it allows any sequence of short values, not restricted to those in the Unicode standard). File names are limited to 255 [UTF-16](http://en.wikipedia.org/wiki/UTF-16) code units. Certain names are reserved in the volume root directory and cannot be used for files. These are $MFT, $MFTMirr,$LogFile, $Volume, $AttrDef, . (dot), $Bitmap, $Boot, $BadClus, $Secure,$Upcase, and $Extend. (dot) and $Extend are both directories; the others are files. The NT kernel limits full paths to 32,767 UTF-16 code units. There are some additional restrictions on code points and file names.

### Master File Table

In NTFS, all file, directory and [metafile](http://en.wikipedia.org/wiki/NTFS#Metafiles) data—file name, creation date, access permissions (by the use of [access control lists](http://en.wikipedia.org/wiki/Access_control_list)), and size—are stored as metadata in the Master File Table (MFT). This abstract approach allowed easy addition of file system features during Windows NT's development—an example is the addition of fields for indexing used by the [Active Directory](http://en.wikipedia.org/wiki/Active_Directory)software. This also enables fast file search software such as [Everything](http://en.wikipedia.org/wiki/Everything_(software)) or [Ultrasearch](http://en.wikipedia.org/w/index.php?title=Ultrasearch&action=edit&redlink=1" \o "Ultrasearch (page does not exist)) to locate named local files and folders included in the MFT very fast, without requiring any other index.

The MFT structure supports algorithms which minimize [disk fragmentation](http://en.wikipedia.org/wiki/File_system_fragmentation).  A directory entry consists of a filename and a "file ID", which is the record number representing the file in the Master File Table. The file ID also contains a reuse count to detect stale references. While this strongly resembles the W\_FID of [Files-11](http://en.wikipedia.org/wiki/Files-11), other NTFS structures radically differ.

Two copies of the MFT are stored in case of corruption. If the first record is corrupted, NTFS reads the second record to find the MFT mirror file. Locations for both files are stored in the boot sector.

### Metafiles (List of Metafiles in Appendix 3 – NTFS Meta Files)

NTFS contains several files that define and organize the file system. In all respects, most of these files are structured like any other user file ($Volume being the most peculiar), but are not of direct interest to file system clients. These metafiles define files, back up critical file system data, buffer file system changes, manage free space allocation, satisfy [BIOS](http://en.wikipedia.org/wiki/BIOS) expectations, track bad allocation units, and store security and disk space usage information. All content is in an unnamed data stream, unless otherwise indicated.

Metafiles are treated specially by Windows and are difficult to directly view: special purpose-built tools are needed. One such tool is the nfi.exe ("NTFS File Sector Information Utility") that is freely distributed as part of the Microsoft "OEM Support Tools". For example to obtain information on the "$MFT"-Master File Table Segment the following command is used:nfi.exe c:\$MFT.

### Attribute Lists, Attributes, and Streams

For each file (or directory) described in the MFT record, there's a linear repository of stream descriptors (also named *attributes*), packed together in one or more MFT records (containing the so-called *attributes list*), with extra padding to fill the fixed 1 KB size of every MFT record, and that fully describes the effective streams associated with that file.

Each attribute has an attribute type (a fixed-size integer mapping to an attribute definition in file $AttrDef), an optional attribute name (for example, used as the name for an alternate data stream), and a value, represented in a sequence of bytes. For NTFS, the standard data of files, the alternate data streams, or the index data for directories are stored as attributes.

According to $AttrDef, some attributes can be either resident or non-resident. The $DATA attribute, which contains file data, is such an example. When the attribute is resident (which is represented by a flag), its value is stored directly in the MFT record. Otherwise, clusters are allocated for the data, and the cluster location information is stored as data runs in the attribute.

* For each file in the MFT, the attributes identified by *attribute type, attribute name* must be unique. Additionally, NTFS has some ordering constraints for these attributes.
* There's a predefined null attribute type, used to indicate the end of the list of attributes in one MFT record. It must be present as the last attribute in the record (all other storage space available after it will be ignored and just consists of padding bytes to match the record size in the MFT).
* Some attribute types are required and must be present in each MFT record, except unused records that are just indicated by null attribute types.
  + This is the case for $STANDARD\_INFORMATION attribute that is stored as a fixed-size record and containing the [timestamps](http://en.wikipedia.org/wiki/Timestamp) and other basic single-bit attributes (compatible with those managed by [FAT](http://en.wikipedia.org/wiki/File_Allocation_Table)/FAT32 in DOS or Windows 95/98 applications).
* Some attribute types cannot have a name and must remain anonymous.
  + This is the case for the standard attributes, or for the preferred NTFS "filename" attribute type, or the "short filename" attribute type, when it is also present (for compatibility with DOS-like applications, see below). It is also possible for a file to only contain a short filename, in which case it will be the preferred one, as listed in the Windows Explorer.
  + The filename attributes stored in the attribute list do not make the file immediately accessible through the hierarchical filesystem. In fact, all the filenames must be indexed separately in at least one separate directory on the same volume, with its own MFT record and its own [security descriptors](http://en.wikipedia.org/wiki/Security_descriptor) and attributes that will reference the MFT record number for that file. This allows the same file or directory to be "hard-linked" several times from several containers on the same volume, possibly with distinct filenames.
* The default data stream of a regular file is a stream of type $DATA but with an anonymous name, and the ADS's are similar but must be named.
* On the opposite, the default data stream of directories has a distinct type, but is not anonymous: they have an attribute name ("$I30" in NTFS 3+) that reflects its indexing format.

All attributes of a given file may be displayed by using the nfi.exe ("NTFS File Sector Information Utility") that is freely distributed as part of the Microsoft "OEM Support Tools".

Windows system calls may handle alternate data streams.  Depending on the operating system, utility and remote file system, a file transfer might silently strip data streams.  A safe way of copying or moving files is to use the Backup Read and Backup Write system calls, which allow programs to enumerate streams, to verify whether each stream should be written to the destination volume and to knowingly skip unwanted streams.

### Resident vs. Non-Resident Attributes

To optimize the storage and reduce the I/O overhead for the very common case of attributes with very small associated value, NTFS prefers to place the value within the attribute itself (if the size of the attribute does not then exceed the maximum size of an MFT record), instead of using the MFT record space to list clusters containing the data; in that case, the attribute will not store the data directly but will just store an allocation map (in the form of *data runs*) pointing to the actual data stored elsewhere on the volume.  When the value can be accessed directly from within the attribute, it is called "resident data" (by [computer forensics](http://en.wikipedia.org/wiki/Computer_forensics) workers). The amount of data that fits is highly dependent on the file's characteristics, but 700 to 800 bytes is common in single-stream files with non-lengthy filenames and no ACLs.

* Some attributes (such as the preferred filename, the basic file attributes) cannot be made non-resident. For non-resident attributes, their allocation map must fit within MFT records.
* Encrypted-by-NTFS, sparse data streams, or compressed data streams cannot be made resident.
* The format of the allocation map for non-resident attributes depends on its capability of supporting sparse data storage. In the current implementation of NTFS, once a non-resident data stream has been marked and converted as sparse, it cannot be changed back to non-sparse data, so it cannot become resident again, unless this data is fully truncated, discarding the sparse allocation map completely.
* When a non-resident attribute is so fragmented, that its effective allocation map cannot fit entirely within one MFT record, NTFS stores the attribute in multiple records. The first one among them is called the base record, while the others are called extension records. NTFS creates a special attribute $ATTRIBUTE\_LIST to store information mapping different parts of the long attribute to the MFT records, which means the allocation map may be split into multiple records. The $ATTRIBUTE\_LIST itself can also be non-resident, but its own allocation map must fit within one MFT record.
* When there are too many attributes for a file (including ADS's, extended attributes, or [security descriptors](http://en.wikipedia.org/wiki/Security_descriptor)), so that they cannot fit all within the MFT record, extension records may also be used to store the other attributes, using the same format as the one used in the base MFT record, but without the space constraints of one MFT record.

The allocation map is stored in a form of *data runs* with compressed encoding. Each data run represents a contiguous group of clusters that store the attribute value. For files on a multi-GB volume, each entry can be encoded as 5 to 7 bytes, which means a 1 KB MFT record can store about 100 such data runs. However, as the $ATTRIBUTE\_LIST also has a size limit, it is dangerous to have more than 1 million fragments of a single file on an NTFS volume, which also implies that it is in general not a good idea to use NTFS compression on a file larger than 10 GB.

The NTFS filesystem driver will sometimes attempt to relocate the data of some of the attributes that can be made non-resident into the clusters, and will also attempt to relocate the data stored in clusters back to the attribute inside the MFT record, based on priority and preferred ordering rules, and size constraints.

Since resident files do not directly occupy clusters ("allocation units"), it is possible for an NTFS volume to contain more files on a volume than there are clusters. For example, a 74.5 GB partition NTFS formats with 19,543,064 clusters of 4 KB. Subtracting system files (a 64 MB log file, a 2,442,888-byte Bitmap file, and about 25 clusters of fixed overhead) leaves 19,526,158 clusters free for files and indices. Since there are four MFT records per cluster, this volume theoretically could hold almost 4 × 19,526,158 = 78,104,632 resident files.

### Opportunistic Locks

Opportunistic locks (oplocks) allow clients to alter their buffering strategy for a given file or stream in order to increase performance and reduce network use.  Oplocks apply to the given open stream of a file and do not affect oplocks on a different stream.

Oplocks can be used to transparently access files in the background. A network client may avoid writing information into a file on a remote server if no other process is accessing the data, or it may buffer read-ahead data if no other process is writing data.

Windows supports four different types of oplocks:

* Level 2 (or shared) oplock: multiple readers, no writers (i.e. read caching).
* Level 1 (or exclusive) oplock: exclusive access with arbitrary buffering (i.e. read and write caching).
* Batch oplock (also exclusive): a stream is opened on the server, but closed on the client machine (i.e. read, write and handle caching).
* Filter oplock (also exclusive): applications and file system filters can "back out" when others try to access the same stream (i.e. read and write caching) (since Windows 2000)

Opportunistic locks have been enhanced in Windows 7 and Windows Server 2008 R2 with per-client oplock keys.

### Time

Windows NT and its descendants keep internal timestamps as [UTC](http://en.wikipedia.org/wiki/UTC) and make the appropriate conversions for display purposes. Therefore, NTFS timestamps are in UTC.

For historical reasons, the versions of Windows that do not support NTFS all keep time internally as local zone time, and therefore so do all file systems other than NTFS that are supported by current versions of Windows. This means that when files are copied or moved between NTFS and non-NTFS partitions, the OS needs to convert timestamps on the fly. But if some files are moved when [daylight saving time](http://en.wikipedia.org/wiki/Daylight_saving_time) (DST) is in effect, and other files are moved when [standard time](http://en.wikipedia.org/wiki/Standard_time) is in effect, there can be some ambiguities in the conversions. As a result, especially shortly after one of the days on which local zone time changes, users may observe that some files have timestamps that are incorrect by one hour. Due to the differences in implementation of DST in different jurisdictions, this can result in a potential timestamp error of up to 4 hours in any given 12 months.

**exFAT**

exFAT was first introduced in late 2006 as part of [Windows CE 6.0](http://en.wikipedia.org/wiki/Windows_CE_6.0), an embedded Windows operating system. Most of the vendors signing on for licenses of exFAT are either for embedded systems or device manufacturers that produce media that will be preformatted with exFAT. The entire FAT family, exFAT included, is used for embedded systems because it is lightweight and is better suited for solutions that have low memory and low power requirements, and can be implemented in firmware.

exFAT allows individual files larger than 4 GiB, facilitating long continuous recording of HD video which can exceed the 4 GiB limit in less than an hour. Current digital cameras using [FAT32](http://en.wikipedia.org/wiki/FAT32) will break the video files into multiple segments of approximately 2 or 4 GiB. With the increase of capacity and the increase of data being transferred, the write operation needs to be made more efficient. SDXC cards, running at UHS-I have a minimum guaranteed write speed of 10 MBps and exFAT plays a factor in achieving that throughput through the reduction of the file system overhead in cluster allocation. This is achieved through the introduction of a cluster bitmap and elimination (or reduction) of writes to the FAT table. A single bit in the directory record indicates that the file is contiguous, telling the exFAT driver to ignore the FAT table. This optimization is analogous to an [extent](http://en.wikipedia.org/wiki/Extent_(file_systems)) in other file systems, except that it only applies to whole files, as opposed to contiguous parts of files.

exFAT is also supported in a number of media devices such as modern flat panel TVs, media centers, and portable media players.

Some vendors of flash media, including USB pen drives, compact flash (CF) and solid state drives (SSD) are shipping from the factory some of their high capacity media pre-formatted with the exFAT file system. For example, Sandisk ships their 256 GiB CF cards as exFAT.

## **Technical Specialties of exFAT**

### File Name Lookup

Like [NTFS](http://en.wikipedia.org/wiki/NTFS) and [HFS+](http://en.wikipedia.org/wiki/HFS%2B), exFAT is a proprietary file system. exFAT is protected under US patent law. Its initial application was issued on July 10, 2009 under the application number US2008168029.[[7]](http://en.wikipedia.org/wiki/ExFAT#cite_note-8)On November 12, 2013, the patent was granted by the US patent office under US8583708. Microsoft had also applied for, and so far in some cases received additional US patents on separate components that are used within exFAT. On November 27, 2012, US patent US8321439 was granted for *Quick File Name Lookup Using Name Hash*, which is an algorithm used in exFAT to speed up file searches. Since Microsoft has not officially released the entire exFAT specification, the *Name Hash* patent application is a key document in providing internal details in the understanding of the file system, since those details were revealed in Appendix A of the application.

### File and Cluster Pre-Allocation

Like [NTFS](http://en.wikipedia.org/wiki/NTFS), exFAT can pre-allocate disk space for a file by just marking arbitrary space on disk as 'allocated'. For each file, exFAT uses two separate 64-bit length fields in the directory: the Valid Data Length (VDL) which indicates the real size of the file, and the physical data length.

To provide improvement in the allocation of cluster storage for a new file, Microsoft incorporated a method to pre-allocate contiguous clusters and bypass the use of updating the FAT table and on December 10, 2013 the US patent office granted patent US8606830. One feature of exFAT (used in the exFAT implementation within embedded systems) provides atomic transactions for the multiple steps of updating the file system metadata. The feature, called *Transaction Safe FAT*, or [TexFAT](http://en.wikipedia.org/wiki/TexFAT" \o "TexFAT), was granted a patent by the US patent office under US7613738 on November 3, 2009.

### Directory File Set

exFAT and the rest of the FAT family of filesystems does not use indexes for filenames, unlike [NTFS](http://en.wikipedia.org/wiki/NTFS) which uses [B-trees](http://en.wikipedia.org/wiki/B-trees) for file searching. When a file is accessed, the directory must be sequentially searched until a match is found. For filenames shorter than 16 characters in length, one filename record is required but the entire file is represented by three 32-byte directory records. This is called a directory file set, and a 256 MiB sub-directory can hold up to 2,796,202 file sets. (If files have longer names, this number will decrease but this is the maximum based on the minimum 3 record file set.) To help improve the sequential searching of the directories (including the root) a hash value of the filename is derived for each file and stored in the directory record. When searching for a file, the file name is first converted to upper case using the upcase table (file names are case insensitive) and then hashed using a proprietary patented algorithm into a 16-bit hash value. Each record in the directory is searched by comparing the hash value. When a match is found, the filenames are compared to ensure that the proper file was located in case of collisions. This improves performance because only two characters have to be compared for each file. This significantly reduces the CPU cycles because most filenames are more than 2 characters in size and each comparison is only performed on two of the characters at a time until the intended file is located.

### Metadata and Checksums

exFAT introduces metadata integrity through the use of checksums. There are three checksums currently in use. The Volume Boot Record (VBR) is a 12 sector region that contains the boot records, BIOS Parameter Block (BPB), OEM parameters and the checksum sector. (There are two VBR type regions, the main VBR and the backup VBR.) The checksum sector is a checksum of the previous 11 sectors, with the exception of three bytes in the boot sector (Flags and percent used). This provides integrity of the VBR by determining if the VBR was modified. The most common cause could be a boot sector virus, but this would also catch any other corruption to the VBR. A second checksum is used for the upcase table. This is a static table and should never change. Any corruption in the table could prevent files from being located because this table is used to convert the filenames to upper case when searching to locate a file. The third checksum is in the directory file sets. Multiple directory records are used to define a single file and this is called a file set. This file set has metadata including the file name, time stamps, attributes, address of first cluster location of the data, file lengths, and the file name. A checksum is taken over the entire file set and a mismatch would occur if the directory file set was accidentally or maliciously changed. When the filesystem is mounted, and the integrity check is conducted, these hashes are verified. Mounting also includes comparison of the version of the exFAT filesystem by the driver to make sure the driver is compatible with the filesystem it is trying to mount, and to make sure that none of the required directory records are missing (for example, the directory record for the upcase table and Allocation Bitmap are required and the filesystem can't run if they are missing). If any of these checks fail, the filesystem should not be mounted, although in certain cases it may mount read-only.

The file system provides extensibility through template based metadata definitions using generic layouts and generic patterns

**Resilient File System** (**ReFS**)

Resilient File System (ReFS), codenamed "Protogon", is a [Microsoft](http://en.wikipedia.org/wiki/Microsoft) [proprietary](http://en.wikipedia.org/wiki/Proprietary_format) [file system](http://en.wikipedia.org/wiki/File_system) introduced with [Windows Server 2012](http://en.wikipedia.org/wiki/Windows_Server_2012) with the intent of becoming the "next generation" [file system](http://en.wikipedia.org/wiki/File_system) after [NTFS](http://en.wikipedia.org/wiki/NTFS).

ReFS was designed to overcome issues that had become significant over the years since NTFS was conceived, related to how data storage requirements had changed. Its key design advantages are intended to include — automatic [integrity checking](http://en.wikipedia.org/wiki/File_integrity_monitoring) and [data scrubbing](http://en.wikipedia.org/wiki/Data_scrubbing), removing the need for [chkdsk](http://en.wikipedia.org/wiki/Chkdsk" \o "Chkdsk) and protecting against [data degradation](http://en.wikipedia.org/wiki/Data_degradation); built-in handling of [hard drive failure](http://en.wikipedia.org/wiki/Hard_disk_drive_failure) and [redundancy](http://en.wikipedia.org/wiki/Redundancy_(engineering)), including [RAID](http://en.wikipedia.org/wiki/RAID) and a switch to [copy/allocate on write](http://en.wikipedia.org/wiki/Copy-on-write) for data and metadata updates; [very long path and filename](http://en.wikipedia.org/wiki/Long_filename) handling; and [storage virtualization](http://en.wikipedia.org/wiki/Storage_virtualization) and pooling, including almost arbitrary [logical volume](http://en.wikipedia.org/wiki/Logical_volume) size (as distinct from the physical sizes of the disks used).

These requirements arose from two major changes in storage systems and usage — the size of storage in use (large or massive arrays of multi-terabyte drives now being fairly common), and the need for [continual reliability](http://en.wikipedia.org/wiki/24/7). As a result, the file system needs to be self-repairing (to prevent disk checking from being impractically slow or disruptive), along with [abstraction](http://en.wikipedia.org/wiki/Abstraction_layer) or virtualization between physical disks and logical volumes.

ReFS was initially added to [Windows Server 2012](http://en.wikipedia.org/wiki/Windows_Server_2012) only, with the aim of gradual migration to consumer systems in future versions (although [modifications](http://en.wikipedia.org/wiki/Modding) were quickly developed by enthusiasts for the latter). The initial versions removed some NTFS features, causing concern among onlookers, such as [quota](http://en.wikipedia.org/wiki/Disk_quota) systems and [extended attributes](http://en.wikipedia.org/wiki/Extended_attributes). Some of these were re-implemented in later versions of ReFS.

In its early versions (2012–2013), ReFS was similar or slightly faster than NTFS in most tests,[[4]](http://en.wikipedia.org/wiki/ReFS" \l "cite_note-firstever-4)but far slower when full integrity checking was enabled, a result attributed to the relative newness of ReFS.[[5]](http://en.wikipedia.org/wiki/ReFS" \l "cite_note-bariseris-5)[[6]](http://en.wikipedia.org/wiki/ReFS#cite_note-jasoneckert-6) Concerns were also raised over [Storage Spaces](http://en.wikipedia.org/wiki/Storage_Spaces), the storage system designed to underpin ReFS, which is able to fail in a manner that prevents ReFS itself from recovering.

## **Changes Compared to NTFS**

**Improved reliability for on-disk structures**

ReFS uses B+ trees for all on-disk structures, including all metadata and file data. Metadata and file data are organized into tables similar to a relational database. The file size, number of files in a folder, total volume size and number of folders in a volume are limited by 64-bit numbers; as a result, ReFS supports a maximum file size of 16 exabytes, a maximum of 18.4 × 1018 directories and a maximum volume size of 1 yottabyte (with 64 KB clusters) which allows large scalability with no practical limits on file and directory size (hardware restrictions still apply). Free space is counted by a hierarchical allocator which includes three separate tables for large, medium, and small chunks. Data scrubbing can be enabled optionally.

**Built-in resilience**

ReFS employs an allocation-on-write update strategy for metadata, which allocates new chunks for every update transaction and uses large IO batches. All ReFS metadata has built-in 64-bit checksums which are stored independently. The file data can have an optional checksum in a separate "integrity stream", in which case the file update strategy also implements allocation-on-write; this is controlled by a new "integrity" attribute applicable to both files and directories. If nevertheless file data or metadata becomes corrupt, the file can be deleted without taking down the whole volume offline for maintenance, and then restored from the backup. As a result of built-in resiliency, administrators do not need to periodically run error-checking tools such as CHKDSK when using ReFS.

**Compatibility with Existing APIs and Technologies**

ReFS supports only a subset of NTFS features, and only Win32 APIs that are "widely adopted"; but does not require new system APIs and most file system filters continue to work with ReFS volumes. ReFS supports many existing Windows and NTFS features such as BitLocker encryption, Access Control Lists, USN Journal, change notifications, symbolic links, junction points, mount points, reparse points, volume snapshots, file IDs, and oplock. ReFS seamlessly integrates with Storage Spaces, a storage virtualization layer that allows data mirroring and striping, as well as sharing storage pools between machines. ReFS resiliency features enhance the mirroring feature provided by Storage Spaces and can detect whether any mirrored copies of files become corrupt using background data scrubbing process, which periodically reads all mirror copies and verifies their checksums then replaces bad copies with good ones.

Windows Registry Concepts and Keys

The Windows Registry is a hierarchical database that stores configuration settings and options on Microsoft Windows operating systems. It contains settings for low-level operating system components and for applications running on the platform that have opted to use the Registry. The kernel, device drivers, services, SAM, user interface and third party applications can all make use of the Registry. The Registry also provides a means to access counters for profiling system performance.

When first introduced with Windows 3.1, Windows Registry's primary purpose was to store configuration information for COM-based components. With the introduction of Windows 95 and Windows NT, its use was extended to tidy up the profusion of per-program INI files that had previously been used to store configuration settings for Windows programs.  It is not a requirement for a Windows application to use Windows Registry—for example, .NET Framework applications use XML files for configuration, while portable applications usually keep their configuration data within files in the folder where the application executable resides.

**Rationale**

Prior to the Registry, .INI files stored each program's settings into a text file, often located in a shared location that did not allow for user-specific settings in a multi-user scenario. By contrast, Windows Registry stores all application settings in one logical repository (but a number of discrete files) and in a standardized form. According to Microsoft this offers several advantages over .INI files. Since file parsing is done much more efficiently with a binary format, it may be read from or written to more quickly than an INI file. As well, strongly typed data can be stored in the Registry, as opposed to the text information stored in .INI files. This is a benefit when editing keys manually using regedit.exe, the built-in Windows Registry editor. Because user-based Registry settings are loaded from a user-specific path rather than from a read-only system location, the Registry allows multiple users to share the same machine, and also allows programs to work for less privileged users. Backup and restoration is also simplified as the Registry can be accessed over a network connection for remote management/support, including from scripts, using the standard set of APIs, as long as the Remote Registry service is running and firewall rules permit this.

As the Registry is constructed as a database, it offers improved system integrity with features such as atomic updates. If two processes attempt to update the same Registry value at the same time, one process's change will precede the others and the overall consistency of the data will be maintained. Where changes are made to .INI files, such race conditions can result in inconsistent data which doesn't match either attempted update. Windows Vista and later operating systems provide transactional updates to the Registry by means of the Kernel Transaction Manager, extending the atomicity guarantees across multiple key and/or value changes, with traditional commit-abort semantics. (Note however that NTFS provides such support for the file system as well, so the same guarantees could, in theory, be obtained with traditional configuration files.)

**Structure**

**Keys and Values**

The Registry contains two basic elements: **keys** and **values**. Registry *keys* are container objects similar to folders. Registry *values* are non-container objects similar to files. Keys may contain values or further keys. Keys are referenced with a syntax similar to Windows' path names, using backslashes to indicate levels of hierarchy. Keys must have a case insensitive name without backslashes.

The hierarchy of Registry keys can only be accessed from a known root key handle (which is anonymous but whose effective value is a constant numeric handle) that is mapped to the content of a Registry key preloaded by the kernel from a stored "hive", or to the content of a subkey within another root key, or mapped to a registered service or DLL that provides access to its contained subkeys and values.

There are seven predefined root keys, traditionally named according to their constant handles defined in the Win32 API, or by synonymous abbreviations (depending on applications):

* HKEY\_LOCAL\_MACHINE or HKLM
* HKEY\_CURRENT\_CONFIG or HKCC (only in Windows 9x and NT)
* HKEY\_CLASSES\_ROOT or HKCR
* HKEY\_CURRENT\_USER or HKCU
* HKEY\_USERS or HKU
* HKEY\_PERFORMANCE\_DATA (only in Windows NT, but invisible in the Windows Registry Editor)
* HKEY\_DYN\_DATA (only in Windows 9x, and visible in the Windows Registry Editor)

Like other files and services in Windows, all Registry keys may be restricted by access control lists(ACLs), depending on user privileges, or on security tokens acquired by applications, or on system security policies enforced by the system (these restrictions may be predefined by the system itself, and configured by local system administrators or by domain administrators). Different users, programs, services or remote systems may only see some parts of the hierarchy or distinct hierarchies from the same root keys.

Registry *values* are name/data pairs stored within keys. Registry values are referenced separately from Registry keys. Each Registry value stored in a Registry key has a unique name whose letter case is not significant. The Windows API functions that query and manipulate Registry values take value names separately from the key path and/or handle that identifies the parent key. Registry values may contain backslashes in their names, but doing so makes them difficult to distinguish from their key paths when using some legacy Windows Registry API functions (whose usage is deprecated in Win32).

The terminology is somewhat misleading, as each Registry key is similar to an associative array, where standard terminology would refer to the name part of each Registry value as a "key". The terms are a holdout from the 16-bit Registry in Windows 3, in which Registry keys could not contain arbitrary name/data pairs, but rather contained only one unnamed value (which had to be a string). In this sense, the entire Registry was like a single associative array where the Registry keys (in both the Registry sense and dictionary sense) formed a hierarchy, and the Registry values were all strings. When the 32-bit Registry was created, so was the additional capability of creating multiple named values per key, and the meanings of the names were somewhat distorted. For compatibility with the previous behavior, each Registry key may have a "default" value, whose name is the empty string.

Each value can store arbitrary data with variable length and encoding, but which is associated with a symbolic type (defined as a numeric constant) defining how to parse this data. The standard types are:

|  |  |  |
| --- | --- | --- |
| **List of standard Registry value types** | | |
| **Type ID** | **Symbolic type name** | **Meaning and encoding of the data stored in the Registry value** |
| 0 | REG\_NONE | No type (the stored value, if any) |
| 1 | REG\_SZ | A string value, normally stored and exposed in UTF-16LE (when using the Unicode version of Win32 API functions), usually terminated by a NUL character |
| 2 | REG\_EXPAND\_SZ | An "expandable" string value that can contain environment variables, normally stored and exposed in UTF-16LE, usually terminated by a NUL character |
| 3 | REG\_BINARY | Binary data (any arbitrary data) |
| 4 | REG\_DWORD / REG\_DWORD\_LITTLE\_ENDIAN | A DWORD value, a 32-bit unsigned integer(numbers between 0 and 4,294,967,295 [232 – 1]) (little-endian) |
| 5 | REG\_DWORD\_BIG\_ENDIAN | A DWORD value, a 32-bit unsigned integer(numbers between 0 and 4,294,967,295 [232 – 1]) (big-endian) |
| 6 | REG\_LINK | A symbolic link (UNICODE) to another Registry key, specifying a root key and the path to the target key |
| 7 | REG\_MULTI\_SZ | A multi-string value, which is an ordered list of non-empty strings, normally stored and exposed in UTF-16LE, each one terminated by a NUL character, the list being normally terminated by a second NUL character. |
| 8 | REG\_RESOURCE\_LIST | A resource list (used by the *Plug-n-Play* hardware enumeration and configuration) |
| 9 | REG\_FULL\_RESOURCE\_DESCRIPTOR | A resource descriptor (used by the *Plug-n-Play* hardware enumeration and configuration) |
| 10 | REG\_RESOURCE\_REQUIREMENTS\_LIST | A resource requirements list (used by the *Plug-n-Play* hardware enumeration and configuration) |
| 11 | REG\_QWORD / REG\_QWORD\_LITTLE\_ENDIAN | A QWORD value, a 64-bit integer (either big- or little-endian, or unspecified) (introduced in Windows XP) |

**Root Keys**

The keys at the root level of the hierarchical database are generally named by their Windows API definitions, which all begin "HKEY". They are frequently abbreviated to a three- or four-letter short name starting with "HK" (e.g. HKCU and HKLM). Technically, they are predefined handles (with known constant values) to specific keys that are either maintained in memory, or stored in hive files stored in the local filesystem and loaded by the system kernel at boot time and then shared (with various access rights) between all processes running on the local system, or loaded and mapped in all processes started in a user session when the user logs on the system.

The HKEY\_LOCAL\_MACHINE (local machine-specific configuration data) and HKEY\_CURRENT\_USER (user-specific configuration data) nodes have a similar structure to each other; user applications typically look up their settings by first checking for them in "HKEY\_CURRENT\_USER\Software\Vendor's name\Application's name\Version\Setting name", and if the setting is not found, look instead in the same location under the HKEY\_LOCAL\_MACHINE key. However, the converse may apply for administrator-enforced policy settings where HKLM may take precedence over HKCU. The Windows Logo Program has specific requirements for where different types of user data may be stored, and that the concept of least privilege be followed so that administrator-level access is not required to use an application.

**HKEY\_LOCAL\_MACHINE (HKLM)**

Abbreviated HKLM, HKEY\_LOCAL\_MACHINE stores settings that are specific to the local computer.

The key located by HKLM is actually not stored on disk, but maintained in memory by the system kernel in order to map all the other subkeys. Applications cannot create any additional subkeys. On Windows NT, this key contains four subkeys, "SAM", "SECURITY", "SYSTEM", and "SOFTWARE" that are loaded at boot time within their respective files located in the %SystemRoot%\System32\config folder. A fifth subkey, "HARDWARE", is volatile and is created dynamically, and as such is not stored in a file (it exposes a view of all the currently detected Plug-and-Play devices). On Windows Vista and above, a sixth and seventh subkey, "COMPONENTS" and "BCD", are mapped in memory by the kernel on-demand and loaded from %SystemRoot%\system32\config\COMPONENTS or from boot configuration data, \boot\BCD on the system partition.

* The "HKLM\SAM" key usually appears as empty for most users (unless they are granted access by administrators of the local system or administrators of domains managing the local system). It is used to reference all "Security Accounts Manager" (SAM) databases for all domains into which the local system has been administratively authorized or configured (including the local domain of the running system, whose SAM database is stored a subkey also named "SAM": other subkeys will be created as needed, one for each supplementary domain). Each SAM database contains all built-in accounts (mostly group aliases) and configured accounts (users, groups and their aliases, including guest accounts and administrator accounts) created and configured on the respective domain, for each account in that domain, it notably contains the user name which can be used to log on that domain, the internal unique user identifier in the domain, a cryptographic hash of each user's password for each enabled authentication protocol, the location of storage of their user Registry hive, various status flags (for example if the account can be enumerated and be visible in the logon prompt screen), and the list of domains (including the local domain) into which the account was configured.
* The "HKLM\SECURITY" key usually appears empty for most users (unless they are granted access by users with administrative privileges) and is linked to the Security database of the domain into which the current user is logged on (if the user is logged on the local system domain, this key will be linked to the Registry hive stored by the local machine and managed by local system administrators or by the built-in "System" account and Windows installers). The kernel will access it to read and enforce the security policy applicable to the current user and all applications or operations executed by this user. It also contains a "SAM" subkey which is dynamically linked to the SAM database of the domain onto which the current user is logged on.
* The "HKLM\SYSTEM" key is normally only writable by users with administrative privileges on the local system. It contains information about the Windows system setup, data for the secure random number generator (RNG), the list of currently mounted devices containing a filesystem, several numbered "HKLM\SYSTEM\Control Sets" containing alternative configurations for system hardware drivers and services running on the local system (including the currently used one and a backup), a "HKLM\SYSTEM\Select" subkey containing the status of these Control Sets, and a "HKLM\SYSTEM\CurrentControlSet" which is dynamically linked at boot time to the Control Set which is currently used on the local system. Each configured Control Set contains:
  + an "Enum" subkey enumerating all known Plug-and-Play devices and associating them with installed system drivers (and storing the device-specific configurations of these drivers),
  + a "Services" subkey listing all installed system drivers (with non-device specific configuration, and the enumeration of devices for which they are instantiated) and all programs running as services (how and when they can be automatically started),
  + a "Control" subkey organizing the various hardware drivers and programs running as services and all other system-wide configuration,
  + a "Hardware Profiles" subkey enumerating the various profiles that have been tuned (each one with "System" or "Software" settings used to modify the default profile, either in system drivers and services or in the applications) as well as the "Hardware Profiles\Current" subkey which is dynamically linked to one of these profiles.
* The "HKLM\SOFTWARE" subkey contains software and Windows settings (in the default hardware profile). It is mostly modified by application and system installers. It is organized by software vendor (with a subkey for each), but also contains a "Windows" subkey for some settings of the Windows user interface, a "Classes" subkey containing all registered associations from file extensions, MIME types, Object Classes IDs and interfaces IDs (for OLE, COM/DCOM and ActiveX), to the installed applications or DLLs that may be handling these types on the local machine (however these associations are configurable for each user, see below), and a "Policies" subkey (also organized by vendor) for enforcing general usage policies on applications and system services (including the central certificates store used for authenticating, authorizing or disallowing remote systems or services running outside the local network domain).
* The "HKLM\SOFTWARE\Wow6432Node" key is used by 32-bit applications on a 64-bit Windows OS, and is equivalent but separate to "HKLM\SOFTWARE". The key path is transparently presented to 32-bit applications by WoW64 as HKLM\SOFTWARE (in a similar way that 32-bit applications see %SystemRoot%\Syswow64 as %SystemRoot%\System32)

**HKEY\_CURRENT\_CONFIG (HKCC)**

Abbreviated HKCC, HKEY\_CURRENT\_CONFIG contains information gathered at runtime; information stored in this key is not permanently stored on disk, but rather regenerated at boot time. It is a handle to the key "HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Hardware Profiles\Current", which is initially empty but populated at boot time by loading one of the other subkeys stored in "HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Hardware Profiles".

**HKEY\_CLASSES\_ROOT (HKCR)**

Abbreviated HKCR, HKEY\_CLASSES\_ROOT contains information about registered applications, such as file associations and OLE Object Class IDs, tying them to the applications used to handle these items. On Windows 2000 and above, HKCR is a compilation of user-based HKCU\Software\Classes and machine-based HKLM\Software\Classes. If a given value exists in both of the subkeys above, the one in HKCU\Software\Classes takes precedence.  The design allows for either machine- or user-specific registration of COM objects.

**HKEY\_USERS (HKU)**

Abbreviated HKU, HKEY\_USERS contains subkeys corresponding to the HKEY\_CURRENT\_USER keys for each user profile actively loaded on the machine, though user hives are usually only loaded for currently logged-in users.

**HKEY\_CURRENT\_USER (HKCU)**

Abbreviated HKCU, HKEY\_CURRENT\_USER stores settings that are specific to the currently logged-in user. The HKEY\_CURRENT\_USER key is a link to the subkey of HKEY\_USERS that corresponds to the user; the same information is accessible in both locations. On Windows NT systems, each user's settings are stored in their own files called NTUSER.DAT and USRCLASS.DAT inside their own Documents and Settings subfolder (or their own Users sub folder in Windows Vista and above). Settings in this hive follow users with a roaming profile from machine to machine.

**HKEY\_PERFORMANCE\_DATA**

This key provides runtime information into performance data provided by the NT kernel itself or running system drivers, programs and services that provide performance data. This key is not stored in any hive and not displayed in the Registry Editor, but it is visible through the Registry functions in the Windows API, or in a simplified view via the Performance tab of the Task Manager (only for a few performance data on the local system) or via more advanced control panels (such as the Performances Monitor or the Performances Analyzer which allows collecting and logging these data, including from remote systems).

**HKEY\_DYN\_DATA**

This key is used only on Windows 95, Windows 98 and Windows ME. It contains information about hardware devices, including Plug and Play and network performance statistics. The information in this hive is also not stored on the hard drive. The Plug and Play information is gathered and configured at startup and is stored in memory.

**Hives**

Even though the Registry presents itself as an integrated hierarchical database, branches of the Registry are actually stored in a number of disk files called *hives*.

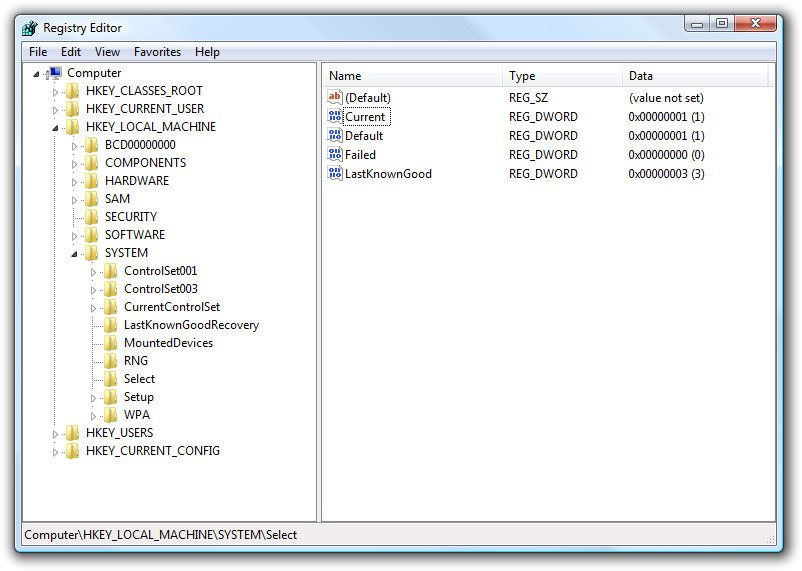
Some hives are volatile and are not stored on disk at all. An example of this is the hive of branch starting at HKLM\HARDWARE. This hive records information about system hardware and is created each time the system boots and performs hardware detection.

Individual settings for users on a system are stored in a hive (disk file) per user. During user login the system loads the user hive under the HKEY\_USERS key, but it also sets the HKCU (HKEY\_CURRENT\_USER) symbolic reference to point to the current user. This allows applications to store/retrieve settings for the current user implicitly under the HKCU key.

Not all hives are loaded at any one time. At boot time only a minimal set of hives are loaded and after that hives are loaded as the operating system initializes and as users log in or whenever a hive is explicitly loaded by an application.

**Editing**

**Registry Editors**



*Registry Editor* in Windows Vista

The Registry contains important configuration information for the operating system, for installed applications as well as individual settings for each user and application. A careless change to the operating system configuration in the Registry could cause irreversible damage, so it is usually only installer programs which perform changes to the Registry database during installation/configuration and removal. If a user wants to edit the Registry manually, Microsoft recommends that a backup of the Registry is performed before the change. Editing the Registry is sometimes necessary when working around Windows-specific issues e.g. problems when logging onto a domain can be resolved by editing the Registry.

Windows Registry can be edited manually using programs such as regedit.exe, although these tools do not expose some of Registry's metadata such as the last modified date.

Registry Editor allows users to perform the following functions:

* Creating, manipulating, renaming and deleting Registry keys, subkeys, values and value data
* Importing and exporting .REG files, exporting data in the binary hive format
* Loading, manipulating and unloading Registry hive format files (Windows NT systems only)
* Setting permissions based on ACLs (Windows NT systems only)
* Bookmarking user-selected Registry keys as Favorites
* Finding particular strings in key names, value names and value data
* Remotely editing the Registry on another networked computer

**.REG files**

.REG files (also known as Registration entries) are text-based human-readable files for exporting and importing portions of the Registry. On Windows 2000 and later, they contain the string *Windows Registry Editor Version 5.00* at the beginning and are Unicode-based. On Windows 9x and NT 4.0 systems, they contain the string *REGEDIT4* and are ANSI-based. Windows 9x format .REG files are compatible with Windows 2000 and later. The Registry Editor on Windows on these systems also supports exporting .REG files in Windows 9x/NT format. Data is stored in .REG files in the following syntax:

[<Hive name>\<Key name>\<Subkey name>]

"Value name"=<Value type>:<Value data>

The Default Value of a key can be edited by using @ instead of "Value Name":

[<Hive name>\<Key name>\<Subkey name>]

@=<Value type>:<Value data>

String values do not require a <Value type> (see example), but backslashes ('\') need to be written as a double-backslash ('\\'), and quotes ('"') as backslash-quote ('\"').

For example, to add the values "Value A", "Value B", "Value C", "Value D", "Value E", "Value F", "Value G", "Value H", "Value I", "Value J", "Value K", "Value L", and "Value M" to the HKLM\SOFTWARE\Foobar key,

Windows Registry Editor Version 5.00  
[HKEY\_LOCAL\_MACHINE\SOFTWARE\Foobar]  
"Value A"="<String value data with escape characters>"  
"Value B"=hex:<Binary data (as comma-delimited list of hexadecimal values)>  
"Value C"=dword:<DWORD value integer>  
"Value D"=hex(0):<REG\_NONE (as comma-delimited list of hexadecimal values)>  
"Value E"=hex(1):<REG\_SZ (as comma-delimited list of hexadecimal values representing a UTF-16LE NUL-terminated string)>  
"Value F"=hex(2):<Expandable string value data (as comma-delimited list of hexadecimal values representing a UTF-16LE NUL-terminated string)>  
"Value G"=hex(3):<Binary data (as comma-delimited list of hexadecimal values)> ; equal to "Value B"  
"Value H"=hex(4):<DWORD value (as comma-delimited list of 4 hexadecimal values, in little endian byte order)>  
"Value I"=hex(5):<DWORD value (as comma-delimited list of 4 hexadecimal values, in big endian byte order)>  
"Value J"=hex(7):<Multi-string value data (as comma-delimited list of hexadecimal values representing UTF-16LE NUL-terminated strings)>  
"Value K"=hex(8):<REG\_RESOURCE\_LIST (as comma-delimited list of hexadecimal values)>  
"Value L"=hex(a):<REG\_RESOURCE\_REQUIREMENTS\_LIST (as comma-delimited list of hexadecimal values)>  
"Value M"=hex(b):<QWORD value (as comma-delimited list of 8 hexadecimal values, in little endian byte order)>

Data from .REG files can be added/merged with the Registry by double-clicking these files or using the /s switch in the command line. .REG files can also be used to remove Registry data.

To remove a key (and all subkeys, values and data), the key name must be preceded by a minus sign ("-").

For example, to remove the HKLM\SOFTWARE\Foobar key (and all subkeys, values and data),

[-HKEY\_LOCAL\_MACHINE\SOFTWARE\Foobar]

To remove a value (and its data), the values to be removed must have a minus sign ("-") after the equal sign ("=").

For example, to remove only the "Value A" and "Value B" values (and their data) from the HKLM\SOFTWARE\Foobar key,

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Foobar]

"Value A"=-

"Value B"=-

To remove only the (Default) value of the key HKLM\SOFTWARE\Foobar (and its data),

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Foobar]

@=-

Lines beginning with a semicolon are considered as comments:

; This is a comment. This can be placed in any part of a .reg file

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Foobar]

"Value"="Example string"

**Group policies**

Windows group policies can change Registry keys for a number of machines or individual users based on policies. When a policy first takes effect for a machine or for an individual user of a machine, the Registry settings specified as part of the policy is applied to the machine or user settings.

Windows will also look for updated policies and apply them periodically, typically every 90 minutes.

Through its *scope* a policy defines which machines and/or users the policy is to be applied to. Whether a machine or user is within the scope of a policy or not is defined by a set of rules which can filter on the location of the machine/user account in organizational directory, specific user/machine accounts or security groups. More advanced rules can be set up using Windows Management Instrumentation expressions. Such rules can filter on properties such as computer vendor name, CPU architecture, installed software, networks connected to etc.

For instance, the administrator can create a policy with one set of Registry settings for machines in the accounting department and policy with another (lock-down) set of Registry settings for kiosk terminals in the visitors area. When a machine is moved from one scope to another (e.g. changing its name or moving it to another organizational unit), the correct policy is automatically applied. When a policy is changed it is automatically re-applied to all machines currently in its scope.

The policy is edited through a number of administrative templates which provides a user interface for picking and changing settings. The set of administrative templates is extensible and software packages which support such remote administration can register their own templates.

**Command Line Editing**

The Registry can be manipulated in a number of ways from the command line. The Reg.exe and RegIni.exe utility tools are included in Windows XP and later versions of Windows. Alternative locations for legacy versions of Windows include the Resource Kit CDs or the original Installation CD of Windows.

Also, a .REG file can be imported from the command line with the following command:

RegEdit.exe /s *file*

The /s means the file will be *silent merged* to the Registry. If the /s parameter is omitted the user will be asked to confirm the operation. In Windows 98, Windows 95 and at least some configurations of Windows XP the /s switch also causes RegEdit.exe to ignore the setting in the Registry that allows administrators to disable it. When using the /s switch RegEdit.exe does not return an appropriate return code if the operation fails, unlike Reg.exe which does.

RegEdit.exe /e *file*

exports the whole Registry in V5 format to a UNICODE .REG file, while any of

RegEdit.exe /e *file* HKEY\_CLASSES\_ROOT[\<key>]

RegEdit.exe /e *file* HKEY\_CURRENT\_CONFIG[\<key>]

RegEdit.exe /e *file* HKEY\_CURRENT\_USER[\<key>]

RegEdit.exe /e *file* HKEY\_LOCAL\_MACHINE[\<key>]

RegEdit.exe /e *file* HKEY\_USERS[\<key>]

export the specified (sub)key (which has to be enclosed in quotes if it contains spaces) only.

RegEdit.exe /a *file*

exports the whole Registry in V4 format to an ANSI .REG file.

RegEdit.exe /a *file* *<key>*

exports the specified (sub)key (which has to be enclosed in quotes if it contains spaces) only.

We can use also Reg.exe. Here is a sample to display the value of the Registry value Version:

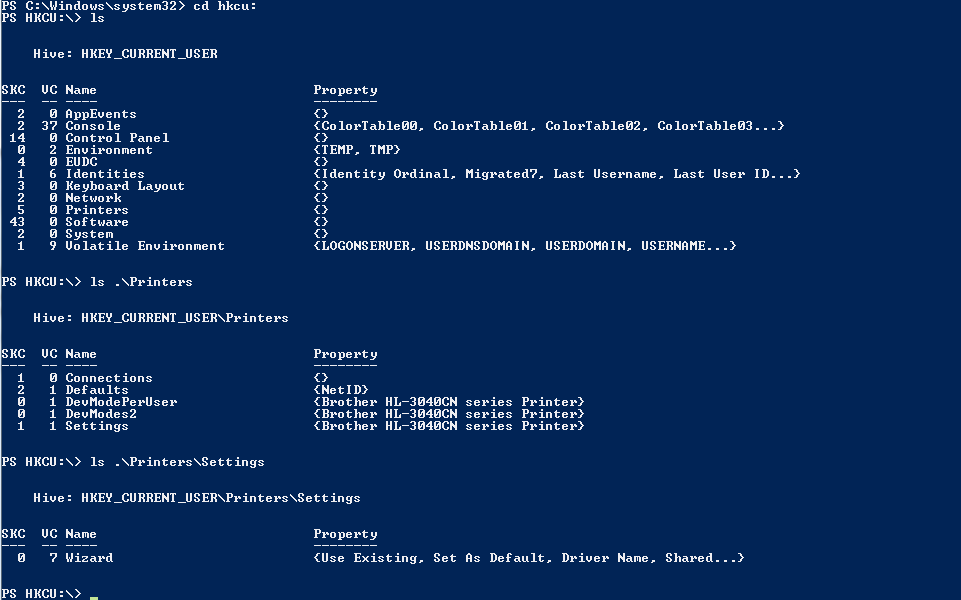
Reg.exe QUERY HKLM\Software\Microsoft\ResKit /v Version

Other command line options include a VBScript or JScript together with CScript, WMI orWMIC.exe and Windows PowerShell.

Registry permissions can be manipulated through the command line using RegIni.exe and theSubInACL.exe tool. For example, the permissions on the HKEY\_LOCAL\_MACHINE\SOFTWARE key can be displayed using:

SubInACL.exe /keyreg HKEY\_LOCAL\_MACHINE\SOFTWARE /display

**PowerShell Commands and Scripts**



Using PowerShell to navigate the Registry

Windows PowerShell comes with a Registry provider which presents the Registry as a location type similar to the file system. The same commands used to manipulate files/directories in the file system can be used to manipulate keys/values of the Registry.

Also like the file system, PowerShell uses the concept of a *current location* which defines the context on which commands by default operate. The Get-ChildItem (also available through the alias ls or dir) retrieves the child keys of the current location. By using the Set-Location command (or the alias cd) the user can change the current location to another key of the Registry. Commands which rename items, remove items, create new items or set content of items or properties can be used to rename keys, remove keys or entire sub-trees or change values.

Through PowerShell scripts files a user/administrator can prepare scripts which, when executed, make changes to the Registry. Such scripts can be distributed to users/administrators who can execute them on individual machines.

The PowerShell Registry provider supports transactions, i.e. multiple changes to the Registry can be bundled into a single atomic transaction. An atomic transaction ensures that either all of the changes are committed to the database, or if the script fails, none of the changes is committed to the database.

**Programs or Scripts**

The Registry can be edited through the APIs of the Advanced Windows 32 Base API Library (advapi32.dll).

|  |  |  |  |
| --- | --- | --- | --- |
| **List of Registry API functions** | | | |
| RegCloseKey | RegOpenKey | RegConnectRegistry | RegOpenKeyEx |
| RegCreateKey | RegQueryInfoKey | RegCreateKeyEx | RegQueryMultipleValues |
| RegDeleteKey | RegQueryValue | RegDeleteValue | RegQueryValueEx |
| RegEnumKey | RegReplaceKey | RegEnumKeyEx | RegRestoreKey |
| RegEnumValue | RegSaveKey | RegFlushKey | RegSetKeySecurity |
| RegGetKeySecurity | RegSetValue | RegLoadKey | RegSetValueEx |
|  | RegNotifyChangeKeyValue | RegUnLoadKey |  |

Many programming languages offer built-in runtime library functions or classes that wrap the underlying Windows APIs and thereby enable programs to store settings in the Registry (e.g.Microsoft.Win32.Registry in VB.NET and C#, or TRegistry in Delphi and Free Pascal).COM-enabled applications like Visual Basic 6 can use the WSH WScript.Shell object. Another way is to use the Windows Resource Kit Tool, Reg.exe by executing it from code, although this is considered poor programming practice.

Similarly, scripting languages such as Perl (with Win32::TieRegistry), Python (with winreg),TCL(which comes bundled with the Registry package), Windows Powershell and Windows Scripting Host also enable Registry editing from scripts.

**Offline Editing**

The offreg.dll available from the Windows Driver Kit offers a set of APIs for the creation and manipulation of currently not loaded Registry hives similar to those provided by advapi32.dll.

It is also possible to edit the Registry (hives) of an offline system from Windows PE or Linux (in the latter case using the open source tools).

**COM Self-Registration**

Prior to the introduction of registration-free COM, developers were encouraged to add initialization code to in-process and out-of-process binaries to perform the Registry configuration required for that object to work. For in-process binaries such as .DLL and .OCX files, the modules typically exported a function called DllInstall()that could be called by installation programs or invoked manually with utilities like Regsvr32.exe;out-of-process binaries typically support the command line arguments /Regserver and /Unregserver that created or deleted the required Registry settings. COM applications that break because of DLL issues can commonly be repaired with RegSvr32.exe or the /RegServer switch without having to re-invoke installation programs.

**Advanced Functionality**

Windows exposes APIs that allows user-mode applications to register to receive a notification event if a particular Registry key is changed. APIs are also available to allow kernel-mode applications to filter and modify Registry calls made by other applications.

Windows also supports remote access to the Registry of another computer via the RegConnectRegistry function if the Remote Registry service is running, correctly configured, and its network traffic is not firewalled.

**File Locations**

The Registry is physically stored in several files, which are generally obfuscated from the user-mode APIs used to manipulate the data inside the Registry. Depending upon the version of Windows, there will be different files and different locations for these files, but they are all on the local machine. The location for system Registry files in Windows NT is %SystemRoot%\System32\Config; the user-specific HKEY\_CURRENT\_USER user Registry hive is stored in Ntuser.dat inside the user profile. There is one of these per user; if a user has a roaming profile, then this file will be copied to and from a server at logout and login respectively. A second user-specific Registry file named UsrClass.dat contains COM Registry entries and does not roam by default.

**Windows NT**

Windows NT systems store the Registry in a binary file format which can be exported, loaded and unloaded by the Registry Editor in these operating systems. The following system Registry files are stored in %SystemRoot%\System32\Config\:

* Sam – HKEY\_LOCAL\_MACHINE\SAM
* Security – HKEY\_LOCAL\_MACHINE\SECURITY
* Software – HKEY\_LOCAL\_MACHINE\SOFTWARE
* System – HKEY\_LOCAL\_MACHINE\SYSTEM
* Default – HKEY\_USERS\.DEFAULT
* Userdiff – Not associated with a hive. Used only when upgrading operating systems.

The following file is stored in each user's profile folder:

* %USERPROFILE%\Ntuser.dat – HKEY\_USERS\<User SID> (linked to by HKEY\_CURRENT\_USER)

For Windows 2000, Server 2003 and Windows XP, the following additional user-specific file is used for file associations and COM information:

* %USERPROFILE%\Local Settings\Application Data\Microsoft\Windows\Usrclass.dat(path is localized) – HKEY\_USERS\<User SID>\_Classes (HKEY\_CURRENT\_USER\Software\Classes)

For Windows Vista and later, the path was changed to:

* %USERPROFILE%\AppData\Local\Microsoft\Windows\Usrclass.dat (path is not localized) alias %LocalAppData%\Microsoft\Windows\Usrclass.dat – HKEY\_USERS\<User SID>\_Classes (HKEY\_CURRENT\_USER\Software\Classes)

Windows 2000 kept an alternate copy of the Registry hives (.ALT) and attempts to switch to it when corruption is detected. Windows XP and Windows Server 2003 do not maintain a System.alt hive because NTLDR on those versions of Windows can process the System.log file to bring up to date a System hive that has become inconsistent during a shutdown or crash. In addition, the %SystemRoot%\Repair folder contains a copy of the system's Registry hives that were created after installation and the first successful startup of Windows.

Each Registry data file has an associated file with a ".log" extension that acts as a transaction log that is used to ensure that any interrupted updates can be completed upon next startup. Internally, Registry files are split into 4 kB "bins" that contain collections of "cells".

**Windows 9x**

The Registry files are stored in the %WINDIR% directory under the names USER.DAT andSYSTEM.DAT with the addition of CLASSES.DAT in Windows Me. Also, each user profile (if profiles are enabled) has its own USER.DAT file which is located in the user's profile directory in %WINDIR%\Profiles\<User name>\.

**Windows 3.11**

The only Registry file is called REG.DAT and it is stored in the %WINDIR% directory.

**Security**

Each key in the Registry of Windows NT versions can have an associated security descriptor. The security descriptor contains an access control list (ACL) that describes which user groups or individual users are granted or denied access permissions. The set of Registry permissions include 10 rights/permissions which can be explicitly allowed or denied to a user or a group of users.

|  |  |
| --- | --- |
| **Registry permissions** | |
| **Permission** | **Description** |
| Query Value | The right to read the Registry key value. |
| Set Value | The right to write a new value |
| Create Subkey | The right to create subkeys. |
| Enumerate Subkeys | Allow the enumeration of subkeys. |
| Notify | The right to request change notifications for Registry keys or subkeys. |
| Create Link | Reserved by the operating system. |
| Delete | The right to delete a key. |
| Write DACL | The right to modify permissions of the container’s DACL. |
| Write Owner | The right to modify the container’s owner. |
| Read Control | The right to read the DACL. |

As with other securable objects in the operating system, individual access control entries (ACE) on the security descriptor can be explicit or inherited from a parent object.

Windows Resource Protection is a feature of Windows Vista and later versions of Windows that uses security to deny Administrators and the system WRITE access to some sensitive keys to protect the integrity of the system from malware and accidental modification.

Special ACEs on the security descriptor can also implement mandatory integrity control for the Registry key and subkeys. A process running at a lower integrity level cannot write, change or delete a Registry key/value, even if the account of the process has otherwise been granted access through the ACL. For instance, Internet Explorer running in Protected Mode can *read* medium and low integrity Registry keys/values of the currently logged on user, but it can only modify low integrity keys.

Outside security, Registry keys cannot be deleted or edited due to other causes. Registry keys containing NUL characters cannot be deleted with standard registry editors and require a special utility for deletion, such as RegDelNull.

**Backups and Recovery**

Different editions of Windows have supported a number of different methods to back up and restore the Registry over the years, some of which are now deprecated:

* System Restore can back up the Registry and restore it as long as Windows is bootable, or from the Windows Recovery Environment starting with Windows Vista.
* NTBackup can back up the Registry as part of the *System State* and restore it. Automated System Recovery in Windows XP can also restore the Registry.
* On Windows NT, the *Last Known Good Configuration* option in startup menu relinks the HKLM\SYSTEM\CurrentControlSet key, which stores hardware and device driver information.
* Windows 98 and Windows Me include command line (Scanreg.exe) and GUI (Scanregw.exe) Registry checker tools to check and fix the integrity of the Registry, create up to five automatic regular backups by default and restore them manually or whenever corruption is detected. The Registry checker tool backs up the Registry, by default, to %Windir%\SysbckupScanreg.exe can also run from MS-DOS.
* The Windows 95 CD-ROM included an Emergency Recovery Utility (ERU.exe) and a Configuration Backup Tool (Cfgback.exe) to back up and restore the Registry. Additionally Windows 95 backs up the Registry to the files system.da0 and user.da0 on every successful boot.
* Windows NT 4.0 included RDISK.EXE, a utility to back up and restore the entire Registry.
* Windows 2000 Resource Kit contained an unsupported pair of utilities called Regback.exe and RegRest.exe for backup and recovery of the Registry.

**Group policy**

Windows 2000 and later versions of Windows use Group Policy to enforce Registry settings through a Registry-specific client extension in the Group Policy processing engine. Policy may be applied locally to a single computer using gpedit.msc, or to multiple users and/or computers in a domain using gpmc.msc.

**Legacy Systems**

With Windows 95, Windows 98, Windows ME and Windows NT 4.0, administrators can use a special file to be merged into the Registry, called a policy file (POLICY.POL). The policy file allows administrators to prevent non-administrator users from changing Registry settings like, for instance, the security level of Internet Explorer and the desktop background wallpaper. The policy file is primarily used in a business with a large number of computers where the business needs to be protected from rogue or careless users.

The default extension for the policy file is .POL. The policy file filters the settings it enforces by user and by group (a "group" is a defined set of users). To do that the policy file merges into the Registry, preventing users from circumventing it by simply changing back the settings. The policy file is usually distributed through a LAN, but can be placed on the local computer.

The policy file is created by a free tool by Microsoft that goes by the filename poledit.exe for Windows 95/Windows 98 and with a computer management module for Windows NT. The editor requires administrative permissions to be run on systems that use permissions. The editor can also directly change the current Registry settings of the local computer and if the remote Registry service is installed and started on another computer it can also change the Registry on that computer. The policy editor loads the settings it can change from .ADM files, of which one is included, that contains the settings the Windows shell provides. The .ADM file is plain text and supports easy localization by allowing all the strings to be stored in one place.

Networking in Windows

**Server Message Block (SMB)**

Server Message Block, which was also known as Common Internet File System (CIFS), operates as an application-layer network protocol mainly used for providing shared access to files, printers, serial ports, and miscellaneous communications between nodes on a network. It also provides an authenticated inter-process communication mechanism. Most usage of SMB involves computers running Microsoft Windows, where it was known as "Microsoft Windows Network" before the subsequent introduction of Active Directory. Corresponding Windows services are LAN Manager Server (for the server component) and LAN Manager Workstation (for the client component).

SMB can run on top of the Session (and lower) network layers in several ways:

* Directly over TCP, port 445;
* Via the NetBIOS API, which in turn can run on several transports:
  + On UDP ports 137, 138 & TCP ports 137, 139 (NetBIOS over TCP/IP);
  + On several legacy protocols such as NBF

The SMB "Inter-Process Communication" (IPC) system provides named pipes and was one of the first inter-process mechanisms commonly available to programmers that provides a means for services to inherit the authentication carried out when a client first connected to an SMB server.

Some services that operate over named pipes, such as those which use Microsoft's own implementation of DCE/RPC over SMB, known as MSRPC over SMB, also allow MSRPC client programs to perform authentication, which over-rides the authorization provided by the SMB server, but only in the context of the MSRPC client program that successfully makes the additional authentication.

*SMB signing*: Windows NT 4.0 Service Pack 3 and upwards have the capability to use cryptography to digitally sign SMB connections. The most common official term is "SMB signing". Other terms that have been used officially are "[SMB] Security Signatures", "SMB sequence numbers" and "SMB Message Signing". SMB signing may be configured individually for incoming SMB connections (handled by the "LanManServer" service) and outgoing SMB connections (handled by the "LanManWorkstation" service). The default setting from Windows 98 and upwards is to opportunistically sign outgoing connections whenever the server also supports this, and to fall back to unsigned SMB if both partners allow this. The default setting for Windows domain controllers from Server 2003 and upwards is to not allow fall back for incoming connections. The feature can also be turned on for any server running Windows NT 4.0 Service Pack 3 or later. This protects from man-in-the-middle attacks against the Clients retrieving their policies from domain controllers at login.

The design of Server Message Block version 2 (SMB2) aims to mitigate this performance-limitation by coalescing SMB signals into single packets.

SMB supports opportunistic locking — a special type of locking-mechanism — on files in order to improve performance.

SMB serves as the basis for Microsoft's Distributed File System implementation.

**SMB Implementation**

**Client-server approach**

SMB works through a client-server approach, where a client makes specific requests and the server responds accordingly. One section of the SMB protocol specifically deals with access tofilesystems, such that clients may make requests to a file server; but some other sections of the SMB protocol specialize in inter-process communication (IPC). The Inter-Process Communication (IPC) share, or ipc$, is a network share on computers running Microsoft Windows. This virtual share is used to facilitate communication between processes and computers over SMB, often to exchange data between computers that have been authenticated.

Developers have optimized the SMB protocol for local subnet usage, but users have also put SMB to work to access different subnets across the Internet—exploits involving file-sharing or print-sharing in MS Windows environments usually focus on such usage.

SMB servers make their file systems and other resources available to clients on the network. Client computers may want access to the shared file systems and printers on the server, and in this primary functionality SMB has become best-known and most heavily used. However, the SMB file-server aspect would count for little without the NT domains suite of protocols, which provide NT-style domain-based authentication at the very least. Almost all implementations of SMB servers use NT Domain authentication to validate user-access to resources.

**Samba**

Andrew Tridgell started the development of Samba, a free-software re-implementation of the SMB/CIFS networking protocol for Unix-like systems, in 1991. As of version 3 (2003), Samba provides file and print services for Microsoft Windows clients and can integrate with a Windows NT 4.0 server domain, either as a Primary Domain Controller (PDC) or as a domain member. Samba4 installations can act as an Active Directory domain controller or member server, at Windows 2008 domain and forest functional levels.

Note in relation to Samba the use of the Linux cifs-utils package.

**Kerberos**

**Kerberos** is a computer network authentication protocol which works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Its designers aimed it primarily at a client–server model and it provides mutual authentication—both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks.

Kerberos builds on symmetric key cryptography and requires a trusted third party, and optionally may use public-key cryptography during certain phases of authentication. Kerberos uses UDP port 88 by default.

Windows 2000 and later uses Kerberos as its default authentication method. Some Microsoft additions to the Kerberos suite of protocols are documented in RFC 3244 "Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols". RFC 4757 documents Microsoft's use of the RC4 cipher. While Microsoft uses the Kerberos protocol, it does not use the MIT software.

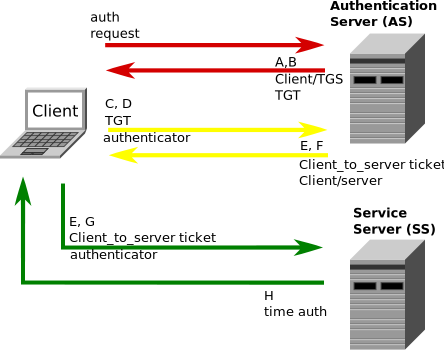
Kerberos is used as preferred authentication method: In general, joining a client to a Windows domain means enabling Kerberos as default protocol for authentications from that client to services in the Windows domain and all domains with trust relationships to that domain.

In contrast, when either client or server or both are not joined to a domain (or not part of the same trusted domain environment) Windows will instead use NTLM for authentication between client and server.

**Protocol Operation**

The client authenticates itself to the Authentication Server (AS) which forwards the username to a key distribution center (KDC). The KDC issues a ticket-granting ticket (TGT), which is time stamped, encrypts it using the user's password and returns the encrypted result to the user's workstation. This is done infrequently, typically at user logon; the TGT expires at some point, though may be transparently renewed by the user's session manager while they are logged in.

When the client needs to communicate with another node ("principal" in Kerberos parlance) the client sends the TGT to the ticket-granting service (TGS), which usually shares the same host as the KDC. After verifying the TGT is valid and the user is permitted to access the requested service, the TGS issues a ticket and session keys, which are returned to the client. The client then sends the ticket to the service server (SS) along with its service request.



**Kerberos negotiations**

The protocol is described in detail below.

**User Client-based Logon**

1. A user enters a username and password on the client machines. Other credential mechanisms like pkinit (RFC4556) allow for the use of public keys in place of a password.
2. The client transforms the password into the key of a symmetric cipher. This either uses the built in key scheduling or a one-way hash depending on the cipher-suite used.

**Client Authentication**

1. The client sends a cleartext message of the user ID to the AS requesting services on behalf of the user. (Note: Neither the secret key nor the password is sent to the AS.) The AS generates the secret key by hashing the password of the user found at the database (e.g., Active Directory in Windows Server).
2. The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
   * Message A: *Client/TGS Session Key* encrypted using the secret key of the client/user.
   * Message B: *Ticket-Granting-Ticket* (TGT, which includes the client ID, client network address, ticket validity period, and the *client/TGS session key*) encrypted using the secret key of the TGS.
3. Once the client receives messages A and B, it attempts to decrypt message A with the secret key generated from the password entered by the user. If the user entered password does not match the password in the AS database, the client's secret key will be different and thus unable to decrypt message A. With a valid password and secret key the client decrypts message A to obtain the *Client/TGS Session Key*. This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.) At this point, the client has enough information to authenticate itself to the TGS.

**Client Service Authorization**

1. When requesting services, the client sends the following two messages to the TGS:
   * Message C: Composed of the TGT from message B and the ID of the requested service.
   * Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the *Client/TGS Session Key*.
2. Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
   * Message E: *Client-to-server ticket* (which includes the client ID, client network address, validity period and *Client/Server Session Key*) encrypted using the service's secret key.
   * Message F: *Client/Server Session Key* encrypted with the *Client/TGS Session Key*.

**Client Service Request**

1. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
   * Message E from the previous step (the *client-to-server ticket*, encrypted using service's secret key).
   * Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using *Client/Server Session Key*.
2. The SS decrypts the ticket using its own secret key to retrieve the *Client/Server Session Key*. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
   * Message H: the timestamp found in client's Authenticator plus 1, encrypted using the *Client/Server Session Key*.
3. The client decrypts the confirmation using the *Client/Server Session Key* and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
4. The server provides the requested services to the client.

Command Line

**CMD**

Command Prompt, also known as cmd.exe or just cmd (after its executable file name), is the command-line interpreter on OS/2 and eComStation, Windows CE and Windows NT operating systems. It is the counterpart of COMMAND.COM in DOS and Windows 9x systems (where it is also called "MS-DOS Prompt"), or some of the Unix shells used on Unix-like systems.

**Powershell**

Windows PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems as well as WS-Management and CIM enabling management of remote Linux systems and network devices.

In PowerShell, administrative tasks are generally performed by cmdlets which are specialized .NET classes implementing a particular operation. Sets of cmdlets may be combined into scripts, executables (which are standalone applications), or by instantiating regular .NET classes (or WMI/COM Objects). These work by accessing data in different data stores, like the file system or registry, which are made available to the PowerShell runtime via Windows PowerShell providers.

Windows PowerShell also provides a hosting API with which the Windows PowerShell runtime can be embedded inside other applications. These applications can then use Windows PowerShell functionality to implement certain operations, including those exposed via the graphical interface. This capability has been used by Microsoft Exchange Server 2007 to expose its management functionality as PowerShell cmdlets and providers and implement the graphical management tools as PowerShell hosts which invoke the necessary cmdlets. Other Microsoft applications including Microsoft SQL Server 2008 also expose their management interface via PowerShell cmdlets. With PowerShell, graphical interface-based management applications on Windows are layered on top of Windows PowerShell. A PowerShell scripting interface for Windows products is mandated by Microsoft's Common Engineering Criteria.

Security and Logging

**Windows Firewall**

**Windows XP**

Windows Firewall was first introduced as part of Windows XP Service Pack 2. Every type of network connection, whether it is wired, wireless, VPN, or even FireWire, has the firewall enabled by default, with some built-in exceptions to allow connections from machines on the local network. It also fixed a problem whereby the firewall policies would not be enabled on a network connection until several seconds after the connection itself was created, thereby creating a window of vulnerability. A number of additions were made to Group Policy, so that Windows system administrators could configure the Windows Firewall product on a company-wide level. XP's Windows Firewall cannot block outbound connections; it is only capable of blocking inbound ones.

Windows Firewall turned out to be one of the two most significant reasons (the other being DCOM activation security) that many corporations did not upgrade to Service Pack 2 in a timely fashion. Around the time of SP2's release, a number of Internet sites were reporting significant application compatibility issues, though the majority of those ended up being nothing more than ports that needed to be opened on the firewall so that components of distributed systems (typically backup and antivirus solutions) could communicate.

The firewall does not filter IPv6 traffic.

Note that the DCOM problem can be solved by moving applications to DComLab's ComBridge protocol.

**Windows Vista**

Windows Vista improved the firewall to address a number of concerns around the flexibility of *Windows Firewall* in a corporate environment:

* The firewall is based on the Windows Filtering Platform.
* A new management console snap-in named *Windows Firewall with Advanced Security* which provides access to many advanced options, and enables remote administration. This can be accessed via Start -> Control Panel -> Administrative Tools -> Windows Firewall with Advanced Security, or by running "wf.msc"
* Outbound packet filtering, reflecting increasing concerns about spyware and viruses that attempt to "phone home". Outbound rules are configured using the management console. Notifications are not shown however for outbound connections.
* With the advanced packet filter, rules can also be specified for source and destination IP addresses and port ranges.
* Rules can be configured for services by its service name chosen by a list, without needing to specify the full path file name.
* IPsec is fully integrated, allowing connections to be allowed or denied based on security certificates, Kerberos authentication, etc. Encryption can also be required for any kind of connection.
* Improved interface for managing separate firewall profiles. Ability to have three separate firewall profiles for when computers are domain-joined, connected to a private network, or connected to a public network (XP SP2 supports two profiles—domain-joined and standard). Support for the creation of rules for enforcing server and domain isolation policies.

**Windows Server 2008 and Windows 7**

Windows Server 2008 contains the same firewall as Windows Vista. The firewall in Windows Server 2008 R2 and Windows 7 contains some improvements, such as multiple active profiles

**Windows Logs**

**Security**

The Security Log, in Microsoft Windows, is a log that contains records of login/logout activity or other security-related events specified by the system's audit policy. Auditing allows administrators to configure Windows to record operating system activity in the Security Log. The Security Log is one of three logs viewable under Event Viewer. Local Security Authority Subsystem Service writes events to the log. The Security Log is one of the primary tools used by Administrators to detect and investigate attempted and successful unauthorized activity and to troubleshoot problems.

If the audit policy is set to record logins, a successful login results in the user's user name and computer name being logged as well as the user name they are logging into.  Depending on the version of Windows and the method of login, the IP address may or may not be recorded. Windows 2000 Web Server, for instance, does not log IP addresses for successful logins, but Windows Server 2003 includes this capability. The categories of events that can be logged are:

* Account logon events
* Account management
* Directory service access
* Logon events
* Object access
* Policy change
* Privilege use
* Process tracking
* System events

The sheer number of logable events means that security log analysis can be a time-consuming task. Third-party utilities have been developed to help identify suspicious trends. It is also possible to filter the log using customized criteria. Logging is dangerous.

**Attacks and countermeasures**

Administrators are allowed to view and clear the log (there is no way to separate the rights to view and clear the log). In addition, an Administrator can use open source tools to delete specific events from the log. For this reason, once the Administrator account has been compromised, the event history as contained in the Security Log is unreliable.  A defense against this is to set up a remote log server with all services shut off, allowing only console access.

As the log approaches its maximum size, it can either overwrite old events or stop logging new events. This makes it susceptible to attacks in which an intruder can flood the log by generating a large number of new events. A partial defense against this is to increase the maximum log size so that a greater number of events will be required to flood the log.

Another way to defeat the Security Log would be for a user to log in as Administrator and change the auditing policies to stop logging the unauthorized activity he intends to carry out. The policy change itself could be logged, depending on the "audit policy change" setting, but this event could be deleted from the log using tools; and from that point onward, the activity would not generate a trail in the Security Log.

Microsoft notes, "It is possible to detect attempts to elude a security monitoring solution with such techniques, but it is challenging to do so because many of the same events that can occur during an attempt to cover the tracks of intrusive activity are events that occur regularly on any typical business network".

Log manipulation is not needed for all attacks. Simply being aware of how the Security Log works can be enough to take precautions against detection. For instance, a user wanting to log into a fellow employee's account on a corporate network might wait until after hours to gain unobserved physical access to the computer in their cubicle; surreptitiously use a hardware keylogger to obtain their password; and later log into that user's account through Terminal Services from a Wi-Fi hotspot whose IP address cannot be traced back to the intruder.

After the log is cleared through Event Viewer, one log entry is immediately created in the freshly cleared log noting the time it was cleared and the admin who cleared it. This information can be a starting point in the investigation of the suspicious activity. In addition to the Windows Security Log, administrators can check the Internet Connection Firewall security log for clues.

**Writing false events to the log**

It is theoretically possible to write false events to the log. Microsoft notes, "To be able to write to the Security log, SeAuditPrivilege is required. By default, only Local System and Network Service accounts have such privilege". *Microsoft Windows Internals* states, "Processes that call audit system services must have the SeAuditPrivilege privilege to successfully generate an audit record". Server 2003 added some API calls so that applications could register with the security event logs and write security audit entries. Specifically, the AuthzInstallSecurityEventSource function installs the specified source as a security event source.

**Application Log**

The application log file contains events that are logged by the applications used on a computer system. Events that are written to the application log are determined by the developers of the software program, not the operating system.

**System Log**

System events are logged by Windows and Windows system services, and are classified as error, warning, or information. Logs created here are of note when a particular vulnerability is exploited that causes a service, system or otherwise, to crash and restart.

User Accounts

A user account is a collection of information that tells Windows which files and folders you can access, what changes you can make to your computer, and your personal preferences, such as your desktop background or screen saver. User accounts let you share a computer with several people while having your own files and settings. Each person accesses his or her user account with a user name and password.

There are three types of accounts. Each type gives users a different level of control over the computer:

* **Standard** accounts are for everyday computing.
* **Administrator** accounts provide the most control over a computer and should only be used when necessary.
* **Guest** accounts are intended primarily for people who need temporary use of a computer.

**Power Users vs Administrators**

While administrators have full control over a particular machine, Power Users can:

* Run legacy applications, in addition to Windows 2000 or Windows XP Professional certified applications.
* Install programs that do not modify operating system files or install system services.
* Customize system-wide resources including printers, date, time, power options, and other Control Panel resources.
* Create and manage local user accounts and groups.
* Stop and start system services which are not started by default.

Power Users do not have permission to add themselves to the Administrators group. Power Users do not have access to the data of other users on an NTFS volume, unless those users grant them permission.

**User Account Control** (**UAC**) is a technology and security infrastructure introduced with Microsoft's Windows Vista and Windows Server 2008 operating systems, with a more relaxed version also present in Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012. It aims to improve the security of Microsoft Windows by limiting application software to standard user privileges until an administrator authorizes an increase or elevation. In this way, only applications trusted by the user may receive administrative privileges, and malware should be kept from compromising the operating system. In other words, a user account may have administrator privileges assigned to it, but applications that the user runs do not inherit those privileges unless they are approved beforehand or the user explicitly authorizes it.

To reduce the possibility of lower-privilege applications communicating with higher-privilege ones, another new technology, User Interface Privilege Isolation, is used in conjunction with User Account Control to isolate these processes from each other. One prominent use of this is Internet Explorer 7's "Protected Mode".

Volume Shadow Copy

Shadow Copy (also known as Volume Snapshot Service, Volume Shadow Copy Service or VSS) is a technology included in Microsoft Windows that allows taking manual or automatic backup copies or snapshots of computer files or volumes, even when they are in use. It is implemented as a Windows service called the Volume Shadow Copy service. The software VSS provider service is also included as part of Windows to be used by Windows applications. Shadow Copy technology requires the file system to be NTFS to be able to create and store shadow copies. Shadow Copies can be created on local and external (removable or network) volumes by any Windows component that uses this technology, such as when creating a scheduled Windows Backup or automatic System Restore point.

**Windows XP and Server 2003**

Volume Snapshot Service was first added to Microsoft Windows in Windows XP. It can only create temporary snapshots, used for accessing stable on-disk version of files that are opened for editing (and therefore locked). This version of VSS is used by NTBackup.

The creation of persistent snapshots (which remain available across reboots until specifically deleted) has been added in Windows Server 2003, allowing up to 512 snapshots to exist simultaneously for the same volume. In Windows Server 2003, VSS is used to create incremental periodic snapshots of data of changed files over time. A maximum of 64 snapshots are stored on the server and are accessible to clients over the network. This feature is known as Shadow Copies for Shared Folders and is designed for a client–server model. Its client component is included with Windows XP SP2 or later, and is available for installation on Windows 2000 SP3 or later, as well as Windows XP RTM or SP1.

Windows XP and later include a command line utility called vssadmin that can list, create or delete volume shadow copies and list installed shadow copy writers and providers.

**Windows Vista, 7 and Server 2008**

A number of Microsoft Windows components have been updated to make use of Shadow Copy. Backup and Restore in Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2 use shadow copies of files in both file-based and sector-by-sector backup. VSS is also used by the System Protection component which creates and maintains periodic copies of system and user data on the same local volume (similar to the Shadow Copies for Shared Folders feature in Windows Server) but allows it to be locally accessed by System Restore.

System Restore allows reverting to an entire previous set of shadow copies called a restore point. Prior to Windows Vista, System Restore was based on a file-based filter that watched changes for a certain set of file extensions, and then copied files before they were overwritten. In addition, a part of Windows Explorer called Previous Versions allows restoring individual files or folders locally from restore points as they existed at the time of the snapshot, thus retrieving an earlier version of a file or recovering a file deleted by mistake. Finally, Windows Server 2008 introduces the disk shadow utility which exposes VSS functionality through 20 different commands.

Shadow copies are created automatically once per day, or manually when triggered by the backup utility or installer applications which create a restore point. The "Previous Versions" feature is available in the Business, Enterprise, and Ultimate editions of Windows Vista and in all Windows 7 editions. The Home Editions of Vista lack the "Previous Versions" feature, even though the Volume Snapshot Service is included and running. Using third party tools it is still possible to restore previous versions of files on the local volume. Some of these tools also allow users to schedule snapshots at user-defined intervals, configure the storage used by volume shadow copies and compare files or directories from different points-in-time using snapshots.  Windows 7 also adds native support through a GUI to configure the storage used by volume shadow copies.

**Windows 8 and Server 2012**

On Windows 8 persistent shadow copies are no longer available. Therefore the ability to browse, search and/or recover older versions of files via the Previous Versions tab of the Properties dialog of files was removed for local volumes. The feature is still available in Windows Server 2012.

Memory Dumps

**Complete Memory Dump**

A complete memory dump records all the contents of system memory when your computer stops unexpectedly. A complete memory dump may contain data from processes that were running when the memory dump was collected.   
  
**Kernel Memory Dump**

A kernel memory dump records only the kernel memory. This speeds up the process of recording information in a log when your computer stops unexpectedly. You must have a pagefile large enough to accommodate your kernel memory. For 32-bit systems, kernel memory is usually between150MB and 2GB. Additionally, on Windows 2003 and Windows XP, the page file must be on the boot volume. Otherwise, a memory dump cannot be created.  
  
This dump file does not include unallocated memory or any memory that is allocated to User-mode programs. It includes only memory that is allocated to the kernel and hardware abstraction layer (HAL) in Windows 2000 and later, and memory allocated to Kernel-mode drivers and other Kernel-mode programs. For most purposes, this dump file is the most useful. It is significantly smaller than the complete memory dump file, but it omits only those parts of memory that are unlikely to have been involved in the problem.  
  
**Small Memory Dump**

A small memory dump records the smallest set of useful information that may help identify why your computer stopped unexpectedly. This option requires a paging file of at least 2 MB on the boot volume and specifies that Windows 2000 and later create a new file every time your computer stops unexpectedly. A history of these files is stored in a folder.  
  
This dump file type includes the following information:

* The Stop message and its parameters and other data
* A list of loaded drivers
* The processor context (PRCB) for the processor that stopped
* The process information and kernel context (EPROCESS) for the process that stopped
* The process information and kernel context (ETHREAD) for the thread that stopped
* The Kernel-mode call stack for the thread that stopped

This kind of dump file can be useful when space is limited. However, because of the limited information included, errors that were not directly caused by the thread that was running at the time of the problem may not be discovered by an analysis of this file.  
  
If a second problem occurs and a second small memory dump file is created, the previous file is preserved. Each additional file is given a distinct name. The date is encoded in the file name. For example, Mini022900-01.dmp is the first memory dump generated on February 29, 2000. A list of all small memory dump files is kept in the %SystemRoot%\Minidump folder.

Active Directory

Active Directory (AD) is a directory service that Microsoft developed for Windows domain networks and is included in most Windows Server operating systems as a set of processes and services.

An AD domain controller authenticates and authorizes all users and computers in a Windows domain type network—assigning and enforcing security policies for all computers and installing or updating software. For example, when a user logs into a computer that is part of a Windows domain, Active Directory checks the submitted password and determines whether the user is a system administrator or normal user.

Active Directory makes use of Lightweight Directory Access Protocol (LDAP) versions 2 and 3, Microsoft's version of Kerberos, and DNS.

**Logical Structure**

As a directory service, an Active Directory instance consists of a database and corresponding executable code responsible for servicing requests and maintaining the database. The executable part, known as Directory System Agent, is a collection of Windows services and processes that run on Windows 2000 and later. Objects in Active Directory databases can be accessed via LDAP, ADSI (a component object model interface), messaging API and Security Accounts Manager services.

**Objects**

An Active Directory Structure is an arrangement of information about objects. The objects fall into two broad categories: resources (e.g., printers) and security principals (user or computer accounts and groups). Security principals are assigned unique security identifiers (SIDs).

Each object represents a single entity—whether a user, a computer, a printer, or a group—and its attributes. Certain objects can contain other objects. An object is uniquely identified by its name and has a set of attributes—the characteristics and information that the object represents— defined by a schema, which also determines the kinds of objects that can be stored in Active Directory.

The schema object lets administrators extend or modify the schema when necessary. However, because each schema object is integral to the definition of Active Directory objects, deactivating or changing these objects can fundamentally change or disrupt a deployment. Schema changes automatically propagate throughout the system. Once created, an object can only be deactivated—not deleted. Changing the schema usually requires planning.

**Forests, Trees, and Domains**

The Active Directory framework that holds the objects can be viewed at a number of levels. The forest, tree, and domain are the logical divisions in an Active Directory network.

Within a deployment, objects are grouped into domains. The objects for a single domain are stored in a single database (which can be replicated). Domains are identified by their DNS name structure, the namespace.

A domain is defined as a logical group of network objects (computers, users, devices) that share the same active directory database.

A tree is a collection of one or more domains and domain trees in a contiguous namespace, linked in a transitive trust hierarchy.

At the top of the structure is the forest. A forest is a collection of trees that share a common global catalog, directory schema, logical structure, and directory configuration. The forest represents the security boundary within which users, computers, groups, and other objects are accessible.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  |  | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-Boston | |  |  | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-New York | |  |  | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-Philly | |  | [Icons-mini-page tree.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_tree.gif) | Tree-Southern | | |  |  | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-Atlanta | |  |  | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-Dallas | | |  |  |  |  | | --- | --- | --- | --- | | [Icons-mini-page url.gif](http://en.wikipedia.org/wiki/File:Icons-mini-page_url.gif) | Domain-Dallas | | | |  | [Icons-mini-folder.gif](http://en.wikipedia.org/wiki/File:Icons-mini-folder.gif) | OU-Marketing | | |  |  | [Icons-mini-icon user.gif](http://en.wikipedia.org/wiki/File:Icons-mini-icon_user.gif) | Hewitt | |  |  | [Icons-mini-icon user.gif](http://en.wikipedia.org/wiki/File:Icons-mini-icon_user.gif) | Aon | |  |  | [Icons-mini-icon user.gif](http://en.wikipedia.org/wiki/File:Icons-mini-icon_user.gif) | Steve | |  | [Icons-mini-folder.gif](http://en.wikipedia.org/wiki/File:Icons-mini-folder.gif) | OU-Sales | | |  |  | [Icons-mini-icon user.gif](http://en.wikipedia.org/wiki/File:Icons-mini-icon_user.gif) | Bill | |  |  | [Icons-mini-icon user.gif](http://en.wikipedia.org/wiki/File:Icons-mini-icon_user.gif) | Ralph | |
| Example of the geographical organizing of zones of interest within trees and domains. | |

**Organizational Units**

The objects held within a domain can be grouped into Organizational Units (OUs). OUs can provide hierarchy to a domain, ease its administration, and can resemble the organization's structure in managerial or geographical terms. OUs can contain other OUs—domains are containers in this sense. Microsoft recommends using OUs rather than domains for structure and to simplify the implementation of policies and administration. The OU is the recommended level at which to apply group policies, which are Active Directory objects formally named Group Policy Objects (GPOs), although policies can also be applied to domains or sites (see below). The OU is the level at which administrative powers are commonly delegated, but delegation can be performed on individual objects or attributes as well.

Organizational units are not mutually exclusive database; e.g. it is not possible to create user accounts with an identical username (SAMAccountName) in separate OUs, such as "fred.staff-ou.domain" and "fred.student-ou.domain", where "staff-ou" and "student-ou" are the OUs. This is so because SAMAccountName, a user object attribute, must be unique within the domain.  However, two users in different OUs can have the same Common Name (CN), the name under which they are stored in the directory itself.

In general the reason for this lack of allowance for duplicate names through hierarchical directory placement, is that Microsoft primarily relies on the principles of NetBIOS, which is a flat-file method of network object management that for Microsoft software, goes all the way back to Windows NT 3.1 and MS-DOS LAN Manager. Allowing for duplication of object names in the directory, or completely removing the use of NetBIOS names, would prevent backward compatibility with legacy software and equipment.

As the number of users in a domain increases, conventions such as "first initial, middle initial, last name" (Western order) or the reverse (Eastern order) fail for common family names like Li (李), Smith or Garcia. Workarounds include adding a digit to the end of the username. Alternatives include creating a separate ID system of unique employee/student id numbers to use as account names in place of actual user's names, and allowing users to nominate their preferred word sequence within an acceptable use policy.

Because duplicate usernames cannot exist within a domain, account name generation poses a significant challenge for large organizations that cannot be easily subdivided into separate domains, such as students in a public school system or university who must be able to use any computer across the network.

**Shadow Groups**

In Active Directory, organizational units cannot be assigned as owners or trustees. Only groups are selectable, and members of OUs cannot be collectively assigned rights to directory objects.

In Microsoft's Active Directory, OUs do not confer access permissions, and objects placed within OUs are not automatically assigned access privileges based on their containing OU. This is a design limitation specific to Active Directory. Other competing directories such as Novell NDS are able to assign access privileges through object placement within an OU.

Active Directory requires a separate step for an administrator to assign an object in an OU as a member of a group also within that OU. Relying on OU location alone to determine access permissions is unreliable, because the object may not have been assigned to the group object for that OU.

A common workaround for an Active Directory administrator is to write a custom PowerShell or Visual Basic script to automatically create and maintain a user group for each OU in their directory. The scripts are run periodically to update the group to match the OU's account membership, but are unable to instantly update the security groups anytime the directory changes, as occurs in competing directories where security is directly implemented into the directory itself. Such groups are known as Shadow Groups. Once created, these shadow groups are selectable in place of the OU in the administrative tools.

Microsoft refers to shadow groups in the Server 2008 Reference documentation, but does not explain how to create them. There are no built-in server methods or console snap-ins for managing shadow groups.

The division of an organization's information infrastructure into a hierarchy of one or more domains and top-level OUs is a key decision. Common models are by business unit, by geographical location, by IT Service, or by object type and hybrids of these. OUs should be structured primarily to facilitate administrative delegation, and secondarily, to facilitate group policy application. Although OUs form an administrative boundary, the only true security boundary is the forest itself and an administrator of any domain in the forest must be trusted across all domains in the forest.

**Partitions**

The Active Directory database is organized in partitions, each holding specific object types and following a specific replication pattern. Microsoft often refers to these partitions as 'naming contexts'. The 'Schema' partition contains the definition of object classes and attributes within the Forest. The 'Configuration' partition contains information on the physical structure and configuration of the forest (such as the site topology). Both replicate to all domains in the Forest. The 'Domain' partition holds all objects created in that domain and replicates only within its domain.

**Physical structure**

Sites are physical (rather than logical) groupings defined by one or more IP subnets.  AD also holds the definitions of connections, distinguishing low-speed (e.g., WAN, VPN) from high-speed (e.g., LAN) links. Site definitions are independent of the domain and OU structure and are common across the forest. Sites are used to control network traffic generated by replication and also to refer clients to the nearest domain controllers (DCs). Microsoft Exchange Server 2007 uses the site topology for mail routing. Policies can also be defined at the site level.

Physically, the Active Directory information is held on one or more peer domain controllers, replacing the NT PDC/BDC model. Each DC has a copy of the Active Directory. Servers joined to Active Directory that are not domain controllers are called Member Servers.  A subset of objects in the domain partition replicate to domain controllers that are configured as global catalogs. Global catalog (GC) servers provide a global listing of all objects in the Forest.  Global Catalog servers replicate to themselves all objects from all domains and hence, provide a global listing of objects in the forest. However, to minimize replication traffic and keep the GC's database small, only selected attributes of each object are replicated. This is called the partial attribute set (PAS). The PAS can be modified by modifying the schema and marking attributes for replication to the GC.  Earlier versions of Windows used NetBIOS to communicate. Active Directory is fully integrated with DNS and requires TCP/IP—DNS. To be fully functional, the DNS server must support SRV resource records, also known as service records.

**Replication**

Active Directory synchronizes changes using multi-master replication. Replication by default is 'pull' rather than 'push', meaning that replicas pull changes from the server where the change was effected.  The Knowledge Consistency Checker (KCC) creates a replication topology of site links using the defined sites to manage traffic. Intrasite replication is frequent and automatic as a result of change notification, which triggers peers to begin a pull replication cycle. Intersite replication intervals are typically less frequent and do not use change notification by default, although this is configurable and can be made identical to intrasite replication.

Each link can have a 'cost' (e.g., DS3, T1, ISDN etc.) and the KCC alters the site link topology accordingly. Replication may occur transitively through several site links on same-protocol site link bridges, if the cost is low, although KCC automatically costs a direct site-to-site link lower than transitive connections. Site-to-site replication can be configured to occur between bridgehead servers in each site, which then replicates the changes to other DCs within the site. Replication for Active Directory zones is automatically configured when DNS is activated in the domain based by site.

Replication of Active Directory uses Remote Procedure Calls (RPC) over IP (RPC/IP). Between Sites SMTP can be used for replication, but only for changes in the Schema, Configuration, or Partial Attribute Set (Global Catalog) GCs. SMTP cannot be used for replicating the default Domain partition.

Unix/Linux

Boot Process

The Linux startup process is the multi-stage initialization process performed during booting a Linux installation. It is in many ways similar to the BSD and other Unix-style boot processes, from which it derives.

Booting a Linux installation involves multiple stages and software components, including firmware initialization, execution of a boot loader, loading and startup of a Linux kernel image, and execution of various startup scripts and daemons. For each of these stages and components there are different variations and approaches; for example, GRUB, LILO, SYSLINUX or Loadlin can be used as boot loaders, while the startup scripts can be either traditional init-style, or the system configuration can be performed through more modern alternatives such as systemd or Upstart.

**Boot Loader Phase**

The boot loader phase varies by computer architecture. Since the earlier phases are not specific to the operating system, the BIOS-based boot process for x86 and x86-64 architectures is considered to start when the master boot record (MBR) code is executed in real mode and the first-stage boot loader is loaded, which is typically a part of LILO or GRUB.  From that point, the boot process continues as described below.

The first-stage boot loader (stored within the MBR or the volume boot record) loads the remainder of the boot loader, which typically gives a prompt asking which operating system (or type of session) the user wishes to initialize. LILO and GRUB differ in some ways:

* Under LILO, the map installer, lilo, reads the configuration file /etc/lilo.conf to identify the available bootable systems (it is executed from a running Linux system). The configuration file can include data such as boot partition and kernel pathname for each, as well as customized options if needed. To be precise, /etc/lilo.conf is prior-parsed and used to create fixed-offset information saved in the boot sector and the map file, which will be used at the next boot. This information is discovered by asking the Linux kernel, at map installer time, where (i.e. on which disk sectors) the object of interest (such as an initrd, a kernel image file, or the like) is stored. At boot time, the default or selected operating system is loaded into RAM, a minimal initial file system is possibly set up in RAM from an image file ("initrd"), and along with the appropriate boot parameters, and control is passed to the newly loaded kernel. LILO does not "understand" file systems, so it uses raw disk offsets and the BIOS to load any needed code or data based on data in the boot sector and map file. It cannot "find" /etc/lilo.conf at boot time because it does not understand file systems; instead it searches fixed locations on the disk memorized the last time the LILO map installer (lilo) was run to generate new offsets in the boot sector and the map file image. Boot time LILO logic loads the menu code, and then, depending on the lilo.conf directives used to make the map file, along with any user interaction, loads either the boot sector for another system such as Microsoft Windows, or the kernel image for Linux.
* GRUB by contrast does have understanding of the common ext2, ext3 and ext4 file systems. Because GRUB stores its data in a configuration file rather than the MBR and contains a command-line interface, it is often easier to rectify or modify GRUB if misconfigured or corrupt.

**LILO**

LILO, the older of the two boot loaders, is almost identical to GRUB in process, except that its command line interface allows only selection of options previously recorded in the boot sector and map file. Thus all changes must be made to its configuration and written to the boot sector and map file, and then the system restarted. An error in configuration can therefore leave a disk unable to be booted without use of a separate boot device (floppy disk etc.) containing a program capable of fixing this. Additionally, it does not understand filesystems. Instead, locations of image files are stored within the boot sector and map file directly and the BIOS is used to access them directly.

**GRUB**

* The first-stage loader (stage1) is loaded and executed either by the BIOS from the Master boot record (MBR) or by another boot loader from the partition boot sector.
* If necessary, an intermediate stage loader (stage1.5) is loaded and executed by the first-stage loader. This may be necessary if the second-stage loader is not contiguous, or if the filesystem or hardware requires special handling in order to access the second-stage loader.
* The second-stage loader (stage2) is then loaded and executed. This displays the GRUB startup menu, which allows the user to choose an operating system or to examine and edit startup parameters.
* After an operating system is chosen, GRUB loads its kernel into memory and passes control to the kernel. Alternatively, GRUB can pass control of the boot process to another boot loader, using chain loading.

**SYSLINUX**

SYSLINUX is not normally used for booting full Linux installations since Linux is not normally installed on FAT filesystems. Instead, it is often used for boot or rescue floppy discs, live USBs, or other lightweight boot systems. ISOLINUX is generally used by Linux live CDs and bootable install CDs.

**Loadlin**

Yet another way to boot Linux is from DOS or Windows 9x, where the Linux kernel completely replaces the running copy of this operating system. This can be useful in the case of hardware which needs to be switched on via software and for which such configuration programs are only available for DOS, whereas not for Linux, those being proprietary to the manufacturer and kept an industry secret. This tedious booting method is less necessary nowadays, as Linux has drivers for a multitude of hardware devices, but it has seen some use in mobile devices.

Another case is when the Linux is located on a storage device which is not available to the BIOS for booting: DOS or Windows can load the appropriate drivers to make up for the BIOS limitation, and boot Linux from there.

**Kernel Phase**

The kernel in Linux handles all operating system processes, such as memory management, task scheduling, I/O, interprocess communication, and overall system control. This is loaded in two stages - in the first stage the kernel (as a compressed image file) is loaded into memory and decompressed, and a few fundamental functions such as basic memory management are set up. Control is then switched one final time to the main kernel start process. Once the kernel is fully operational – and as part of its startup, upon being loaded and executing – the kernel looks for an init process to run, which (separately) sets up a user space and the processes needed for a user environment and ultimate login. The kernel itself is then allowed to go idle, subject to calls from other processes.

**Kernel Loading Stage**

The kernel as loaded is typically an image file, compressed into either zImage or bzImage formats with zlib. A routine at the head of it does a minimal amount of hardware setup, decompresses the image fully into high memory, and takes note of any RAM disk if configured.  It then executes kernel startup via ./arch/i386/boot/head and the startup\_32 () (for x86 based processors) process.

**Kernel Startup Stage**

The startup function for the kernel (also called the swapper or process 0) establishes memory management (paging tables and memory paging), detects the type of CPU and any additional functionality such as floating point capabilities, and then switches to non-architecture specific Linux kernel functionality via a call to start\_kernel().

start\_kernel() executes a wide range of initialization functions. It sets up interrupt handling (IRQs), further configures memory, starts the Init process (the first user-space process), and then starts the idle task via cpu\_idle(). Notably, the kernel startup process also mounts the initial RAM disk ("initrd") that was loaded previously as the temporary root file system during the boot phase. The initrd allows driver modules to be loaded directly from memory, without reliance upon other devices (e.g. a hard disk) and the drivers that are needed to access them (e.g. a SATA driver). This split of some drivers statically compiled into the kernel and other drivers loaded from initrd allows for a smaller kernel. The root file system is later switched via a call to pivot\_root()which unmounts the temporary root file system and replaces it with the use of the real one, once the latter is accessible. The memory used by the temporary root file system is then reclaimed.

Thus, the kernel initializes devices, mounts the root filesystem specified by the boot loader as read only, and runs Init (/sbin/init) which is designated as the first process run by the system (PID = 1). A message is printed by the kernel upon mounting the file system, and by Init upon starting the Init process. It may also optionally run Initrdto allow setup and device related matters (RAM disk or similar) to be handled before the root file system is mounted.

According to Red Hat, the detailed kernel process at this stage is therefore summarized as follows:

"When the kernel is loaded it immediately initializes and configures the computer's memory and the various hardware attached to the system. This includes all processors, I/O subsystems, and storage devices. It then looks for the compressed initrd image in a predetermined location in memory, decompresses it, mounts it, and loads all necessary drivers. Next, it initializes virtual devices related to the file system, such as LVM or software RAID before unmounting the initrd disk image and freeing up all the memory the disk image once occupied. The kernel then creates a root device, mounts the root partition read-only, and frees any unused memory. At this point, the kernel is loaded into memory and operational. However, since there are no user applications that allow meaningful input to the system, not much can be done with it."

At this point, with interrupts enabled, the scheduler can take control of the overall management of the system, to provide pre-emptive multi-tasking, and the init process is left to continue booting the user environment in user space.

**Early user space**

Initramfs, also known as early user space, has been available since version 2.5.46 of the Linux kernel, with the intent to replace as many functions as possible that previously the kernel would have performed during the start-up process. Typical uses of early user space are to detect what device drivers are needed to load the main user space file system and load them from a temporary filesystem.

**Init process**

**SysV init**

An excerpt from the man page for init states:

“Init is the parent of all processes on the system, it is executed by the kernel and is responsible for starting all other processes; it is the parent of all processes whose natural parents have died and it is responsible for reaping those when they die. Processes managed by init are known as jobs and are defined by files in the/etc/init directory.”

Init's job is "to get everything running the way it should be" once the kernel is fully running. Essentially it establishes and operates the entire user space. This includes checking and mounting file systems, starting up necessary user services, and ultimately switching to a user-environment when system startup is completed. It is similar to the Unix and BSD init processes, from which it derived, but in some cases has diverged or became customized. In a standard Linux system, Init is executed with a parameter, known as a run level, that takes a value from 0 to 6, and that determines which subsystems are to be made operational. Each run level has its own scripts which codify the various processes involved in setting up or leaving the given run level, and it is these scripts which are referenced as necessary in the boot process. Init scripts are typically held in directories with names such as "/etc/rc". The top level configuration file for init is at/etc/inittab.

During system boot, it checks whether a default run level is specified in /etc/inittab, and requests the run level to enter via the system console if not. It then proceeds to run all the relevant boot scripts for the given run level, including loading modules, checking the integrity of the root file system (which was mounted read-only) and then remounting it for full read-write access, and sets up the network.

After it has spawned all of the processes specified, init goes dormant, and waits for one of three events to happen: processes that started to end or die, a power failure signal, or a request via /sbin/telinit to further change the run level.

This applies to SysV-style init.

**systemd**

The developers of systemd aimed to replace the Linux init system inherited from UNIX System V and Berkeley Software Distribution (BSD) operating systems. Like init, systemd is a daemon that manages other daemons. All daemons, including systemd, are background processes. Systemd is the first daemon to start (during booting) and the last daemon to terminate (during shutdown).

Lennart Poettering and Kay Sievers, software engineers that initially developed systemd, sought to surpass the efficiency of the init daemon in several ways. They wanted to improve the software framework for expressing dependencies, to allow more processing to be done concurrently or in parallel during system booting, and to reduce the computational overhead of the shell.

Systemd's initialization instructions for each daemon are recorded in a declarative configuration file rather than a shell script. For inter-process communication, systemd makes Unix domain sockets and D-Bus available to the running daemons. Systemd is also capable of aggressive parallelization.

**Upstart**

The traditional init process was originally only responsible for bringing the computer into a normal running state after power-on, or gracefully shutting down services prior to shutdown. As a result, the design is strictly synchronous, blocking future tasks until the current one has completed. Its tasks must also be defined in advance, as they are limited to this prep or cleanup function. This leaves it unable to handle various non-startup-tasks on a modern desktop computer.

Upstart operates asynchronously; it handles starting of the tasks and services during boot and stopping them during shutdown, and also supervises the tasks and services while the system is running.

Easy transition and perfect backward compatibility with sysvinit were the explicit design goals; accordingly, Upstart can run unmodified sysvinit scripts. In this way it differs from most other init replacements (beside systemd and OpenRC), which usually assume and require complete transition to run properly, and do not support a mixed environment of traditional and new startup methods.

Upstart allows for extensions to its event model through the use of initctl to input custom, single events, or event bridges to integrate many or more-complicated events.[[12]](http://en.wikipedia.org/wiki/Linux_startup_process#cite_note-12) By default, Upstart includes bridges for socket, dbus, udev, file, and dconf events; additionally, more bridges (for example, a Mach ports bridge, or a devd (found on FreeBSD systems) bridge) are possible.

**runit**

Runit is an init scheme for Unix-like operating systems that initializes, supervises, and ends processes throughout the operating system. It is a "reimplementation" of the "seminal" Daemontoolsprocess supervision toolkit that runs on the Linux, Mac OS X, \*BSD, and Solaris operating systems. Runit features parallelization of the startup of system services, which can speed up the boot time of the operating system.

Runit is an init daemon, so it is the direct or indirect ancestor of all other processes. It is the first process started during booting, and continues running until the system is shut down.

**Linux Run Levels**

The term “run level” refers to a mode of operation in one of the computer operating systems that implement Unix System V-style initialization. Conventionally, seven run levels exist, numbered from zero to six; though up to ten, from zero to nine,, may be used. **S** is sometimes used as a synonym for one of the levels. Only one "run level" is executed on bootup - run levels are **not** executed sequentially, i.e. either run level 2 OR 3 OR 4 is executed, **not** 2 then 3 then 4.

"Run level" defines the state of the machine after boot. Different run levels are typically assigned to:

* single-user mode
* multi-user mode without network services started
* multi-user mode with network services started
* system shutdown
* system reboot

The exact setup of these configurations will vary from OS to OS and from one Linux distribution to another. For example, run level 4 might be multi-user, GUI, no-server on one distribution, and nothing on another. Note the difference in the Red Hat and Slackware distributions detailed below. However, "run levels" do commonly follow patterns described in this article. It is best to consult the particular distribution's available user guides.

In standard practice, when a computer enters run level zero, it halts, and when it enters run level six, it reboots. The intermediate run levels (1-5) differ in terms of which drives are mounted, and which network services are started. Default run levels are typically 3, 4, or 5. Lower run levels are useful for maintenance or emergency repairs, since they usually don't offer any network services at all. The particular details of run level configuration differ widely among operating systems, and also among system administrators.

The run level system replaced the traditional /etc/rc script used in Version 7 Unix.

**Standard run levels**

|  |  |  |
| --- | --- | --- |
| **Standard run levels** | | |
| **ID** | **Name** | **Description** |
| **0** | JNB | Shuts down the system. |
| **S** | Single user mode | Does not configure network interfaces or start daemons. |
| **6** | Reboot | Reboots the system. |

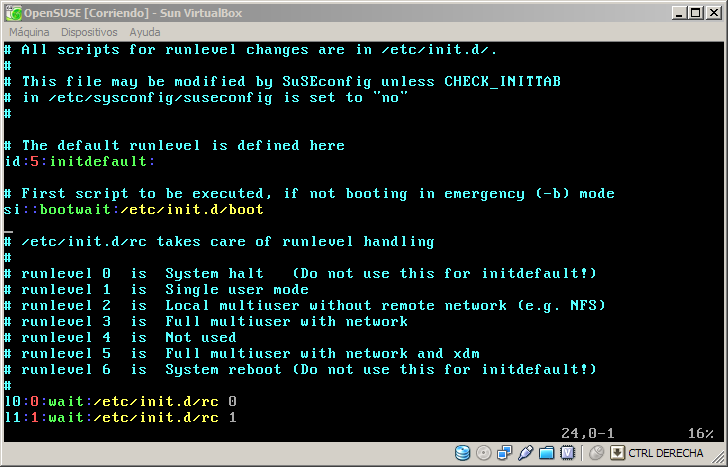
S = Almost all systems use run level 1 for this purpose. This mode is intended to provide a safe environment to perform system maintenance. Originally this run level provided a single terminal (console) interface running a root login shell. The increasing trend towards physical access to the computer during the boot process has led to changes in this area.

**Linux**

The Linux operating system can make use of run levels through the programs of the sysvinit project. After the Linux kernel has booted, the init program reads the /etc/inittab file to determine the behavior for each run level. Unless the user specifies another value as a kernel boot parameter, the system will attempt to enter (start) the default run level.

**Linux Standard Base Specification**

Systems conforming to the Linux Standard Base (LSB) need not provide the exact run levels given here or give them the meanings described here, and may map any level described here to a different level which provides the equivalent functionality.



Run levels in a standard Linux distro.

|  |  |  |
| --- | --- | --- |
| **LSB 4.1.0** | | |
| **ID** | **Name** | **Description** |
| **0** | Halt | Shuts down the system. |
| **1** | Single-user Mode | Mode for administrative tasks. |
| **2** | Multi-user Mode | Does not configure network interfaces and does not export networks services. |
| **3** | Multi-user Mode with Networking | Starts the system normally. |
| **4** | Not used/User-definable | For special purposes. |
| **5** | Start the system normally with appropriate display manager. ( with GUI ) | Same as run level 3 + display manager. |
| **6** | Reboot | Reboots the system. |

**Note 1 -** The additional behavior of run level 1 varies greatly. All distributions provide at least one virtual terminal. Some distributions start a login shell as the superuser; some require correctly entering the superuser's password; others provide a login prompt, allowing access to any registered user.

**Note 2 -** In some cases, run levels 2 and 3 function identically; offering a Multi-user Mode with Networking.

**Debian GNU/Linux**

Debian, as well as most of the distributions based on it, does not make any distinction between run levels 2 to 5, leaving that choice to the system administrator.

|  |  |
| --- | --- |
| **Debian GNU/Linux Run Levels** | |
| **ID** | **Description** |
| **S** | Only run on boot (replaces /etc/rc.boot) |
| **0** | Halt |
| **1** | Single-user mode |
| **2**-**5** | Full Multi-user with console logins and display manager if installed |
| **6** | Reboot |

**Ubuntu**

Ubuntu 6.10 (Edgy Eft) and later contain Upstart as a replacement for the traditional init-process, but they still use the traditional init scripts and Upstart's SysV-rc compatibility tools to start most services and emulate run levels.

|  |  |
| --- | --- |
| **Ubuntu Run Levels** | |
| **Code** | **Information** |
| **0** | Halt |
| **1** | Single-user mode |
| **2** | Graphical multi-user with networking |
| **3-5** | Unused but configured the same as run level 2 |
| **6** | Reboot |

**Red Hat Linux and Fedora**

Red Hat and most of its derivatives (such as CentOS) use run levels like this:

|  |  |
| --- | --- |
| **Red Hat Linux/Fedora run levels** | |
| **Code** | **Information** |
| **0** | Halt |
| **1** | Single-user text mode (without networking) |
| **2** | Not used (user-definable) |
| **3** | Full multi-user text mode |
| **4** | Not used (user-definable) |
| **5** | Full multi-user graphical mode (with an X-based login screen) |
| **6** | Reboot |

The chkconfig utility updates and queries run level information for system services.

**SUSE Linux**

SUSE uses a similar setup to Red Hat:

|  |  |
| --- | --- |
| **SUSE Linux run levels** | |
| **ID** | **Description** |
| **0** | Halt |
| **1** or **S** | Single-user mode |
| **2** | Multi-user mode without networking |
| **3** | Multi-user mode, console logins only |
| **4** | Not used/User-definable |
| **5** | Multi-user mode with display manager |
| **6** | Reboot |

System administrators can modify the services that run under a specific run level by using YaST | System Services (run level), insserv or chkconfig  - as in the distributions based on Red Hat.

**Slackware Linux**

Slackware Linux uses run level 1 for maintenance, as on other Linux distributions; run levels 2, 3 and 5 identically configured for a console (with all services active); and run level 4 adds the X Window System.

|  |  |
| --- | --- |
| **Slackware Linux Run Levels** | |
| **ID** | **Description** |
| **0** | Halt |
| **1** | Single-user mode |
| **2** | Unused but configured the same as run level 3 |
| **3** | Multi-user mode without display manager |
| **4** | Multi-user mode with display manager |
| **5** | Unused but configured the same as run level 3 |
| **6** | Reboot |

**Gentoo Linux**

|  |  |
| --- | --- |
| **Gentoo Linux Run Levels** | |
| **ID** | **Description** |
| **0** | Halt |
| **1** or **S** | Single-user mode |
| **2** | Multi-user mode without networking |
| **3** | Multi-user mode |
| **4** | Aliased for run level **3** |
| **5** | Aliased for run level **3** |
| **6** | Reboot |

**Unix**

**System V Releases 3 and 4**

|  |  |
| --- | --- |
| **System V run levels** | |
| **ID** | **Description** |
| **0** | Shut down system, power-off if hardware supports it (only available from the console) |
| **1** | Single-user mode, all filesystems unmounted but not root, all processes except console processes killed |
| **2** | Multi-user mode |
| **3** | Multi-user mode with RFS (and NFS in Release 4) filesystems exported |
| **4** | Multi-user, User-definable |
| **5** | Halt the operating system, go to firmware |
| **6** | Reboot |
| **s**,**S** | Identical to **1**, except current terminal acts as the system console |

**Solaris**

Starting from Solaris 10, SMF (Service Management Facility) is used instead of SVR4 run levels. The latter are emulated to preserve compatibility with legacy startup scripts.

|  |  |
| --- | --- |
| **Solaris run levels** | |
| **ID** | **Description** |
| **0** | Operating system halted; (SPARC only) drop to OpenBoot prompt |
| **S** | Single-user mode with only root filesystem mounted (as read-only) -- Solaris 10+: svc:/milestone/single-user |
| **1** | Single-user mode with all local filesystems mounted (read-write) |
| **2** | Multi-user mode with most daemons started – Solaris 10+: svc:/milestone/multi-user |
| **3** | Multi-user mode; identical to **2** (run level **3** runs both /sbin/rc2 and /sbin/rc3), with filesystems exported, plus some other network services started. -- Solaris 10+: svc:/milestone/multi-user-server |
| **4** | Alternative Multi-user mode, User-definable |
| **5** | Shut down, power-off if hardware supports it |
| **6** | Reboot |

**HP-UX**

|  |  |
| --- | --- |
|  | |
| **ID** | **Description** |
| **0** | System halted |
| **S** | Single-user mode, booted to system console only, with only root filesystem mounted (as read-only) |
| **s** | Single-user mode, identical to **S** except the current terminal acts as the system console |
| **1** | Single-user mode with local filesystems mounted (read-write) |
| **2** | Multi-user mode with most daemons started and Common Desktop Environment launched |
| **3** | Identical to run level **2** with NFS exported |
| **4** | Multi-user mode with VUE started instead of CDE |
| **5**,**6** | Not used/User-definable |

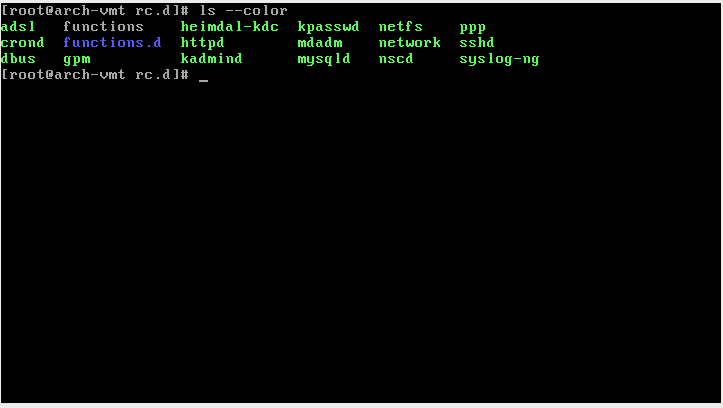
**AIX**

AIX does not follow the System V R4 (SVR4) run level specification, with run levels from 0 to 9 available, as well as from a to c. 0 and 1 are reserved, 2 is the default normal multi-user mode and run levels from 3 to 9 are free to be defined by the administrator. Run levels from a to c allow the execution of processes in that run level without killing processes started in another.

|  |  |  |
| --- | --- | --- |
| **AIX Run Levels** | | |
| **ID** | **Name** | **Description** |
| 0 |  | reserved |
| 1 |  | reserved |
| 2 | Normal Multi-user mode | default mode |

**Using Run Levels To Start Services**

Let's start by looking at how the system is set up, and in particular at the directory /etc/rc.d. Here you will find either a set of files named rc.0, rc.1, rc.2, rc.3, rc.4, rc.5, and rc.6, or a set of directories named rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d, and rc6.d. You will also find a file named /etc/inittab. The system uses these files (and/or directories) to control the services to be started.



Services started on startup in a barebones configuration.

If you look in the file /etc/inittab you will see something like:

id:4:initdefault:l

0:0:wait:/etc/rc.d/rc.0l

6:6:wait:/etc/rc.d/rc.6x

1:4:wait:/etc/rc.d/rc.4

The boot process uses these parameters to identify the default run level and the files that will be used by that run level. In this example, run level 4 is the default and the scripts that define run level 4 can be found in /etc/rc.d/rc.4.

If you wanted something to be run on startup you would just put it in the respective rc.d/rc.# directory and like magic it is automatically run by the operating system.

Filesystems and Structure

**Unix and Unix-like Operating Systems**

Unix-like operating systems create a virtual file system, which makes all the files on all the devices appear to exist in a single hierarchy. This means, in those systems, there is one root directory, and every file existing on the system is located under it somewhere. Unix-like systems can use a RAM disk or network shared resource as its root directory.

Unix-like systems assign a device name to each device, but this is not how the files on that device are accessed. Instead, to gain access to files on another device, the operating system must first be informed where in the directory tree those files should appear. This process is called mounting a file system. For example, to access the files on a CD-ROM, one must tell the operating system "Take the file system from this CD-ROM and make it appear under such-and-such directory". The directory given to the operating system is called the mount point – it might, for example, be/media. The /media directory exists on many Unix systems (as specified in the Filesystem Hierarchy Standard) and is intended specifically for use as a mount point for removable media such as CDs, DVDs, USB drives or floppy disks. It may be empty, or it may contain subdirectories for mounting individual devices. Generally, only the administrator (i.e. root user) may authorize the mounting of file systems.

Unix-like operating systems often include software and tools that assist in the mounting process and provide it new functionality. Some of these strategies have been coined "auto-mounting" as a reflection of their purpose.

* In many situations, file systems other than the root need to be available as soon as the operating system has booted. All Unix-like systems therefore provide a facility for mounting file systems at boot time. [System administrators](http://en.wikipedia.org/wiki/System_administrator) define these file systems in the configuration file fstab (vfstab in Solaris), which also indicates options and mount points.
* In some situations, there is no need to mount certain file systems at boot time, although their use may be desired thereafter. There are some utilities for Unix-like systems that allow the mounting of predefined file systems upon demand.
* Removable media have become very common with microcomputer platforms. They allow programs and data to be transferred between machines without a physical connection. Common examples include USB flash drives, CD-ROMs, and DVDs. Utilities have therefore been developed to detect the presence and availability of a medium and then mount that medium without any user intervention.
* Progressive Unix-like systems have also introduced a concept called **supermounting**; see, for example, the Linux supermount-ng project. For example, a floppy disk that has been supermounted can be physically removed from the system. Under normal circumstances, the disk should have been synchronized and then unmounted before its removal. Provided synchronization has occurred, a different disk can be inserted into the drive. The system automatically notices that the disk has changed and updates the mount point contents to reflect the new medium.
* An automounter will automatically mount a file system when a reference is made to the directory atop which it should be mounted. This is usually used for file systems on network servers, rather than relying on events such as the insertion of media, as would be appropriate for removable media.

**Linux**

Linux supports many different file systems, but common choices for the system disk on a block device include the ext\* family (such as ext2, ext3 and ext4), XFS, JFS, ReiserFS and btrfs. For raw flash without a flash translation layer (FTL) or Memory Technology Device (MTD), there is UBIFS, JFFS2, and YAFFS, among others. SquashFS is a common compressed read-only file system.

**Ext2**

The space in ext2 is split up into blocks. These blocks are grouped into block groups, analogous to cylinder groups in the Unix File System. There are typically thousands of blocks on a large file system. Data for any given file is typically contained within a single block group where possible. This is done to minimize the number of disk seeks when reading large amounts of contiguous data.

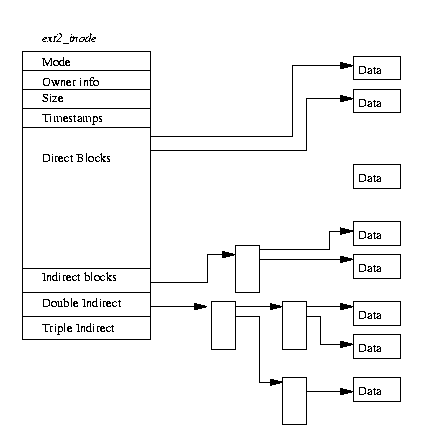
Each block group contains a copy of the superblock and block group descriptor table, and all block groups contain a block bitmap, an inode bitmap, an inode table and finally the actual data blocks.

The superblock contains important information that is crucial to the booting of the operating system. Thus backup copies are made in multiple block groups in the file system. However, typically only the first copy of it, which is found at the first block of the file system, is used in the booting.

The group descriptor stores the location of the block bitmap, inode bitmap and the start of the inode table for every block group. These, in turn, are stored in a group descriptor table.

**Inodes**

Every file or directory is represented by an inode. The term "inode" comes from "index node" (over the time, it became i-node and then inode). The inode includes data about the size, permission, ownership, and location on disk of the file or directory.



Example of ext2 inode structure

Quote from the Linux kernel documentation for ext2:

**"There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect block."**

So, there is a structure in ext2 that has 15 pointers. Pointers 1 to 12 point to direct blocks, pointer 13 points to an indirect block, pointer 14 points to a doubly indirect block, and pointer 15 points to a trebly indirect block.

**Directories**

Each directory is a list of directory entries. Each directory entry associates one file name with one inode number, and consists of the inode number, the length of the file name, and the actual text of the file name. To find a file, the directory is searched front-to-back for the associated filename. For reasonable directory sizes, this is fine. But for very large directories this is inefficient, and ext3 offers a second way of storing directories (HTree) that is more efficient than just a list of filenames.

The root directory is always stored in inode number two, so that the file system code can find it at mount time. Subdirectories are implemented by storing the name of the subdirectory in the name field, and the inode number of the subdirectory in the inode field. Hard links are implemented by storing the same inode number with more than one file name. Accessing the file by either name results in the same inode number, and therefore the same data.

The special directories "." (current directory) and ".." (parent directory) are implemented by storing the names "." and ".." in the directory, and the inode number of the current and parent directories in the inode field. The only special treatment these two entries receive is that they are automatically created when any new directory is made, and they cannot be deleted.

**Allocating Data**

When a new file or directory is created, ext2 must decide where to store the data. If the disk is mostly empty, then data can be stored almost anywhere. However, clustering the data with related data will minimize seek times and maximize performance.

ext2 attempts to allocate each new directory in the group containing its parent directory, on the theory that accesses to parent and children directories are likely to be closely related. ext2 also attempts to place files in the same group as their directory entries, because directory accesses often lead to file accesses. However, if the group is full, then the new file or new directory is placed in some other non-full group.

The data blocks needed to store directories and files can be found by looking in the data allocation bitmap. Any needed space in the inode table can be found by looking in the inode allocation bitmap.

**File System Limits**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Theoretical ext2 limits under Linux** | | | | |
| **Block size:** | 1 KiB | 2 KiB | 4 KiB | 8 KiB |
| **max. file size:** | 16 GiB | 256 GiB | 2 TiB | 2 TiB |
| **max. filesystem size:** | 4 TiB | 8 TiB | 16 TiB | 32 TiB |

The reasons for some limits of ext2 are the file format of the data and the operating system's kernel. Mostly these factors will be determined once when the file system is built. They depend on the block size and the ratio of the number of blocks and inodes. In Linux the block size is limited by the architecture page size.

There are also some userspace programs that can't handle files larger than 2 GiB.

If *b* is the block size, the maximum file size is limited to *min*( ((*b*/4)3+(*b*/4)2+*b*/4+12)\**b*, (232-1)\*512 ) due to the i\_block structure (an array of direct/indirect EXT2\_N\_BLOCKS) and i\_blocks (32-bit integer value) representing the number of 512-byte "blocks" in the file.

The max number of sublevel-directories is 31998, due to the link count limit. Directory indexing is not available in ext2, so there are performance issues for directories with a large number of files (10,000+). The theoretical limit on the number of files in a directory is 1.3 × 1020, although this is not relevant for practical situations.

Note: In Linux 2.4 and earlier, block devices were limited to 2 TiB, limiting the maximum size of a partition, regardless of block size.

**Ext3**

Ext3, or third extended filesystem, is a journaled file system that is commonly used by the Linux kernel. It is the default file system for many popular Linux distributions.  The filesystem was merged with the mainline Linux kernel in November 2001 from 2.4.15 onward. Its main advantage over ext2 is journaling, which improves reliability and eliminates the need to check the file system after an unclean shutdown.

**Advantages**

The performance (speed) of ext3 is less attractive than competing Linux filesystems, such as ext4, JFS, ReiserFS and XFS. But ext3 has a significant advantage in that it allows in-place upgrades from ext2 without having to back up and restore data. Benchmarks suggest that ext3 also uses less CPU power than ReiserFS and XFS. It is also considered safer than the other Linux file systems, due to its relative simplicity and wider testing base.

ext3 adds the following features to ext2:

* A journal.
* Online file system growth.
* HTree indexing for larger directories.

Without these features, any ext3 file system is also a valid ext2 file system. This situation has allowed well-tested and mature file system maintenance utilities for maintaining and repairing ext2 file systems to also be used with ext3 without major changes. The ext2 and ext3 file systems share the same standard set of utilities, e2fsprogs, which includes an fsck tool. The close relationship also makes conversion between the two file systems (both forward to ext3 and backward to ext2) straightforward.

ext3 lacks "modern" filesystem features, such as dynamic inode allocation and extents. This situation might sometimes be a disadvantage, but for recoverability, it is a significant advantage. The file system metadata is all in fixed, well-known locations, and data structures have some redundancy. In significant data corruption, ext2 or ext3 may be recoverable, while a tree-based file system may not.

**Size Limits**

The max number of blocks for ext3 is 232. The size of a block can vary, affecting the max number of files and the max size of the file system:

|  |  |  |
| --- | --- | --- |
| Block size | Maximum file size | Maximum file system size |
| 1 KiB | 16 GiB | 4 TiB |
| 2 KiB | 256 GiB | 8 TiB |
| 4 KiB | 2 TiB | 16 TiB |
| 8 KiB | 2 TiB | 32 TiB |

Note 1 - In Linux, 8 KiB block size is only available on architectures which allow 8 KiB pages, such as Alpha.

**Journaling Levels**

There are three levels of journaling available in the Linux implementation of ext3:

**Journal** (lowest risk)

Both metadata and file contents are written to the journal before being committed to the main file system. Because the journal is relatively continuous on disk, this can improve performance, if the journal has enough space. In other cases, performance gets worse, because the data must be written twice—once to the journal, and once to the main part of the filesystem.

**Ordered** (medium risk)

Only metadata is journaled; file contents are not, but it's guaranteed that file contents are written to disk before associated metadata is marked as committed in the journal. This is the default on many Linux distributions. If there is a power outage or kernel panic while a file is being written or appended to, the journal will indicate that the new file or appended data has not been "committed", so it will be purged by the cleanup process. (Thus appends and new files have the same level of integrity protection as the "journaled" level.) However, files being overwritten can be corrupted because the original version of the file is not stored. Thus it's possible to end up with a file in an intermediate state between new and old, without enough information to restore either one or the other (the new data never made it to disk completely, and the old data is not stored anywhere). Even worse, the intermediate state might intersperse old and new data, because the order of the write is left up to the disk's hardware. XFS uses this form of journaling.

**Writeback** (highest risk)

Only metadata is journaled; file contents are not. The contents might be written before or after the journal is updated. As a result, files modified right before a crash can become corrupted. For example, a file being appended to may be marked in the journal as being larger than it actually is, causing garbage at the end. Older versions of files could also appear unexpectedly after a journal recovery. The lack of synchronization between data and journal is faster in many cases. JFS uses this level of journaling, but ensures that any "garbage" due to unwritten data is zeroed out on reboot.

In all three modes, the internal structure of file system is assured to be consistent even after a crash. In any case, only the data content of files or directories which were being modified when the system crashed will be affected; the rest will be intact after recovery.

**Disadvantages**

**Functionality**

Since ext3 aims to be backwards compatible with the earlier ext2, many of the on-disk structures are similar to those of ext2. Consequently, ext3 lacks recent features, such as extents, dynamic allocation of inodes, and block suballocation. A directory can have at most 31998 subdirectories, because an inode can have at most 32000 links.

ext3, like most current Linux filesystems, fsck cannot be run while the filesystem is mounted for writing. Attempting to check a file system that is already mounted may detect bogus errors where changed data has not reached the disk yet, and corrupt the file system in an attempt to "fix" these errors.

**Defragmentation**

There is no online ext3 defragmentation tool that works on the filesystem level. There is an offline ext2 defragmenter, e2defrag, but it requires that the ext3 filesystem be converted back to ext2 first. But e2defrag may destroy data, depending on the feature bits turned on in the filesystem; it does not know how to treat many of the newer ext3 features.

There are userspace defragmentation tools, like Shake and defrag. Shake works by allocating space for the whole file as one operation, which will generally cause the allocator to find contiguous disk space. If there are files which are used at the same time, Shake will try to write them next to one another. Defrag works by copying each file over itself. However, this strategy works only if the file system has enough free space. A true defragmentation tool does not exist for ext3.

However, as the Linux System Administrator Guide states, "Modern Linux filesystem(s) keep fragmentation at a minimum by keeping all blocks in a file close together, even if they can't be stored in consecutive sectors. Some filesystems, like ext3, effectively allocate the free block that is nearest to other blocks in a file. Therefore it is not necessary to worry about fragmentation in a Linux system."

While ext3 is more resistant to file fragmentation than the FAT filesystem, ext3 can get fragmented over time or for specific usage patterns, like slowly writing large files. Consequently, ext4 (the successor to ext3) has an online filesystem defragmentation utility e4defragand currently supports extents (contiguous file regions).

**Undelete**

ext3 does not support the recovery of deleted files. The ext3 driver actively deletes files by wiping file inodes, for crash safety reasons.

There are still several techniquesand some freeand commercial software for recovery of deleted or lost files using file system journal analysis; however, they do not guarantee any specific file recovery.

**Compression**

e3compris an unofficial patch for ext3 that does transparent compression. It is a direct port of e2compr and still needs further development. It compiles and boots well with upstream kernels, but journaling is not implemented yet.

**Lack of Snapshots Support**

Unlike a number of modern file systems, ext3 does not have native support for snapshots—the ability to quickly capture the state of the filesystem at arbitrary times. Instead, it relies on less-space-efficient, volume-level snapshots provided by the Linux LVM. The Next3 file system is a modified version of ext3 which offers snapshots support, yet retains compatibility with the ext3 on-disk format.

**No Checksum Generation in Journal**

ext3 does not generate a checksum when writing to the journal. On a storage device with extra cache, if barrier=1 is not enabled as a mount option (in /etc/fstab), and if the hardware is doing out-of-order write caching, one runs the risk of severe filesystem corruption during a crash. This is because storage devices with write caches report to the system that the data has been completely written, even if it was written to the (volatile) cache.

Consider the following scenario: If hard disk writes are done out-of-order (due to modern hard disks caching writes in order to amortize write speeds), it is likely that one will write a commit block of a transaction before the other relevant blocks are written. If a power failure or unrecoverable crash should occur before the other blocks get written, the system will have to be rebooted. Upon reboot, the file system will replay the log as normal, and replay the "winners" (transactions with a commit block, including the invalid transaction above, which happened to be tagged with a valid commit block). The unfinished disk write above will thus complete but using corrupt journal data. The file system will thus mistakenly overwrite normal data with corrupt data while replaying the journal. There is a test program available to trigger the problematic behavior. If checksums had been used, where the blocks of the "fake winner" transaction were tagged with a mutual checksum, the file system could have known better and not replayed the corrupt data onto the disk. Journal checksumming has been added to ext4.

Filesystems going through the device mapper interface (including software RAID and LVM implementations) may not support barriers, and will issue a warning if that mount option is used. There are also some disks that do not properly implement the write cache flushing extension necessary for barriers to work, which causes a similar warning.  In these situations, where barriers are not supported or practical, reliable write ordering is possible by turning off the disk's write cache and using the data=journal mount option. Turning off the disk's write cache may be required even when barriers are available.

Applications like databases expect that a call to fsync() will flush pending writes to disk, and the barrier implementation doesn't always clear the drive's write cache in response to that call. There is also a potential issue with the barrier implementation related to error handling during events, such as a drive failure.  It is also known that sometimes some virtualization technologies do not properly forward fsync or flush commands to the underlying devices (files, volumes, disk) from a guest operating system.  Similarly, some hard disks or controllers implement cache flushing incorrectly or not at all, but still advertise that it is supported, and do not return any error when it is used.  There are so many ways to handle fsync and write cache handling incorrectly, it is safer to assume that cache flushing does not work unless it is explicitly tested, regardless of how reliable individual components are believed to be.

**Ext4**

The ext4 or fourth extended filesystem is a journaling file system for Linux, developed as the successor to ext3. Key features include:

**Large File System**

The ext4 filesystem can support volumes with sizes up to 1 exabyte and files with sizes up to 16 terabyte.  However, Red Hat recommends using XFS instead of ext4 for volumes larger than 100 TB.

**Extents**

Extents replace the traditional block mapping scheme used by ext2 and ext3. An extent is a range of contiguous physical blocks, improving large file performance and reducing fragmentation. A single extent in ext4 can map up to 128 MB of contiguous space with a 4 KB block size.  There can be four extents stored in the inode. When there are more than four extents to a file, the rest of the extents are indexed in an HTree.

**Backward Compatibility**

ext4 is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 as ext4. This will slightly improve performance, because certain new features of ext4 can also be used with ext3 and ext2, such as the new block allocation algorithm.

ext3 is partially forward compatible with ext4. Practically, it won't mount as an ext3 filesystem out of the box, unless you disable all new features of ext4 when creating it, and those would be: ^extent,^flex\_bg,^huge\_file,^uninit\_bg,^dir\_nlink,^extra\_isize as -O options of the mke2fs tool.

**Persistent Pre-Allocation**

ext4 can pre-allocate on-disk space for a file. To do this on most file systems, zeros would be written to the file when created. In ext4 (and some other files systems such as XFS) fallocate(), a new system call in the Linux kernel, can be used. The allocated space would be guaranteed and likely contiguous. This situation has applications for media streaming and databases.

**Delayed Allocation**

ext4 uses a performance technique called allocate-on-flush also known as delayed allocation. That is, ext4 delays block allocation until data is flushed to disk. (In contrast, some file systems allocate blocks immediately, even when the data goes into a write cache.) Delayed allocation improves performance and reduces fragmentation by effectively allocating larger amounts of data at a time.

**Increasing the 32,000 Subdirectory Limit**

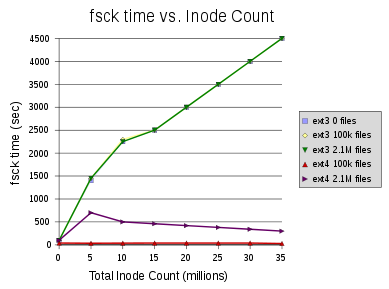
In ext3 a directory can have at most 32,000 subdirectories. Ext4 allows an unlimited number of subdirectories. To allow for larger directories and continued performance, ext4 turns on HTree indexes (a specialized version of a B-tree) by default. This feature is implemented in Linux 2.6.23. In ext3 HTrees can be used by enabling the dir\_index feature.

**Journal Checksumming**

ext4 uses checksums in the journal to improve reliability, since the journal is one of the most used files of the disk. This feature has a side benefit: it can safely avoid a disk I/O wait during journaling, improving performance slightly. Journal checksumming was inspired by a research paper from the University of Wisconsin, titled IRON File Systems (specifically, section 6, called "transaction checksums"), with modifications within the implementation of compound transactions performed by the IRON file system (originally proposed by Sam Naghshineh in the RedHat summit).

**Faster File System Checking**

In ext4 unallocated block groups and sections of the inode table are marked as such. This enables e2fsck to skip them entirely and greatly reduces the time it takes to check the file system. Linux 2.6.24 implements this feature.



[fsck](http://en.wikipedia.org/wiki/Fsck) time/Inode Count (ext3 vs. ext4)

**Multiblock Allocator**

When ext3 appends to a file, it calls the block allocator, once for each block. Consequently, if there are multiple concurrent writers files can easily become fragmented on disk. However, ext4 uses delayed allocation which allows it to buffer data and allocate groups of blocks. Consequently the multiblock allocator can make better choices about allocating files contiguously on disk. The multiblock allocator can also be used when files are opened in O\_DIRECT mode. This feature does not affect the disk format.

**Improved Timestamps**

As computers become faster in general and as Linux becomes used more for mission-critical applications, the granularity of second-based timestamps becomes insufficient. To solve this, ext4 provides timestamps measured in nanoseconds. In addition, 2 bits of the expanded timestamp field are added to the most significant bits of the seconds field of the timestamps to defer the year 2038 problem for an additional 204 years.

ext4 also adds support for date-created timestamps. But, as Theodore Ts'o points out, while it is easy to add an extra creation-date field in the inode (thus technically enabling support for date-created timestamps in ext4), it is more difficult to modify or add the necessary system calls, like stat() (which would probably require a new version) and the various libraries that depend on them (like glibc). These changes would require coordination of many projects. So even if ext4 developers implement initial support for creation-date timestamps, this feature will not be available to user programs for now.

**XFS**

XFS is a high-performance 64-bit journaling file system created by Silicon Graphics, Inc (SGI) in 1993.  It was the default file system in the SGI's IRIX operating system starting with its version 5.3; the file system was ported to the Linux kernel in 2001. As of June 2014, XFS is supported by most Linux distributions, some of which use it as the default file system.

XFS excels in the execution of parallel input/output(I/O) operations due to its design, which is based on allocation groups (a type of subdivision of the physical volumes in which XFS is used- also shortened to AGs). Because of this, XFS enables extreme scalability of I/O threads, file system bandwidth, and size of files and of the file system itself when spanning multiple physical storage devices.

XFS ensures the consistency of data by employing metadata journaling and supporting write barriers. Space allocation is performed via extents with data structures stored in B+ trees, improving the overall performance of the file system, especially when handling large files. Delayed allocation assists in the prevention of file system fragmentation; online defragmentation is also supported. A feature unique to XFS is the pre-allocation of I/O bandwidth at a pre-determined rate, which is suitable for many real-time applications; however, this feature was supported only on IRIX, and only with specialized hardware.

Key features of XFS include:

**Capacity**

XFS is a 64-bit file system.  It supports a maximum file system size of 8 exabytes minus one byte (263-1 bytes), but this limitation can be decreased by limitations imposed by the host operating system. 32-bit Linux systems limit the size of both the file and file system to 16 tebibytes.

**Journaling**

In modern computing, journaling is a capability which ensures consistency of data in the file system, despite any power outages or system crash that may occur. XFS provides journaling for file system metadata, where file system updates are first written to a serial journal before the actual disk blocks are updated. The journal is a circular buffer of disk blocks that is not read in normal file system operation.

The XFS journal is limited to a maximum size of both 64 KB blocks and 128 MB,with the minimum size dependent upon a calculation of the file system block size and directory block size. Placing the journal on an external device larger than the maximum journal size will simply leave the extra space unused by the journal. It can be stored within the data section of the file system (as an internal log), or on a separate device to minimize disk contention.

In XFS, the journal contains "logical" entries that describe, in a humanly understandable way, what operations are being performed (as opposed to a "physical" journal that stores a copy of the blocks modified during each operation). Journal updates are performed asynchronously to avoid a decrease in performance speed.

In the event of a system crash, file system operations which occurred immediately prior to the crash can be reapplied and completed as recorded in the journal, which is how data stored in XFS file systems remain consistent. Recovery is performed automatically the first time the file system is mounted after the crash. The speed of recovery is independent of the size of the file system, instead depending on the amount of file system operations to be reapplied.

**Allocation Groups**

XFS file systems are internally partitioned into allocation groups, which are equally sized linear regions within the file system. Files and directories can span allocation groups. Each allocation group manages its own inodes and free space separately, providing scalability and parallelism — multiple threads and processes can perform I/O operations on the same file system simultaneously.

This architecture helps to optimize parallel I/O performance on systems with multiple processors and/or cores, as metadata updates can also be parallelized. The internal partitioning provided by allocation groups can be especially beneficial when the file system spans multiple physical devices, allowing for optimal usage of throughput of the underlying storage components.

**Striped Allocation**

If an XFS file system is to be created on a striped RAID array, a stripe unit can be specified when the file system is created. This maximizes throughput by ensuring that data allocations, inode allocations and the internal log (the journal) are aligned with the stripe unit.

**Extent Based Allocation**

Blocks used in files stored on XFS file systems are managed with variable length extents where one extent describes one or more contiguous blocks. This can shorten the list of blocks considerably, compared to file systems that list all blocks used by a file individually.

Also, many file systems manage space allocation with one or more block oriented bitmaps — in XFS, these structures are replaced with an extent oriented structure consisting of a pair of B+ trees for each file system allocation group. One of the B+ trees is indexed by the length of the free extents, while the other is indexed by the starting block of the free extents. This dual indexing scheme allows for the highly efficient location of free extents for file system operations.

**Variable Block Sizes**

The file system block size represents the minimum allocation unit. XFS allows file systems to be created with block sizes ranging between 512 bytes and 64 KB, allowing the file system to be tuned for the expected degree of usage. When many small files are expected, a small block size would typically maximize capacity, but for a system dealing mainly with large files, a larger block size can provide a performance efficiency advantage.

**Delayed Allocation**

XFS makes use of lazy evaluation techniques for file allocation. When a file is written to the buffer cache, rather than allocating extents for the data, XFS simply reserves the appropriate number of file system blocks for the data held in memory. The actual block allocation occurs only when the data is finally flushed to disk. This improves the chance that the file will be written in a contiguous group of blocks, reducing fragmentation problems and increasing performance.

**Sparse Files**

XFS provides a 64-bit sparse address space for each file, which allows both for very large file sizes, and for "holes" within files in which no disk space is allocated. As the file system uses an extent map for each file, the file allocation map size is kept small. Where the size of the allocation map is too large for it to be stored within the inode, the map is moved into a B+ tree which allows for rapid access to data anywhere in the 64-bit address space provided for the file.

**Extended Attributes**

XFS provides multiple data streams for files; this is made possible by its implementation of extended attributes. These allow the storage of a number of name/value pairs attached to a file. Names are null-terminated printable character strings which are up to 256 bytes in length, while their associated values can contain up to 64 KB of binary data.

They are further subdivided into two namespaces: root and user. Extended attributes stored in the root namespace can be modified only by the superuser, while attributes in the user namespace can be modified by any user with permission to write to the file.

Extended attributes can be attached to any kind of XFS inode, including symbolic links, device nodes, directories, etc. The attr utility can be used to manipulate extended attributes from the command line, and the xfsdump and xfsrestore utilities are aware of extended attributes, and will back up and restore their contents. Most other backup systems do not support working with extended attributes.

**Direct I/O**

For applications requiring high throughput to disk, XFS provides a direct I/O implementation that allows non-cached I/O operations to be applied directly to the userspace. Data is transferred between the buffer of the application and the disk using DMA, which allows access to the full I/O bandwidth of the underlying disk devices.

**Guaranteed-rate I/O**

The XFS guaranteed-rate I/O system provides an API that allows applications to reserve bandwidth to the filesystem. XFS dynamically calculates the performance available from the underlying storage devices, and will reserve bandwidth sufficient to meet the requested performance for a specified time. This is a feature unique to the XFS file system. Guaranteed rates can be "hard" or "soft", representing a trade-off between reliability and performance; however, XFS will only allow "hard" guarantees if the underlying storage subsystem supports it. This facility is used mostly for real-time applications, such as video streaming.

Guaranteed-rate I/O was only supported under IRIX, and required special hardware for that purpose.

**DMAPI**

XFS implemented the DMAPI interface to support Hierarchical Storage Management in IRIX. As of October 2010, the Linux implementation of XFS supported the required on-disk metadata for DMAPI implementation, but the kernel support was reportedly not usable. For some time, SGI hosted a kernel tree which included the DMAPI hooks, but this support has not been adequately maintained, although kernel developers have stated an intention to bring this support up to date.

**Snapshots**

XFS does not provide direct support for snapshots, as it expects the snapshot process to be implemented by the volume manager. Taking a snapshot of an XFS filesystem involves temporarily halting I/O to the filesystem using the xfs\_freeze utility, having the volume manager perform the actual snapshot, and then resuming I/O to continue with normal operations. The snapshot can then be mounted read-only for backup purposes.

Releases of XFS in IRIX incorporated an integrated volume manager called XLV. This volume manager has not been ported to Linux, and XFS works with standard LVM in Linux systems instead.

In recent Linux kernels, the xfs\_freeze functionality is implemented in the VFS layer, and is executed automatically when the Volume Manager's snapshot functionality is invoked. This was once a valuable advantage as the ext3 file system could not be suspended and the volume manager was unable to create a consistent "hot" snapshot to back up a heavily busy database. Fortunately this is no longer the case. Since Linux 2.6.29, the file systems ext3, ext4, GFS2 and JFS have the freeze feature as well.

**Online Defragmentation**

Although the extent-based nature of XFS and the delayed allocation strategy it uses significantly improves the file system's resistance to fragmentation problems, XFS provides a filesystem defragmentation utility (xfs\_fsr, short for XFS filesystem reorganizer) that can defragment the files on a mounted and active XFS filesystem.

**Online Resizing**

XFS provides the xfs\_growfs utility to perform online resizing of XFS file systems. XFS filesystems can be grown so long as there is remaining unallocated space on the device holding the filesystem. This feature is typically used in conjunction with volume management, as otherwise the partition holding the filesystem will need enlarging separately. XFS partitions cannot (as of August 2010) be shrunk in place, although several possible workarounds have been discussed.

**Native Backup/Restore Utilities**

XFS provides the xfsdump and xfsrestore utilities to aid in the backup of data stored in XFS file systems. The xfsdump utility backs up an XFS filesystem in inode order, and in contrast to traditional UNIX file systems which must be unmounted before dumping to guarantee a consistent dump image, XFS file systems can be dumped while the file system is in use. This is not the same as a snapshot, since files are not frozen during the dump.

XFS dumps and restores are also resumable, and can be interrupted without difficulty. The multi-threaded operation of xfsdump provides high performance of backup operations by splitting the dump into multiple streams, which can be sent to different dump destinations. The multi stream capabilities have not been fully ported to Linux yet, however..

**Atomic Disk Quotas**

Quotas for XFS filesystems are turned on when initially mounted; this fixes a race window that is present with most other filesystems that first require to be mounted and where no quotas are enforced until quotaon(8) is called.

**ReiserFS**

ReiserFS is a general-purpose, journaled computer file system formerly designed and implemented by a team at Namesys led by Hans Reiser. ReiserFS is currently supported on Linux (without quota support). Introduced in version 2.4.1 of the Linux kernel, it was the first journaling file system to be included in the standard kernel. ReiserFS is the default file system on the Elive, Xandros, Linspire, GoboLinux, and Yoper Linux distributions. ReiserFS was the default file system in Novell's SUSE Linux Enterprise until Novell decided to move to ext3 on October 12, 2006 for future releases.

At the time of its introduction, ReiserFS offered features that had not been available in existing Linux file systems:

* Metadata-only journaling (also block journaling, since Linux 2.6.8), its most-publicized advantage over what was the stock Linux file system at the time, ext2.
* Online resizing (growth only), with or without an underlying volume manager such as LVM. Since then, Namesys has also provided tools to resize (both grow and shrink) ReiserFS file systems offline.
* Tail packing, a scheme to reduce internal fragmentation. Tail packing, however, can have a significant performance impact. Reiser4 may have improved this by packing tails where it does not hurt performance.

**Btrfs**

Btrfs (B-tree file system) is a GPL-licensed copy-on-write file system for Linux. Development began at Oracle Corporation in 2007. As of August 2014, the file system's on-disk format has been marked as stable.

Btrfs is intended to address the lack of pooling, snapshots, checksums, and integral multi-device spanning in Linux file systems

**Solaris**

The Sun Microsystems Solaris operating system in earlier releases defaulted to (non-journaled or non-logging) UFS for bootable and supplementary file systems. Solaris defaulted to, supported, and extended UFS.

Support for other file systems and significant enhancements were added over time, including Veritas Software Corp. (Journaling) VxFS, Sun Microsystems (Clustering) QFS, Sun Microsystems (Journaling) UFS, and Sun Microsystems (open source, poolable, 128 bit compressible, and error-correcting) ZFS.

Kernel extensions were added to Solaris to allow for bootable Veritas VxFS operation. Logging or Journaling was added to UFS in Sun's Solaris 7. Releases of Solaris 10, Solaris Express, OpenSolaris, and other open source variants of the Solaris operating system later supported bootable ZFS.

Logical Volume Management allows for spanning a file system across multiple devices for the purpose of adding redundancy, capacity, and/or throughput. Legacy environments in Solaris may use Solaris Volume Manager (formerly known as Solstice DiskSuite). Multiple operating systems (including Solaris) may use Veritas Volume Manager. Modern Solaris based operating systems eclipse the need for Volume Management through leveraging virtual storage pools in ZFS.

**OS X**

OS X uses a file system inherited from classic Mac OS called HFS Plus. Apple also uses the term "Mac OS Extended". HFS Plus is a metadata-rich and case-preserving but (usually) case-insensitive file system. Due to the Unix roots of OS X, Unix permissions were added to HFS Plus. Later versions of HFS Plus added journaling to prevent corruption of the file system structure and introduced a number of optimizations to the allocation algorithms in an attempt to defragment files automatically without requiring an external defragmenter.

Filenames can be up to 255 characters. HFS Plus uses Unicode to store filenames. On OS X, the filetype can come from the type code, stored in file's metadata, or the filename extension.

HFS Plus has three kinds of links: Unix-style hard links, Unix-style symbolic links and aliases. Aliases are designed to maintain a link to their original file even if they are moved or renamed; they are not interpreted by the file system itself, but by the File Manager code in userland.

OS X also supported the UFS file system, derived from the BSD Unix Fast File System via NeXTSTEP. However, as of Mac OS X Leopard, OS X could no longer be installed on a UFS volume, nor can a pre-Leopard system installed on a UFS volume be upgraded to Leopard.  As of Mac OS X Lion UFS support was completely dropped.

Newer versions of OS X are capable of reading and writing to the legacy FAT file systems (16 & 32) common on Windows. They are also capable of reading the newer NTFS file systems for Windows. In order to write to NTFS file systems on OS X versions prior to version 10.6 (Snow Leopard) third party software is necessary. Mac OS X 10.6 (Snow Leopard) and later allows writing to NTFS file systems, but only after a non-trivial system setting change (third party software exists that automates this).

Finally, OS X supports reading and writing of the exFAT file system since Mac OS X Snow Leopard, starting from version 10.6.5.

**PC-BSD**

PC-BSD is a desktop version of FreeBSD, which inherits FreeBSD's ZFS support, similarly to FreeNAS. The new graphical installer of PC-BSD can handle / (root) on ZFS and RAID-Z pool installs and disk encryption using Geli right from the start in an easy convenient (GUI) way. The current PC-BSD 9.0+ 'Isotope Edition' has ZFS filesystem version 5 and ZFS storage pool version 28.

Networking

**Samba**

**Samba** is a free software re-implementation of the SMB/CIFS networking protocol. As of version 3, Samba provides file and print services for various Windows clients and can integrate with a Windows Server domain, either as a Primary Domain Controller (PDC) or as a domain member. It can also be part of an Active Directory domain.

Samba runs on most Unix, OpenVMS and Unix-like systems, such as Linux, Solaris, AIX and the BSD variants, including Apple's OS X Server, and OS X client (version 10.2 and greater). Samba is standard on nearly all distributions of Linux and is commonly included as a basic system service on other Unix-based operating systems as well. Samba is released under the terms of the GNU General Public License. The name Samba comes from SMB (Server Message Block), the name of the standard protocol used by the Microsoft Windows network file system.

Samba allows file and print sharing between computers running Windows and computers running Unix. It is an implementation of dozens of services and a dozen protocols, including:

* NetBIOS over TCP/IP (NBT)
* SMB
* CIFS (an enhanced version of SMB)
* DCE/RPC or more specifically, MSRPC, the Network Neighborhood suite of protocols
* A WINS server also known as a NetBIOS Name Server (NBNS)
* The NT Domain suite of protocols which includes NT Domain Logons
* Security Accounts Manager (SAM) database
* Local Security Authority (LSA) service
* NT-style printing service (SPOOLSS), NTLM and more recently Active Directory Logon which involves a modified version of Kerberos and a modified version of LDAP.
* DFS server

All these services and protocols are frequently incorrectly referred to as just NetBIOS or SMB. The NetBIOS and WINS protocols are deprecated on Windows.

Samba sets up network shares for chosen Unix directories (including all contained subdirectories). These appear to Microsoft Windows users as normal Windows folders accessible via the network. Unix users can either mount the shares directly as part of their file structure using the smbmount command or, alternatively, can use a utility, smbclient (libsmb) installed with Samba to read the shares with a similar interface to a standard command line FTP program. Each directory can have different access privileges overlaid on top of the normal Unix file protections. For example: home directories would have read/write access for all known users, allowing each to access their own files. However they would still not have access to the files of others unless that permission would normally exist. Note that the netlogon share, typically distributed as a read only share from/etc/samba/netlogon, is the logon directory for user logon scripts.

Samba services are implemented as two daemons:

* smbd, which provides the file and printer sharing services, and
* nmbd, which provides the NetBIOS-to-IP-address name service. NetBIOS over TCP/IP requires some method for mapping NetBIOS computer names to the IP addresses of a TCP/IP network.

Samba configuration is achieved by editing a single file (typically installed as /etc/smb.conf or/etc/samba/smb.conf). Samba can also provide user logon scripts and group policy implementation through poledit.

Samba is included in most Linux distributions and is started during the boot process.

**SSH**

Secure Shell, or SSH, is a cryptographic (encrypted) network protocol for initiating secure text-based shell sessions on remote machines. This allows a user to run commands on a machine's command prompt without them being physically present near the machine. It also allows a user to establish a secure channel over an insecure network in a client-server architecture, connecting an SSH client application with an SSH server.Common applications include remote command-line login and remote command execution, but any network service can be secured with SSH. The protocol specification distinguishes between two major versions, referred to as SSH-1 and SSH-2.

The most visible application of the protocol is for access to shell accounts on Unix-like operating systems, but it can also be used in a similar fashion on Windows.

SSH was designed as a replacement for Telnet and other insecure remote shell protocols such as the Berkeley rsh and rexec protocols, which send information, notably passwords, in plaintext, rendering them susceptible to interception and disclosure using packet analysis. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network.

**Public Key Crypto and Key Management**

SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user, if necessary.  There are several ways to use SSH; one is to use automatically generated public-private key pairs to simply encrypt a network connection, and then use password authentication to log on.

Another is to use a manually generated public-private key pair to perform the authentication, allowing users or programs to log in without having to specify a password. In this scenario, anyone can produce a matching pair of different keys (public and private). The public key is placed on all computers that must allow access to the owner of the matching private key (the owner keeps the private key secret). While authentication is based on the private key, the key itself is never transferred through the network during authentication. SSH only verifies whether the same person offering the public key also owns the matching private key. In all versions of SSH it is important to verify unknown public keys, i.e. associate the public keys with identities, before accepting them as valid. Accepting an attacker's public key without validation will authorize an unauthorized attacker as a valid user.

On Unix-like systems, the list of authorized public keys is typically stored in the home directory of the user that is allowed to log in remotely, in the file ~/.ssh/authorized\_keys.  This file is respected by ssh only if it is not writable by anything apart from the owner and root. When the public key is present on the remote end and the matching private key is present on the local end, typing in the password is no longer required (some software like Message Passing Interface (MPI) stack may need this password-less access to run properly). However, for additional security the private key itself can be locked with a passphrase.

The private key can also be looked for in standard places, and its full path can be specified as a command line setting (the option *-i* for ssh). The ssh-keygen utility produces the public and private keys, always in pairs.

SSH also supports password-based authentication that is encrypted by automatically generated keys. In this case the attacker could imitate the legitimate server side, ask for the password, and obtain it (man-in-the-middle attack). However, this is possible only if the two sides have never authenticated before, as SSH remembers the key that the server side previously used. The SSH client raises a warning before accepting the key of a new, previously unknown server. Password authentication can be disabled.

**Uses**

SSH is a protocol that can be used for many applications across many platforms including most Unix variants (Linux, the BSDs including Apple's OS X, & Solaris), as well as Microsoft Windows. Some of the applications below may require features that are only available or compatible with specific SSH clients or servers. For example, using the SSH protocol to implement a VPN is possible, but presently only with the OpenSSH server and client implementation.

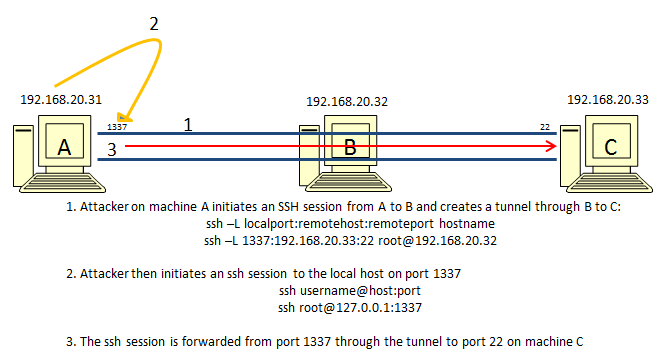
* For login to a shell on a remote host (replacing Telnet and rlogin)
* For executing a single command on a remote host (replacing rsh)
* For setting up automatic (password-less) login to a remote server (for example, using OpenSSH)
* Secure file transfer
* In combination with rsync to back up, copy and mirror files efficiently and securely
* For forwarding or tunneling a port (not to be confused with a VPN, which routes packets between different networks, or bridges two broadcast domains into one).
* For using as a full-fledged encrypted VPN. Note that only OpenSSH server and client supports this feature.
* For forwarding X from a remote host (possible through multiple intermediate hosts)
* For browsing the web through an encrypted proxy connection with SSH clients that support the SOCKS protocol.
* For securely mounting a directory on a remote server as a filesystem on a local computer using SSHFS.
* For automated remote monitoring and management of servers through one or more of the mechanisms discussed above.
* For development on a mobile or embedded device that supports SSH.

There are multiple mechanisms for transferring files using the Secure Shell protocols:

* Secure copy (SCP), which evolved from RCP protocol over SSH
* rsync, intended to be more efficient than SCP
* SSH File Transfer Protocol (SFTP), a secure alternative to FTP (not to be confused with FTP over SSH)
* Files transferred over shell protocol (a.k.a. FISH), released in 1998, which evolved from Unix shell commands over SSH

**Secure Shell Tunneling**

A Secure Shell (SSH) tunnel consists of an encrypted tunnel created through an SSH protocol connection. Users may set up SSH tunnels to transfer unencrypted traffic over a network through an encrypted channel. For example, Microsoft Windows machines can share files using the Server Message Block (SMB) protocol, a non-encrypted protocol. If one were to mount a Microsoft Windows file-system remotely through the Internet, someone snooping on the connection could see transferred files. To mount the Windows file-system securely, one can establish a SSH tunnel that routes all SMB traffic to the remote fileserver through an encrypted channel. Even though the SMB protocol itself contains no encryption, the encrypted SSH channel through which it travels offers security.



To set up an SSH tunnel, one configures an SSH client to forward a specified local port to a port on the remote machine. Once the SSH tunnel has been established, the user can connect to the specified local port to access the network service. The local port does not have to be the same as the remote port.

SSH tunnels provide a means to bypass firewalls that prohibit certain Internet services – so long as a site allows outgoing connections. For example, an organization may prohibit a user from accessing Internet web pages (port 80) directly without passing through the organization's proxy filter (which provides the organization with a means of monitoring and controlling what the user sees through the web). But users may not wish to have their web traffic monitored or blocked by the organization's proxy filter. If users can connect to an external SSH server, they can create an SSH tunnel to forward a given port on their local machine to port 80 on a remote web server. To access the remote web server, users would point their browser to the local port at http://localhost/

Some SSH clients support dynamic port forwarding that allows the user to create a SOCKS 4/5 proxy. In this case users can configure their applications to use their local SOCKS proxy server. This gives more flexibility than creating an SSH tunnel to a single port as previously described. SOCKS can free the user from the limitations of connecting only to a predefined remote port and server. If an application doesn't support SOCKS, a proxifier can be used to redirect the application to the local SOCKS proxy server. Some proxifiers, such as Proxycap, support SSH directly, thus avoiding the need for an SSH client.

Users & Authentication

**/etc/passwd & /etc/shadow**

/etc/passwd is a tool on most [Unix](http://en.wikipedia.org/wiki/Unix) and [Unix-like](http://en.wikipedia.org/wiki/Unix-like) operating systems used to change a user's password. The password entered by the user is run through a key derivation function to create a hashed version of the new password, which is saved. Only the hashed version is stored; the entered password is not saved for security reasons.

When the user logs on, the password entered by the user during the log on process is run through the same key derivation function and the resulting hashed version is compared with the saved version. If the hashes are identical, the entered password is considered to be identical, and so the user is authenticated. In theory, it is possible to occur that two different passwords produce the same hash. However, cryptographic hash functions are designed in such a way that finding any password that produces the given hash is very difficult, so if the produced hash matches the stored one, the user can be authenticated.

The passwd command may be used to change passwords for local accounts, and on most systems, can also be used to change passwords managed in a distributed authentication mechanism such as NIS, Kerberos, or LDAP.

**Password File**

The /etc/passwd file is a text-based database of information about users that may log in to the system or other operating system user identities that own running processes.

In many operating systems this file is just one of many possible back-ends for the more general passwd name service.

The file's name originates from one of its initial functions as it contained the data used to verify passwords of user accounts. However, on modern Unix systems the security-sensitive password information is instead often stored in a different file using shadow passwords, or other database implementations.

The /etc/passwd file typically has file system permissions that allow it to be readable by all users of the system (*world-readable*), although it may only be modified by the superuser or by using a few special purpose privileged commands.

The /etc/passwd file is a text file with one record per line, each describing a user account. Each record consists of seven fields separated by colons. The ordering of the records within the file is generally unimportant.

An example record may look like:

jsmith:x:1001:1000:Joe Smith,phonenum,email:/home/jsmith:/bin/sh

The fields, in order from left to right, are:

1. User name: the string a user would type in when logging into the operating system: the login name. Must be unique across users listed in the file.
2. Information used to validate a user's password; in most modern uses, this field is usually set to "x" (or some other indicator) with the actual password information being stored in a separate shadow password file. Setting this field to an asterisk ("\*") is a common way to disable direct logins to an account while still preserving its name. Another possible value is "\*NP\*" which indicates to use an NIS server to obtain the password.
3. user identifier number, used by the operating system for internal purposes. It need not be unique.
4. group identifier number, which identifies the primary group of the user; all files that are created by this user may initially be accessible to this group.
5. Gecos field, commentary that describes the person or account. Typically, this is a set of comma-separated values including the user's full name and contact details.
6. Path to the user's home directory.
7. Program that is started every time the user logs into the system. For an interactive user, this is usually one of the system's command line interpreters (shells).

**Shadow File**

/etc/shadow is used to increase the security level of passwords by restricting all but highly privileged users' access to hashed password data. Typically, that data is kept in files owned by and accessible only by the super user.

Systems administrators can reduce the likelihood of brute force attacks by making the list of hashed passwords unreadable by unprivileged users. The obvious way to do this is to make the passwd database itself readable only by the root user. However, this would restrict access to other data in the file such as username-to-userid mappings, which would break many existing utilities and provisions. One solution is a "shadow" password file to hold the password hashes separate from the other data in the world-readable *passwd* file. For local files, this is usually /etc/shadow on Linux and Unix systems, or /etc/master.passwd on BSD systems; each is readable only by *root*. (Root access to the data is considered acceptable since on systems with the traditional "all-powerful root" security model, the root user would be able to obtain the information in other ways in any case). Virtually all recent Unix-like operating systems use shadowed passwords.

The shadow password file does not entirely solve the problem of attacker access to hashed passwords, as some network authentication schemes operate by transmitting the hashed password over the network (sometimes in cleartext, e.g., Telnet), making it vulnerable to interception. Copies of system data, such as system backups written to tape or optical media, can also become a means for illicitly obtaining hashed passwords. In addition, the functions used by legitimate password-checking programs need to be written in such a way that malicious programs cannot make large numbers of authentication checks at high rates of speed.

On a system without shadowed passwords (typically older Unix systems dating from before 1990 or so), the passwd file holds the following user information for each user account:

* Username
* Salt combined with the current hash of the user's password (usually produced from a cryptographic hash function)
* Password expiration information
* User ID (UID)
* Default group ID (GID)
* Full name
* Home directory path
* Login shell

The passwd file is readable by all users so that name service switch can work (e.g., to ensure that user names are shown when the user lists the contents of a folder), but only the root user can write to it. This means that an attacker with unprivileged access to the system can obtain the hashed form of every user's password. Those values can be used to mount a brute force attack offline, testing possible passwords against the hashed passwords relatively quickly without alerting system security arrangements designed to detect an abnormal number of failed login attempts. Especially when the hash is not salted it is also possible to lookup these hashed passwords in Rainbow Tables.

With a shadowed password scheme in use, the /etc/passwd file typically shows a character such as '\*', or 'x' in the password field for each user instead of the hashed password, and /etc/shadow usually contains the following user information:

* User login name
* salt and hashed password OR a status exception value e.g.:
  + "$id$salt$hashed", the printable form of a password hash as produced by crypt (C), where "$id" is the algorithm used. (On GNU/Linux, "$1$" stands for MD5, "$2a$" is Blowfish, "$2y$" is Blowfish (correct handling of 8-bit chars), "$5$" is SHA-256 and "$6$" is SHA-512, crypt(3) man page, other Unix may have different values, like NetBSD. Key stretching is used to increase password cracking difficulty, using by default 1000 rounds of modified MD5, 64 rounds of Blowfish, 5000 rounds of SHA-256 or SHA-512. The number of rounds may be varied for Blowfish, or for SHA-256 and SHA-512 by using e.g. "$6$rounds=50000$".)
  + Empty string - No password, the account has no password. (Reported by passwd on Solaris with "NP")
  + "!" - the account is password Locked, user will be unable to log-in via password authentication but other methods (e.g. ssh key) may be still allowed)
  + "\*LK\*" or "\*" - the account is Locked, user will be unable to log-in via password authentication but other methods (e.g. ssh key) may be still allowed)
  + "!!" - the password has never been set (RedHat)
* Days since epoch of last password change
* Days until change allowed
* Days before change required
* Days warning for expiration
* Days before account inactive
* Days since Epoch when account expires
* Reserved

The format of the shadow file is simple, and basically identical to that of the password file, one line per user, ordered fields on each line, and fields separated by colons. Many systems require the order of user lines in the shadow file be identical to the order of the corresponding users in the password file.

Command Line

**Bourne Shell**

The Bourne shell was one of the major shells used in early versions of the Unix operating system and became a *de facto* standard. It was written by Stephen R. Bourne at Bell Labs and was first distributed with Version 7 Unix, circa 1977. Every Unix-like system has at least one shell compatible with the Bourne shell. The Bourne shell program name is sh and its path in the Unix file system hierarchy is typically /bin/sh. On many systems, however, this may be a symbolic link or hard link to a compatible, but more feature-rich shell than the Bourne shell.

The POSIX standard specifies its standard shell as a strict subset of the Korn shell, an enhanced version of the Bourne shell. From a user's perspective the Bourne shell was immediately recognized when active by its characteristic default command line prompt character, the dollar sign ($).

**C Shell**

The C shell was developed by Bill Joy for the Berkeley Software Distribution (BSD), a line of Unix operating systems derived from Unix and developed at the University of California, Berkeley. It was originally derived from the 6th Edition Unix shell (Thompson shell), with its syntax modeled after the C programming language. The C shell is used primarily for interactive terminal use, and less frequently for scripting and operating system control. It has interactive keyboard shortcuts in form of special control-key sequences for special effects such as job control.

**Bourne Again Shell**

Bash is a Unix shell written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. Released in 1989, it is a command processor that typically runs in a text window, where the user types commands that cause actions. Bash can also read commands from a file, called a script. Like all Unix shells, it supports filename wildcarding, piping, here documents, command substitution, variables and control structures for condition-testing and iteration. The keywords, syntax and other basic features of the language were all copied from sh. Other features, e.g., history, were copied from csh and ksh. Bash is a POSIX shell, but with a number of extensions.

Linux Logs

Almost all log files are located under /var/log directory and its sub-directories on Linux. You can change to this directory using the cd command. You need be the root user to view or access log files on Linux or Unix like operating systems.

Common Linux log files names and usage

* /var/log/messages : General message and system related stuff
* /var/log/auth.log : Authenication logs
* /var/log/kern.log : Kernel logs
* /var/log/cron.log : Crond logs (cron job)
* /var/log/maillog : Mail server logs
* /var/log/qmail/ : Qmail log directory (more files inside this directory)
* /var/log/httpd/ : Apache access and error logs directory
* /var/log/lighttpd/ : Lighttpd access and error logs directory
* /var/log/boot.log : System boot log
* /var/log/mysqld.log : MySQL database server log file
* /var/log/secure or /var/log/auth.log : Authentication log
* /var/log/utmp or /var/log/wtmp : Login records file
* /var/log/yum.log : Yum command log file.

Mission Phases

Reconnaissance

**Footprinting –** The process of conducting target analysis, identification and discovery. This could include dumpster diving, social engineering and the use of utilities such as web-search hacking, traceroutes, pings, network lookups, etc.

**Scanning –** This step will take the findings from footprinting and begin to dig a little deeper. In a traditional sense, this step includes port scanning, OS identification, and determining whether or not a machine is accessible.

Enumeration

This is the phase where you further interrogate specific services to determine exact operating systems, software, etc. Normal enumeration techniques include searching for network share information, specific versions of applications running, user accounts, SNMP traffic, etc.

**Network Mapping –** This step involves laying out an illustration of the target network. This includes taking all available resources (logs, target surveys, etc.) to create a visualization of the target environment. The perspective of the map will often differ from the exploiters to the system administrators.

Gaining/Maintaining Access

**Gaining Access –** This step is the exploitation process. Basically, this is gaining access to the machine or the network by a client-side exploit, insider threat, interdiction attacks, or a remote exploitation opportunity. This could be conducted via spear-phishing attacks, buffer overflows, embedded device exploitation, credential masquerade attacks, etc.

**Privilege Escalation –** Depending on the exploitation opportunity that was utilized, the attacker may need to elevate privileges to a different user. There are various different scenarios in which the attacker will need to use this procedure. Typically, this is conducted through the use of a local exploit opportunity in order to gain root, or system level, privileges.

**Maintaining Access –** This step typically involves placing a means of persistence on the exploited system.

**Rootkit** (aka Backdoor, Implant, persistence) – This step involves installing an application, hooking the kernel, or laying down come mechanism which allows the operator to maintain continued access to the host or network. If the implant is well designed, the operator could live in the network for extended periods of time.

Post Exploitation

**Target Survey & Remote Forensics –** This step is to conduct analysis on the target machine for potential security mechanisms, files, or uses which could either assist in obtaining the objective or harm the operation. This is the process of analyzing the targets operating system and applications.

**Data Collection -** The focus of this step is to syphon as much quality data as possible. Network traffic analysis and efficient searching is key.

**Computer Network Attack -** In this step the operator has already identified the network as a target or opportunity and has identified plans to launch an attack. This attack could be remote or local in nature and could come from already established access or with no access to the targeted environment. The attack will typically identify core and vital network processes and perform various attacks to disrupt, deny, degrade, destroy, or deceive the adversary.

**Cover Tracks –** This step is the process of removing any forensically relevant residue that was left behind as the result of exploitation or presence. This is one of the most important steps that an operator can perform.

Documentation

Target Evaluations

This phase of documentation involves the documentation of targets. Think about the following:

* Who uses the target system
* What is the target system used for
* What vulnerabilities are potentially exploitable
* Is the system a trusted node on a domain
* What security software/appliances are on the system

Currently there is no defined process, or location, for target evaluations in the unit.

Security Software Evaluations

This involves documenting what security applications detect tools and ways to avoid or mitigate detection.

Mission Execution

Changeover Debrief

Operators will brief the oncoming crew of any notable items prior to the next crew assuming control of the operations floor. Items that should be covered are:

* Ongoing operations to include status
* Significant events impacting operations
* Read Files/eCIFs

Attribution

The art of obfuscating the true source of operator actions. This could involve using a VPN, proxy, robust infrastructure, redirection, pivots or spoofing IP addresses.

Tunneling

Tunneling protocol works by using the data portion of a packet (the payload) to carry the packets that actually provide the service. Tunneling uses a layered protocol model such as those of the [OSI](http://en.wikipedia.org/wiki/Open_Systems_Interconnection) or [TCP/IP](http://en.wikipedia.org/wiki/TCP/IP) protocol suite, but usually violates the layering when using the payload to carry a service not normally provided by the network. Typically, the delivery protocol operates at an equal or higher level in the layered model than the payload protocol.

Tunneling protocols may use data encryption to transport insecure payload protocols over a public network (such as the Internet), thereby providing [VPN](http://en.wikipedia.org/wiki/VPN) functionality. [IPsec](http://en.wikipedia.org/wiki/IPsec) has an end-to-end Transport Mode, but can also operate in a tunneling mode through a trusted security gateway.

File Transfers

File transfers are the act of transmitting files over a computer network. There are multiple ways to do so, both secure and non-secure.

**FTP**

The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet.

FTP is built on a client-server architecture and uses separate control and data connections between the client and the server.  FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS). SSH File Transfer Protocol (SFTP) is sometimes also used instead, but is technologically different.

The first FTP client applications were command-line applications developed before operating systems had graphical user interfaces, and are still shipped with most Windows, Unix, and Linux operating systems

**CP**

cp is a UNIX command for copying files and directories. The command has three principal modes of operation, expressed by the types of arguments presented to the program for copying a file to another file, one or more files to a directory, or for copying entire directories to another directory.

The utility further accepts various command line option flags to detail the operations performed. The two major specifications are POSIX cp and GNU cp. GNU cp has many additional options over the POSIX version

**SCP**

Secure copy is a means of securely transferring computer files between a local host and a remote host or between two remote hosts. It is based on the Secure Shell (SSH) protocol.

**SCP Binary**

The SCP binary is a software tool implementing the SCP protocol as a service daemon or client. It is a program to perform secure copying. The SCP server program is typically the same program as the SCP client.

Perhaps the most widely used SCP program is the command line scp program, which is provided in most SSH implementations. The scp program is the secure analog of the rcp command. The scp program must be part of all SSH servers that want to provide SCP service, as scp functions as SCP server too.

Some SSH implementations provide the scp2 program, which uses the SFTP protocol instead of SCP, but provides the very same command line interface as scp. scp is then typically a symbolic link to scp2.

Typically, a syntax of scp program  is like the syntax of cp:

Copying file to host:

scp *SourceFile* *user*@*host*:*directory*/*TargetFile*

Copying file from host:

scp *user*@*host*:*directory*/*SourceFile* *TargetFile*

scp -r *user*@*host*:*directory*/*SourceFolder* *TargetFolder*

Note that if the remote host uses a port other than the default of 22, it can be specified in the command. For example, copying a file from host:

scp -P 2222 *user*@*host*:*directory*/*SourceFile* *TargetFile*

**XCOPY**

XCOPY is a command used on PC DOS, MS-DOS, OS/2, Microsoft Windows, and related operating systems for copying multiple files or entire directory trees from one directory to another and for copying files across a network. XCOPY stands for *extended copy*, and was created as a more functional file copying utility than the copy command found in these operating systems. XCOPY first appeared in DOS 3.2.

Create a new directory by copying all contents of the existing directory, including any files or subdirectories having the "hidden" or "system" attributes and empty directories:

xcopy e:\existing e:\newcopy /s /e /i /h

If the directory names include blank signs(spaces), the names can be put in quotation marks:

xcopy "D:\Documents and Settings\MY.USERNAME\My Documents\\*" "E:\MYBACKUP\My Documents\" /D/E/C/Y

Copy entire drive in to a mapped network drive while ignoring any errors in network restartable mode:

xcopy \*.\* z:\Netmirror /E /V /C /F /H /Y /Z 1>out.txt 2>err.txt

Copy a single file without prompt if it is a file or a directory:

cmd /c **echo** F | xcopy "c:\directory 1\myfile" "c:\directory 2\myfile"

**Robocopy**

Robocopy, or "Robust File Copy", is a command-line directory and/or file replication command. Robocopy functionally replaces Xcopy, with more options. It has been available as part of the Windows Resource Kit starting with Windows NT 4.0, and was first introduced as a standard feature in Windows Vista and Windows Server 2008

Logging Operator Actions

**History File**

The various Unix shells maintain a record of the commands issued by the user during the current session. The history command manipulates this history list. In its simplest form, it prints the history list. Options allow for the recall and editing of particular commands and for setting parameters such as the number of past commands to retain in the list. All potential uses for the history command are available in the man page.

**TCPDump/Wireshark**

Records all packets that leave the operational system on the specified interface. This allows for accurate reconstruction of events when paired with other means of logging.

Emergency Actions

Cohabitation

Cohabitation involves operators acting on a machine concurrently with other services or other nations.

All instances of cohabitation should be thoroughly evaluated prior to engagement. Typical questions you should ask include:

Blue/Blue Cohabitation

* What effect will **their** operations have on my operations
* What effect will **my** operations have on their operations
* Are the other cohabitants cleared to see/have access to our tools

Red/Blue Cohabitation

* Are we able to reside at a lower level than the Red actors and remain undetected
* What is the TGL level of our access/tools

Administrator Presence on Target

During operations SA should be maintained at all times. One thing all operators should do when accessing a remote system is to determine what users are currently logged on. The worst case scenario is that an administrator is actively engaged on the system. Any instance where an administrator is observed on a system should be routed through the Operations Controller, Crew Commander, or against relative RoEs for that specific operation.

Undocumented/Unexpected Security Solution

Implant/Presence Discovered by Adversary

Appendix 3 – NTFS Meta Files

| **List of NTFS metafiles** | | |
| --- | --- | --- |
| **Segment Number** | **File Name** | **Purpose** |
| 0 | $MFT | Describes all files on the volume, including file names, timestamps, stream names, and lists of cluster numbers where data streams reside, indexes, [security identifiers](http://en.wikipedia.org/wiki/Security_identifier), and file attributes like "read only", "compressed", "encrypted", etc. |
| 1 | $MFTMirr | Duplicate of the first vital entries of $MFT, usually 4 entries (4 [Kilobytes](http://en.wikipedia.org/wiki/Kilobyte)). |
| 2 | $LogFile | Contains transaction log of file system metadata changes. |
| 3 | $Volume | Contains information about the volume, namely the volume object identifier, [volume label](http://en.wikipedia.org/wiki/Volume_label), file system version, and volume flags (mounted, chkdsk requested, requested $LogFile resize, mounted on NT 4, volume serial number updating, structure upgrade request). This data is not stored in a data stream, but in special MFT attributes: If present, a volume object ID is stored in an $OBJECT\_ID record; the volume label is stored in a $VOLUME\_NAME record, and the remaining volume data is in a $VOLUME\_INFORMATION record. Note: volume serial number is stored in file $Boot (below). |
| 4 | $AttrDef | A table of MFT attributes that associates numeric identifiers with names. |
| 5 | . | [Root directory](http://en.wikipedia.org/wiki/Root_directory). Directory data is stored in $INDEX\_ROOT and $INDEX\_ALLOCATION attributes both named $I30. |
| 6 | $Bitmap | An array of bit entries: each bit indicates whether its corresponding cluster is used (allocated) or free (available for allocation). |
| 7 | $Boot | [Volume boot record](http://en.wikipedia.org/wiki/Volume_boot_record). This file is always located at the first clusters on the volume. It contains [bootstrap code](http://en.wikipedia.org/w/index.php?title=Bootstrap_code&action=edit&redlink=1) (see[NTLDR](http://en.wikipedia.org/wiki/NTLDR)/[BOOTMGR](http://en.wikipedia.org/wiki/Windows_Vista_startup_process#Windows_Boot_Manager)) and a [BIOS parameter block](http://en.wikipedia.org/wiki/BIOS_parameter_block) including a [volume serial number](http://en.wikipedia.org/wiki/Volume_serial_number) and cluster numbers of $MFT and $MFTMirr. $Boot is usually 8192 bytes long.[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] |
| 8 | $BadClus | A file that contains all the clusters marked as having [bad sectors](http://en.wikipedia.org/wiki/Bad_sector). This file simplifies cluster management by the chkdsk utility, both as a place to put newly discovered bad sectors, and for identifying unreferenced clusters. This file contains two data streams, even on volumes with no bad sectors: an unnamed stream contains bad sectors—it is zero length for perfect volumes; the second stream is named $Bad and contains all clusters on the volume not in the first stream. |
| 9 | $Secure | [Access control list](http://en.wikipedia.org/wiki/Access_control_list) database that reduces overhead having many identical ACLs stored with each file, by uniquely storing these ACLs in this database only (contains two indices: $SII *(Standard\_Information ID)* and $SDH*([Security Descriptor](http://en.wikipedia.org/wiki/Security_Descriptor" \o "Security Descriptor) Hash)*, which index the stream named $SDS containing actual ACL table).[[2]](http://en.wikipedia.org/wiki/NTFS#cite_note-insidewin2kntfs-2) |
| 10 | $UpCase | A table of unicode uppercase characters for ensuring case-insensitivity in Win32 and DOS namespaces. |
| 11 | $Extend | A filesystem directory containing various optional extensions, such as $Quota, $ObjId, $Reparse or $UsnJrnl. |
| 12–23 | Reserved for $MFT extension entries. Extension entries are additional MFT records that contain additional attributes that do not fit in the primary record. This could occur if the file is sufficiently fragmented, has many streams, long filenames, complex security, or other rare situations. | |
| 24 | $Extend\$Quota | Holds disk quota information. Contains two index roots, named $O and $Q. |
| 25 | $Extend\$ObjId | Holds [link tracking](http://en.wikipedia.org/wiki/NTFS#Distributed_Link_Tracking_.28DLT.29) information. Contains an index root and allocation named $O. |
| 26 | $Extend\$Reparse | Holds [reparse point](http://en.wikipedia.org/wiki/Reparse_point) data (such as [symbolic links](http://en.wikipedia.org/wiki/Symbolic_link)). Contains an index root and allocation named $R. |
| 27— | Beginning of regular file entries. | |