Configuration Management in Computer Science

Lavonda Forsyth

First Edition, 2012

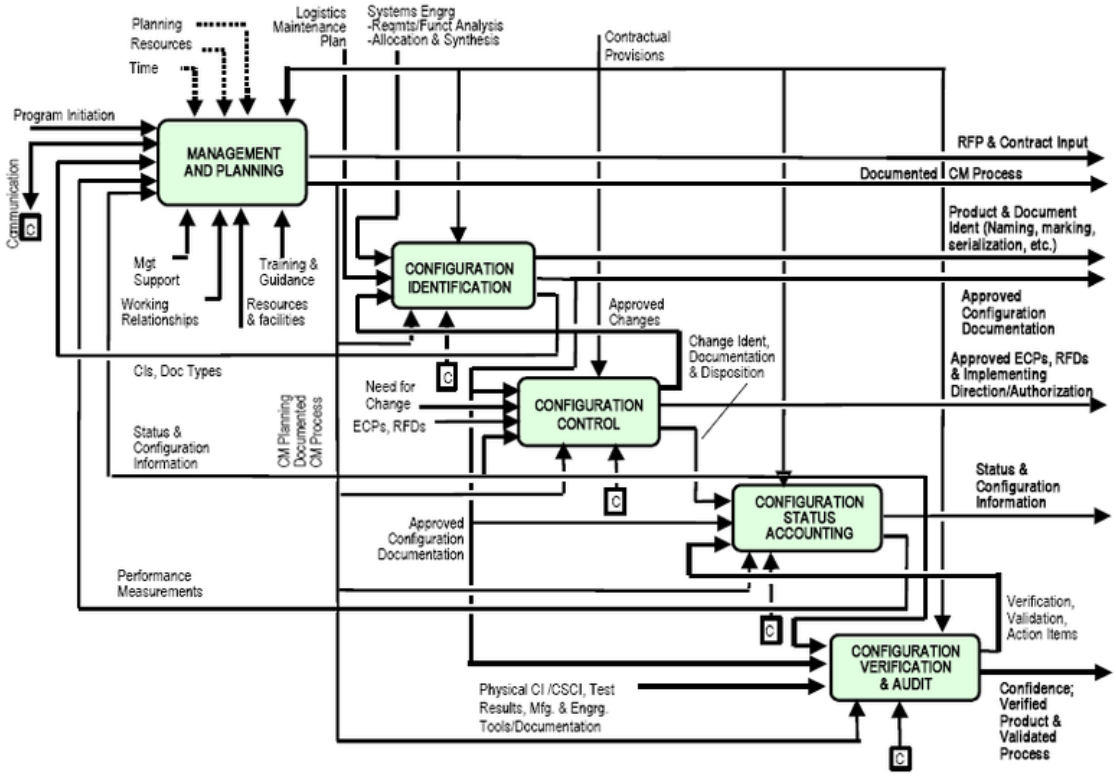
ISBN 978-81-323-4115-4

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

Table of Contents

Chapter 1

Configuration Management



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 2

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 3

Component Repository Management and Cfengine

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.

Cfengine

	Cfengine
Developer(s)	Mark Burgess
Stable release	3.0.5 / June 07, 2010
Operating system	Cross-platform
Platform	Unix, Linux, Windows
Type	Configuration management System administration Network management
License	GNU General Public License

Cfengine is a policy-based configuration management system written by Mark Burgess at Oslo University College. Its primary function is to provide automated configuration and maintenance of computers, from a policy specification.

The cfengine project was started in 1993 as a reaction to the complexity and non-portability of shell scripting for Unix configuration management, and continues today. The aim was to absorb frequently used coding paradigms into a declarative, domain-specific language that would offer self-documenting configuration.

Portability

Cfengine provides an operating system-independent interface to Unix-like host configuration. It requires some expert knowledge to deal with peculiarities of different operating systems, but has the power to perform maintenance actions across multiple Unix-like hosts. Cfengine can be used on Windows hosts, and is widely used for managing large numbers of Unix hosts that run heterogeneous operating systems e.g. Solaris, Linux, AIX, and HPUX. Statistics collected by the supporting commercial company Cfengine AS indicate hundreds of thousands of hosts running cfengine, with the largest sites recorded at 50,000.

Research-based

Shortly after its inception, cfengine inspired a field of research into automated configuration management. The cfengine project claims to attempt to place the problem of configuration management in a scientific framework. Its author Mark Burgess has developed a range of theoretical tools and results to talk about the problem, and has written several text books and monographs explaining them.

Commercialization

In June 2008 the company Cfengine AS was formed as a collaboration between author Mark Burgess, Oslo University College and the Oslo Innovation Centre in order to support users of cfengine. 20 April 2009, the company launched the first commercial version of Cfengine - Cfengine Nova. Current version of Cfengine Nova is 2.0.

Convergence

One of the main ideas in cfengine is that changes in computer configuration should be carried out in a *convergent* manner. This means that each change operation made by the agent should have the character of a fixed point. Rather than describing the steps needed to make a change, cfengine describes the final state in which one wants to end up. The agent then ensures that the necessary steps are taken to end up in this "policy compliant state". Thus, cfengine can be run again and again, whatever the initial state of a system, and it will end up with a predictable result. Cfengine supports the item of statistical compliance with policy, meaning that a system can never guarantee to be exactly in an ideal or desired state, rather one approaches (converges) towards the desired state by

best-effort, at a rate that is determined by the ratio of the frequency of environmental change to the rate of Cfengine execution.

Promise Theory

A model of distributed cooperation called Promise theory was developed to describe and re-design Cfengine after ten years of experience using the software. Cfengine major release 3 was ostensibly an implementation of this theory to the end of simplifying and rationalizing the Cfengine project. Burgess and Couch showed that Cfengine supported the notion of self-healing automation using promise theory.

User base

Cfengine is used in both large and small companies, as well as in many universities and governmental institutions. Sites as large as 50,000 machines are reported, while sites of several thousand hosts running under cfengine are common. According to statistics from the Cfengine Company, probably several million computers run Cfengine around the world.

Chapter 4

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

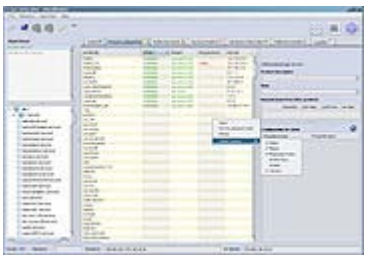
Some of its functionality varies by Windows versions:

- 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 5

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 6

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 7

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.

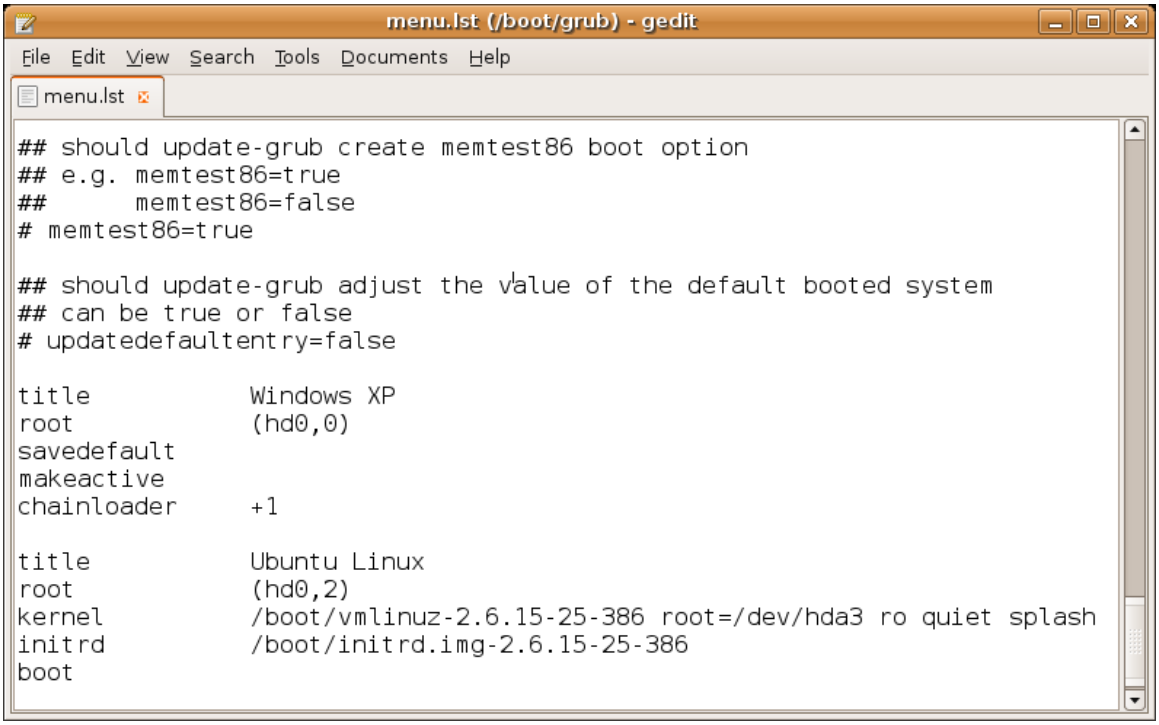
The system also included an automated, distributed build and continuous integration system, much like today's Maven and Hudson tools. The product was first released in 1990. Later a bug-tracking system was also built on the platform.

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 8

Configuration File and AUTOEXEC.BAT

Configuration file



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 9

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 10

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 11

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 12

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.

Chapter 13

Hosts (File)

The **hosts file** is a computer file used in an operating system to map hostnames to IP addresses. The hosts file is a plain-text file and is traditionally named *hosts*.

Purpose

The hosts file is one of several system facilities to assist in addressing network nodes in a computer network. It is a common part in a operating system's Internet Protocol (IP) implementation, and serves the function of translating human-friendly hostnames into numeric protocol addresses, called IP addresses, that identify and locate a host in an IP network.

In some operating systems, the host file content is used preferentially over other methods, such as the Domain Name System (DNS), but many systems implement name service switches (.e.g., nsswitch.conf) to provide customization. Unlike the DNS, the hosts file is under the direct control of the local computer's administrator.

File content

The hosts file contains lines of text consisting of an IP address in the first text field followed by one or more hostnames, each field is separated by white space (blanks or tabulation characters). Comment lines may be included; they are indicated by a hash character (#) in the first position of such lines. Entirely blank lines in the file are ignored. For example a typical hosts file may contain the following:

```
#This is an example of the hosts file
127.0.0.1  localhost loopback
::1        localhost
```

This example only contains entries for the loopback addresses of the system and their hostnames, a typical default content of the host file. The example illustrates that an IP address may have multiple hostnames, and that a hostname may be mapped to several IP addresses.

Location in the file system

The location of the hosts file in the file system hierarchy of operating systems varies.

Operating System	Version(s)	Location
Unix, Unix-like, POSIX		`/etc/hosts`
Microsoft Windows	3.1	`%Windir%\HOSTS.SAM`
	95, 98/98SE, Me	`%WinDir%\`
	NT, 2000, and 32-bit versions of XP, 2003, Vista, 7	`%SystemRoot%\system32\drivers\etc\`
	64-bit versions	`%SystemRoot%\system32\drivers\etc\` (Many sources, including several Microsoft support pages, will incorrectly state that the hosts file is at `%SystemRoot%\SysWOW64\drivers\etc\`)
Windows Mobile		Registry key under `\HKEY_LOCAL_MACHINE\Comm\Tcpip\Hosts`
Apple Macintosh	9 and earlier	System Folder: Preferences or System folder
	Mac OS X 10.0 – 10.1.5	(Added through NetInfo or niload)
	Mac OS X 10.2 and newer, iOS	`/private/etc/hosts` or, since `/etc` is a symbolic link to `/private/etc`, `/etc/hosts` (just like POSIX)
Novell NetWare		`SYS:etc\hosts`
OS/2 & eComStation		`"bootdrive":\mptn\etc\`
Symbian	Symbian OS 6.1–9.0	`C:\system\data\hosts`
	Symbian OS 9.1+	`C:\private\10000882\hosts`
MorphOS	NetStack	`ENVARC:sys/net/hosts`
Android		`/system/etc/hosts`
iOS	iOS 2.0 and newer	`/etc/hosts`

History

The ARPANET, the predecessor of the Internet, had no distributed host name database. Each network node maintained its own map of the network nodes as needed and assigned them names that were memorable to the users of the system. There was no method for ensuring that all references to a given node in a network were using the same name, nor was there a way to read the hosts file of another computer to automatically obtain a copy.

The small size of the ARPANET kept the administrative overhead small to maintain an accurate hosts file. Network nodes typically had one address and could have many names. As local area TCP/IP computer networks gained popularity, however, the maintenance of hosts files became a larger burden on system administrators as networks and network nodes were being added to the system with increasing frequency.

Standardization efforts, such as the format specification of the file *HOSTS.TXT* in RFC 952, and distribution protocols, e.g., the hostname server described in RFC 953, helped with these problems, but the centralized and monolithic nature of host files eventually necessitated the creation of the distributed Domain Name System.

Extended applications

In its function of resolving host names, the hosts file may be used to define any hostname or domain name for use in the local system. This may be used either beneficially or maliciously for various effects.

Redirecting local domains
: Some web service and intranet developers and administrators define locally defined domains in a LAN for various purposes, such as accessing the company's internal resources or to test local websites in development.

Internet resource blocking
: Specially crafted entries in the hosts file may be used to block online advertising, or the domains of known malicious resources and servers that contain spyware, adware, and other malware. This may be achieved by adding entries for those sites to redirect requests to another address that does not exist or to a harmless destination.

Various software applications exist that populate the hosts file with entries of undesirable Internet resources automatically.

Security issues

Because of its role in local name resolution, the hosts file represents an attack vector for malicious software. The file may be *hijacked*, for example, by adware, computer viruses, trojan horse software, and may be modified to redirect traffic from the intended destination to sites hosting content that may be offensive or intrusive to the user or the user's computer system. The widespread computer worm Mydoom.B blocked users from

visiting sites about computer security and antivirus software and also affected users' ability to access the Microsoft Windows Update website.

Chapter 14

Windows Registry

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

When first introduced with Windows 3.1, the Windows registry's primary purpose was to store configuration information for COM-based components. With the introduction of Windows 95 and Windows NT, its use was extended to tidy up the profusion of per-program INI files that had previously been used to store configuration settings for Windows programs.

Rationale

.INI files stored each program's settings into a text file, often located in a shared location that did not allow for user-specific settings in a multi-user scenario. By contrast, the Windows registry stores all application settings in one central repository and in a standardized form. This offers several advantages over INI files. Since accessing the registry does not require parsing, it may be read from or written to more quickly than an INI file. As well, strongly typed data can be stored in the registry, as opposed to the text information stored in INI files. Because user-based registry settings are loaded from a user-specific path rather than from a read-only system location, the registry allows multiple users to share the same machine, and also allows programs to work for less privileged users. Backup and restoration is also simplified as the registry can be accessed over a network connection for remote management/support, including from scripts, using the standard set of APIs, as long as the Remote Registry service is running and firewall rules permit this.

The registry has features that improve system integrity, as the registry is constructed as a database and offers database-like features such as atomic updates. If two processes attempt to update the same registry value at the same time, one process's change will

precede the other's and the overall consistency of the data will be maintained. Where changes are made to INI files, such race conditions can result in inconsistent data which doesn't match either attempted update. Windows Vista and Windows 7 provide transactional updates to the registry, extending the atomicity guarantees across multiple key and/or value changes, with traditional commit-abort semantics. (Note however that NTFS provides such support for the file system as well, so the same guarantees could, in theory, be obtained with traditional configuration files.)

Criticism

While offering improvements over application-specific .INI files, the organization and implementation of the registry also had potential problems:

- The registry duplicates much of the functionality of the file system.
- Centralizing configurations makes it more difficult to back up and recover individual applications.
- Installers and uninstallers may become more complicated if applications rely on Registry configuration settings that need to be created by installation applications because these Registry settings cannot be transferred by simply copying the application files that comprise the application. Use of the Registry by non-COM based applications is optional; .Net applications leverage a configuration file instead of the Registry. Some other operating systems (e.g., OS X) also support installation through simple file copy.
- Because information required for loading device drivers is stored in the registry, a damaged System registry may prevent a Windows system from booting successfully, since some device drivers won't be loaded, preventing the affected devices from working. Device drivers are stored in a key called HKLM\System\CurrentControlSet, which is a symbolic link that alternates between HKLM\System\CurrentControlSet001 and HKLM\System\CurrentControlSet002, thereby allowing the "last known configuration" boot option to provide a mechanism of reverting to the last configuration that successfully started the system.
- The parts of the registry may have to be kept in sync with the file system (e.g., deleting an application rather than uninstalling it, may leave associated configuration items such as COM registration entries in the registry if the application is legacy and does not leverage side-by-side registry-free configuration.)
- Applications that make use of the registry to store and retrieve their settings may be unsuitable for use on portable devices used to carry applications from one system to another. Similarly, it is often not possible to copy installed applications that use the Registry to another computer. This means that software usually has to be reinstalled from original media after a computer upgrade or rebuild. Application virtualization addresses this problem.
- The Windows Registry is claimed by some third parties to be a single point of failure.

Structure

Keys and values

The registry contains two basic elements: keys and values.

Registry **keys** are similar to folders — in addition to values, each key can contain subkeys, which may contain further subkeys, and so on. Keys are referenced with a syntax similar to Windows' path names, using backslashes to indicate levels of hierarchy. Each subkey has a mandatory name which is a non-empty string that cannot contain any backslash or null character, and whose letter case is insignificant.

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

E.g. HKEY_LOCAL_MACHINE\Software\Microsoft\Windows refers to the subkey "Windows" of the subkey "Microsoft" of the subkey "Software" of the HKEY_LOCAL_MACHINE root key.

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

- HKEY_LOCAL_MACHINE or HKLM
- HKEY_CURRENT_CONFIG or HKCC (only in Windows 9x/ME and NT-based versions of Windows)
- HKEY_CLASSES_ROOT or HKCR
- HKEY_CURRENT_USER or HKCU
- HKEY_USERS or HKU
- HKEY_PERFORMANCE_DATA (only in NT-based versions of Windows, but invisible in the Windows Registry Editor)
- HKEY_DYN_DATA (only in Windows 9x/ME, and visible in the Windows Registry Editor)

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

Registry **values** are name/data pairs stored within keys. Registry values are referenced separately from registry keys. All registry values stored in a registry key have a unique

name whose letter case is not significant. The Windows API functions that query and manipulate registry values take value names separately from the key path and/or handle that identifies the parent key. Registry values may contain backslashes in their name but doing so makes them difficult to distinguish from their key paths when using some legacy Windows Registry API functions (whose usage is deprecated in Win32).

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

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

List of standard registry value types

Type ID	Symbolic type name	Meaning and encoding of the data stored in the registry value
0	REG_NONE	No type (the stored value, if any)
1	REG_SZ	A string value, normally stored and exposed in UTF-16LE (when using the Unicode version of Win32 API functions), usually terminated by a null character
2	REG_EXPAND_SZ	An "expandable" string value that can contain environment variables, normally stored and exposed in UTF-16LE, usually terminated by a null character
3	REG_BINARY	Binary data (any arbitrary data)

4	REG_DWORD / REG_DWORD_LITTLE_ENDIAN	A DWORD value, a 32-bit unsigned integer (numbers between 0 and 4,294,967,295 [$2^{32} - 1$]) (little-endian)
5	REG_DWORD_BIG_ENDIAN	A DWORD value, a 32-bit unsigned integer (numbers between 0 and 4,294,967,295 [$2^{32} - 1$]) (big-endian)
6	REG_LINK	A symbolic link (UNICODE) to another registry key, specifying a root key and the path to the target key
7	REG_MULTI_SZ	A multi-string value, which is an ordered list of non-empty strings, normally stored and exposed in UTF-16LE, each one terminated by a null character, the list being normally terminated by a second null character.
8	REG_RESOURCE_LIST	A resource list (used by the *Plug-n-Play* hardware enumeration and configuration)
9	REG_FULL_RESOURCE_DESCRIPTOR	A resource descriptor (used by the *Plug-n-Play* hardware enumeration and configuration)
10	REG_RESOURCE_REQUIREMENTS_LIST	A resource requirements list (used by the *Plug-n-Play* hardware enumeration and configuration)
11	REG_QWORD / REG_QWORD_LITTLE_ENDIAN	A QWORD value, a 64-bit integer (either big- or little-endian, or unspecified) (Introduced in Windows 2000)

Hives

The Registry comprises a number of logical sections, or "hives" (the word hive constitutes an in-joke). Hives are generally named by their Windows API definitions,

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

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

HKEY_LOCAL_MACHINE (HKLM)

Abbreviated HKLM, HKEY_LOCAL_MACHINE stores settings that are specific to the local computer.

The key located by HKLM is actually not stored on disk, but maintained in memory by the system kernel in order to map there all other subkeys. Applications cannot create any additional subkeys. On NT-based versions of Windows, this key contains four subkeys, "SAM", "SECURITY", "SYSTEM", and "SOFTWARE", that are loaded at boot time within their respective files located in the %SystemRoot%\System32\config folder. A fifth subkey, "HARDWARE", is volatile and is created dynamically, and as such is not stored in a file (it exposes a view of all the currently detected Plug-n-Play devices). On Windows Vista, Windows Server 2008, Windows Server 2008 R2, and Windows 7, a sixth subkey is mapped in memory by the kernel and populated from boot configuration data (BCD).

- The "HKLM\SAM" key usually appears as empty for most users (unless they are granted access by administrators of the local system or administrators of domains managing the local system). It is used to reference all "Security and Accounts Management" (SAM) databases for all domains into which the local system has been administratively authorized or configured (including the local domain of the running system, whose SAM database is stored a subkey also named "SAM": other subkeys will be created as needed, one for each supplementary domain). Each SAM database contains all builtin accounts (mostly, group aliases) and configured accounts (users, groups, and their aliases, including guest accounts and administrator accounts) created and configured on the respective domain; for each account in that domain, it notably contains the user name which can be used to log

on that domain, the internal unique user identifier in the domain, their cryptographically hashed password on that domain, the location of storage of their user registry hive, various status flags (for example is the account can be enumerated and be visible in the logon prompt screen), and the list of domains (including the local domain) into which the account was configured.

- The "HKLM\SECURITY" key usually appears empty for most users (unless they are granted access by users with administrative privileges) and is linked to the Security database of the domain into which the current user is logged on (if the user is logged on the local system domain, this key will be linked to the registry hive stored by the local machine and managed by local system administrators or by the builtin "System" account and Windows installers). The kernel will access it to read and enforce the security policy applicable to the current user and all applications or operation executed by this user. It also contains a "SAM" subkey which is dynamically linked to the SAM database of the domain onto which the current user is logged on.
- The "HKLM\SYSTEM" key is normally only writable by users with administrative privileges on the local system. It contains information about the Windows system setup, data for the secure random number generator (RNG), the list of currently mounted devices containing a filesystem, several numbered "HKLM\SYSTEM\Control Sets" containing alternative configurations for system hardware drivers and services running on the local system (including the currently used one and a backup), a "HKLM\SYSTEM\Select" subkey containing the status of these Control Sets, and a "HKLM\SYSTEM\CurrentControlSet" which is dynamically linked at boot time to the Control Set which is currently used on the local system. Each configured Control Set contains:
    - a "Enum" subkey enumerating all known Plug-and-Play devices and associating them with installed system drivers (and storing the device-specific configurations of these drivers),
    - a "Services" subkey listing all installed system drivers (with non device-specific configuration, and the enumeration of devices for which they are instanciated) and all programs running as services (how and when they can be automatically started),
    - a "Control" subkey organizing the various hardware drivers and programs running as services and all other system-wide configuration,
    - a "Hardware Profiles" subkey enumerating the various profiles that have been tuned (each one with "System" or "Software" settings used to modify the default profile, either in system drivers and services or in the applications) as well as the "Hardware Profiles\Current" subkey which is dynamically linked to one of these profiles.
- The "HKLM\SOFTWARE" subkey contains software and Windows settings (in the default hardware profile). It is mostly modified by application and system installers. It is organized by software vendor (with a subkey for each), but also contains a "Windows" subkey for some settings of the Windows user interface, a "Classes" subkey containing all registered associations from file extensions, MIME types, Object Classes IDs and interfaces IDs (for OLE, COM/DCOM and ActiveX), to the installed applications or DLLs that may be handling these types

on the local machine (however these associations are configurable for each user, see below), and a "Policies" subkey (also organized by vendor) for enforcing general usage policies on applications and system services (including the central certificates store used for authentifying, authorizing or disallowing remote systems or services running outside of the local network domain).

HKEY_CURRENT_CONFIG (HKCC)

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

HKEY_CLASSES_ROOT (HKCR)

Abbreviated HKCR, HKEY_CLASSES_ROOT contains information about registered applications, such as file associations and OLE Object Class IDs, tying them to the applications used to handle these items. On Windows 2000 and above, HKCR is a compilation of user-based HKCU\Software\Classes and machine-based HKLM\Software\Classes. If a given value exists in both of the subkeys above, the one in HKCU\Software\Classes takes precedence. The design allows for either machine- or user-specific registration of COM objects. The user-specific classes hive, unlike the HKCU hive, does not form part of a roaming user profile.

HKEY_USERS (HKU)

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

HKEY_CURRENT_USER (HKCU)

Abbreviated HKCU, HKEY_CURRENT_USER stores settings that are specific to the currently logged-in user. The HKCU key is a link to the subkey of HKEY_USERS that corresponds to the user; the same information is accessible in both locations. On Windows NT-based systems, each user's settings are stored in their own files called NTUSER.DAT and USRCLASS.DAT inside their own Documents and Settings subfolder (or their own Users sub folder in Windows Vista and Windows 7). Settings in this hive follow users with a roaming profile from machine to machine.

HKEY_PERFORMANCE_DATA

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

HKEY_DYN_DATA

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

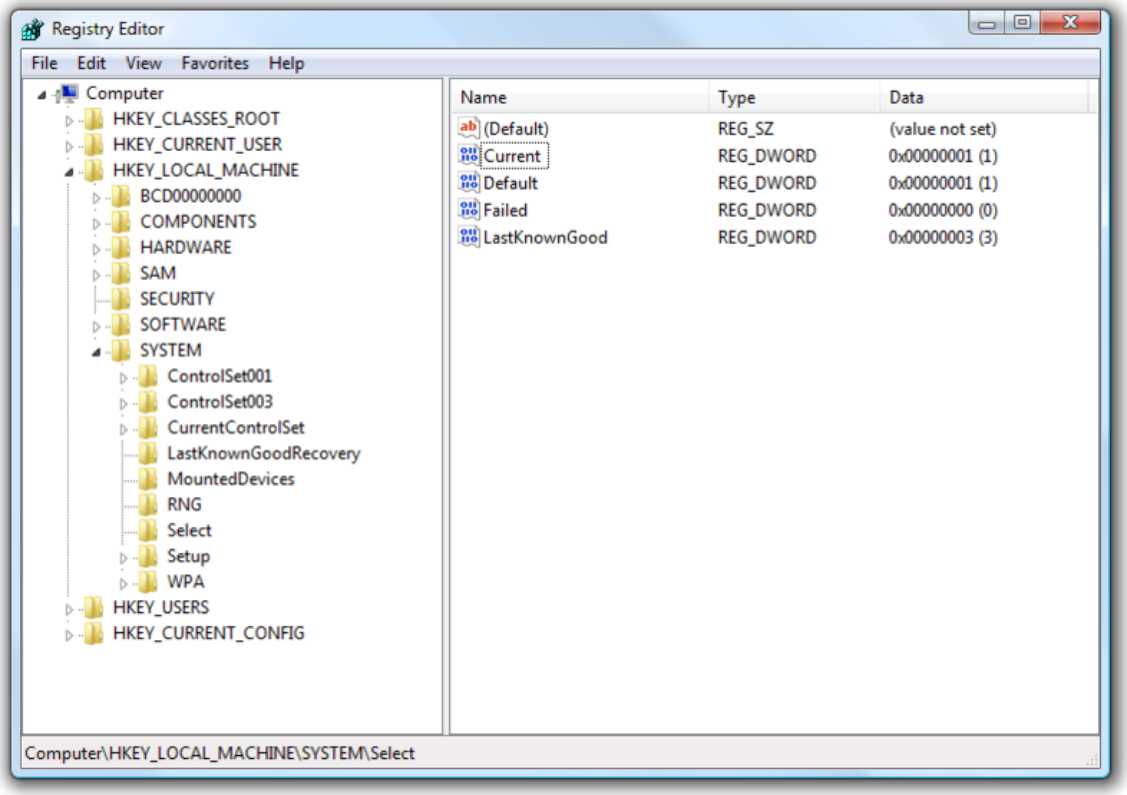
Aliases

Aliases in the Windows 9x registry:

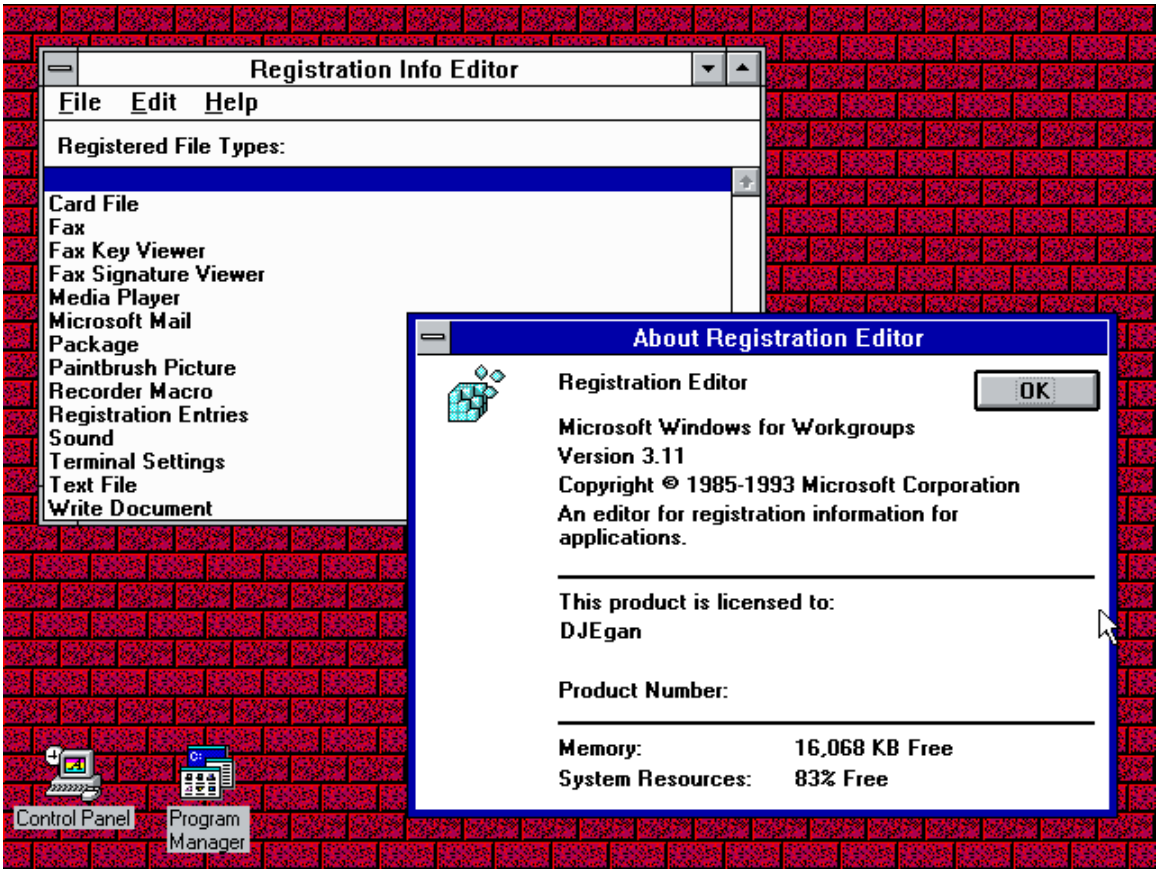
Root	Aliases for
HKEY_CLASSES_ROOT	HKEY_LOCAL_MACHINE\Software\Classes (as modified by HKEY_CURRENT_USER\Software\Classes)
HKEY_CURRENT_USER	User's branch within HKEY_USERS
HKEY_CURRENT_CONFIG	Hardware profile within HKEY_LOCAL_MACHINE\Config

Editing

Manual editing



The *Registry Editor* in Windows Vista



Windows 3.11 Registry Editor

The Windows registry can be edited manually using programs such as regedit.exe and on older versions of Windows, regedt32.exe.

As a careless change could cause irreversible damage, a backup of the registry before editing is recommended by Microsoft.

A simple implementation of the current registry tool appeared in Windows 3.x, called the "Registration Info Editor" or "Registration Editor". This was basically just a database of applications used to edit embedded OLE objects in documents.

Windows 9x operating systems included REGEDIT.EXE which could be used in Windows and also in real mode MS-DOS. Windows NT introduced permissions for Registry editing. Windows NT 4.0 and Windows 2000 were distributed with both the Windows 9x REGEDIT.EXE program and Windows NT 3.x's REGEDT32.EXE program. There were several differences between the two editors on these platforms:

- REGEDIT.EXE had a left-side tree view that begins at "My Computer" and lists all loaded hives. REGEDT32.EXE had a left-side tree view, but each hive had its own window, so the tree displays only keys.

- REGEDIT.EXE represented the three components of a value (its name, type, and data) as separate columns of a table. REGEDT32.EXE represented them as a list of strings.
- REGEDIT.EXE supported right-clicking of entries in a tree view to adjust properties and other settings. REGEDT32.EXE required all actions to be performed from the top menu bar.
- REGEDIT.EXE supported searching for key names, values, or data throughout the entire registry, whereas REGEDT32.EXE only supported searching for key names in one hive at a time.
- Earlier versions of REGEDIT.EXE did not support editing permissions. Therefore, on those early versions, only REGEDT32.EXE could access the full functionality of an NT registry. REGEDIT.EXE in Windows XP, VISTA, and Windows 7, supported editing permissions.
- REGEDIT.EXE only supported string (REG_SZ), binary (REG_BINARY), and DWORD (REG_DWORD) values. REGEDT32.EXE supported those, plus expandable string (REG_EXPAND_SZ) and multi-string (REG_MULTI_SZ). Attempting to edit unsupported key types with REGEDIT.EXE on Windows 2000 or Windows NT 4.0 would result in irreversible conversion to a supported type.

Windows XP was the first system to integrate these two programs into one, adopting the old REGEDIT.EXE interface and adding the REGEDT32.EXE functionality. The differences listed above are not applicable on Windows XP and newer systems; REGEDIT.EXE is the improved editor, and REGEDT32.EXE is deprecated.

The Registry Editor allows users to perform the following functions:

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

It is also possible to edit the registry under Linux using the open source Offline NT Password & Registry Editor to edit the files.

`.REG` files

`.REG` files (also known as Registration entries) are text-based human-readable files for storing portions of the registry. On Windows 2000 and later NT-based operating systems, they contain the string *Windows Registry Editor Version 5.00* at the beginning and are Unicode-based. On Windows 9x and NT 4.0 systems, they contain the string *REGEDIT4* and are ANSI-based. Windows 9x format `.REG` files are compatible with Windows 2000

and later NT-based systems. The Registry Editor on Windows on these systems also supports exporting .REG files in Windows 9x/NT format. Data is stored in .REG files in the following syntax:

```
[<Hive Name>\<Key Name>\<Subkey Name>]
"Value Name"=<Value type>:<Value data>
```

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

```
@=<Value type>:<Value data>
```

String values do not require a <Value type> (see example), but backslashes ("\") needs to written/escaped with a double-backslash ("\\").

For example, to add the values "Value A", "Value B", "Value C", "Value D", "Value E", "Value F", "Value G", "Value H" and "Value I" to the HKLM\SOFTWARE\Microsoft key,

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft]
"Value A"="<String value data>"
"Value B"=hex:<Binary data>
"Value C"=dword:<DWORD value integer>
"Value D"=hex(7):<Multi-string value data (as comma-delimited list of
hexadecimal values)>
"Value E"=hex(2):<Expandable string value data (as comma-delimited list
of hexadecimal values)>
"Value F"=hex(b):<QWORD value (as comma-delimited list of 8 hexadecimal
values, in little endian byte order)>
"Value G"=hex(4):<DWORD value (as comma-delimited list of 4 hexadecimal
values, in little endian byte order)>
"Value H"=hex(5):<DWORD value (as comma-delimited list of 4 hexadecimal
values, in big endian byte order)>
"Value I"=hex(0):
```

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

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

For example, to remove the <Key Name> key (and all subkeys, values and data),

```
[-HKEY_LOCAL_MACHINE\SOFTWARE\<Key Name>]
```

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

For example, to remove only the "Value A" and "Value B" values (and their data) from the <Key Name> key,

```
[HKEY_LOCAL_MACHINE\SOFTWARE\<Key Name>]
"Value A"=-
"Value B"=-
```

To remove only the (Default) value of the key <Key Name> (and its data),

```
[HKEY_LOCAL_MACHINE\SOFTWARE\<Key Name>]
@=-
```

Command line editing

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

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

```
regedit.exe /s file
```

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

The default association for `.REG` files in many versions of Microsoft Windows.

We can use also `REG.EXE`. Here is a sample to displays the value of the registry value Version:

```
REG QUERY HKLM\Software\Microsoft\ResKit /v Version
```

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

Registry permissions can be manipulated through the command line using `regini.exe` and the `SubInACL.exe` tool. For example, the permissions on the HKEY_LOCAL_MACHINE\SOFTWARE key can be displayed using:

```
subinacl /keyreg HKEY_LOCAL_MACHINE\SOFTWARE /display
```

Programs or scripts

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

List of Registry API functions

RegCloseKey	RegOpenKey	RegConnectRegistry	RegOpenKeyEx
RegCreateKey	RegQueryInfoKey	RegCreateKeyEx	RegQueryMultipleValues
RegDeleteKey	RegQueryValue	RegDeleteValue	RegQueryValueEx
RegEnumKey	RegReplaceKey	RegEnumKeyEx	RegRestoreKey
RegEnumValue	RegSaveKey	RegFlushKey	RegSetKeySecurity
RegGetKeySecurity	RegSetValue	RegLoadKey	RegSetValueEx
	RegNotifyChangeKeyValue	RegUnLoadKey	

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

Similarly, scripting languages such as Perl (with `Win32::TieRegistry`), Windows Powershell and Windows Scripting Host also enable registry editing from scripts.

Locations

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

Windows NT-based operating systems

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

- `Sam` – HKEY_LOCAL_MACHINE\SAM
- `Security` – HKEY_LOCAL_MACHINE\SECURITY
- `Software` – HKEY_LOCAL_MACHINE\SOFTWARE
- `System` – HKEY_LOCAL_MACHINE\SYSTEM
- `Default` – HKEY_USERS\.DEFAULT
- `Userdiff` – Not associated with a hive. Used only when upgrading operating systems.

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

- `%UserProfile%\Ntuser.dat` – HKEY_USERS\<User SID> (linked to by HKEY_CURRENT_USER)

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

- `%UserProfile%\Local Settings\Application Data\Microsoft\Windows\Usrclass.dat` (path is localized) – HKEY_USERS\<User SID>_Classes (HKEY_CURRENT_USER\Software\Classes)

For Vista and later, the path was changed to:

- `%UserProfile%\AppData\Local\Microsoft\Windows\Usrclass.dat` (path is not localized) alias `%LocalAppData%\Microsoft\Windows\Usrclass.dat` – HKEY_USERS\<User SID>_Classes (HKEY_CURRENT_USER\Software\Classes)

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

Windows 95, 98, and Me

The registry files are named `USER.DAT` and `SYSTEM.DAT` are stored in the `%WINDIR%` directory. In Windows Me, `Classes.dat` was added. Also, each user profile (if profiles are enabled) has its own `USER.DAT` in profile's directory.

Windows 3.11

The registry file is `Reg.dat, system.dat` and is stored in the `C:\WINDOWS` directory.

These files also contain system registry info: `user.dat` and is stored in the `C:\windows\profiles\<username>\user.dat` directory.

Backups and recovery

Windows supports several methods to back up and restore the registry:

- System Restore can back up the registry and restore it as long as Windows is bootable, or from the Windows Recovery Environment starting with Windows Vista.

- NTBackup can back up the registry as part of the *System State* and restore it.
- On Windows NT-based systems, the *Last Known Good Configuration* option in startup menu relinks the `HKLM\SYSTEM\CurrentControlSet` key, which stores hardware and device driver information.
- Windows 98 and Windows Me include command line (Scanreg.exe) and GUI (Scanregw.exe) registry checker tools to check and fix the integrity of the registry, create up to five automatic regular backups by default and restore them manually or whenever corruption is detected. The registry checker tool backs up the registry, by default, to `%Windir%\Sysbckup` Scanreg.exe can also run from MS-DOS.
- The Windows 95 CD-ROM included an Emergency Recovery Utility (ERU.exe) and a Configuration Backup Tool (Cfgback.exe) to back up and restore the registry. Additionally Windows 95 backs up the registry to the files system.da0 and user.da0 on every successful boot.

Policy

Group policy

Windows 2000 and later versions of Windows use Group Policy to enforce Registry settings. Policy may be applied locally to a single computer using `gpedit.msc`, or to multiple users and/or computers in a domain using `gpmc.msc`.

Legacy systems

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

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

The policy file is created by a free tool by Microsoft that goes by the filename `poledit.exe` for Windows 95/Windows 98 and with a computer management module for NT-based systems. The editor requires administrative permissions to be run on systems that uses permissions. The editor can also directly change the current registry settings of the local computer and if the remote registry service is installed and started on another computer it can also change the registry on that computer. The policy editor loads the settings it can change from `.ADM` files, of which one is included, that contains the

settings the Windows shell provides. The `.ADM` file is plain text and supports easy localisation by allowing all the strings to be stored in one place.

.INI file virtualization

Windows NT kernels support redirection of INI file-related APIs into a virtual file in a Registry location such as HKEY_CURRENT_USER using a feature called "InifileMapping". This functionality was introduced to allow legacy applications written for 16-bit versions of Windows to be able to run under Windows NT platforms on which the System folder is no longer considered an appropriate location for user-specific data or configuration. Non-compliant 32-bit applications can also be redirected in this manner, even though the feature was originally intended for 16-bit applications.

Registry virtualization

Windows Vista has introduced limited Registry virtualization, whereby poorly written applications that do not respect the principle of least privilege and instead try to write user data to a read-only system location (such as the HKEY_LOCAL_MACHINE hive), can be redirected to a more appropriate location, without changing the application itself. The operation is transparent to the application, as it does not know that its Registry operations have been directed elsewhere.

Similarly, application virtualization redirects all of an application's Registry operations to a non-Registry backed location, such as a file. Used together with file virtualization, this approach allows applications to run without being installed on the location machine.

Low integrity processes may also leverage registry virtualization. For example as Internet Explorer 7 or 8 running in "Protected Mode" on Windows Vista or Windows 7 will automatically redirect registry writes by ActiveX controls to a sandboxed location in order to frustrate some classes of security exploits.

Lastly, the Application Compatibility Toolkit provides shims that can transparently redirect HKEY_LOCAL_MACHINE or HKEY_CLASSES_ROOT Registry operations to HKEY_CURRENT_USER to address "LUA" bugs that cause applications not to work for limited users.

Equivalents in other operating systems

In contrast to the Windows registry's binary-based database model, some other operating systems use separate plain-text files for daemon and application configuration, but group these configurations together for ease of management.

- Under Unix-like operating systems e.g. Linux that follow the Filesystem Hierarchy Standard, system-wide configuration files (information similar to what would appear in HKEY_LOCAL_MACHINE on Windows) are traditionally stored in files in `/etc/` and its subdirectories, or sometimes in `/usr/local/etc`.

Per-user information (information that would be roughly equivalent to that in HKEY_CURRENT_USER) is stored in hidden directories and files (that start with a period/full stop) within the user's home directory. However XDG-compliant applications should refer to the environment variables defined in the Base Directory specification.

- Applications running on Apple Inc.'s Mac OS X operating system typically store settings in property list files which are usually stored in each user's Library folder.
- RISC OS also allows applications to be copied into directories easily, as opposed to the separate installation program that typifies Windows applications. If one wishes to remove the application, it is possible to simply delete the folder belonging to the application. This will often not remove configuration settings which are stored independently from the application, usually within the computer's !Boot structure, in !Boot.Choices, but potentially anywhere on a network fileserver.
- IBM AIX (a Unix variant) uses a registry component called Object Data Manager (ODM). The ODM is used to store information about system and device configuration. An extensive set of tools and utilities provides users with means of extending, checking, correcting the ODM database. The ODM stores its information in several files, default location is /etc/objrepos.
- The GNOME desktop environment uses a registry-like interface called GConf for storing configuration settings for the desktop and applications. However, in GConf, all application settings are stored in separate files, thereby eliminating a single point of failure. Conversely, this also creates multiple points of failure, and the likelihood of one or more files being destroyed is increased.
- The Elektra Initiative provides an alternative back-end for text configuration files for the Linux operating system, similar to the registry.
- While not an operating system, the Wine compatibility layer, which allows Windows software to run on a Unix-like system, also employs a Windows-like registry as text files in the WINEPREFIX folder: system.reg (HKEY_LOCAL_MACHINE), user.reg (HKEY_CURRENT_USER) and userdef.reg.