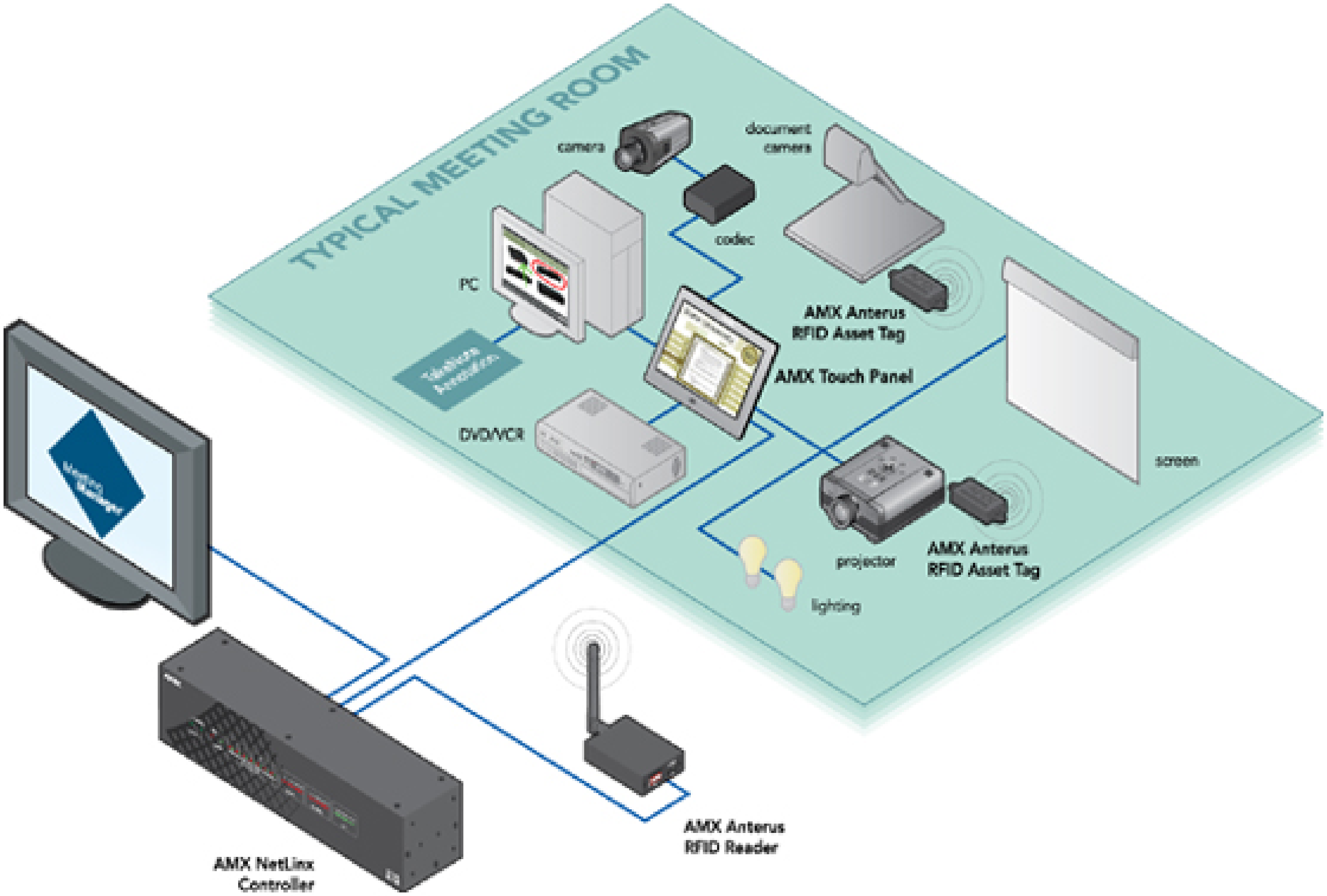
Principles, Concepts and Elements of Computer Configuration



Herschel Jolly

Lavonda Forsyth

First Edition, 2012

ISBN 978-81-323-0971-0

Published by:
Academic Studio
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Autoconfig and Autoconf

Autoconfig

Autoconfig is an auto-configuration feature of Amiga computers which assigns resources to expansion devices without the need for jumpers. It is analogous to PCI configuration.

When the computer is switched on, AmigaOS interrogates each expansion device in turn and assigns address space as needed. In the case of a memory card, the OS would also add the memory to the available system memory. Autoconfig also supports boot ROMs.

Protocol

Expansion devices respond to certain fixed memory addresses starting at hexadecimal E80000 (or FF000000 for Zorro III) if the /CFGIN signal is asserted and the device is not already configured. The CPU reads nibbles of configuration information (usually supplied by a PAL) such as manufacturer ID, product ID, and the amount of address space the device requires. The CPU then writes a base memory address to the device (or tells it to "shut up" if for some reason it can't be configured), and the device asserts /CFGOUT.

The /CFGIN of the first device is tied to ground. The second device's /CFGIN is controlled by the first device's /CFGOUT, and so on.

In a backplane design such as the Amiga 2000, connecting the /CFGOUT of one slot directly to the /CFGIN of the next would create the problem that an unoccupied slot would break the configuration chain. To solve this, the backplane ORs the /CFGIN and /CFGOUT signals to form the /CFGIN for the next slot (/CFGOUT is pulled low if undriven), which allows empty slots to be bypassed. This requires one 74LS32 (quad OR gate) on the Amiga 2000, which is the only motherboard hardware required by Autoconfig.

Hardware specifications and BUS connections

Autoconfig was initially how the Amiga 2000 (which first sported Zorro II) configured its ZorroII cards. In the A3000 architecture, it was deemed desirable for all enumerable hardware to submit to Autoconfig. It was OS-legal for non-Autoconfig hardware to be completely ignored and this standard was adopted in AmigaOS 3.1.

Autoconfig was inherent to the Zorro II specification and as such required the MC68k data and address bus to be present on each card. A virtual address system, such as PCI, would require a minor revision to Autoconfig as it stood.

Amiga 2000 could address 5 Zorro expansion cards such as RAM expansions, SCSI controllers for hard disks, graphic cards, etc. but the actual standard provided for an unlimited number of different cards. Two of these cards are front end with ISA bus of Amiga 2000, and could be joined with Bridgeboard Amiga Janus Hardware Emulator, which allows Amiga to run PC Intel based software 80286 and 80386 supported by the two Amiga cards.

Zorro III was the 32 bit autoconfiguring slot system of Amiga 3000 and Amiga 4000.

Comparison with PCI configuration

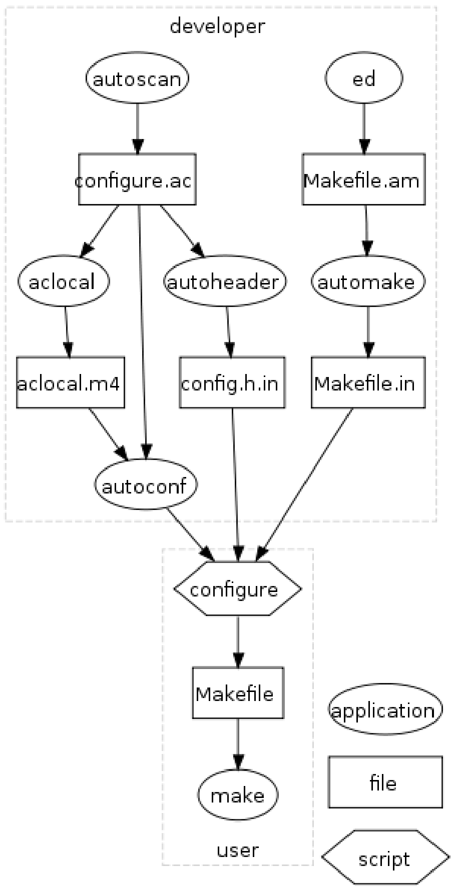
Compared with PCI configuration, Autoconfig is much simpler, yet provides the same basic functions. PCI allows random access to the configuration space of devices, which requires system registers and I/O lines. Autoconfig requires no such system hardware, but has the restriction that devices can only be configured in sequence, and they remain configured until reset. Autoconfig does support hot-plugging but only for one device (the last one). Most manufacturers which required hot-plugging instead did not use Autoconfig for whatever was being added and removed (e.g. a PCMCIA card) but instead assigned whatever resource was necessary permanently to the port or controller and handled the addition or removal much like inserting a floppy disk.

An Amiga's Autoconfig is performed by the OS at boot-time and may not be changed without rebooting. In theory, PCI can change its resource allocation at any time, though both the popular Linux and Windows operating systems do not allow such changes due to architectural limitations in the respective operating systems. Direct PCI hardware (e.g. A PCI card), however, may not be hot-plugged (PCI configuration registers are a separate part of the specification) due to the synchronous arbited nature of the bus. So, PCI can reallocate resources on the fly, which it does when the OS loads and may override BIOS resource allocation, but cannot change the hardware while the system is running. Autoconfig can change the hardware while the system is running but only for the last hardware in the config sequence, or to add a new piece of hardware. Neither Autoconfig nor PCI PnP actually allow this in any considerable operating system.

Notation

In late official Commodore documents (Zorro III specs) it was spelled in all capitals: *AUTOCONFIG*. Earlier notations include *Auto Configuration* (A500/A2000/B2000 service manual) *auto-config* (same) and *AutoConfig* (A2000 user manual).

Autoconf



Flow diagram of autoconf and automake. Note that "configure.ac" was named "configure.in" in early versions of autoconf.

GNU Autoconf is a tool for producing configure scripts for building, installing and packaging software on computer systems where a Bourne shell is available.

Autoconf is agnostic about the programming languages used, but it is often used for projects using C, C++, Fortran, Fortran 77, Erlang and Objective-C.

A configure script configures a software package for installation on a particular target system. After running a series of tests on the target system the configure script generates header files and a makefile from templates, thus customizing the software package for the target system. Together with Automake and Libtool, Autoconf forms the GNU build system. It comprises several other tools, notably Autoheader.

Usage overview

The developer specifies the desired behaviour of the configure script by writing a list of instructions in the GNU m4 language in a file called "configure.ac". A library of pre-defined m4 macros is available to describe common configure script instructions. Autoconf transforms the instructions in "configure.ac" into a portable configure script. The building system is not supposed to have autoconf installed: autoconf is needed only to build the configure script, that is usually shipped with the software.

configure.ac format

The GNU Autoconf manual suggests the following format for the configure.ac file:

Autoconf requirements
: The AC_PREREQ(*version*) macro can be used to ensure that a recent enough version of the autoconf program is available to process the configure.ac file

AC_INIT(*package*, *version*, *bug-report-address*)
: This macro is required in every configure.ac file. It specifies the name and version of the software package for which to generate a configure script and the email address of the developer.

information on the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
AC_CONFIG_FILES([file...])
AC_OUTPUT

History

Autoconf was begun in the summer of 1991 by David Mackenzie to support his work at the Free Software Foundation. In the subsequent years it grew to include enhancements from a variety of authors and became the most widely used build configuration system for writing portable free or open-source software.

Approach

Autoconf is similar to the Metaconfig package used by Perl. The imake system formerly used by the X Window System (up to X11R6.9) is closely related, but has a different philosophy.

The Autoconf approach to portability is to test for features, not for versions. For example, the native C compiler on SunOS 4 did not support ISO C. However, it is possible for the user or administrator to have installed an ISO C-compliant compiler. A pure version-based approach would not detect the presence of the ISO C compiler, but a feature-testing approach would be able to discover the ISO C compiler the user had installed. The rationale of this approach is to gain the following advantages:

- the configure script can get reasonable results on newer or unknown systems
- it allows administrators to customize their machines and have the configure script take advantage of the customizations
- there is no need to keep track of minute details of versions, patch numbers, etc., to figure out whether a particular feature is supported or not

Criticism

Autoconf is an old and mature product that, when properly used, provides a very simple interface even in complex cross-compiling scenarios. However there is some criticism that states that Autoconf uses dated technologies, has a lot of legacy restrictions, and complicates simple scenarios unnecessarily for the author of *configure.ac* scripts. In particular, often cited weak points of Autoconf are:

- General complexity of used architecture, most projects use multiple repetitions.
- Generated 'configure' is written in Unix shell and thus Makefile generation is slow. Some projects, such as KDE, tried to work around this issue by introducing distinct Makefile generators written in other languages.
- Some people think that 'configure' scripts generated by autoconf provides only manual-driven command-line interface without any standardization. While it is true that some developers do not respect common conventions, such conventions still exist and are widely used.
- M4 is unusual and unknown to many developers. Developers will need to learn it to extend autoconf with non-standard checks.
- Weak backward and forward compatibility requires a wrapper script.
- Autoconf-generated scripts are usually large and rather complex. Although they produce extensive logging, debugging them can still be difficult.

Due to these limitations, some projects that used GNU Build System started to switch to other build systems:

- KDE project switched to CMake, starting with KDE 4.
- Scribus switched to CMake.

- Blender switched to SCons on 2004-02-15.

Chapter 2

Apple Sleep Proxy Service (Bonjour Sleep Proxy) and Control Panel (Windows)

Apple Sleep Proxy Service (Bonjour Sleep Proxy)

The **Sleep Proxy Service** is an open source component of zero configuration networking, designed to assist in reducing power consumption of networked electronic devices. A device acting as a **sleep proxy server** will respond to Multicast DNS queries for another, compatible device which has gone into low power mode. The *low-power-mode* device remains *asleep* while the Sleep Proxy Server responds to any Multicast DNS queries.

When the Sleep Proxy Server sees a query which requires the *low-power-mode* device to *wake up*, the Sleep Proxy Server sends a special wake-up-packet ("magic packet") to the low-power-mode device. Finally, communication parameters are updated via Multicast DNS and normal communications proceed.

Apple Inc. describes the service as **Bonjour Sleep Proxy** in their support documents. The service supports the **Wake on Demand** feature, first offered in Mac OS X Snow Leopard.

Details

Address resolution protocol

The sleep proxy service responds to address resolution protocol requests on behalf of the low-power-mode device:

"When a Sleep Proxy sees an IPv4 ARP or IPv6 ND Request for one of the sleeping device's addresses, it answers on behalf of the sleeping device, without waking it up, giving its own MAC address as the current (temporary) owner of that address."

This may appear confusing to network administrators who are not expecting the behaviour of changing MAC addresses.

Wireless magic packet

In case the low-power-mode device is communicating via Wi-Fi, the wake-up-packet is sent via Wireless Multimedia Extensions (WMM). This was not possible in previous implementations of Wake on LAN. The wireless hardware must be updated enough to include WMM support. Apple provides instructions for checking compatibility with this feature for Macintosh computers.

Supported services and examples

The sleep proxy service is able to advertise any Bonjour-supported services, while the host computer sleeps. Some examples of supported services are:

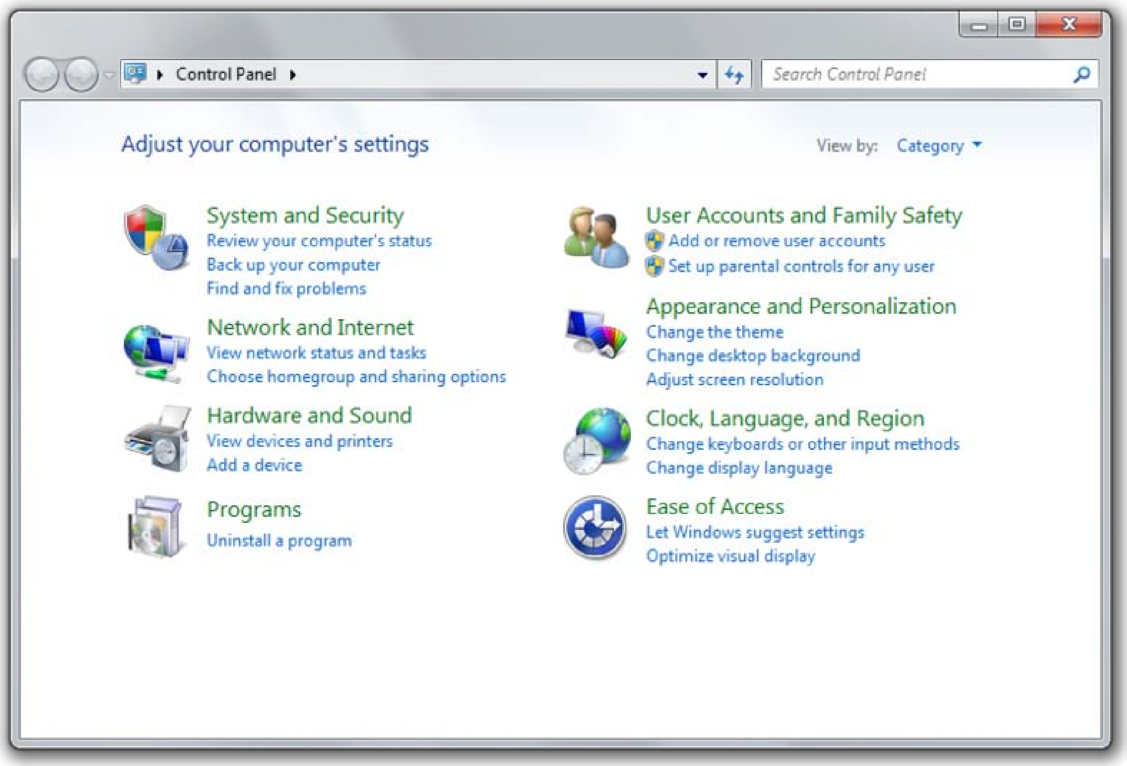
- File sharing: a host supporting the sleep proxy service, which offers file services, may go to sleep as needed. When someone needs to access shared files, the host will wake up automatically.
- iTunes library sharing: the computer hosting the iTunes library may go to sleep, and will automatically wake when someone wishes to browse the iTunes library from a different PC.
- Printer sharing: a printer may be connected and shared from a computer supporting sleep proxy service. The computer can go to sleep when not in use, but will wake when needed to service a print job being sent from a different computer.
- SSH: a computer offering SSH access may go to sleep, and awaken via the sleep proxy service when an SSH login is initiated. On Darwin or Macintosh computers, the host can be put back to sleep using the command line instruction: pmset *sleepnow*.
- Desktop sharing: similar to above examples.

Implementations

Implementations on a local area network can be seen with Bonjour Browser.

- Apple AirPort Express with firmware version 7.4.1 or 7.4.2
- Apple AirPort Extreme with firmware version 7.4.1 or 7.4.2
- Legacy AppleTVs (confirmed for version 3.0.2)
- Apple Time Capsule
- Computers running Mac OS X Snow Leopard act as a Bonjour sleep proxy server when Internet sharing is enabled.

Control Panel (Windows)


Control Panel
Search Control Panel
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Back up your computer
Find and fix problems
Network and Internet
View network status and tasks
Choose homegroup and sharing options
Hardware and Sound
View devices and printers
Add a device
Programs
Uninstall a program
User Accounts and Family Safety
Add or remove user accounts
Set up parental controls for any user
Appearance and Personalization
Change the theme
Change desktop background
Adjust screen resolution
Clock, Language, and Region
Change keyboards or other input methods
Change display language
Ease of Access
Let Windows suggest settings
Optimize visual display

Control Panel under Windows 7

The **Control Panel** is a part of the Microsoft Windows graphical user interface which allows users to view and manipulate basic system settings and controls via applets, such as adding hardware, adding and reoving software, controlling user accounts, and changing accessibility options. Additional applets can be provided by third party software.

The Control Panel has been an inherent part of the Microsoft Windows operating system since its first release (Windows 2.0), with many of the current applets being added in later versions. Beginning with Windows 95, the Control Panel is implemented as a special folder, i.e. the folder does not physically exist, but only contains shortcuts to various applets such as *Add or Remove Programs* and *Internet Options*. Physically, these applets are stored as *.cpl* files. For example, the *Add or Remove Programs* applet is stored under the name *appwiz.cpl* in the *SYSTEM32* folder.

In recent versions of Windows, the Control Panel has two views, *Classic View* and *Category View*, and it is possible to switch between these through an option that appears on the left side of the window.

Many of the individual Control Panel applets can be accessed in other ways. For instance, *Display Properties* can be accessed by right-clicking on an empty area of the desktop and choosing *Properties*.

The classic view consists of shortcuts to the various control panel applets, usually without any description (other than the name). The categories are seen if the user use "Details" view.

The category view consists of categories, which when clicked on display the control panel applets related to the category. In Windows Vista, the category displays links to the most commonly used applets below the name of the category.

Chapter 3

Environment Variable

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They can be said in some sense to create the operating environment in which a process runs. For example, an environment variable with a standard name can store the location that a particular computer system uses to store temporary files—this may vary from one computer system to another. A process which invokes the environment variable by (standard) name can be sure that it is storing temporary information in a directory that exists and is expected to have sufficient space.

Synopsis

In all Unix and Unix-like systems, each process has its own private set of environment variables. By default, when a process is created it inherits a duplicate environment of its parent process, except for explicit changes made by the parent when it creates the child. At API level, these changes must be done between `fork` and `exec`. Alternatively, from shells such as bash, you can change environment variables for a particular command invocation by indirectly invoking it via `env` or using the `ENVIRONMENT_VARIABLE=VALUE <command>` notation. All Unix operating system flavors, MS-DOS, and Microsoft Windows have environment variables; however, they do not all use the same variable names. Running programs can access the values of environment variables for configuration purposes. Examples of environment variables include:

- PATH - lists directories the shell searches, for the commands the user may type without having to provide the full path.
- HOME (Unix-like) and userprofile (Microsoft Windows) - indicate where a user's home directory is located in the file system.
- TERM (Unix-like) - specifies the type of computer terminal or terminal emulator being used (e.g., **vt100** or **dumb**).
- PS1 (Unix-like) - specifies how the prompt is displayed in the Bourne shell and variants.
- MAIL (Unix-like) - used to indicate where a user's mail is to be found.
- TEMP - location where processes can store temporary files

Shell scripts and batch files use environment variables to communicate data and preferences to child processes. They can also be used to store temporary values for reference later in the script, although in Unix other variables are usually used for this.

In Unix, an environment variable that is changed in a script or compiled program will only affect that process and possibly child processes. The parent process and any unrelated processes will not be affected. In MS-DOS changing or removing a variable's value inside a BATCH file will change the variable for the duration of command.com's existence.

In Unix, the environment variables are normally initialized during system startup by the system init scripts, and hence inherited by all other processes in the system. Users can, and often do, augment them in the profile script for the shell they are using. In Microsoft Windows, environment variables defaults are stored in the windows registry or set in autoexec.bat.

Getting and setting environment variables

The variables can be used both in scripts and on the command line. They are usually referenced by putting special symbols in front of or around the variable name. For instance, to display the program search path, in most scripting environments, the user has to type:

```
echo $PATH
```

On DOS or Windows system, the user has to type this:

```
echo %PATH%
```

Unix

The commands **`env`**, **`set`**, and **`printenv`** display all environment variables and their values. `env` and `set` are also used to set environment variables and are often incorporated directly into the shell. `printenv` can also be used to print a single variable by giving that variable name as the sole argument to the command.

In Unix, the following commands can also be used, but are often dependent on a certain shell.

```
export VARIABLE=value  # for Bourne, bash, and related shells
setenv VARIABLE value  # for csh and related shells
```

Working principles of environment variables

A few simple principles govern how environment variables BY INSTALLING, achieve their effect.

Local to process

Environment variables are local to the process in which they were set. That means if we open two terminal windows (Two different processes running shell) and change value of environment variable in one window, that change will not be seen by other window.

Inheritance

When Parent process creates a child process, the child process inherits all the environment variable and their values which parent process had.

Case sensitive

The names of environment variables are case sensitive.

Persistence

Environment variables persistence can be session-wide or system-wide.

DOS and Windows

In DOS and Windows, the **`set`** command without any arguments displays all environment variables along with their values.

To set a variable to a particular value, use:

```
set VARIABLE=value
```

However, this is temporary. Permanent change to the environment variable can be achieved through editing the registry (not recommended for novices) and using the Windows Resource Kit application `setx.exe`. With the introduction of Windows Vista, the `setx` command became part of Windows.

Users of the Windows GUI can manipulate variables via <Control Panel:System:Advanced:Environment Variables>; through the Windows Registry this is done changing the values under HKCU\Environment (for user specific variables) and HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment (for System variables).

To see the current value of a particular variable, use:

```
set VARIABLE
```

Note: Please take note that doing so will print out all variables beginning with 'VARIABLE'. Another example is:

```
C:\> set p
Path=c:\.. ..
PATHEXT=.COM;.EXE;.BAT;
PROCESSOR_ARCHITECTURE=.. ..
PROCESSOR_IDENTIFIER=x8..
PROCESSOR_LEVEL=6..
PROCESSOR_REVISION=1706..
ProgramFiles=C:\Program.. .
PROMPT=$P$G
```

or

```
echo %VARIABLE%
```

To delete a variable, the following command is used:

```
set VARIABLE=
```

Unexported variables

In Unix shells, variables may be assigned without the **`export`** keyword. Variables defined in this way are displayed by the **`set`** command, but are **not** true environment variables, as they are stored only by the shell and not recognized by the kernel. The `printenv` command will not display them, and child processes do not inherit them.

```
VARIABLE=value
```

However, if used in front of a program to run, the variables will be exported to the environment and thus appear as real environment variables to the program:

```
VARIABLE=value program_name [arguments]
```

The tool that gives closest parallel in Windows is the SETLOCAL/ENDLOCAL commands that prevent variables from being set globally.

Security

On Unix, a setuid program is given an environment chosen by its caller, but it runs with different authority from its caller. The dynamic linker will usually load code from locations specified by the environment variables LD_LIBRARY_PATH and LD_PRELOAD and run it with the process's authority. If a setuid program did this, it would be insecure, because its caller could get it to run arbitrary code and hence misuse its authority. For this reason, libc unsets these environment variables at startup in a setuid process. setuid programs usually unset unknown environment variables and check others or set them to reasonable values.

Common environment variables

Examples of Unix environment variables

$PATH

Contains a colon-separated list of directories that the shell searches for commands that do not contain a slash in their name (commands with slashes are interpreted as file names to execute, and the shell attempts to execute the files directly).

$HOME

Contains the location of the user's home directory. Although the current user's home directory can also be found out through the C functions `getpwuid` and `getuid`, `$HOME` is often used for convenience in various shell scripts (and other contexts). Using the environment variable also gives the user the possibility to point to another directory.

$PWD

This variable points to the current directory. Equivalent to the output of the command pwd when called without arguments.

$DISPLAY

Contains the identifier for the display that X11 programs should use by default.

$LD_LIBRARY_PATH

On many Unix systems with a dynamic linker, contains a colon-separated list of directories that the dynamic linker should search for shared objects when building a process image after `exec`, before searching in any other directories.

$LANG, $LC_ALL, $LC_...

LANG is used to set to the default locale. For example, if the locale values are `pt_BR`, then the language is set to (Brazilian) Portuguese and Brazilian practice is used where relevant. Different aspects of localization are controlled by individual LC_-variables (LC_CTYPE, LC_COLLATE, LC_DATE etc.). LC_ALL can be used to force the same locale for all aspects.

$TZ

Refers to Time zone. It can be in several formats, either specifying the timezone itself or referencing a file (in /usr/share/zoneinfo).

Examples of DOS environment variables

%COMSPEC%

This variable contains the full path to the command processor, command.com.

%PATH%

This variable contains a semicolon-delimited list of directories in which the command interpreter will search for executable files. Equivalent to the Unix $PATH variable (although note that PATH on Windows additionally performs the same task as LD_LIBRARY_PATH on Unix-like systems). Note that %PATH% can also be set like this PATH=c:\dos; where SET isn't required.

%TEMP% and %TMP%

These variables contain the path to the directory where temporary files should be stored.

Examples from Microsoft Windows

Discrete value variables

These variables generally expand to discrete values, such as the current working directory, the current date, or a random number. Some of these are true environment variables and will be expanded by all functions that handle environment variables. Others, like %CD% simply look like environment variables and will only be expanded by some functions and shells. They are not case sensitive.

%CD%

This variable points to the current directory. Equivalent to the output of the command `cd` when called without arguments.

%DATE%

This variable expands to the current date. The date is displayed according to the current user's date format preferences.

The following is a way of reformatting the date and time for use in file copies. The example assumes UK format of day month year and the time is set for a 24 hour clock.

```
@echo off
echo %DATE% %TIME%
set MTH=%DATE:~4,2%
set DAY=%DATE:~7,2%
set YR=%DATE:~10,4%
set HR=%TIME:~0,2%
set HR0=%TIME:~0,1%
if "%HR0%"==" " set HR=0%TIME:~1,1%
set MIN=%TIME:~3,2%
set SEC=%TIME:~6,2%
set MYDATE=%YR%%MTH%%DAY%-%HR%%MIN%%SEC%
```

```
echo %MYDATE%
```

%ERRORLEVEL%

This variable points to the current error level. If there was an error in the previous command, this is what you need to check against to find out about that.

%RANDOM%

This variable returns a random number between 0 and 32767

%TIME%

This variable points to the current time. The time is displayed according to the current user's time format preferences.

System path variables

These variables refer to locations of critical operating system resources, and as such generally are not user-dependent.

%AppData%

Contains the full path to the Application Data folder of the logged-in user. Does not work on Windows NT 4.0 SP6 UK.

%ComSpec%

This variable contains the full path to the command processor; on Windows NT based operating systems this is `cmd.exe`, while on Windows 9x and ME it is the DOS command processor, `COMMAND.COM`.

%LOCALAPPDATA%

This variable is the temporary files of Applications. Its uses include storing of Desktop Themes, Windows Error Reporting, Caching and profiles of web browsers.

%PATH%

This variable contains a semicolon-delimited (do not put spaces in between) list of directories in which the command interpreter will search for an executable file that matches the given command. Equivalent to the Unix `$PATH` variable.

%ProgramFiles%

This variable points to Program Files directory, which stores all the installed program of Windows and others. The default on English-language systems is `C:\Program Files`. In

64-bit editions of Windows (XP, 2003, Vista), there are also **%ProgramFiles(x86)%** which defaults to `C:\Program Files (x86)` and **%ProgramW6432%** which defaults to `C:\Program Files`. The %ProgramFiles% itself depends on whether the process requesting the environment variable is itself 32-bit or 64-bit (this is caused by Windows-on-Windows 64-bit redirection).

%CommonProgramFiles%

This variable points to Common Files directory. The default is `C:\Program Files\Common Files`.

%SystemDrive%

The %SystemDrive% variable is a special system-wide environment variable found on Microsoft Windows NT and its derivatives. Its value is the drive upon which the system folder was placed.

The value of %SystemDrive% is in most cases `C:`.

%SystemRoot%

The %SystemRoot% variable is a special system-wide environment variable found on Microsoft Windows NT and its derivatives. Its value is the location of the system folder, including the drive and path.

The drive is the same as %SystemDrive% and the default path on a clean installation depends upon the version of the operating system. By default, on a clean installation:

- Windows NT 5.1 (Windows XP) and newer versions use `\WINDOWS`
- Windows NT 5.0 (Windows 2000), Windows NT 4.0 and Windows NT 3.1 use `\WINNT`
- Windows NT 3.5x uses `\WINNT35`

%WinDir%

This variable points to the Windows directory (on Windows NT-based operating systems it is identical to the %SystemRoot% variable, above). If the System is on drive C: then the default values are:

- `C:\WINDOWS` on Windows 95, Windows 98, Windows Me, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008 and Windows 7
- `C:\WINNT` for Windows NT 4, and Windows 2000

Note that Windows NT 4 Terminal Server Edition by default installs to `C:\WTSRV`.

User management variables

These variables store information related to resources and settings owned by various user profiles within the system. As a general rule, these variables do not refer to critical system resources or locations that are necessary for the OS to run.

%AllUsersProfile% (%PROGRAMDATA% for Windows Vista, Windows 7)

The %AllUsersProfile%(%PROGRAMDATA%) variable expands to the full path to the All Users profile directory. This profile contains resources and settings that are used by all system accounts. Shortcut links copied to the All Users' Start menu or Desktop folders will appear in every user's Start menu or Desktop, respectively.

%UserDomain%

The variable holds the name of the Workgroup or Windows Domain to which the current user belongs. The related variable, %LOGONSERVER%, holds the hostname of the server that authenticated the current user's logon credentials (name and password). For Home PCs, and PCs in a Workgroup, the authenticating server is usually the PC itself. For PCs in a Windows Domain, the authenticating server is a domain controller (a primary domain controller, or PDC, in Windows NT 4-based domains).

%UserProfile%

The %UserProfile% variable is a special system-wide environment variable found on Microsoft Windows NT and its derivatives. Its value is the location of the current user's profile directory, in which is found that user's HKCU registry hive (`NTUSER`).

Users can also use the **%USERNAME%** variable to determine the active users login identification.

Windows GUI forced variable expansion

In certain cases it is not possible to create file paths containing environment variables using the Windows GUI, and it is necessary to fight with the user interface to make things work as intended.

- In Windows 7, a shortcut may not contain the variable %USERNAME% in unexpanded form. Trying to create shortcut to *\\server\share\accounts\%USERNAME%* or *C:\users\%USERNAME%* will be silently changed to replace %USERNAME% with the account name of the currently logged-in user, when the OK button is pressed on the shortcut properties.
    - This can only be overridden if the %USERNAME% variable is part of a parameter to some other program in the shortcut. For example, *%SYSTEMROOT%\Explorer.exe C:\Users\%USERNAME%* is not

expanded when OK is clicked, but this shortcut is treated as unsafe and displays a warning when opened.

- In Group Policy Management on Server 2008 R2, a profile folder can not be redirected a custom folder hierarchy. For example, the desktop can not be redirected to *\\server\share\accounts\%USERNAME%\custom\path\desktop*. Upon pressing OK, this is silently changed to "Create a folder for each user in the root path" with the path *\\server\share\accounts\* pointing to "\username\desktop".
  - This behavior can only be overridden if the path contains a variable or drive letter that is not currently resolvable at the time of editing the GPO. For example if a mapping for drive O: does not exist on the server, then the path *O:\folder\%username%\CustomTarget* is not expanded when OK is clicked.

- A domain user account may not contain a profile path or home folder path containing an unexpanded %USERNAME% variable. Upon clicking OK, this is silently replaced with the user's account name.
  - This causes problems for new user creation that is performed by copying an existing user account, if there are additional folders listed after the username in the path. For a pre-existing account with a profile path of *\server\share\accounts\DomainUser\profile* the Microsoft Management Console doesn't know which part of the path contains the previous user's name and doesn't change the path during the copy, resulting in the new account pointing to the other account's profile/home paths. The profile/home paths must be manually re-edited to point to the correct location.

Default Values on Microsoft Windows

Variable	Windows XP	Windows Vista/7
%ALLUSERSPROFILE%	C:\Documents and Settings\All Users	C:\Users\Public\Desktop
%APPDATA%	C:\Documents and Settings\{username}\Application Data	C:\Users\{username}\AppData\Roaming
%COMPUTERNAME%	{computername}	{computername}
%COMMONPROGRAMFILES%	C:\Program Files\Common Files	C:\Program Files\Common Files
%COMMONPROGRAMFILES(x86)%	C:\Program Files (x86)\Common Files	C:\Program Files (x86)\Common Files
%COMSPEC%	C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe
%HOMEDRIVE%	C:	C:
%HOMEPATH%	\Documents and Settings\{username}	\Users\{username}

%LOCALAPPDATA%		C:\Users\{username}\AppData\Local
%LOGONSERVER%	\\{domain_logon_server}	\\{domain_logon_server}
%PATH%	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;{plus program paths}	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;{plus program paths}
%PATHEXT%	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.WSF;.WSH	.com;.exe;.bat;.cmd;.vbs;.vbe;.js;.jse;.wsf;.wsh;.msc
%PROGRAMFILES%	%SystemDrive%\Program Files	%SystemDrive%\Program Files
%PROGRAMFILES(X86)%	%SystemDrive%\Program Files (x86) (only in 64-bit version)	%SystemDrive%\Program Files (x86) (only in 64-bit version)
%PROMPT%	Code for current command prompt format. Code is usually PG	Code for current command prompt format. Code is usually PG
%SystemDrive%	C:	C:
%SystemRoot%	The Windows directory, usually C:\Windows, formerly C:\WINNT	%SystemDrive%\Windows
%TEMP% and %TMP%	%SystemDrive%\Documents and Settings\{username}\Local Settings\Temp	%SystemDrive%\Users\{username}\AppData\Local\Temp
%USERDOMAIN%	{userdomain}	{userdomain}
%USERNAME%	{username}	{username}
%USERPROFILE%	%SystemDrive%\Documents and Settings\{username}	%SystemDrive%\Users\{username}
%WINDIR%	C:\Windows	C:\Windows
%PUBLIC%		%SystemDrive%\Users\Public
%PROGRAMDATA%		%SystemDrive%\ProgramData
%PSModulePath%		%SystemRoot%\system32\WindowsPowerShell\v1.0\Modules\

Chapter 4

Link-Local Address and System Preferences

Link-local address

A **link-local address** is an IP address that is intended only for communications within the local subnetwork. Routers do not forward packets with link-local addresses.

Link-local addresses are assigned using **stateless address autoconfiguration** procedures for Internet Protocol Version 4 (IPv4) and IPv6. On IPv4, link-local address may be used when no external, stateful mechanism of address configuration, such as the Dynamic Host Configuration Protocol (DHCP), exists or another primary configuration method has failed. On IPv6, link-local addresses are required for the internal functioning of various protocol components.

Link-local addresses for IPv4 are defined in the address block `169.254.0.0/16`. In IPv6, they are allocated with the `fe80::/10` prefix.

IPv4

The Internet Engineering Task Force has reserved the address block `169.254.1.0` through `169.254.254.255` for link-local addressing in Internet Protocol Version 4. They are assigned to interfaces by host-internal, i.e. stateless, address autoconfiguration when other means of address assignment are not available.

Because it is undesirable to have multiple addresses assigned to an interface, before automatically assigning an IP address, hosts generally search for a DHCP server on the network.

In the automatic address configuration process, network hosts select a random candidate address within the reserved range and use Address Resolution Protocol (ARP) probes to ascertain that the address is not in use by another host. Otherwise, a new address is selected. When a globally routable or a private address become available after a link-local address has been assigned, the use of the new address must be preferred to the link-local address for new connections.

Microsoft refers to this address autoconfiguration method as **Automatic Private IP Addressing** (**APIPA**). It is sometimes also casually referred to as **auto-IP**.

IPv6

Internet Protocol Version 6 (IPv6) requires operating systems to assign link-local addresses to network interfaces even when routable addresses are also assigned. A link-local unicast address has the prefix `fe80::/10` in standard IPv6 CIDR notation.

IPv6 hosts usually have more than one IPv6 address assigned to each of their network interfaces. The link-local address is required for IPv6 sublayer operations within the Neighbor Discovery Protocol. Link-local addresses may be assigned by automatic (stateless) or stateful (DHCPv6) mechanisms.

In IPv6 stateless address autoconfiguration is performed as a component of the Neighbor Discovery Protocol (NDP), as specified in RFC 4862. The address is formed from its routing prefix and the interface's MAC address.

Link-local IPv6 address assignment automatically implies the on-link presence for this routing prefix, unlike for the addresses of other scopes.

IPv6 introduced additional means of assigning addresses to host interfaces. Through NDP routing prefix advertisements a router or a dedicated server host may announce configuration information to all link-attached interfaces which causes additional IP address assignment on the receiving interfaces for local or global routing purposes. This process is sometimes also considered stateless, as the prefix server does not receive or log any individual assignments to hosts. Uniqueness is guaranteed automatically by the MAC-address based methodology and duplicate address detection algorithms.

System Preferences

System Preferences is an application included with the Mac OS X operating system that allows users to modify various system settings which are divided into separate preference panes. The System Preferences application was introduced in the first version of Mac OS X to replace the control panel that was included in previous versions of the Mac operating system.

Overview

History

Before the release of Mac OS X in 2001, users modified system settings using control panels. Control panels, unlike the preference panes found in System Preferences, were separate applications that were accessed through the Apple menu. Mac OS 9, the last release of the Mac OS before Mac OS X, included 32 control panels.

Organization

When Mac OS X was released, preference panes replaced control panels. Preference panes are not applications but subsections of the System Preferences application. By default, System Preferences organizes preference panes into several categories. In the latest version of System Preferences, included with Mac OS X v10.6, these categories are "Personal", "Hardware", "Internet & Wireless", and "System". A fifth category, "Other", appears when third-party preference panes are installed. Users can also choose to sort preference panes alphabetically. Originally, System Preferences included a customizable toolbar into which frequently-used preference pane icons could be dragged. This was removed in Mac OS X v10.4 and replaced with a static toolbar that featured back and forward navigation buttons and a search field.

Apple has added new preference panes when major features are added to the operating system and occasionally merges multiple panes into one. When Exposé was introduced with Mac OS X v10.3, a corresponding preference pane was added to System Preferences. This was replaced by a single "Dashboard & Exposé" pane in Mac OS X v10.4, which introduced Dashboard. When the .Mac service was replaced by MobileMe, the corresponding preference pane was also renamed.

Included preference panes

Mac OS X v10.5 "Leopard" includes the following preference panes:

Option	Description
Accounts	control user creation/deletion, administrator privileges and user limitations.
Appearance	changes the general color scheme of the OS (Aqua or Graphite), as well as placement of scroll arrows and font smoothing.
Bluetooth	pair Bluetooth devices and edit Bluetooth settings.
CDs & DVDs	used to set default settings upon inserting blank CD/DVDs, as well as music CDs, picture CDs and video DVDs.
Date & Time	used to set the date and time of the computer, as well as how the clock appears on the menu bar.

Desktop & Screensaver	used to set the desktop picture as well as the screensaver, and their settings.
Displays	used to set screen resolution and color settings.
Dock	adjust the dock size as well as magnification and position on screen.
Energy Saver	optimize energy settings as well as set sleep times and processor usage.
Exposé and Spaces	set Active Screen Corners and keyboard and mouse settings to activate Exposé, and set application settings for Spaces.
Ink	set handwriting recognition settings (only appears when a graphics tablet is connected).
International	set the default OS language as well as numerical, measurement, currency, date, and time formats.
Keyboard & Mouse	set keyboard settings, as well as change keyboard shortcuts, and mouse settings.
MobileMe	used to set preferences for the user's MobileMe account and iDisk.
Network	set Ethernet, AirPort, Modem and VPN Settings.
Parental Controls	manage parental controls for accounts, and view account usage data.
Print & Fax	set the default printer as well as fax settings.
QuickTime	set network speeds, plug-in settings, update plug-ins, and register Quicktime Pro.
Security	set "FileVault" and account security settings, and setup the firewall.
Sharing	set the computer name, and sharing and remote management services.
Software Update	set default times to check for updates, and view updates already installed.
Sound	set alert sound, volume and input/output options.
Speech	set the computer's default voice, set up speech recognition, and other speech settings.
Startup Disk	set the default disk, for the computer to boot into.
Time Machine	set the Time Machine drive and backup options.
Trackpad	(only appears on Apple notebooks) adjust tracking, clicking, and scrolling speed. Also allows users to adjust multi-touch gestures on newer MacBooks
Universal Access	make the system more accessible for those with sight, hearing and other impairments.

Chapter 5

Udev and Zero Configuration Networking

udev

udev is the device manager for the Linux kernel. Primarily, it manages device nodes in `/dev`. It is the successor of devfs and hotplug, which means that it handles the `/dev` directory and all user space actions when adding/removing devices, including firmware load.

History

udev was new in Linux 2.5.

The Linux kernel version 2.6.13 introduced or updated a new version of the uevent interface. A system using a new version of udev will not boot with kernels older than 2.6.13 unless udev is disabled and a traditional `/dev` directory is used for device access.

Overview

Unlike traditional Unix systems, where the device nodes in the `/dev` directory have been a static set of files, the Linux udev device manager dynamically provides only the nodes for the devices actually present on a system. Although devfs used to provide similar functionality, advocates of udev cited a number of reasons for preferring its implementation over devfs:

- udev supports persistent device naming, which does not depend on, for example, the order in which the devices are plugged into the system. The default udev setup provides persistent names for storage devices. Any hard disk is recognized by its unique filesystem id, the name of the disk and the physical location on the hardware it is connected to.
- udev executes entirely in user space, as opposed to devfs' kernel space. One consequence is that udev moved the naming policy out of the kernel and can run

arbitrary programs to compose a name for the device from the device's properties, before the node is created.

Operation

udev is a generic kernel device manager. It runs as a daemon on a Linux system and listens to uevents the kernel sends out (via netlink socket) if a new device is initialized or a device is removed from the system. The system provides a set of rules that match against exported values of the event and properties of the discovered device. A matching rule will possibly name and create a device node and run configured programs to set-up and configure the device.

udev rules can match on properties like the kernel subsystem, the kernel device name, the physical location of the device, or properties like the device's serial number. Rules can also request information from external programs to name a device or specify a custom name that will always be the same, regardless of the order devices are discovered by the system.

A common way to use udev on Linux systems is to let it send events through a socket to HAL or DeviceKit, which will perform further device-specific actions. For example, HAL/DeviceKit will notify other software running on the system that the new hardware has arrived by issuing a broadcast message on the D-Bus IPC system to all interested processes. In this way, desktops such as GNOME or KDE can open a file browser to newly attached USB flash drives and SD cards.

Architecture

The system is divided into three parts:

- The library *libudev*, that allows access to device information.
- The daemon *udevd*, in user space, that manages the virtual `/dev`.
- The administrative command *udevadm* for diagnostics.

The system gets calls from the kernel via netlink socket. Earlier versions used hotplug, adding a link to themselves in `/etc/hotplug.d/default` with this purpose.

Authors

udev was developed by Greg Kroah-Hartman and Kay Sievers, with much help from Dan Stekloff, among others.

Zero configuration networking

Zero configuration networking (**zeroconf**), is a set of techniques that automatically creates a usable Internet Protocol (IP) network without manual operator intervention or special configuration servers.

Zero configuration networking allows inexpert users to connect computers, networked printers, and other network devices and expect a functioning network to be established automatically. Without zeroconf, a user must either set up special services, like Dynamic Host Configuration Protocol (DHCP) and Domain Name System services (DNS), or set up each computer's network settings manually, which may be difficult for non-technical or novice users.

Zeroconf is built on three core technologies:

- Assignment of numeric network addresses for networked devices (link-local address autoconfiguration)
- Automatic resolution and distribution of computer hostnames (multicast DNS)
- Automatic location of network services, such as printing devices through DNS service discovery.

Address selection

Both IPv4 and IPv6 have standard methods for address autoconfiguration. For link-local addressing IPv4 uses the special block 169.254.0.0/16 as described in RFC 3927 while IPv6 hosts use the prefix fe80::/10.

Most IPv4 hosts use link-local addressing (IPv4LL) only as a last resort when a DHCP server is unavailable. An IPv4 host otherwise uses its DHCP-assigned address for all communications, global or link-local. One reason is that IPv4 hosts are not required to support multiple addresses per interface, although many do. Another is that not every IPv4 host implements distributed name resolution (e.g., multicast DNS), so discovering the autoconfigured link-local address of another host on the network can be difficult. However, discovering the DHCP-assigned address of another host also requires either distributed name resolution or a unicast DNS server with this information, and some networks feature DNS servers that are automatically updated with DHCP-assigned host and address information.

Because IPv6 hosts are required to support multiple addresses per interface, nearly every IPv6 host configures a link-local address even when global addresses are available. IPv6 hosts may additionally self-configure one or more global addresses on receipt of one or more router advertisement messages, thus eliminating the need for a DHCP6 server.

Both IPv4 and IPv6 hosts may randomly generate the host-specific part of an autoconfigured address. IPv6 hosts generally combine a prefix of up to 64 bits with a 64-

bit EUI-64 derived from the factory-assigned 48-bit IEEE MAC address. The MAC address have the advantage of being globally unique, a property inherited by the EUI-64. The host is normally required to ensure, through broadcast queries, that the addresses it generates are not in use by any other host on the local network.

The technique is called *Link-Local* address assignment in RFC 3927. However, Microsoft refers to this as *Automatic Private IP Addressing* (APIPA) or *Internet Protocol Automatic Configuration* (IPAC) (supported since at least Windows 98).

Name resolution

In 2000, Bill Manning and Bill Woodcock described the *Multicast Domain Name Service* which spawned the implementations by Apple and Microsoft. Both implementations are very similar. Apple's Multicast DNS (mDNS) is an open specification, while Microsoft's Link-local Multicast Name Resolution (LLMNR) is little used and the specification is not an IETF standards track publication. The latter was published as informational RFC 4795.

The two protocols have minor differences in their approach to name resolution. mDNS allows a network device to choose a domain name in the `local` namespace and announce it using a special multicast IP address. This introduces special semantics for the `local` domain, which is considered a problem by some members of the IETF. The current LLMNR draft allows a network device to choose any domain name, which is considered a security risk by some members of the IETF. mDNS is compatible with DNS-SD as described in the next section, while LLMNR is not.

Service discovery

Apple's protocol: Multicast DNS/DNS-SD

Multicast DNS (mDNS) is a protocol that uses APIs similar to unicast Domain Name System but implemented over a multicast protocol. Each computer on the LAN stores its own list of DNS resource records (e.g., A, MX, SRV) and joins the mDNS multicast group. When an mDNS client wants to know the IP address of a PC given its name, mDNS client sends a request to a well-known multicast address; the PC with the corresponding A record replies with its IP address. The mDNS multicast address is 224.0.0.251 for IPv4 and ff02::fb for IPv6 link-local addressing.

DNS based Service Discovery (DNS-SD) is the other half of Apple's solution, built on top of the Domain Name System. It is used in Apple products, many network printers and a number of third party products and applications on various operating systems. The Apple solution uses DNS messages, in contrast to Microsoft's competing technology, SSDP, which uses HTTP messages. It uses DNS SRV, TXT, and PTR records to advertise Service Instance Names. The hosts offering services publish details of available services: instance, service type, domain name and optional configuration parameters.

Service types are given informally on a first-come basis. A service type registry is maintained and published by DNS-SD.org.

Many Mac OS X networking clients, such as the Safari browser and the iChat instant messaging software, use DNS-SD to locate nearby servers. On Windows, some instant messaging and VoIP clients, for example Gizmo5, support DNS-SD. Some Linux distributions also include DNS-SD functionality.

mDNS/DNS-SD was developed by Apple Computer employee Stuart Cheshire in the company's move from AppleTalk to IP.

Microsoft's protocol: UPnP SSDP

Simple Service Discovery Protocol (SSDP) is a UPnP protocol, used in Windows XP and several brands of network equipment. SSDP uses HTTP notification announcements that give a service-type URI and a Unique Service Name (USN). Service types are regulated by the Universal Plug and Play Steering Committee.

SSDP is supported in many SOHO firewall appliances, where host computers behind it may pierce holes for applications. It is also used in media center systems, where media exchange between host computers and the media center is facilitated using SSDP.

Efforts toward an IETF standard protocol

Service Location Protocol (SLP), the only protocol for service discovery to have reached the IETF Proposed Standard status, is supported by Hewlett-Packard's network printers, Novell, and Sun Microsystems, but ignored by some other large vendors. SLP is described in RFC 2608 and RFC 3224 and implementations are available for both Solaris and Linux.

Standardization

RFC 3927, a standard for choosing addresses for networked items, was published in March 2005 by the Zeroconf IETF working group, which included individuals from Apple, Sun, and Microsoft.

LLMNR was submitted for official adoption in the DNSEXT IETF working group, however failed to gain consensus and thus has been published as informational RFC only: RFC 4795. Following the failure of LLMNR to become an Internet standard Apple was asked by the IETF to submit the mDNS/DNS-SD specs for publishing as informational RFC as well, given that mDNS/DNS-SD is used much more widely than LLMNR. They are currently published as an Internet Draft.

RFC 2608, the SLP standard for figuring out where to get services, was published by the SVRLOC IETF working group.

As of December 2009, the IETF is actively planning to publish the Apple Multicast DNS specification (draft-cheshire-multicastdns-08.txt) as an RFC. That document has been placed into *Last Call* by the IESG, which is the IETF's management committee. IETF approval of the mDNS document is likely to happen by end of December 2009. It is not yet clear whether this will be an Informational RFC or a standards-track RFC.

Security Issues

Because mDNS operates under a different trust model than unicast DNS—trusting the entire network rather than a designated DNS server—it is vulnerable to spoofing attacks by any system within the multicast IP range. Like SNMP and many other network management protocols, it can also be used by attackers to quickly gain detailed knowledge of the network and its machines.

Major implementations

Apple Bonjour

Bonjour (formerly known as Rendezvous) from Apple Inc., uses multicast DNS and DNS Service Discovery. Apple changed its preferred zeroconf technology from SLP to mDNS and DNS-SD between Mac OS X 10.1 and 10.2, though SLP continues to be supported by Mac OS X.

Apple's mDNSResponder has interfaces for C and Java and is available on BSD, Mac OS X, Linux, other POSIX based operating systems and Windows. The Windows downloads are available from Apple's website.

Avahi

Avahi is a Zeroconf implementation for Linux and BSDs. It implements IPv4LL, mDNS and DNS-SD. It is part of most Linux distributions, and is installed by default on some. If run in conjunction with nss-mdns it also offers host name resolution.

Avahi also implements binary compatibility libraries that emulate Bonjour and the historical mDNS implementation Howl, so software made to use those implementations can also utilize Avahi through the emulation interfaces.

Windows CE 5.0

Windows CE 5.0 includes Microsoft's own implementation of LLMNR.

Link-local IPv4 addresses

There are some implementations available:

- Windows and Mac OS have both supported link-local addresses since 1998. Apple released its open-source implementation in the Darwin bootp package.
- Avahi contains an implementation of IPv4LL in the avahi-autoipd tool.
- zcip (Zero-Conf IP)
- BusyBox can embed a simple IPv4LL implementation
- Stablebox, a fork from Busybox, offers a slightly modified IPv4LL implementation named llad.
- zeroconf, a package based on Simple IPv4LL, a shorter implementation by Arthur van Hoff.

The above implementations are all stand-alone daemons or plugins for DHCP clients that only deal with link-local IP addresses. Another approach is to include support in new or existing DHCP clients:

- Elvis Pfützenreuter has written a patch for the uDHCP client/server.
- dhcpcd is an opensource DHCP client for Linux and BSD that includes IPv4LL support. It is included as standard in NetBSD.

Neither of these implementations addresses kernel issues like the broadcasting of ARP replies or closing of existing network connections.

Chapter 6

Make Compatible

Make Compatible is a program developed by Microsoft that is included with Windows 9x operating systems. It changes per-program system settings in Windows to allow Windows 3.1 programs that are tailored specifically to that platform to execute under newer versions. The name of the program image file for Make Compatible is `mkcompat.exe`, and it is stored in the `\Windows\System` directory.

When it is invoked, one can choose the name of the Windows 3.1 application program image file using the "Choose Program" option on the "File" menu. After the program image file is chosen, Make Compatible by default displays a list of five options that can be set to alter the behaviour of Windows for that program when it is executed:

- Don't spool to enhanced meta files
- Give application more stack space
- Lie about printer device mode size
- Lie about Windows version number
- Windows 3.1-style controls

An advanced options mode, selectable via the "Advanced Options" selection on the "File" menu presents a longer list of options, allowing finer control of Windows 3.1 emulation if the particular application requires it.

Each of the options is recorded in a system database of so-called "compatibility bits". This is a database of 1-bit flags, one for each of the options displayed by Make Compatible.

This database already existed in earlier versions of Windows. In Windows 3.1, the database is stored in the `[Compatibility]` section of `win.ini`, with entries such as:

```
[Compatibility]
ACAD=0x8000
AMIPRO=0x04000010
```

Each line names an application program, and gives a hexadecimal numeric constant to associate with that program. The hexadecimal numeric constant encodes the compatibility bitflags for that particular application, that Windows applies when the application is executed. Make Compatible merely provides a graphical user interface for editing these flags in an easy way, rather than editing `win.ini` manually, with a text editor. It allows one to set and unset individual flags without having to know their numeric values.

The compatibility bitflags settable in `win.ini` are not documented in the `WININI.WRI` file that ships with Windows 3.1, or in the *Microsoft Windows 3.1 Resource Kit* published by Microsoft. They are listed as a simple set of defined constants (with names beginning "GACF_" for "GetAppCompatFlags"), without explanation, in the `windows.h` header file that is shipped with the Microsoft Windows 3.1 Device driver Development Kit. In Windows 3.1, the compatibility flags that are in effect for any given task in the kernel's Task Database are readable via the undocumented `GetAppCompatFlags()` function that is exported from the `KERNEL` module.

The flags are documented in Microsoft KnowledgeBase article #82860. They correspond to the "advanced mode" flags that are settable by Make Compatible's "Advanced Options" menu:

30 average width metrics

> This is bit #19 of the compatibility bits word, with hexadecimal value `0x80000`, known by the symbolic name `GACF_30AVGWIDTH` in `windows.h`. This flag causes Windows to re-scale all fonts by a factor of 7 / 8 when calculating their average character widths. The reason for this is that one particular Windows 3.0 application, TurboTax, hard-coded the values that it was using for such size calculations, which failed to work correctly with the new TrueType-compatible font average width calculation method employed by Windows 3.1. This prevented people from using TurboTax to print their income tax return forms on PostScript printers.

Always send NC_Paint

> This is bit #6 of the compatibility bits word, with hexadecimal value `0x40`, known by the symbolic name `GACF_ALWAYSSENDNCPAINT` in `windows.h`. This flag forces any call to `SetWindowPos()` to cause the sending of `WM_NCPAINT` message to all child windows. This is the Windows 3.0 behaviour. In Windows 3.1, the behaviour was changed so that the window message is only sent to those windows that need their non-client areas to be repainted. Some Windows 3.0 applications, however, relied upon always receiving this message, to determine whether child windows needed repositioning.

Don't enum device fonts

> This is bit #14 of the compatibility bits word, with hexadecimal value `0x4000`, known by the symbolic name `GACF_ENUMTTNOTEDEVICE` in `windows.h`. This flag causes Windows 3.1 to turn the `DEVICE_FONTTYPE` flag off in particular circumstances when an application that is wanting to print enumerates fonts. The particular circumstances are when the target printer is not either a dot matrix

printer or a PostScript printer, and the fonts are TrueType fonts that are not resident in the printer itself. In such circumstances, some applications (including PageMaker and MGXDraw) misinterpret the flag and believe the font to be device-resident. (TrueType fonts can be uploaded by Windows to printers, and need not be resident on the device itself to be usable.) They then query the printer to see what sizes of the font it supports, and when that fails (because the printer doesn't know about the font until it is uploaded), they incorrectly assume that the font cannot be resized. Setting the `DEVICE_FONTTYPE` flag to false in such situations prevents the applications from going wrong, and that is what this compatibility bitflag does.

Don't send calcsize `WM_MOVE`

This is bit #17 of the compatibility bits word, with hexadecimal value `0x20000`, known by the symbolic name `GACF_NCCALCSIZEONMOVE` in `windows.h`. This flag forces the `WM_NCCALCSIZE` message to be sent to a window that is being moved or resized. This is the Windows 3.0 behaviour. In Windows 3.1, the behaviour was changed so that the window message is only sent to those windows that were being resized. It was not sent if the window was merely moved. Some Windows 3.0 applications, however, such as Lotus Notes for example, relied upon always receiving this message.

Enum Helv and Times Roman fonts

This is bit #12 of the compatibility bits word, with hexadecimal value `0x1000`, known by the symbolic name `GACF_ENUMHELVTMSRMN` in `windows.h`. This flag works around a problem with applications that refused to work properly unless fonts with the exact names "Helv" and "Tms Rmn" were listed as present on the system by the Windows font enumeration API. The names are trademarks of Linotype company for particular fonts, Helvetica and Times Roman, and since Microsoft didn't ship those Linotype fonts with Windows 3.1, it could not enumerate them as present. This flag causes Windows to enumerate the "MS Sans Serif" and "MS Serif" fonts under the names "Helv" and "Tms Rmn", for compatibility with the applications that don't work without those exact names being used.

Force extra windows words

This is bit #8 of the compatibility bits word, with hexadecimal value `0x100`, known by the symbolic name `GACF_MOREEXTRAWNDWORDS` in `windows.h`. This flag works around a problem with applications that assumed the existence of window words (extra items of data associated by Windows with GUI windows) when they had not in fact informed Windows that extra window words were required. Windows 3.1, unlike Windows 3.0, bounds checks all uses of extra window words, and applications that assumed that they could use more space than they had asked for would thus fail. For applications run with this flag set, Windows 3.1 silently increased the total number of words requested by the application by an extra 4 bytes.

Force printer text to new band

This is bit #1 of the compatibility bits word, with hexadecimal value `0x2`, known by the symbolic name `GACF_FORCETEXTBAND` in `windows.h`. This flag causes Windows 3.1 to always use two bands when printing, the first for graphics and the second for text. Normally Windows 3.1 tries to print both in a single band. But applications such as WordPerfect assumed that a second band would always exist, and would always be where the text was, as had been the case in Windows 3.0. Freelance Graphics had a similar problem.

Force TT fonts to graphics band

This is bit #15 of the compatibility bits word, with hexadecimal value `0x8000`, known by the symbolic name `GACF_FORCETTGRAPHICS` in `windows.h`. This flag was to work around a problem with Freelance Graphics, where it wouldn't print using TrueType fonts unless they were printed as graphics.

Global hooks only called for Win16 apps

Ignore discardable segment attributes

This is bit #0 of the compatibility bits word, with hexadecimal value `0x1`, known by the symbolic name `GACF_IGNORENODISCARD` in `windows.h`. This flag forces the `GEM_NODISCARD` flag passed to `GlobalAlloc()` by a program to be ignored. It worked around a bug in the run-time library supplied with Microsoft's own C compiler, Microsoft C version 6. The run-time library would erroneously set that flag in calls to `GlobalAlloc()`, and any application compiled with that compiler would thus exhibit the behaviour.

Ignore raster fonts

This is bit #9 of the compatibility bits word, with hexadecimal value `0x200`, known by the symbolic name `GACF_TTIGNORERASTERDUPE` in `windows.h`. This flag prevents fonts of the same sizes from being enumerated as both bitmap and TrueType fonts. This was because several applications, including WordPerfect and Visual Basic, were not capable of handling that particular situation correctly.

Ignore topmost windows

This is bit #3 of the compatibility bits word, with hexadecimal value `0x8`, known by the symbolic name `GACF_IGNORETOPMOST` in `windows.h`. This flag fixes a particular problem with cc:Mail that caused it to fail on Windows 3.1. It assumed that accessing the first window with `GetWindow(HWND,GW_HWND_FIRST)` would return it the window of the application that it had just started with `WinExec()`. But on Windows 3.1, which introduced the idea of "topmost" windows, this was no longer true. The flag caused Windows 3.1 to skip topmost windows when that particular API request was made.

Module specific hack

No HRGN 1

This is bit #16 of the compatibility bits word, with hexadecimal value `0x10000`, known by the symbolic name `GACF_NOHRGN1` in `windows.h`. This flag reinstates a bug that existed in Windows 3.0 that was fixed in Windows 3.1. It did so because several applications, such as Microsoft Draw, worked around the bug themselves, and would fail to work correctly when the bug was eliminated. The bug was an error in the return value of the `GetUpdateRect()` function, whereby in certain situations (where the entire window was invalid) it would return the coördinates of the update rectangle in window coördinates, rather than in logical coördinates as it was supposed to. In Windows 3.1, it always returned the update rectangle in logical coördinates. The applications that worked around the bug would perform the coördinate transform themselves to work around the bug, and end up updating the wrong parts of their windows on Windows 3.1.

One graphic band and use print escapes

This is bit #2 of the compatibility bits word, with hexadecimal value `0x4`, known by the symbolic name `GACF_ONELANDGRXBAND` in `windows.h`. This flag causes Windows 3.1 to use a single graphics band when printing in Landscape mode, consuming as much memory as needed for that band, and discarding whatever content would not fit into the band.

Subtract clip siblings

This is bit #14 of the compatibility bits word, with hexadecimal value `0x4000`, known by the symbolic name `GACF_SUBTRACTCLIPSIBS` in `windows.h`. This flag causes Windows 3.1 to handle window invalidation differently for top-level windows that do not have the `WS_CLIPSIBLINGS` window style set, and their chlid windows. (In other words: It affects dialogue boxes and the controls on them.) With the flag set, Windows would not invalidate sibling child windows underneath (in the z-order) other `WS_CLIPSIBLINGS` child windows. The main reason for the flag was applications such as Lotus Notes 2.1, which implemented its own combo boxes as child windows, rather than as top-level windows (the system default combo box implementation). With this flag set, odd display problems with such windows would disappear.

Support multiple printing bands

This is bit #5 of the compatibility bits word, with hexadecimal value `0x20`, known by the symbolic name `GACF_MULTIPLEBANDS` in `windows.h`. This flag causes Windows 3.1 to always use multiple bands for printing, even when one band would be sufficient. This was to work around a problem in Freelance Graphics, which would assume that if only one band existed, and it was the entire page, it was the text band, and would not even attempt to print graphics. On Windows 3.1, the universal printer driver would sometimes be able to handle both text and graphics with a single band. By forcing the use of multiple bands, the problems that this would cause for Freelance Graphics were avoided.

TT fonts are device fonts

This is bit #4 of the compatibility bits word, with hexadecimal value `0x10`, known by the symbolic name `GACF_CALLTTDEVICE` in `windows.h`. This flag causes Windows 3.1 to always set the `DEVICE_FONTTYPE` flag on any TrueType fonts that are enumerated using the Windows `EnumFont()` API. This was to fix a problem with applications including AmiPro and WordPerfect, both of which assumed that all TrueType fonts available on a printer would be device-resident.

Windows 3.1 palette behavior

Windows 3.1 defined 20 application compatibility flags. Windows 95 and 98 defined a further 11 flags, not documented in the KnowledgeBase article and not assigned symbolic constant names in `windows.h`, which are the remainder of the options accessible via the "Advanced Options" menu in Make Compatible:

Disable 16 color brush cache and 55ms timer

This is bit #29 of the compatibility bits word, with hexadecimal value `0x20000000`.

Disable EMF spooling

This is bit #26 of the compatibility bits word, with hexadecimal value `0x4000000`.

Disable font associations

This is bit #24 of the compatibility bits word, with hexadecimal value `0x1000000`.

Don't attach input thread when journaling, `SetActiveWindow` == `SetForeGroundWindow`

This is bit #28 of the compatibility bits word, with hexadecimal value `0x10000000`.

Don't Shutdown/Ignore certain faults/dequote commandline

This is bit #25 of the compatibility bits word, with hexadecimal value `0x2000000`.

Enable 3.x UI features

This is bit #27 of the compatibility bits word, with hexadecimal value `0x8000000`.

Force Win31 printer dev mode size

This is bit #23 of the compatibility bits word, with hexadecimal value `0x800000`.

Increase stack size

This is bit #22 of the compatibility bits word, with hexadecimal value `0x400000`.

Lie about device caps/no `SetDIBits` validation

This is bit #20 of the compatibility bits word, with hexadecimal value `0x100000`.

Lie about windows version

This is bit #21 of the compatibility bits word, with hexadecimal value `0x200000`.

Mirror fonts in `win.ini`

This is bit #30 of the compatibility bits word, with hexadecimal value `0x40000000`.

Chapter 7

Engineering Support and Component Repository Management

Engineering support

Configuration management is for most of time dealing with the system that is large, complexed, has a long life duration (more than 10 years) and involve more people. The key issues for **engineering support** are to coordinate the participants and to provide each engineer an environment, also called a workspace where they can work independently in the task duration. The former one refers the cooperative work support and the latter one is mostly called workspace support.

Cooperative work support

Cooperative work support is introduced since many concurrent workspaces may contain and change the same objects (files). So, there are needs to **synchronize objects** and **control concurrent work**. It is also important since the duration of an activity can be very long, which means the files would be locked for too long and severe dead lock would occur, to solve this problem, merge algorithm is used to resynchronizing objects.

Synchronizing method

As shown in the figure, the object A is used in both of the work groups. In order to prevent overlapping, the integration work space is created to coordinate the two development workspaces. Compared with database, the integration work space plays the role of the central DB and the other basic work spaces play the role of the cache of the concurrent transactions. Development work spaces report (integrate arrows in the figure) to the integration work space regularly and receive new version from it to work concurrently with other groups. Seeing from the outside, the complete group behaves as its integration work space along, while a tree where nodes are either groups or basic work spaces can be constructed to record the history. Containment between two workspaces may mean either work decomposition into concurrent activities or different level of validation.

Control concurrent work

The control of concurrent work is dealing with the problems of who can perform a change, at what time, on which attribute of which object. Priority can be introduced to solve part of the problem, but this field is still under research.

In Software configuration management (SCM), Merge is used to combine files based on a line by line comparison method. **Merge control** is commonly applied to changes to the same attribute of different objects or changes to different attributes of the same object. Object concurrent change control subsumes traditional file control and provides homogeneous and elegant solutions to many difficulties which currently hamper concurrent software engineering.

Workspace support

Software configuration management (SCM) system is responsible for providing a workspace for each engineer in the right file system, at the right time to let users work independently, and to save or update the changes automatically when the job is done. Sometimes, the later one is also said as change management.

Merge tools are widely used to facilitate workspace support. The following chart provides a process flow of the **merge tools** which is based on a line by line comparison method.

The upper process flow digram presents the main principle of **merge tools** in software configuration management. When a source file is required by a second workespace, the center DB will deliver a copy of that file to it. And after submitting the 2 versions of the same file, merger tools will start to combine these two version into a new one. It is based on a line by line process, which is: if There are new lines in the submitted version, add them to the source file, and if there are lines which do not exist in the new version, delete these lines in the source file. After several times of iteration, a new version of the sources file, which contains all the changes created by the two (or more) authors, will be upload again to the central DB and acts as a new version of the source file.

Component repository management

Component repository management is a field of configuration management that seeks to ensure the safe storage of different components of a software product and all its versions. This topic includes product model, revision control, and software configuration management.

Product model

The **product model** describes the structure of a software product in a single version system, which means the version model is not taken into consideration. It can be represented by a *product graph* in which nodes and edges represent the software objects and their relationships.

Software object

A **software object** records the result of a development or maintenance activity. A SCM system has to manage all kinds of software objects created throughout the software lifecycle, including requirements specifications, designs, documentations, program code, test plans, test cases, user manuals, project plans and so on.

Relationship

Relationships are the connectors of software objects. In the product model diagram, they are the edges between nodes. *Composition relationships* and *dependency relationships* are the two main relationships used in the product model.

Composition relationships are used to organize software objects with respect to their granularity. For example, the subsystems of a software product which in turn consist of modules.

Dependency relationships or in short dependency establish directed connections between objects that are orthogonal to composition relationships. They include lifecycle dependencies between requirements specifications, designs and module implementations.

Version model

A **version model** defines the items to be versioned, the common properties shared by all versions of an item, and the deltas. It also determines the way version sets are organized. It introduces dimensions of evolution such as revisions and variants, it defines whether a version is characterized in terms of the state it represents or in terms of some changes relative to some baseline, it selects a suitable representation for the version graphs, and it also provides operations for retrieving old versions and constructing new versions.

Another field related to **version model** is software history. It records all the versions of a software product safely and properly, it also records who created the revision along with what comment. Delta is used to reduce the spaces needed for storage, since two successive versions are often very similar (98% same on average).

It also provides supports for multi-user management. The traditional locking method is still used while a new advanced synchronizing method can greatly improve the efficiency.

Version and product model structures

Because **product model** only considers the software structure of a single version model, and the **version model** does not take software structure in to consideration, methods to integrate the two models will be discussed here. In particular, we will investigate which items are put under version control and how versions of different items are interrelated with each other.

According to the selection order during the configuration process, we can classify the structures into 3 categories, product first, version first and intertwined:

Product first means that the product structure is selected before the version models of components. This approach is followed, for example, by SCCS and RCS. But this method suffers from the restriction that structural versioning can not be expressed (it is because the product structure is fixed for all configurations).

Version first means the version of product is selected first and uniquely determines the component versions. Different product versions may be structured in different ways. For example, a version of a component is contained only in the product version 1.0 but not 2.0. PCTE is an example of an SCM system using this organization.

Intertwined means that the selections of version model and component structure are performed in alternating order. Intertwined structure is relatively more flexible than the two methods mentioned above, because when a new change happens or a new component need to be added in the configuration, for most of time it is not necessary to change the whole configuration structure.

In addition, *version first* and *intertwined* both take in to account that different versions of a certain *object* may vary with respect to their *relationships* of other objects. This means that in addition to objects, relationships are versioned as well.

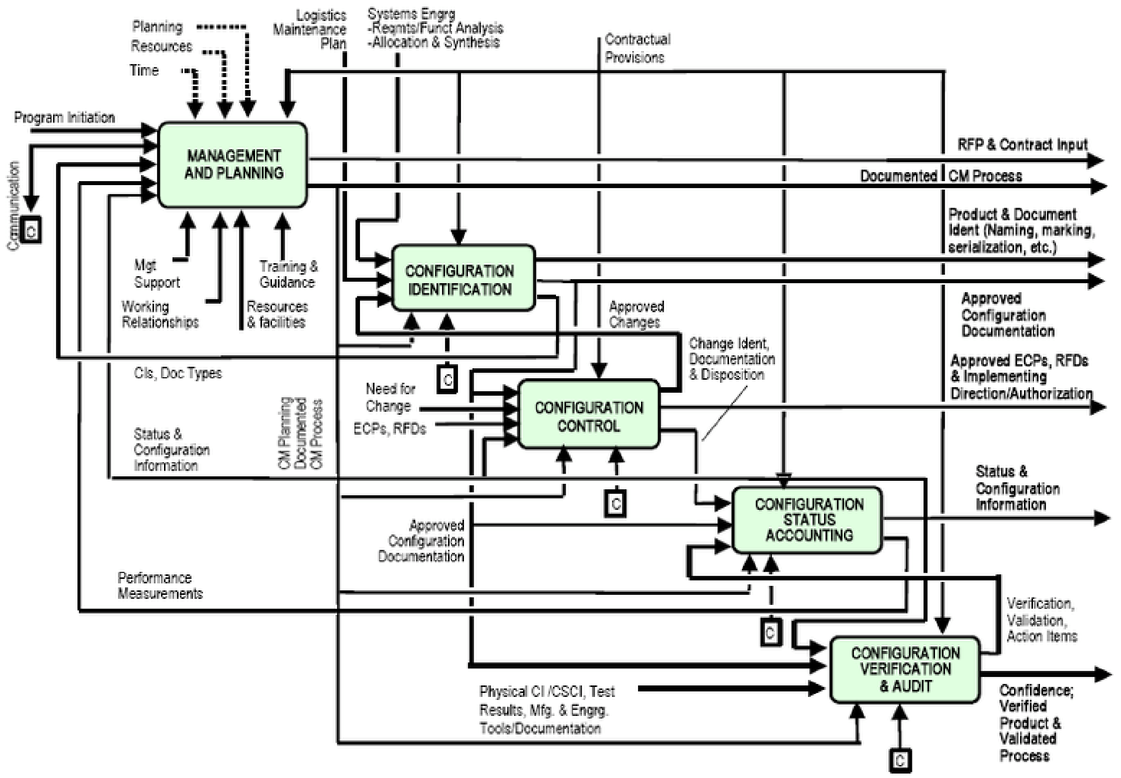
A meta-model of the AND / OR method

As shown in the intertwined configuration structure, an **AND / OR graphs** can be used to illustrate the interplay of *product model* and *version model*. The graphs consist of *nodes* and *edges*, and *selections* of the nodes represent the configuration of a certain software product of a certain version. There are two types of nodes — *AND nodes* and *OR nodes*. Analogously, a distinction is made between AND and OR edges, which emanate from AND and OR nodes, respectively. A meta-model is presented in the following figure:

An unversioned product graph can be represented by exclusively AND nodes/edges. Then this AND/OR graph will be the same as the *product model* or product compositions. While versioning of the product graph is modeled by introducing *OR nodes*. Versioned objects and their versions are represented by OR nodes and AND nodes, respective.

Chapter 8

Configuration Management



Planning
Resources
Time
Logistics Maintenance Plan
Systems Engrg -Reqmts/Funct Analysis -Allocation & Synthesis
Contractual Provisions
Program Initiation
Communication
C
MANAGEMENT AND PLANNING
RFP & Contract Input
Documented CM Process
Product & Document Ident (Naming, marking, serialization, etc.)
Mgt Support
Working Relationships
Resources & facilities
Training & Guidance
CONFIGURATION IDENTIFICATION
Approved Configuration Documentation
Approved Changes
Change Ident, Documentation & Disposition
Approved ECPs, RFDs & Implementing Direction/Authorization
CIs, Doc Types
Need for Change
ECPs, RFDs
CONFIGURATION CONTROL
Status & Configuration Information
CM Planning Documented CM Process
CONFIGURATION STATUS ACCOUNTING
Approved Configuration Documentation
Performance Measurements
Verification, Validation, Action Items
CONFIGURATION VERIFICATION & AUDIT
Physical CI /CSCI, Test Results, Mfg. & Engrg. Tools/Documentation
Confidence; Verified Product & Validated Process

Top level Configuration Management Activity model

Configuration management (CM) is a field of management that focuses on establishing and maintaining consistency of a system or product's performance and its functional and physical attributes with its requirements, design, and operational information throughout its life.

For information assurance, CM can be defined as the management of security features and assurances through control of changes made to hardware, software, firmware, documentation, test, test fixtures, and test documentation throughout the life cycle of an information system. CM for information assurance, sometimes referred to as **S**ecure **C**onfiguration **M**anagement, relies upon performance, functional, and physical attributes of IT platforms and products and their environments to determine the appropriate security features and assurances that are used to measure a system configuration state.

For example, configuration requirements may be different for a network firewall that functions as part of an organization's Internet boundary versus one that functions as an internal local network firewall.

History

Configuration management was first developed by the United States Air Force for the Department of Defense in the 1950s as a technical management discipline of hardware. The concepts of this discipline have been widely adopted by numerous technical management functions, including systems engineering (SE), integrated logistics support (ILS), Capability Maturity Model Integration (CMMI), ISO 9000, Prince2 project management methodology, COBIT, Information Technology Infrastructure Library (ITIL), product lifecycle management, and application lifecycle management. Many of these functions and models have redefined configuration management from its traditional holistic approach to technical management. Some treat configuration management as being similar to a librarian activity, and break out change control or change management as a separate or stand alone discipline. However the bottomline is and always shall be Traceability.

Software configuration management

The traditional software configuration management (SCM) process is looked upon by practitioners as the best solution to handling changes in software projects. It identifies the functional and physical attributes of software at various points in time, and performs systematic control of changes to the identified attributes for the purpose of maintaining software integrity and traceability throughout the software development life cycle.

The SCM process further defines the need to trace changes, and the ability to verify that the final delivered software has all of the planned enhancements that are supposed to be included in the release. It identifies four procedures that must be defined for each software project to ensure that a sound SCM process is implemented. They are:

1. Configuration identification
2. Configuration control
3. Configuration status accounting
4. Configuration audits

These terms and definitions change from standard to standard, but are essentially the same.

- Configuration identification is the process of identifying the attributes that define every aspect of a configuration item. A configuration item is a product (hardware and/or software) that has an end-user purpose. These attributes are recorded in configuration documentation and baselined. Baselining an attribute forces formal configuration change control processes to be effected in the event that these attributes are changed.

- Configuration change control is a set of processes and approval stages required to change a configuration item's attributes and to re-baseline them.

- Configuration status accounting is the ability to record and report on the configuration baselines associated with each configuration item at any moment of time.

- Configuration audits are broken into functional and physical configuration audits. They occur either at delivery or at the moment of effecting the change. A functional configuration audit ensures that functional and performance attributes of a configuration item are achieved, while a physical configuration audit ensures that a configuration item is installed in accordance with the requirements of its detailed design documentation.

Configuration management is widely used by many military organizations to manage the technical aspects of any complex systems, such as weapon systems, vehicles, and information systems. The discipline combines the capability aspects that these systems provide an organization with the issues of management of change to these systems over time.

Outside of the military, CM is appropriate to a wide range of fields and industry and commercial sectors.

Computer hardware configuration management

Computer hardware configuration management is the process of creating and maintaining an up-to-date record of all the components of the infrastructure, including related documentation. Its purpose is to show what makes up the infrastructure and illustrate the physical locations and links between each item, which are known as configuration items.

Computer hardware configuration goes beyond the recording of computer hardware for the purpose of asset management, although it can be used to maintain asset information. The extra value provided is the rich source of support information that it provides to all interested parties. This information is typically stored together in a configuration management database (CMDB). This concept was introduced by ITIL.

The scope of configuration management is assumed to include, at a minimum, all configuration items used in the provision of live, operational services.

Computer hardware configuration management provides direct control over information technology (IT) assets and improves the ability of the service provider to deliver quality IT services in an economical and effective manner. Configuration management should work closely with change management.

All components of the IT infrastructure should be registered in the CMDB. The responsibilities of configuration management with regard to the CMDB are:

- identification
- control
- status accounting
- verification

The scope of configuration management is assumed to include:

- physical client and server hardware products and versions
- operating system software products and versions
- application development software products and versions
- technical architecture product sets and versions as they are defined and introduced
- live documentation
- networking products and versions
- live application products and versions
- definitions of packages of software releases
- definitions of hardware base configurations
- configuration item standards and definitions

The benefits of computer hardware configuration management are:

- helps to minimize the impact of changes
- provides accurate information on CIs
- improves security by controlling the versions of CIs in use
- facilitates adherence to legal obligations
- helps in financial and expenditure planning

Maintenance systems

Configuration management is used to maintain an understanding of the status of complex assets with a view to maintaining the highest level of serviceability for the lowest cost. Specifically, it aims to ensure that operations are not disrupted due to the asset (or parts of the asset) overrunning limits of planned lifespan or below quality levels.

In the military, this type of activity is often classed as "mission readiness", and seeks to define which assets are available and for which type of mission; a classic example is whether aircraft on-board an aircraft carrier are equipped with bombs for ground support or missiles for defense.

A theory of configuration maintenance was worked out by Mark Burgess , with a practical implementation on present day computer systems in the software Cfengine able to perform real time repair as well as preventive maintenance.

Preventive maintenance

Understanding the "as is" state of an asset and its major components is an essential element in preventive maintenance as used in maintenance, repair, and overhaul and enterprise asset management systems.

Complex assets such as aircraft, ships, industrial machinery etc. depend on many different components being serviceable. This serviceability is often defined in terms of the amount of usage the component has had since it was new, since fitted, since repaired, the amount of use it has had over its life and several other limiting factors. Understanding how near the end of their life each of these components is has been a major undertaking involving labor intensive record keeping until recent developments in software.

Predictive maintenance

Many types of component use electronic sensors to capture data which provides live condition monitoring. This data is analyzed on board or at a remote location by computer to evaluate its current serviceability and increasingly its likely future state using algorithms which predict potential future failures based on previous examples of failure through field experience and modeling. This is the basis for "predictive maintenance".

Availability of accurate and timely data is essential in order for CM to provide operational value and a lack of this can often be a limiting factor. Capturing and disseminating the operating data to the various support organizations is becoming an industry in itself.

The consumers of this data have grown more numerous and complex with the growth of programs offered by original equipment manufacturers (OEMs). These are designed to offer operators guaranteed availability and make the picture more complex with the operator managing the asset but the OEM taking on the liability to ensure its serviceability. In such a situation, individual components within an asset may communicate directly to an analysis center provided by the OEM or an independent analyst.

Standards

- ANSI/EIA-649-1998 National Consensus Standard for Configuration Management
- EIA-649-A 2004 National Consensus Standard for Configuration Management
- ISO 10007:2003 Quality management systems - Guidelines for configuration management
- Federal Standard 1037C

- GEIA Standard 836-2002 Configuration Management Data Exchange and Interoperability
- IEEE Std. 828-1998 IEEE Standard for Software Configuration Management Plans
- MIL-STD-973 Configuration Management (cancelled on September 20, 2000)
- STANAG 4159 NATO Materiel Configuration Management Policy and Procedures for Multinational Joint Projects
- STANAG 4427 Introduction of Allied Configuration Management Publications (ACMPs)
- CMMI CMMI for Development, Version 1.2 CONFIGURATION MANAGEMENT
- CMII - The Path to Integrated Process Excellence

Guidelines

- IEEE Std. 1042-1987 IEEE Guide to Software Configuration Management
- MIL-HDBK-61A CONFIGURATION MANAGEMENT GUIDANCE 7 February 2001
- 10007 Quality management - Guidelines for configuration management
- GEIA-HB-649 - Implementation Guide for Configuration Management
- ANSI/EIA-649-1998 National Consensus Standard for Configuration Management
- EIA-836 Consensus Standard for Configuration Management Data Exchange and Interoperability
- ANSI/EIA-632-1998 Processes for Engineering a System

Construction Industry

More recently configuration management has been applied to large construction projects which can often be very complex and have a huge amount of details and changes that need to be documented. Construction agencies such as the Federal Highway Administration have used configuration management for their infrastructure projects. There have been several construction based configuration management software developed that aim to document change orders and RFIs in order to ensure a project stays on schedule and on budget. These programs can also store information to aid in the maintenance and modification of the infrastructure when it is completed. One such application, ccsNet, was tested in a case study funded by the Federal Transportation Administration (FTA) in which the efficacy of configuration management was measured through comparing the approximately 80% complete construction of the Los Angeles County Metropolitan Transit Agency (LACMTA) 1st and 2nd segments of the Red Line, a $5.3 billion rail construction project. This study yielded results indicating a benefit to using configuration management on projects of this nature.

Chapter 9

Baseline (Configuration Management) and CA Software Change Manager

Baseline (configuration management)

Configuration management is the process of managing change in hardware, software, firmware, documentation, measurements, etc. As change requires an initial state and *next* state, the marking of significant states within a series of several changes becomes important. The identification of significant states within the revision history of a configuration item is the central purpose of **baseline** identification.

Typically, significant states are those that receive a formal approval status, either explicitly or implicitly (approval statuses may be marked individually, when such a marking has been defined, or signified merely by association to a certain baseline). Nevertheless, this approval status is usually recognized publicly. Thus, a baseline may also mark an approved configuration item, e.g. a project plan that has been signed off for execution. In a similar manner, associating multiple configuration items with such a baseline indicates those items as being approved.

Generally, a baseline may be a single work product, or set of work products that can be used as a logical basis for comparison. A baseline may also be established (whose work products meet certain criteria) as the basis for subsequent select activities. Such activities may be attributed with formal approval.

Conversely, the configuration of a project often includes one or more baselines, the status of the configuration, and any metrics collected. The current configuration refers to the current status, current audit, current metrics, and latest revision of all configuration items. Similarly, but less frequently, a baseline may refer to all items associated with a specific project. This may include all revisions of all items, or only the latest revision of all items in the project, depending upon context, e.g. "the baseline of the project is proceeding as planned."

A baseline may be specialized as a specific type of baseline. Some examples include:

- Functional Baseline: initial specifications established; contract, etc.
- Allocated Baseline: state of work products once requirements are approved
- Developmental Baseline: state of work products amid development
- Product Baseline: contains the releasable contents of the project
- others, based upon proprietary business practices

Capabilities of Baselines

While marking approval status covers the majority of uses for a baseline, baselines may also be established merely to signify the progress of work through the passage of time. In this case, a baseline is a visible stake through an endured collective effort, e.g. a developmental baseline. Baselines may also mark milestones; albeit some milestones also signify approval.

Baselines themselves are valued not only for their ability to identify the notable state of work product(s) but also provide particular importance in their ability to be retrieved. Once retrieved, the state of the work product(s) in that subset share the same significance in their history of changes that this significance was observed. The baseline is then regarded with poignant qualities (either favorably or disfavorably). For this reason, baseline identification, monitoring, and retrieval are critical to the success of configuration management. However, the ease of retrieving any given baseline varies according to the system employed for performing configuration management which may use a manual, automated, or hybrid approach. Once retrieved, the baseline may be compared to a particular configuration or another baseline.

Most baselines are established at a fixed point in time and serve to continue to reference that point (identification of state). However, some baselines are established to carry forward as a reference to the item itself regardless of any changes to the item. These latter baselines evolve with the progression of the work effort but continue to identify notable work products in the project.

Baselining Configuration Items

In the process of performing configuration management, configuration items (or work products) may be baselined so as to establish them with a certain status to interested parties. In this sense, to baseline a work product may require certain change(s) to the work product to ensure its conformance to the characteristics associated with the baseline referenced. This varies upon context, but in many cases this has the implication that the work product is "reset" to an initial (possibly inherently approved) state from which work may proceed.

Baseline Control

In many environments, baselines are controlled such that certain subsequent activities against work products in that baseline are either prohibited or permitted. These activities are carefully selected and controlled, and again, depending upon the configuration management system, are also monitored. Consequently, baselines are ordinarily subjected to configuration management audits. Audits may include an examination of specific actions performed against the baseline, identification of individuals involved in any action, an evaluation of change within the baseline, (re-)certification for approval, accounting, metric collection, comparison to another baseline, or all of these.

Application

Though common in software revision control systems as **labels** or **tags**, the existence of baselines is found in several other technology-related domains. Baselines can be found in UML modeling systems and business rule management systems, among others.

In addition to the field of Hardware and Software Engineering, baselines can be found in Medicine (e.g. monitoring health progress), Politics (e.g. statistics), Physics & Chemistry (e.g. observations and changes), Finance (e.g. budgeting), and others.

CA Software Change Manager

CA Software Change Manager	
Developer(s)	Softool (–1995) Platinum Technology (1995–1999) CA Technologies (1999–present)
Stable release	r12.1.01 / December 22, 2010; 3 months ago
Operating system	Microsoft Windows
Type	Revision control
License	Proprietary EULA

CA Software Change Manager (originally known as **CCC/Harvest**) is a software tool for the configuration management (revision control, SCM, etc.) of source code and other software development assets.

History

The first CCC (acronym for 'Change and Configuration Control') product was released in the early 70s and was designed as a project for a Defense Department contractor in Santa Barbara CA. (The company at the time was Hughes Aircraft, now Santa Barbara Research Center for Raytheon.) It became the first commercially available CM tool.

CCC was designed to manage all the components that went into an aircraft engine, and seeing as the same engine was used by both the U.S. Air Force and U.S. Navy (for the F-14 Tomcat and F-15 Eagle) it required robust and reliable parallel development.

The first version of **CCC/Harvest** was commercially developed by Softool Corporation, a CM-focused software company founded in 1977 in Goleta, CA. Other CCC tools included CCC/Manager, CCC/DM Turnkey and CCC/QuickTrak.

Softool was acquired in late 1995 by Platinum Technology, which was later acquired in May 1999 by Computer Associates (now known as CA Technologies) who added CCC/Harvest to their AllFusion suite. In 2002, the 'CCC' part of the name was dropped, and 'Change Manager' was added so it became known as **AllFusion Harvest Change Manager**. Later this was changed to just **CA Harvest Change Manager**. On October 7, 2008, the 'Harvest' part of the name was dropped and changed to 'Software' and it is now known as **CA Software Change Manager**.

Distinguishing features

- **Change Packages:** Harvest can provide both version control and change management. The developer makes changes in Harvest against a change package (creating a "change set"). The change package(s) will initially consist of a number of files that the developer has either created or amended. This is the version control component of Harvest.
- **Life Cycles:** Once the developer is satisfied with his/her changes, the changes progress through a pre-defined life cycle (i.e. into a number of sequential TEST stages and finally into PRODUCTION). At all these stages of this "life cycle", the package must have approvals from the appropriate users or user groups. These approvals are recorded permanently for audit purposes. For example, a test manager may have to approve packages prior to moving to the TEST stage, and the production change management team may have to approve packages prior to moving to the PROD state.
- **Projects (Environments):** Central to Harvest's philosophy is the concept of a Harvest "project". Projects are fully customizable according to an application's, organization's, or team's needs. The term project refers to the entire control framework in Harvest and includes:
    - A branch or separate line of development where changes can be isolated (the version control component)
    - The definition of processes and how changes progress through the promotional life-cycle

- Access control for processes and file

Chapter 10

Merge (Revision Control) and MSConfig

Merge (revision control)

Merging (also called integration) in revision control, is a fundamental operation that reconciles multiple changes made to a revision-controlled collection of files. Most often, it is necessary when a file is modified by two people on two different computers at the same time. When two branches are merged, the result is a single collection of files that contains both sets of changes.

In some cases, the merge can be performed automatically, because the changes do not conflict. In other cases, a person must decide exactly what the resulting files should contain. Many revision control software tools include merge capabilities.

Merge can be used as a verb ("to merge branches,") but can also be a noun ("this merge will be difficult.")

Types of merges

There are two primary types of merges performed by automated merge tools: 2-way merge and 3-way merge. A 3-way merge is a more powerful and reliable method of merging than is afforded by the 2-way merge.

Two-way merge

A two-way merge performs an automated difference analysis between a file 'A' and a file 'B'. This method considers the differences between the two files alone to conduct the merge and makes a "best-guess" analysis to generate the resulting merge. Consequently, this type of merge is usually the most error prone and requires user intervention to verify and sometimes correct the result of the merge prior to completing the merge event.

Three-way merge

A three-way merge is performed after an automated difference analysis between a file 'A' and a file 'B' while also considering the origin, or parent, of both files (usually the parent is the same for both). This type of merge is more likely to be usable in revision control systems, which can guarantee that such a parent exists and is known. The merge tool examines the differences and patterns appearing in the changes between both files as well as the parent, building a relationship model to generate a merge of files 'A', 'B', and the parent 'C', to produce a new revision 'D'.

This merge is the most reliable and has performed well in practice. It has also required the least amount of user intervention, and in many cases, requiring no intervention at all (depending upon the complexity of the merge) making the process eligible for task automation.

Trends

The technological advancements in the 3-way merge method have led to the increase in popularity among software development environments to institute concurrent modification through branching in their practices of software configuration management (SCM). In the early to mid-1990s branching was a discouraged practice in smaller software development groups due to the complexities and conflicts introduced through the merging process and the low availability of cost-effective 3-way merge tools. However, this practice was more in demand among larger groups merely due to the increased likelihood that two developers would need to modify the same file at the same time. Merging, at that time, was indeed a challenge and in some environments, additional proprietary conventions were introduced to simplify the necessary merge.

In the early 2000s, the increased availability of reliable 3-way merge tools reduced the time that software development groups had to spend concerning themselves with the technical limitations of their infrastructure. Even smaller software groups are more inclined to approach concurrent modification in their revision control systems. Nevertheless, merges still often cause problems; even intelligent merge tools can't resolve all conflicts automatically. Consequently, human interaction is required, which can lead to human errors.

3-way merges still remain one of the more taxing tasks of any software development team. This is especially because the person resolving the merge needs prior knowledge of the original code, the intermediate commit and the changes wanted.

Recent developments

In recent years, some new merge algorithms have been developed and are gaining popularity:

- the patch commutation of Darcs

- the two-way merge with history of Codeville

Standalone merging tools

- Araxis Merge 2/3-way file comparison, merging and folder synchronization for Windows and Mac OS X
- Beyond Compare Professional Folder and file comparison/synchronization and 3-way merge utility for Windows and Linux
- DeltaWalker Oro Two and three-way file & folder comparison, merge and synchronization for Mac OS X, Windows, and Linux
- DiffMerge file and folder compare and merge (also supports three way merge) for Mac OS X, Windows and Linux.
- ECMerge 2-way and 3-way diff/merge tool for text, images and directories
- Apple FileMerge, a Mac OS X development tool derived from Merge from NeXT
- Guiffy SureMerge File Compare, Folder Compare, and Merge tool
- diff3 3-way merge tool, widely spread on Unix-like systems
- KDiff3 3-way merge tool
- meld
- MergePlant 3-way merge tool
- SimMerge 3-way merge tool for Simulink models.
- tkmerge
- WinMerge a free, open source graphic windows based diff and merge tool
- UltraCompare
- xxdiff - Graphical File And Directories Comparator And Merge Tool

MSConfig

MSConfig, or **Microsoft System Configuration Utility**, (or simply **System Configuration** in Windows Vista and Windows 7) is a utility to troubleshoot the Microsoft Windows startup process. It is bundled with all Microsoft Windows operating systems since Windows 98 except Windows 2000. Windows 95 and Windows 2000 users can download the utility as well, although it was not designed for them. MSConfig modifies which programs run at startup, edits certain configuration files, and simplifies controls over Windows services. As part of the base Windows install, MSConfig has commonly not been linked to in the Start Menu or Control Panel, but is accessible by using the Run dialog to launch 'msconfig' on any system on which the user has administrator access.

Files that can be edited through MSConfig include AUTOEXEC.BAT, CONFIG.SYS, WIN.INI, SYSTEM.INI on Windows 9x systems, and WIN.INI, SYSTEM.INI and BOOT.INI on Windows NT systems prior to Windows Vista. The chief benefit to using MSConfig to edit these files is that it provides a simplified GUI to manipulate sections of those files and the Windows registry tree pertaining to the Windows boot sequence.

Using MSConfig, Windows can also be configured to perform a diagnostic startup (load a minimum set of drivers, programs and services).

Features

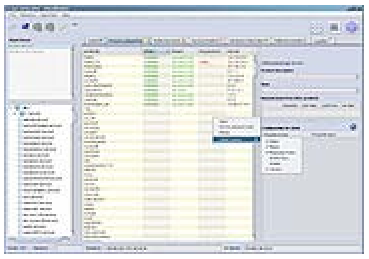
Some of its functionality varies by Windows versions:

- In Windows 98 and Windows Me, it can configure advanced troubleshooting settings pertaining to these operating systems. It can also launch common system tools.
- In Windows 98, it can back up and restore startup files.
- In Windows Me, it has also been updated with three new tabs called "Static VxDs", "Environment" and "International". The Static VxDs tab allows users to enable or disable static virtual device drivers to be loaded at startup, the *Environment* tab allows users to enable or disable environment variables, and the *International* tab allows users to set international language keyboard layout settings that were formerly set via the real-mode MS-DOS configuration files. A "Cleanup" button on the "Startup" tab allows cleaning up invalid or deleted startup entries.
- In Windows Me and Windows XP versions, it can restore an individual file from the original Windows installation set.
- On Windows NT-based operating systems prior to Windows Vista, it can set various BOOT.INI switches.
- In Windows XP and Windows Vista, it can hide all operating system services for troubleshooting.
- In Windows Vista and later, the tool gained additional support for launching a variety of tools, such as system information, other configuration areas, such as Internet options, and the ability to enable/disable UAC. An update is available for Windows XP and Windows Server 2003 that adds the *Tools* tab. It also allows configuring various switches for Windows Boot Manager and Boot Configuration Data.

Chapter 11

Opsi

opsi



opsi management interface

Developer(s)	uib GmbH, Mainz, Germany
Stable release	4.0 / 1. oct 2010
Written in	Python Java
Operating system	Linux, Windows
Available in	English, French, German, Spanish, Turkish
Type	Network management System administration
License	GPL

Opsi (open pc server integration) is a software distribution and management system for Windows Clients, based on Linux servers. Opsi is developed and maintained by uib GmbH from Mainz, Germany. The main parts of of opsi are Open Source licensed under the GNU General Public License.

Features

The key features of opsi are:

- Automated operating system installation (OS deployment)

- Software distribution
- Patch management
- Inventory (hardware and software)
- License Management / Software Asset Management

A tools for automated installations is important and necessary for standardization, maintainability and cost saving of larger PC networks.

opsi supports the client operating systems Windows XP, Server 2003, Windows Vista, Server 2008, Windows 7 and Server 2008R2. The 32- and the 64-bit versions are supported. For the installation of an opsi-server there are packages available for the Linux distributions Debian, Ubuntu, SLES, Univention Corporate Server, CentOS, RHEL and OpenSuse.

Automated operating system installation

Via management interface a client may be selected for OS-Installation. If the client boots via PXEit loads a boot image from the opsi-depotserver. This bootimage prepares the hard disk, copies the required installation files, drivers and the opsi client agent and starts finally an unattended OS-Installation. Opsi uses the automatic detection of the necessary drivers for PCI-, HD-Audio- and USB-Devices. OS-installation via Disk image is also supported.

Software distribution

For the automatic software distribution some software, the opsi-client-agent, has to be installed on each client. Every time the client boots the opsi-client-agent connects to the opsi-server and asks if there is anything to install (default). If this shall be done a script driven installation program (opsi-winst) starts and installs the required software on the client. During the installation process the user login can be blocked for integrity reasons. To integrate a new software packet into the software deployment system, a script must be written to specify the installation process. This script provides all the information on how this software packet has to be installed silent or unattended or by using tools like AutoIt or Autohotkey. With the opsi-winst steps like copy files or edit the registry can be done. The the opsi-client-agent can also be triggered by other events or via push-installation from the opsi-server

Patch-Management

The mechanism of the software deployment can also be used to deploy software patches and hotfixes.

Inventory (hardware and software)

The hard- and software inventory also uses the opsi-client-agent. The hardware information is collected via calls to WMI while the software information is gathered from the registry. The inventory data are sent back to the opsi-server by a web service. The inventory data may imported via a web service to a CMDB e.g. in OTRS.

License management / Software Asset Management

The opsi License Management module supports the administration of different kinds of licenses like Retail, OEM and Volume licenses. It counts the licenses that are used with the software deployment. Using the combination of the License Management and the software inventory, Software Asset Management reports on the number of free and installed licenses can be generated. The License Management module is part of a Co-funding Project and not released as open source yet.

opsi-server

The opsi-server provides the following services:

- The configuration-server stores the configuration data of the clients and provides the methods to manage these data via a web service or the command-line. The data can be stored in files, in OpenLDAP or in a MySQL Database.
- The depot-server stores software packages that may be installed on the clients. To provide support for multiple locations, multiple depot-servers may be controlled by one configuration-server.
- A TFTP-Server provides the boot images for the OS-Installations.
- A DHCP-Server may be integrated in the opsi-server

Management interface

For managing opsi a graphical user interface is available as an application or as a browser Applet. Management is also possible with a command line tool or via web service.

License

The opsi core features are Open Source according to the GNU General Public License Version 3 and are free of charge. The core features are software distribution (or software deployment), OS deployment and hard- and software-inventory. These free components can be supplemented with closed source add-ons that are fee required. They are called Co-funding Projects.

Co-funding projects

Even though opsi is open source, there are some components which are not for free at the moment. These components are developed in a co-funding project. This means, that these parts are only available for those customers who paid a contribution to the cost of development. As soon as the development of a co-funding project is refinanced, the component will be part of the free opsi-version and can be used free of charge. It will be open source (as long as not prevented caused by technical reasons). The first of these co-funding Projects was the opsi support for Vista/Windows 7. It was completed on 1st of February 2008 and is free of charge since 1st of March 2010. The source code was divided form the not yet paid parts and is open source since since 30th of November 2010. At the moment (January 2011) there are three co-funding projects: Treewiew allows to build hierarchical groups of clients to manage them, MySQL as backend for all data and the license management module. The main focus of co-funding projects is to create software once for a pool of purchaser who share the cost and make it open source as soon as it paid in full.

Chapter 12

Quattor

Quattor' is a generic open-source tool-kit used to install, configure, and manage computers. Quattor was originally developed in the framework of European Data Grid project (2001-2004). Since its first release in 2003, Quattor has been maintained and extended by a volunteer community of users and developers, primarily from the community of grid system administrators. The Quattor tool-kit, like other configuration management systems, reduces the manpower required to maintain a cluster and facilitates reliable change management. However, three unique features make it particularly attractive for managing grid resources:

- Federated Management: The open, modular nature of the tool-kit permits system administrators at different institutes to share the management of their distributed resources.
- Shared Configuration and Management Efficiency: Quattor encourages the re-use of configuration information in such a way that it can be distributed and used with little or no modification at different sites, facilitating the distribution of best practices without the need for each site to implement configuration changes.
- Coherent Site Model: Quattor allows an administrator to develop a site model that, once constructed, can be used to manage a range of different resources, such as real machines, virtual machines and cloud resources.

These features are also attractive beyond the grid context. This has been confirmed by the growing adoption of Quattor, by both commercial companies and academic institutions, most of them using the tool-kit to manage consistently their grid and non-grid systems.

Principles

The challenge of structuring and sharing components in a collaborative system is not new; over the years programming language designers have attacked this problem from many angles. While trends change, the basic principles are well understood. Features such as encapsulation, abstraction, modularity, and typing produce clear benefits. We believe that similar principles apply when sharing configuration information across administrative domains.

The Quattor configuration tool-kit derives its architecture from LCFG, improving it in several aspects. At the core of Quattor is Pan, a high-level, typed language with flexible include mechanisms, a range of data structures, and validation features familiar to modern programmers. Pan allows collaborative administrators to build up a complex set of configuration templates describing service types, hardware components, configuration parameters, users etc. The use of a high-level language facilitates code reuse in a way that goes beyond cut-and-paste of configuration snippets.

The principles embodied in Quattor are in line with those established within the system administration community. In particular, all managed nodes retrieve their configurations from a configuration server backed by a source-control system (or systems in the case of devolved management). This allows individual nodes to be recreated in the case of hardware failure. Quattor handles both distributed and traditional (single-site) infrastructures.

Devolved management includes the following features: consistency over a multi-site infrastructure, multiple management points, and the ability to accommodate the specific needs of constituent sites. There is no single “correct” model for a devolved infrastructure, thus great flexibility is needed in the architecture of the configuration system itself. Sometimes a set of highly autonomous sites wish to collaborate loosely. In this case each site will host a fairly comprehensive set of configuration servers, with common configuration information being retrieved from a shared database and integrated with the local configuration.

Distributing the management task can potentially introduce new costs. For example, transmitting configuration information over the WAN introduces latency and security concerns. Quattor allows servers to be placed at appropriate locations in the infrastructure to reduce latency, and the use of standard tools and protocols means that existing security systems (such as a public key infrastructure) can be harnessed to encrypt and authenticate communications.

Quattor Architecture

Configuration management system

Quattor’s configuration management system is composed of a configuration database that stores high-level configuration templates, the Pan compiler that validates templates and translates them to XML profiles, and a machine profile repository that serves the profiles to client nodes. Only the Pan compiler is strictly necessary in a Quattor system; the other two subsystems can be replaced by any service providing similar functionality.

Devolved management in a cross-domain environment requires users to be authenticated and their operations to be authorized. For the configuration database, we chose to adopt X.509 certificates1 because of the support offered by many standard tools, and access control lists (ACLs) because they allow a fine-grained control (an ACL can be attached to each template). When many users interact with the system, conflicts and misconfiguration may arise which require a roll back mechanism; to this purpose, a simple concurrent transaction mechanism, based on standard version control systems, was implemented.

Quattor's modular architecture allows the three configuration management subsystems to be deployed in either a distributed or centralized fashion. In the distributed approach, profile compilation (at development stage) is carried out on client systems, templates are then checked in to a suitable database, and finally the deployment is initiated by invoking a separate operation on the server. The centralized approach provides strict control of configuration data. The compilation burden is placed onto the central server, and users can only access and modify templates via a dedicated interface.

Since the two paradigms provide essentially the same functionality, the choice between them depends on which fits the management model of an organization better. For instance, the centralized approach fits large computer centres well because of its strictly controlled work-flow, whereas multi-site organizations such as GRIF prefer the distributed approach because it allows different parts of the whole configuration set to be handled autonomously.

Pan language

The Pan language compiler sits at the core of the Quattor tool-kit. It compiles machine configurations written in the Pan configuration language by system administrators and produces XML files (profiles) that are easily consumed by Quattor clients. The Pan language itself has a simple, declarative syntax that allows simultaneous definition of configuration information and an associated schema. Here, we focus only on the Pan features that are relevant to devolved management of distributed sites: validation, configuration reuse, and modularization.

Validation. The extensive validation features in the Pan language maximize the probability of finding configuration problems at compile time, minimizing costly clean-ups of deployed misconfiguration. Pan enables system administrators to define atomic or compound types with associated validation functions; when a part of the configuration schema is bound to a type, the declared constraints are automatically enforced.

Configuration reuse. Pan allows identification and reuse of configuration information through "structure templates." These identify small, reusable chunks of Pan-level configuration information which can be used whenever an administrator identifies an invariant (or nearly invariant) configuration sub-tree.

Modularization. With respect to the original design, two new features have been developed to promote modularization and large-scale reuse of configurations: the name-spacing and load-path mechanisms.

A full site configuration typically consists of a large number of templates organized into directories and subdirectories. The Pan template name-spacing mimics (and enforces) this organization much as is done in the Java language. The name-space hierarchy is independent of the configuration schema. The configuration schema is often organized by low-level services such as firewall settings for ports, account generation, log rotation entries, cron entries, and the like. In contrast, the Pan templates are usually organized based on other criteria like high-level services (web server, mail server, etc.) or by responsible person/group.

The name-spacing allows various parts of the configuration to be separated and identified. To effectively modularize part of the configuration for reuse, administrators must be able to import the modules easily into a site's configuration and to customize them. Users of the Pan compiler combine a load-path with the name-spacing to achieve this. The compiler uses the load-path to search multiple root directories for particular, named templates; the first version found on the load-path is the one that is used by the compiler. This allows modules to be kept in a pristine state while allowing sites to override any particular template.

Further, module developers can also expose global variables to parameterize the module, permitting a system administrator to use a module without having to understand the inner workings of the module's templates.

Quattor Working Group (QWG) templates are used to configure grid middleware services. The QWG templates use all of the features of Pan to allow distributed sites to share grid middleware expertise.

Automated installation management

A key feature for administering large distributed infrastructures is the ability to automatically install machines, possibly from a remote location. To this purpose, Quattor provides a modular framework called the Automated Installation Infrastructure (AII). This framework is responsible for translating the configuration parameters embodied in node profiles into installation instructions suitable for use by standard installation tools. Current AII modules use node profiles to configure DHCP servers, PXE boot and Kickstart-guided installations.

Normally AII is set up with an install server at each site. However, the above mentioned technologies allow the transparent implementation of multi-site installations, by setting up a central server and appropriate relays using standard protocols.

Node configuration management

In Quattor, managed nodes handle their configuration process autonomously; all actions are initiated locally, once the configuration profile has been retrieved from the repository. Each node has a set of configuration agents (components) that are each registered with a particular part of the configuration schema. For example, the component that manages user accounts is registered with the path /software/components/accounts. A dispatcher program running on the node performs an analysis of the freshly retrieved configuration for changes in the relevant sections, and triggers the appropriate components. Run-time dependencies may be expressed in the node's profile, so that a partial order can be enforced on component execution. For example, it is important that the user accounts component runs before the file creation component, to ensure that file ownership can be correctly specified.

By design, no control loop is provided for ensuring the correct execution of configuration components. Site administrators typically use standard monitoring systems to detect and respond to configuration failures. Nagios and Lemon are both being used at Quattor sites for this purpose. In fact, Lemon has been developed in tandem with Quattor, and provides sensors to detect failures in Quattor component execution.

While nodes normally update themselves automatically, administrators can configure the system to disable automatic change deployment. This is crucial in a devolved system where the responsibilities for, respectively, modifying and deploying the configuration may be separated. A typical scenario is that top-level administrators manage the shared configuration of multiple remote sites and local managers apply it according to their policies. For instance, software updates might be scheduled at different times.

Chapter 13

Software Configuration Management and Telelogic Synergy

Software configuration management

In software engineering, **software configuration management (SCM)** is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines.

SCM concerns itself with answering the question "Somebody did something, how can one reproduce it?" Often the problem involves not reproducing "it" identically, but with controlled, incremental changes. Answering the question thus becomes a matter of comparing different results and of analysing their differences. Traditional configuration management typically focused on controlled creation of relatively simple products. Now, implementers of SCM face the challenge of dealing with relatively minor increments under their own control, in the context of the complex system being developed.

Terminology

The history and terminology of SCM (which often varies) has given rise to controversy. Roger Pressman, in his book *Software Engineering: A Practitioner's Approach*, states that SCM is a "set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining' **for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made."**

Source configuration management is a related practice often used to indicate that a variety of artifacts may be managed and versioned, including software code, hardware, documents, design models, and even the directory structure itself.

Atria (later Rational Software, now a part of IBM), used "SCM" to mean "software configuration management". Gartner uses the term *software change and configuration management*.

Purposes

The goals of SCM are generally:

- Configuration identification - Identifying configurations, configuration items and baselines.
- Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline.
- Configuration status accounting - Recording and reporting all the necessary information on the status of the development process.
- Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications and user manuals.
- Build management - Managing the process and tools used for builds.
- Process management - Ensuring adherence to the organization's development process.
- Environment management - Managing the software and hardware that host the system.
- Teamwork - Facilitate team interactions related to the process.
- Defect tracking - Making sure every defect has traceability back to the source.

Telelogic Synergy

Rational Synergy is a software tool that provides software configuration management (SCM) capabilities for all artifacts related to software development including source code, documents and images as well as the final built software executables and libraries. Rational Synergy also provides the repository for the Rational change management tool known as Rational Change. Together these two tools form an integrated configuration management and change management environment that is used in software development organizations that need controlled SCM processes and an understanding of what is in a build of their software.

The name *Synergy* refers to its database level integration with Change Management that provides views into what is in a build in terms of defects

History

Synergy began life in 1988 as a research project for computer-aided software engineering by software developer Pete Orelup at Computers West of Irvine California. Computers West was supporting itself through contract software development and an application for finance and insurance at automobile dealerships on the Pick OS, and probably had fewer than 10 employees.

In 1989, the company decided to pursue development of a software configuration management and version-control product, renamed itself CaseWare, and hired three more developers, Alan Wright, Kris Meissner, and Greg Holmberg. The system was re-imagined as a platform for building SCM systems running on Unix (Sun Solaris).

For such extreme configuration by customers, it was decided that a compiled language such as C++ was not sufficiently flexible, reliable, and productive, and so a new programming language was created, Accent. Accent has many features similar to Java, but pre-dates it by five years. It has a compiler that compiles to machine-independent byte-codes, and a virtual machine execution environment with automatic memory management. Except for the compiler and execution environment, the entire Amplify Control product was written in the Accent language, including a scalable, networked client-server architecture and use of a SQL database with a schema flexible enough to allow customer extension of the built-in data types in Accent without changes to the physical schema.

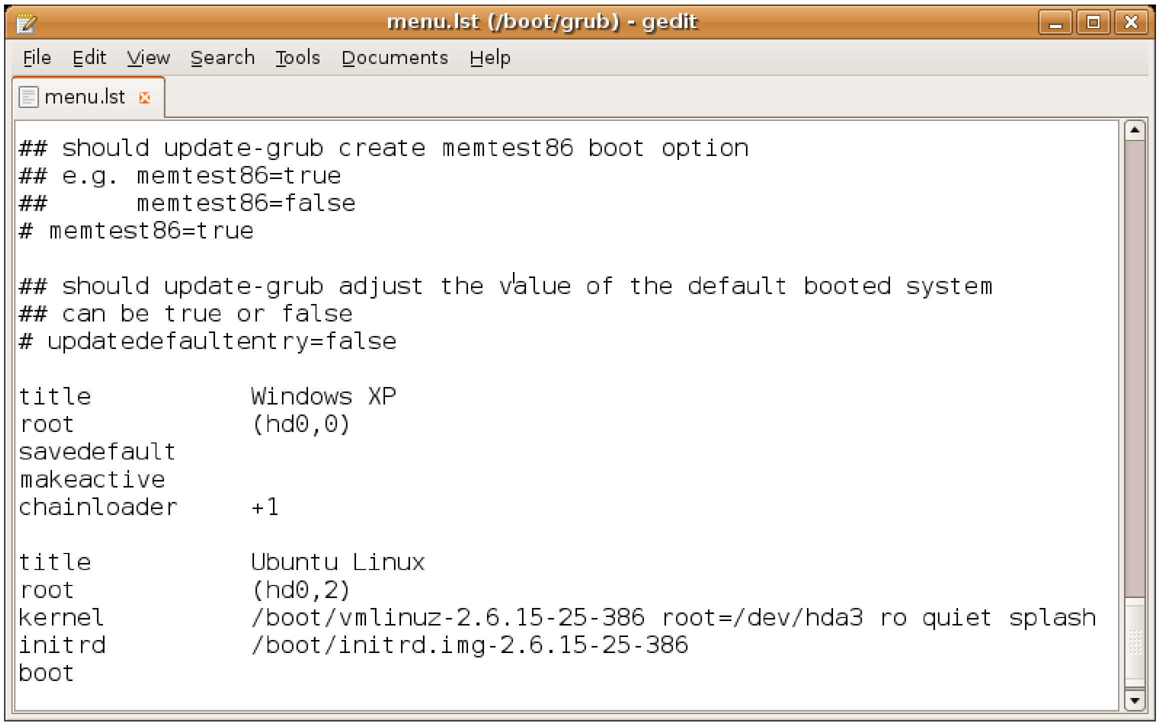
The system also included an automated, distributed build and continuous integration system, much like today's Maven and Hudson tools. The product was first released in 1990. Later a bug-tracking system was also built on the platform.

The company was somewhat successful, but lacked experienced leadership and started to lose market-share to ClearCase. In 1991 the company was nearly broke and the original developers walked out en masse. A new CEO, John Wark, was brought in, and the company was relaunched, although without the developers. Both the company and the product were renamed from CaseWare to Continuus Software in 1993. On July 29th, 1999 Continuus Software announced a public offering listing its stock on the NASDAQ Stock Market. In October 2000, the Swedish software company Telelogic, agreed to purchase Continuus software in a deal worth $42 million. Under Telelogic, Continuus was renamed to Synergy. In 2008 IBM Rational announced that it had purchased Telelogic. Rational Synergy is now part of the IBM Rational family of SCM tools.

Chapter 14

Configuration File and AUTOEXEC.BAT

Configuration file



menu.lst (/boot/grub) - gedit
File Edit View Search Tools Documents Help
menu.lst

```
## should update-grub create memtest86 boot option
## e.g. memtest86=true
##      memtest86=false
# memtest86=true

## should update-grub adjust the value of the default booted system
## can be true or false
# updatedefaultentry=false

title           Windows XP
root            (hd0,0)
savedefault
makeactive
chainloader     +1

title           Ubuntu Linux
root            (hd0,2)
kernel          /boot/vmlinuz-2.6.15-25-386 root=/dev/hda3 ro quiet splash
initrd          /boot/initrd.img-2.6.15-25-386
boot
```

A configuration file for GNU GRUB being modified with gedit. This file contains a list of Operating Systems, which GNU GRUB reads and presents to the user as a menu.

In computing, **configuration files**, or **config files** configure the initial settings for some computer programs. They are used for user applications, server processes and operating system settings. The files are often written in ASCII (rarely UTF-8) and line-oriented, with lines terminated by a newline or carriage return/line feed pair, depending on the operating system. They may be considered a simple database.

Some applications provide tools to create, modify, and verify the syntax of their configuration files; these sometimes have graphical interfaces. For other programs, system administrators may be expected to create and modify files by hand using a text editor. For server processes and operating-system settings, there is often no standard tool, but operating systems may provide their own graphical interfaces such as YaST or debconf.

Some computer programs only read their configuration files at startup. Others periodically check the configuration files for changes. Users can instruct some programs to re-read the configuration files and apply the changes to the current process, or indeed to read arbitrary files as a configuration file. There are no definitive standards or strong conventions.

UNIX/Linux

Across the Unix variants hundreds of configuration-file formats exist. Each application or service may have a unique format. Historically, Unix operating system settings were often modified only by editing configuration files. Almost all formats allow entries to be disabled by prepending a special comment character, turning that command line into a comment.

The configuration files on Unix-type operating systems are traditionally documented using manpages, though other forms of online help are also used. In many cases the default configuration files distributed with a program contain extensive internal documentation in the form of comments. It is rare for a file to be completely undocumented, except in cases where a graphical configuration tool is the preferred method of configuring a program.

Unix user applications often create a file or directory in the home directory of the user upon startup. To hide the file or directory from casual listing of the contents of the home directory, the name of the file or directory is prepended with a period, giving rise to the nickname "dotfile" or "dot file". Server processes often use configuration files stored in `/etc`, but they may also use their installation directory or a location defined by the system administrator.

Configuration files also do more than just modify settings, they often (in the form of an "rc file") run a set of commands upon startup (for example, the "rc file" for a shell might instruct the shell to change directories, run certain programs, delete or create files —- many things which do not involve modifying variables in the shell itself and so were not in the shell's dotfiles); according to the Jargon File, this convention is borrowed from "runcom files" on the CTSS operating system. This functionality can and has been extended for programs written in interpreted languages such that the configuration file is actually another program rewriting or extending or customizing the original program; Emacs is the most prominent such example. The "rc" naming convention of "rc files" was inspired by the "runcom" facility mentioned above and does not stand for "resource configuration" or "runtime configuration" as is often wrongly guessed.

On UNIX variants dot files remain "hidden" from listing by default. On Mac OS X these files are sometimes called "hidden files" although other mechanisms exist on Mac OS X to hide a file from view in various tools. The Explorer interface of Microsoft Windows XP does not allow the user to rename a file with an initial '.' though it does allow access to such files, and Windows' Notepad program does allow files to be saved with such names. Where Unix programs that use dotfiles are ported to Windows, they are sometimes modified to accept some other naming convention; for example, GNU Emacs permits its configuration file to be named `_emacs` instead of `.emacs`.

IBM's AIX uses an Object Data Manager (ODM) database to store some system settings, some of which need to be available at boot time.

Microsoft DOS

DOS primarily relies on two files called `CONFIG.SYS` and `AUTOEXEC.BAT`. These were retained up to Windows 98SE, but were not strictly required to run Windows applications.

Microsoft Windows

The Microsoft Windows family of operating systems and their attendant applications utilize a similar system of configuration files. Windows 3.0 had an API for INI files (from "initialization"). Many Windows programs abandoned configuration files to use the Windows Registry to store information.

IBM OS/2

IBM's OS/2 uses a binary format, also with a .INI suffix, but this differs from the Windows versions. It contains a list of lists of untyped key-value pairs. Two files control system-wide settings: OS2.INI and OS2SYS.INI. Application developers can choose whether to use them or create a specific file for their applications.

Configuration languages

Many language specifications have been created specifically to describe and retain configurations. These are frequently not Turing complete (nor need to be, by definition). A notable exception is Lua, which started out specifically as a configuration language for use in other programs. It evolved into a complete programming language, but retains a phrasing that allows configuration descriptions to be read directly into a native, stateful, tabulated set of variable-key pairings accessible to other programs (via a library), as well as allowing (self or external) invocation of commands to augment configuration activities.

The class includes all markup languages. The trend in the increase of XML and YAML (among other formats) for use as configuration-file formats is at least partially attributable to the increase in popularity of open source and platform neutral software applications and libraries. Moreover, the specifications describing these formats are routinely made available to the public, thus increasing the availability of parsers and emitters across programming languages.

AUTOEXEC.BAT

`AUTOEXEC.BAT` is a system file found originally on DOS-type operating systems. It is a plain-text batch file that is located in the root directory of the boot device. The name of the file stands for "automatic execution", which describes its function in automatically executing commands on system startup; the portmanteau was coined in response to the 8.3 filename limitations of the FAT file system family.

Usage

`AUTOEXEC.BAT` is read upon startup by all versions of DOS, including MS-DOS version 7.x as used in Windows 95 and Windows 98. Windows Me only parses environment variables as part of its attempts to reduce legacy dependencies, but this can be worked around.

Under DOS, the file is executed once the operating system has booted and after the `CONFIG.SYS` file has been processed. Windows NT and its descendants Windows XP and Windows Vista parse `AUTOEXEC.BAT` when a user logs on. As with Windows Me, anything other than setting environment variables is ignored. Unlike `CONFIG.SYS`, the commands in `AUTOEXEC.BAT` can be entered at the interactive command line interpreter. They are just standard commands that the computer operator wants to be executed automatically whenever the computer is started, and can include other batch files.

`AUTOEXEC.BAT` is most often used to set environment variables such as keyboard, soundcard, printer, and temporary file locations. It is also used to initiate low level system utilities, such as the following:

- Virus scanners
- Disk caching software - `SMARTDRV.EXE` from Microsoft the most common
- Mouse drivers
- Keyboard drivers
- CD drivers
- Miscellaneous other drivers

Example

In early versions of DOS, `AUTOEXEC.BAT` was by default extremely simple. The `date` and `time` commands were necessary as early PC and XT class machines did not have a battery backed-up Real Time Clock as default.

```
echo off
```

```
cls
date
time
ver
```

In non US environments the keyboard driver (like KEYBFR for the french keyboard) was also included. Later versions were often much expanded with numerous third party device drivers. The following is a basic DOS 5.x type `AUTOEXEC.BAT` configuration, consisting only of essential commands:

```
@echo off
prompt $P$G
PATH=C:\DOS;C:\WINDOWS
set TEMP=C:\TEMP
set BLASTER=A220 I7 D1 T2
lh smartdrv.exe
lh doskey
lh mouse.com /Y
win
```

This configuration sets common environment variables, loads the disk cache SmartDrive on line six, places common directories into the default path, and initializes the DOS mouse / keyboard drivers, before starting Windows. The `prompt` command sets the command prompt to "C:\>" instead of simply "C>".

In general, .SYS files were called in `CONFIG.SYS`, and .EXE programs such as the popular disk caching software *SmartDrive* provided by Microsoft with MS-DOS 5x, were loaded in the `AUTOEXEC.BAT` file. Some devices, such as mice, could be loaded either as a .SYS file in `CONFIG.SYS`, or as a .COM in `AUTOEXEC.BAT`, depending upon the manufacturer.

Lines prefixed with the string "REM" are comments (remarks) and are not run as part of `AUTOEXEC.BAT`. The "REM" lines are used for comments or to temporarily disable drivers (e.g. for a CD-ROM). An alternative, though less common, method for commenting is using double colons (**::**).

In MS-DOS 6 and higher, a DOS boot menu is configurable. This can be of great help to users who wish to have optimized boot configurations for various programs, such as DOS games and Windows. *(continued from CONFIG.SYS article)*

```
@echo off
prompt $P$G
PATH=C:\DOS;C:\WINDOWS
set TEMP=C:\TEMP
set BLASTER=A220 I7 D1 T2
goto %CONFIG%
:WIN
 lh smartdrv.exe
 lh mouse.com /Y
 win
```

```
goto END
:XMS
 lh smartdrv.exe
 lh doskey
 goto END
:END
```

The `goto %CONFIG%` line informs DOS to look up menu entries that were defined within `CONFIG.SYS`. Then, these profiles are named here and configured with the desired specific drivers and utilities. At the desired end of each specific configuration, a `goto` command redirects DOS to the `:END` section. Lines after `:END` will be used by all profiles.

Issues

One of the problems with the versions of Windows that ran on top of DOS, was a lack of conventional memory. This was due to the archaic design of the original x86 processor, which was originally only able to address 1024kB, or an effective 640kB of memory. While this was later extended with new processor modes, DOS was not able to load low level `AUTOEXEC.BAT` type drivers into extended memory.

Users were therefore presented with the baffling situation, of potentially having 8192K of physical memory, but were not able to run software that required a mere 512K of memory, because the DOS drivers in the `AUTOEXEC.BAT` file, especially CD-ROM and disk compression drivers, had taken up too much conventional memory.

Users were left to experiment with `LOADHIGH/LH` (MS-DOS) or `HILOAD` (DR-DOS) commands, based upon the EMM386 memory manager loaded in the `CONFIG.SYS` files, in order to try to move drivers from the 640K region, into the upper memory area or the high memory area. Lack of conventional memory proved to be a particular issue for gamers, and generated numerous baffled calls to support desks. Many gamers were forced to maintain several boot disks, each with game specific PC configurations.

Resolving driver and conventional memory issues has been cited as a key reason for adoption of the Windows based Direct-X gaming interface, which could access the entire physical memory of the PC, and relied upon Windows drivers to access hardware. This was also solved by using 32-bit DOS programs and standard VESA drivers for graphics.

Dual-booting DOS and Win 9x

When installing Windows 95 over a preexisting DOS/WINDOWS install, `CONFIG.SYS` and `AUTOEXEC.BAT` are renamed to `CONFIG.DOS` and `AUTOEXEC.DOS`. This is intended to ease dual booting between Windows 9.x and DOS. When booting into DOS, they are temporarily renamed `CONFIG.SYS` and `AUTOEXEC.BAT`. Backups of the Win95 versions are made as `.W40` files.

Windows 9x also installs a fake `MSDOS.SYS` file. This file contains some switches that designate how the system will boot, one of which controls whether or not the system automatically goes into Windows. This "BootGUI" option must be set to "0" in order to boot to a DOS prompt. By doing this, the system's operation essentially becomes that of a DOS/Windows pairing like with earlier Windows versions. Windows can be started as desired by typing "WIN" at the DOS prompt.

When installing Caldera DR-DOS 7, the Windows version retains the name `AUTOEXEC.BAT`, while the file preferred by the DR DOS loader is named `AUTODOS7.BAT`. It also differentiates the Config.sys file by using the name `DCONFIG.SYS`.

OS/2 / NT

On Windows NT and its derivatives, Windows 2000, Windows Server 2003 and Windows XP, the equivalent file is called `AUTOEXEC.NT` and is located in the `%SystemRoot%\system32` directory. The file is not used during the operating system boot process; it is executed when the MS-DOS environment is started, which occurs when an MS-DOS application is loaded.

The `AUTOEXEC.BAT` file may often be found on Windows NT, in the root directory of the boot drive. Windows only considers the "SET" and "PATH" statements which it contains, in order to define environment variables global to all users. Setting environment variables through this file may be interesting if for example MS-DOS is also booted from this drive (this requires that the drive be FAT) or to keep the variables across a reinstall. This is an exotic usage today so this file usually remains empty. The TweakUI applet from the *PowerToys* collection allows to control this feature (*Parse Autoexec.bat at logon*).

Chapter 15

Autorun.Inf

An **`autorun.inf`** file is a text file that can be used by the AutoRun and AutoPlay components of Microsoft Windows Operating systems. For the file to be discovered and used by these components, it must be located in the root directory of a volume. As Windows has a case-insensitive view of filenames, the autorun.inf file can be stored as AutoRun.inf or Autorun.INF or any other case combination.

The AutoRun component was introduced in Windows 95 as a way of reducing support costs. AutoRun enabled application CD-ROMs to automatically launch a program which could then guide the user through the installation process. By placing settings in an `autorun.inf` file, manufacturers could decide what actions were taken when their CD-ROM was inserted. The simplest autorun.inf files have just two settings: one specifying an icon to represent the CD in Windows Explorer (or "My Computer") and one specifying which application to run.

Extra settings have been added in successive versions of Windows to support AutoPlay and add new features.

The autorun.inf file

`autorun.inf` is an ASCII text file located in the root folder of a CD-ROM or other *volume* device medium. The structure is that of a classic Windows .ini file, containing information and commands as "key=value" pairs, grouped into sections. These keys specify:

- The name and the location of a program to call when the medium is inserted (the "AutoRun task").
- The name of a file that contains an icon that represents the medium in Explorer (instead of the standard drive icon).
- Commands for the menu that appears when the user right-clicks the drive icon.
- The default command that runs when the user double-clicks the drive icon.
- Settings that alter AutoPlay detection routines or search parameters.
- Settings that indicate the presence of drivers.

Inf handling

The mere existence of an autorun.inf file on a medium does not mean that Windows will automatically read it or use its settings. How an inf file is handled depends on the version of Windows in use, the volume drive type and certain Registry settings.

Assuming Registry settings allow, the following autorun.inf handling takes place:

- Windows versions prior to Windows XP

  On any drive type, the autorun.inf is read, parsed and instructions followed immediately and silently.
  The "AutoRun task" is the application specified by the `open` or `shellexecute` keys. If an AutoRun task is specified it is executed immediately without user interaction.

- Windows XP, prior to Service Pack 2

  Introduction of AutoPlay.
  Drives of type DRIVE_CDROM invoke AutoPlay if no autorun.inf file is found.
  Drives of type DRIVE_REMOVABLE do not use the autorun.inf file. Any discovered removable media are handled by AutoPlay. All other handling is as before.

- XP Service Pack 2 and up (includes Vista)

  Drives of type DRIVE_FIXED are now handled by AutoPlay. Any specified AutoRun task appears as an option within the AutoPlay dialog together with any text specified by the optional `action` key.
  Drives of type DRIVE_REMOVABLE now use autorun.inf but continue to be handled by AutoPlay. Any specified AutoRun task needs to be paired with the mandatory `action` key to appear as an option within the AutoPlay dialog. Otherwise the AutoRun task is omitted.
  All other handling is as before.

- Vista and later

  The AutoRun task is no longer automatically and silently executed on any drive type. All volumes are handled by AutoPlay which, by default, will present an appropriate dialog to the user.

- Windows 7

  For all drive types, *except* DRIVE_CDROM, the only keys available in the [autorun] section are `label` and `icon`. Any other keys in this section will be

ignored. Thus only CD and DVD media types can specify an AutoRun task or affect double-click and right-click behaviour.
There is a patch available, `KB971029` for Windows XP and later, that will change AutoRun functionality to this behaviour.

A simple example

This simple autorun.inf file specifies `setup.exe` as the application to run when AutoRun is activated. The first icon stored within the setup.exe itself will represent the drive in Explorer:

```
[autorun]
open=setup.exe
icon=setup.exe,0
label=My install CD
```

Sections

Following are the sections and keys allowed in a valid autorun.inf. There also exist architecture specific section types for systems such as Windows NT 4 running on RISC. However these are long outdated and not described here.

[autorun]

The `autorun` section contains the default AutoRun commands. An autorun.inf file must contain this section to be valid. Keys allowed are:

`action=`*`text`*
`action=@[`*`filepath`*`\]`*`filename`*`,-`*`resourceID`*

Windows XP SP2 or later; drives of type DRIVE_REMOVABLE and DRIVE_FIXED
Specifies text used in the AutoPlay dialog to represent the program specified in the `open` or `shellexecute` keys. The text is expressed as either text or as a resource reference. The `icon` is displayed next to the text. This item is always first in the AutoPlay dialog and is always selected by default.
If the (action) key does not appear on drives of type:
DRIVE_REMOVABLE
the AutoPlay dialog appears but without additional menu items. Essentially, the AutoRun task is omitted. This makes the action key mandatory for drives of this type.
DRIVE_FIXED
default text is created and used in the AutoPlay dialog.
On all other drive types the key is ignored.

`icon=`*`iconfilename`*`[,`*`index`*`]`

The name of an file resource containing an icon. This icon replaces the standard drive icon in Windows Explorer. This file must be in the same directory as the file specified by the `open` key.

`label=`*`text`*

Specifies a text label representing the drive in Windows Explorer.

`open=[`*`exepath`*`\]exefile [param1 [param2 ...]]`

Specifies the path, file name and optional parameters to the application that AutoRun launches when a user inserts a disc in the drive. It is the `CreateProcess` function that is called by AutoRun.

`shellexecute=[`*`filepath`*`\]filename [`*`param1`*` [`*`param2`*` ...]]`

Windows 2000, Windows ME or later

Similar to open, but using file association information to run the application. The file name can therefore be an executable or a data file. It is the `ShellExecuteEx` function that is called by AutoRun.

`UseAutoPlay=1`

Windows XP or later; drives of type DRIVE_CDROM

Use AutoPlay rather than AutoRun with CD-ROMs. The action taken on CD-ROM insertion will depend on the version of Windows being used.

On versions of Windows earlier than XP, this key has no effect and actions specified by `open` or `shellexecute` are performed.

On Windows XP and later, the user will be presented with the AutoPlay dialog and any actions specified by `open` or `shellexecute` are ignored.

`shell\`*`verb`*`\command=[`*`exepath`*`\]`*`exefile`*` [`*`param1`*` [`*`param2`*` ...]]`

Adds a custom command to the drive's shortcut menu. *verb* is a string with no embedded spaces. *verb* is also the text that will appear in the shortcut menu unless specifically altered to some other text. See below for an example.

`shell\`*`verb`*`=`*`menu text`*

Optionally specify the text displayed in the shortcut menu for the *verb* above. Use an ampersand (&) to select a hotkey for the menu. See below for an example.

`shell=`*`verb`*

Defines the menu command referred to by `shell\`*`verb`* as the default command in the shortcut menu. The default command is the command executed when the drive icon is double-clicked. If missing, the default menu item will be "AutoPlay", which launches the application specified by the `open` entry.

Example:

```
shell\readme\command=notepad readme.txt
shell\readme=Read &Me
shell=readme
```

[Content]

The `Content` section allows authors to communicate the type and intent of content to AutoPlay without AutoPlay having to examine the media.

Valid keys are: `MusicFiles`, `PictureFiles`, `VideoFiles`. Each key can be set to indicate true or false values and values are not case sensitive.

true (or 1, y, yes, t)

display the handlers associated with that content type

false (or 0, n, no, f)

do not display the handlers associated with that content type

Example:

```
[Content]
MusicFiles=Y
PictureFiles=0
VideoFiles=false
```

[ExclusiveContentPaths]

Limits AutoPlay's content search to only those folders listed, and their subfolders. The folder names are always taken as absolute paths (a path from the root directory of the media) whether or not a leading slash is used.

Example:

```
[ExclusiveContentPaths]
\pictures
\music
more music\special
```

[IgnoreContentPaths]

AutoPlay's content search system will not scan the folders listed, nor their subfolders. `IgnoreContentPaths` takes precedence over `ExclusiveContentPaths` so if a path given in a [IgnoreContentPaths] section is a subfolder of a path given in an [ExclusiveContentPaths] section it is still ignored.

Example:

```
[IgnoreContentPaths]
pictures
\music
more music\special
```

[DeviceInstall]

This section is used to indicate where driver files may be located. This prevents a lengthy search through the entire contents of a CD-ROM. Windows XP will fully search:

- floppy disks in drives A or B
- CD/DVD media less than 1 GB in size.

without this section present. All other media should include this section to have Windows XP autodetect any drivers stored on that media.

The section is not used with AutoRun or AutoPlay and is only referred to during a driver installation phase. The only valid key is:

```
      DriverPath=directorypath
```

which lists a path Windows will search for driver files. All subdirectories of that path are also searched. Multiple key entries are allowed.

If no `DriverPath` entry is provided in the `[DeviceInstall]` section or the `DriverPath` entry has no value, then that drive is skipped during a search for driver files.

Example:

```
[DeviceInstall]
DriverPath=drivers\video
DriverPath=drivers\audio
```

Chapter 16

CONFIG.SYS

CONFIG.SYS is the primary configuration file for the DOS and OS/2 operating systems. It is a special file that contains setup or configuration instructions for the computer system.

Usage

The commands in this file configure DOS for use with devices and applications in the system. The commands also set up the memory managers in the system. After processing the CONFIG.SYS file, DOS proceeds to load and execute the command shell specified in the `shell=` line of CONFIG.SYS, or COMMAND.COM if there is no such line. The command shell in turn is responsible for processing the AUTOEXEC.BAT file.

CONFIG.SYS is composed mostly of `name=value` statements which look like variable assignments. In fact these will either define some tunable parameters often resulting in reservation of memory, or load files, mostly TSRs and device drivers, into memory.

In DOS, CONFIG.SYS is located in the root directory of the drive from which DOS was booted. In some versions of DOS it may have an alternate filename, e.g. **FDCONFIG.SYS** in FreeDOS, or **DCONFIG.SYS** in some versions of DR-DOS.

Both CONFIG.SYS and AUTOEXEC.BAT can be found included in the root folder of Windows 95, and Windows 98 boot drives, as they are based on DOS. Typically these files are left empty, with no content, as they are not strictly required to run Windows programs from these versions.

Windows ME does not even parse the CONFIG.SYS file during the Windows boot process, loading those settings from the Windows Registry instead: HKLM\System\CurrentControlSet\Control\SessionManager\Environment

Examples

MS-DOS

Example CONFIG.SYS for MS-DOS with Windows 3.xx:

```
device=c:\dos\himem.sys
device=c:\dos\emm386.exe ram
dos=high,umb
devicehigh=c:\windows\mouse.sys
devicehigh=c:\dos\setver.exe
country=044,437,c:\dos\country.sys
shell=c:\dos\command.com c:\dos /e:512 /p
```

- The first line loads the himem.sys driver that enables DOS to use the high memory area.
- The second line loads the EMM386 memory manager, which emulates expanded memory. The command line argument *ram* allows the use of the upper memory area. Another argument that can be given to emm386.exe is *noems*, which allows use of the upper memory area without emulating expanded memory. The noems switch also frees up more umb blocks.
- The third line causes DOS to use high memory and upper memory when possible, freeing up more conventional memory for applications to use.
- Lines four to five load device drivers into the upper memory area: the first is a mouse driver from Microsoft; the second is a compatibility program.
- Line six sets localisation settings such as setting the country to the UK (code 044) and setting code page 437.
- The final line sets the shell to the default shell, command.com, and starts it with c:\dos as the working directory, with an environment size of 512 bytes, and the /p indicates that it is the parent process and therefore cannot be shut down by using the exit command.

As of MS-DOS version 6, an optional DOS boot menu was configurable. With this, the user could configure any number of boot configurations and choose one on start-up. This was of great use because various DOS applications preferred different settings for optimal functionality. In particular, with Windows 9x, it was best to load as few 16-bit DOS drivers and utilities as possible.

Example CONFIG.SYS with MS-DOS 6+ boot menu:

```
[menu]
 menuitem=WIN, Windows
 menuitem=XMS, DOS with only Extended Memory
 menudefault=WIN, 10
[common]
 device=c:\dos\himem.sys
 dos=high,umb
 shell=c:\dos\command.com c:\dos /e:512 /p
 country=044,437,c:\dos\country.sys
```

```
[WIN]
 device=c:\dos\emm386.exe ram
 devicehigh=c:\windows\mouse.sys
 devicehigh=c:\dos\setver.exe
[XMS]
 device=c:\dos\emm386.exe noems
```

The layout of the DOS boot menu is fairly self-explanatory. The "[menu]" section defines menu entries. The option, "menudefault", allows a default choice with a countdown timer before it starts up (10 seconds here). The "[common]" area holds lines that will start for every menu choice, while the later "[WIN]" and "[XMS]" areas are specific to each configuration.

The later boot file, AUTOEXEC.BAT, would receive the profile names and they could be separately configured there as well.

FreeDOS

Recent FDCONFIG.SYS or CONFIG.SYS of FreeDOS:

```
screen=0x12
device=c:\dos\himem.exe
device=c:\dos\emm386.exe
dos=high,umb
country=044,437,c:\dos\country.sys
shell=c:\dos\freecom.com c:\dos /e:512 /p
```

In general .sys files are called in config.sys, as above, and .exe programs such as the version of the caching software SMARTDRIVE provided by Microsoft with MS-DOS 6.x, or LBACACHE of FreeDOS, are loaded in the autoexec.bat file. However, there are ways to load .SYS like files later from commandline as well as .EXE files from config file.

Issues

The system can still boot if these files are missing or corrupted. However, these two files are essential for the complete bootup process to occur with the DOS operating system. They contain information that is used to change the operating system for personal use. They also contain the requirements of different software application packages. A DOS system would require troubleshooting if either of these files became damaged or corrupted.

If `config.sys` does not contain a "shell" statement (or the file is corrupt or missing), DOS typically searches for `COMMAND.COM` in the root directory. If this is not found, the system will not start up.

Dual Booting DOS and Win 9X

When installing Windows 95 over a preexisting DOS/WINDOWS install, CONFIG.SYS and AUTOEXEC.BAT are renamed to CONFIG.DOS and AUTOEXEC.DOS. This is intended to ease dual booting between Windows 9X and DOS. When booting into DOS, they are temporarily renamed CONFIG.SYS and AUTOEXEC.BAT. Backups of the Win95 versions are made as .W40 files.

When Caldera DR DOS 7 is installed on a system already containing Windows 95, Windows' `CONFIG.SYS` and `AUTOEXEC.BAT` retain those names. DR DOS' startup files are installed as `DCONFIG.SYS` (a name already used in earlier versions of DR DOS) and `AUTODOS7.BAT`.

OS/2 / NT

OS/2 uses the CONFIG.SYS file extensively for setting up its configuration, drivers and environment before the graphical part of the system loads.

In the OS/2 subsystem of Windows NT, what appeared as CONFIG.SYS to OS/2 programs was actually stored in the registry.

There are many undocumented or poorly documented CONFIG.SYS statements used by OS/2.

Chapter 17

Fstab

The **fstab** (/etc/fstab) (or *file systems table*) file is a system configuration file commonly found on Unix systems. The fstab file typically lists all available disks and disk partitions, and indicates how they are to be initialized or otherwise integrated into the overall system's file system. fstab is still used for basic system configuration, notably of a system's main hard drive and startup file system, but for other uses has been superseded in recent years by automatic mounting.

The fstab file is most commonly used by the `mount` command, which reads the fstab file to determine which options should be used when mounting the specified device. It is the duty of the system administrator to properly create and maintain this file.

The file has other names on some versions of Unix, eg it is `/etc/vfstab` on Solaris.

Modern use

Traditionally, the fstab was only read by programs, and not automatically written (it is instead manually written by the sysadmin). However, some administration tools can automatically build and edit fstab, or act as graphical editors for it, such as the Kfstab graphical configuration utility available for KDE.

Modern Linux systems use udev as an automounter to handle hot swapping devices instead of rewriting fstab file on the fly, and thus fstab is less important than in the past. Programs such as pmount allow users to mount and unmount filesystems without a corresponding fstab entry; traditional Unix has always allowed privileged users to mount or unmount without an fstab entry.

Example

The following is an example of an fstab file on a typical Linux system:

```
# device name   mount point     fs-type      options
dump-freq pass-num
```

```
LABEL=/         /               ext3         defaults                1
1
/dev/hda6       swap            swap         defaults                0
0
none            /dev/pts        devpts       gid=5,mode=620          0
0
none            /proc           proc         defaults                0
0
none            /dev/shm        tmpfs        defaults                0
0

# Removable media
/dev/cdrom      /mount/cdrom    udf,iso9660  noauto,owner,kudzu,ro   0
0
/dev/fd0        /mount/floppy   auto         noauto,owner,kudzu      0
0

# NTFS Windows XP partition
/dev/hda1       /mnt/WinXP      ntfs-3g
quiet,defaults,locale=en_US.utf8,umask=0    0 0

# Partition shared by Windows and Linux
/dev/hda7       /mnt/shared     vfat         umask=000              0 0

# mounting tmpfs
tmpfs           /mnt/tmpfschk   tmpfs        size=100m              0 0

# mounting cifs
//pingu/ashare  /store/pingu    cifs
credentials=/root/smbpass.txt 0 0

#mounting NFS
pingu:/store    /store          nfs          rw                     0 0
```

The columns are as follows:

1. The *device name* or other means of locating the partition or data source.
2. The *mount point*, where the data is to be attached to the filesystem.
3. The *filesystem type*, or the algorithm used to interpret the filesystem.
4. *Options*, including if the filesystem should be mounted at boot. (*kudzu* is an option specific to Red Hat and Fedora Core.)
5. *dump-freq* adjusts the archiving schedule for the partition (used by dump).
6. *pass-num* Controls the order in which fsck checks the device/partition for errors at boot time. The root device should be 1. Other partitions should be either 2 (to check after root) or 0 (to disable checking for that partition altogether).

A value of zero in either of the last 2 columns disables the corresponding feature. For the whitespace character in paths the character code "\040" is used.

Options common to all filesystems

As the filesystems in /etc/fstab will eventually be mounted using mount(8) it isn't surprising that the options field simply contains a comma-separated list of options which will be passed directly to mount when it tries to mount the filesystem.

The options common to all filesystems are:

`atime / noatime / relatime / strictatime (Linux-specific)`

The Unix stat structure records when files are last accessed (atime), modified (mtime), and created (ctime). One result is that atime is written every time a file is *read,* which has been heavily criticized for causing performance degradation and increased wear. However, atime is used by some applications and desired by some users, and thus is configurable as atime (update on access), noatime (do not update), or (in Linux) relatime (update atime if older than mtime). Through Linux 2.6.29, atime was the default; as of 2.6.30 (9 June 2009), relatime is the default.

`auto / noauto`

With the auto option, the device will be mounted automatically at bootup or when the mount -a command is issued. auto is the default option. If you don't want the device to be mounted automatically, use the noauto option in /etc/fstab. With noauto, the device can be only mounted explicitly.

`dev / nodev`

Interpret/do not interpret block special devices on the filesystem.

`exec / noexec`

exec lets you execute binaries that are on that partition, whereas noexec doesn't let you do that. noexec might be useful for a partition that contains no binaries, like /var, or contains binaries you don't want to execute on your system, or that can't even be executed on your system. Last might be the case of a Windows partition.

`ro`

Mount read-only.

`rw`

Mount the filesystem read-write. Again, using this option might alleviate confusion on the part of new Linux users who are frustrated because they can't write to their floppies, Windows partitions, or other media.

`sync / async`

How the input and output to the filesystem should be done. sync means it's done synchronously. If you look at the example fstab, you'll notice that this is the option used with the floppy. In plain English, this means that when you, for example, copy a file to the floppy, the changes are physically written to the floppy at the same time you issue the copy command.

`suid / nosuid`

Permit/Block the operation of suid, and sgid bits.

`user / users / nouser`

`user` permits any user to mount the filesystem. This automatically implies noexec, nosuid, nodev unless overridden. If `nouser` is specified, only root can mount the filesystem. If `users` is specified, every user in group *users* will be able to unmount the volume.

`owner` (This is Linux-specific)

Permit the owner of device to mount.

`defaults`

Use default settings. Default settings are defined per file system at the file system level. For ext3 file systems these can be set with the tune2fs command. The normal default for Ext3 file systems is equivalent to `rw,suid,dev,exec,auto,nouser,async`(no acl support). Modern Red Hat based systems set acl support as default on the root file system but not on user created Ext3 file systems. Some file systems such as XFS enable acls by default. Default file system mount attributes can be over ridden in /etc/fstab.

Filesystem specific options

There are many options for the specific filesystems supported by mount. Listed below are some of the more commonly used. The full list may be found in the documentation for mount. Note that these are for Linux; traditional UNIX-like systems have generally provided similar functionality but with slightly different syntax.

ext2

`check={none, normal, strict}`

Sets the fsck checking level.

`debug`

Print debugging info on each remount .

`sb=n`

n is the block which should be used as the superblock for the fs.

fat

`check={r[elaxed], n[ormal], s[trict]}`

Not the same as ext2, but rather deals with allowed filenames.

`conv={b[inary], t[ext], a[uto]}`

Performs DOS <---> UNIX text file conversions automatically.

`uid=n, gid=n`

Sets the user identifier, uid, and group identifier, gid, for all files on the filesystem.

`umask=nnn, dmask=nnn, fmask=nnn`

Sets the user file creation mode mask, umask, the same for directories only, dmask and for files only, fmask.

iso9660

`norock`

Disables Rock Ridge extensions.

Mounting all filesystems

```
mount -a
```

This command will mount all (not-yet-mounted) filesystems mentioned in fstab and is used in system script startup during booting. Note that this command will ignore all those entries containing "`noauto`" in the options section.

Chapter 18

INI File

The **INI file** format is a '***de facto'*** standard for configuration files. INI files are simple text files with a basic structure. They are commonly associated with Microsoft Windows, but are also used on other platforms. The use of the "INI file" has been deprecated in Windows in favor of the registry, and deprecated in .NET in favor of XML .config files. The name "INI file" comes from the filename extension usually used, "`.INI`", that stands for "**ini**tialization". Sometimes files using the INI file format will use a different extension, such as "`.CFG`", "`.conf`", or "`.TXT`".

Format

Properties

The basic element contained in an INI file is the *property*. Every property has a *name* and a *value*, delimited by an equals sign (`=`). The name appears to the left of the equals sign.

```
name=value
```

Sections

Properties may be grouped into arbitrarily named *sections*. The section name appears on a line by itself, in square brackets (`[` and `]`). All properties after the section declaration are associated with that section. There is no explicit "end of section" delimiter; sections end at the next section declaration, or the end of the file. Sections may not be nested.

```
[section]
```

Comments

Semicolons (`;`) indicate the start of a comment. Comments continue to the end of the line. Everything between the semicolon and the end of the line is ignored.

```
; comment text
```

Varying features

The INI file format is not well defined. Many programs support features beyond the basics described above. The following is a list of some common features, which may or may not be implemented in any given program.

Blank Lines

Some rudimentary programs do not allow blank lines. Every line must therefore be a section head, a property, or a comment.

Whitespace

Interpretation of whitespace varies. Most implementations ignore leading and trailing whitespace around the outside of the property name. Some even ignore whitespace within values (for example, making "host name" and "hostname" equivalent). Some implementations also ignore leading and trailing whitespace around the property value; others consider all characters following the equals sign (including whitespace) to be part of the value.

Quoted values

Some implementations allow values to be quoted, typically using double quotes and/or apostrophes. This allows for explicit declaration of whitespace, and/or for quoting of special characters (equals, semicolon, etc.). The standard Windows function GetPrivateProfileString supports this, and will remove quotation marks that surround the values.

Comments

Some software supports the use of the number sign (`#`) as an alternative to the semicolon for indicating comments.

In some implementations, a comment may begin anywhere on a line, including on the same line after properties or section declarations. In others, any comments must occur on lines by themselves.

Duplicate names

Most implementations only support having one property with a given name in a section. The second occurrence of a property name may cause an abort; the second occurrence may be ignored (and the value discarded); the second occurrence may override the first occurrence (discard the first value). Some programs use duplicate property names to implement multi-valued properties.

Interpretation of multiple section declarations with the same name also varies. In some implementations, duplicate sections simply merge their properties together, as if they occurred contiguously. Others may abort, or ignore some aspect of the INI file.

Order of sections and properties.

In most cases the order of properties in a section and the order of sections in a file is irrelevant, but implementations may vary.

Name/value delimiter

Some implementations allow a colon (`:`) as the name/value delimiter (instead of the equals sign).

Hierarchy

Most commonly, INI files have no hierarchy of sections within sections. Some files appear to have a hierarchical naming convention, however. For section A, subsection B, sub-sub-section C, property P and value V, they may accept entries such as `[A.B.C]` and `P=V` (Windows' `xstart.ini`), `[A\B\C]` and `P=V` (the IBM Windows driver file `devlist.ini`), or `[A]` and `B,C,P = V` (Microsoft Visual Studio file `AEMANAGR.INI`).

It is unclear whether these are simply naming conventions that an application happens to use in order to give the *appearance* of a hierarchy, or whether the file is being read by a module that actually presents this hierarchy to the application programmer.

Escape characters

Some implementations also offer varying support for an escape character, typically with the backslash (`\`). Some support "line continuation", where a backslash followed immediately by EOL (end-of-line) causes the line break to be ignored, and the "logical line" to be continued on the next actual line from the INI file. Implementation of various "special characters" with sequences escapes is also seen.

Common escape sequences

Sequence	**Meaning**
\\	\ (a single backslash, escaping the escape character)
\0	Null character
\a	Bell/Alert/Audible
\b	Backspace, Bell character for some applications
\t	Tab character
\r	Carriage return
\n	Newline
\;	Semicolon

\#	Number sign
\=	Equals sign
\:	Colon
\x*????*	Unicode character with hexadecimal codepoint corresponding to *????*

Example

Following is an example INI file for an imaginary program. It has two sections, one for the owner of the software, and one for a payroll database connection. Comments note who modified the file last, and why an IP address is used instead of a DNS name.

```
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
server=192.0.2.62     ; use IP address in case network name resolution
is not working
port=143
file = "payroll.dat"
```

Accessing INI files

Under Windows, the *Profile API* is the programming interface used to read and write settings from classic Windows .ini files. For example, the GetPrivateProfileString function retrieves a string from the specified section in an initialization file.

The following sample C program demonstrates reading property values from the above sample INI file (Let the name of configuration file be **dbsettings.ini**)

```
#include <Windows.h>
int main(int argc, _TCHAR *argv[])
{
  _TCHAR dbserver[1000];
  int dbport;
  GetPrivateProfileString("database", "server", "127.0.0.1", dbserver,
1000, "dbsettings.ini");
  dbport = GetPrivateProfileInt("database", "port", 143,
"dbsettings.ini");
  return 0;
}
```

File mapping

Initialization File Mapping creates a mapping between an INI file and the Registry. It was introduced with Windows NT and Windows 95 as a way to migrate from storing settings in classic .ini files to the new Windows Registry. File mapping traps the Profile API calls and, using settings from the `IniFileMapping` Registry section, directs reads and writes to appropriate places in the Registry.

Using the Example above, a string call could be made to fetch the *name* key from the *owner* section from a settings file called, say, *dbsettings.ini*. The returned value should be the string "John Doe":

```
GetPrivateProfileString("owner", "name", ... ,
"c:\\programs\\oldprogram\\dbsettings.ini");
```

INI mapping takes this Profile API call, ignores any path in the given filename and checks to see if there is a Registry key matching the filename under:

```
    HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\
      CurrentVersion\IniFileMapping
```

If this exists, it looks for an entry name matching the requested section. If an entry is found, INI mapping uses its value as a pointer to another part of the Registry. It then looks up the requested INI setting in that part of the Registry.

If no matching entry name is found and there is an entry under the `(Default)` entry name, INI mapping uses that instead. Thus each section name does not need its own entry.

HKEY_LOCAL_MACHINE\Software\...\IniFileMapping\dbsettings.ini

`(Default)`	@USR:Software\oldprogs\inisettings\all
`database`	USR:Software\oldprogs\inisettings\db

So, in this case the profile call for the [owner] section is mapped through to:

HKEY_CURRENT_USER\Software\oldprogs\inisettings\all

`name`	John Doe
`organization`	Acme Products

where the "`name`" Registry entry name is found to match the requested INI key. The value of "John Doe" is then returned to the Profile call. In this case, the @ prefix on the default prevents any reads from going to the dbsettings.ini file on disk. The result is that any settings not found in the Registry are not looked for in the INI file.

The "`database`" Registry entry does not have the @ prefix on the value; thus, for the [database] section *only*, settings in the Registry are taken first followed by settings in the dbsettings.ini file on disk.

Alternatives

Starting with Windows 95, Microsoft began strongly promoting the use of Windows registry over the INI file.

More recently, XML-based configuration files have become a popular choice for encoding configuration in text files. XML allows arbitrarily complex levels and nesting, and has standard mechanisms for encoding binary data. INI files are typically limited to two levels (sections and properties) and do not handle binary data well. Additionally, data serialization formats, such as JSON and YAML can serve as configuration formats. These latter formats can nest arbitrarily and represent objects of unlimited complexity, but don't have a lightweight syntax like the ini file.