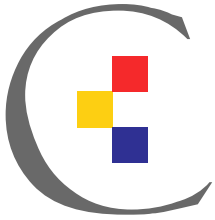


c a f i t

CVS/TortoiseCVS Administration

An enterprise guide

January 2005

Written by Graham Crockford
www.cafit.co.uk

1. Introduction

1.1 Aim of this document

This document aims to give an in-depth view of a successful implementation of CVSNT (Server) and TortoiseCVS (Client) in a small-to-medium-size enterprise environment based on a **Windows 2003 Server** and **Windows 2000/XP Clients**.

It covers as much as possible of the daily and regular maintenance tasks required and best-use practices. Note, however, that these practices are what we have come to over time and may not necessarily suit all environments. Backup and sandbox storage policy have been strong bones of contention (see 3.2.2).

1.2 Scope

- Required applications, download sites, support URLs
- Working with Open Source software and the OSS community
- Server configuration
- Client configuration
- Update checking, deployment, issues arising from updates
- Fault recognition and resolution

Not covered:

- Relative merits of CVS versus other open source or commercial systems. SVN (http://subversion.tigris.org/), intended as a modern replacement for CVS and now fairly stable, has been interesting us for some time and will be worth considering once reserved checkouts are available (expected early 2005)
- Migration to CVS from other source code management systems (we migrated from SourceSafe using a modified version of one of the migration tools available)
- How CVS works (a working knowledge of CVS is required)
- How to use TortoiseCVS, WinCVS, WinMerge or any other applications described except in the cases of specific workflow strategies or configuration settings, which can improve the system in an enterprise environment. It is recommended that some time be taken "playing" with a test CVSNT/TortoiseCVS setup to familiarise yourself with the software before reading this document.

1.3 Intended audience

This document is aimed at:

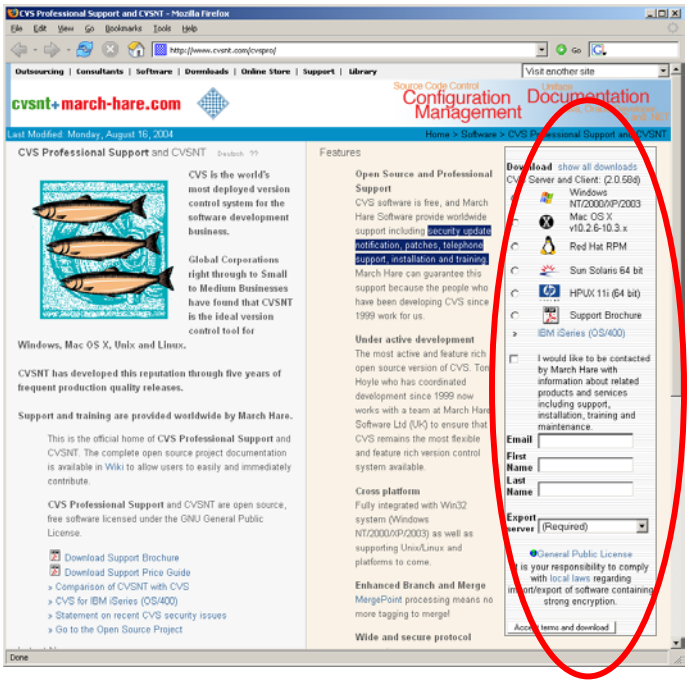
- Any enterprise considering CVS as a source code management platform
- Enterprises with existing CVS installations looking for maintenance tips

2. Resources

2.1 CVSNT

2.1.1 Web links

CVSNT's commercial presence is via March Hare Software, who provide security update notification, patches, telephone support, installation and training. However the product itself is free, and compiled binaries packaged into a windows installer are available from http://www.cvsnt.com



2.1.2 Documentation

There is extensive documentation available at the main CVSNT site, but if this is not sufficient (e.g. for Q&A), we have found that a **Google** search (http://www.google.com)
is the most effective method of quickly finding answers to problems people have encountered before.

A common search string may take the form

CVSNT Commit "Socket error -1"

To search for help on a "Socket error -1" error that occurred whilst trying to commit a file.

2.1.3 Support

If something goes wrong and you can't find the solution, your only recourse is to report the error and then track its progress through resolution and release. Most often, however, the development team will offer advice and help which will circumvent the issue, so **do this ASAP**.

From the CVSNT Support Wiki at http://www.cvsnt.org/wiki :

There are four mailing lists provided:

- cvsnt: General support and help for users of cvsnt
- cvsnt-dev: Development and future plans for 2.1.x branch
- cvsnt-commits: Track commits to cvsnt development tree
- cvsnt-bugs: Bug status reports

Alternatively, they can be accessed via News:

- support.cvsnt: General support and help for users of cvsnt
- cvsnt-dev: Development and future plans for 2.1.x branch
- support.cvsnt-commits: Track commits to cvsnt development tree
- support.cvsnt-bugs: Bug status reports

It is generally good manners to join these groups anyway and offer support to others when possible – open source software many be free but that is because it relies on the support of its users.

2.2 TortoiseCVS

2.2.1 Web links

The main TortoiseCVS site is at http://www.tortoisecvs.org and the SourceForge project can be found at http://sourceforge.net/projects/tortoisecvs/ .

Updates appear *extremely* regularly at times and often fix new issues created by a previous release. It is therefore important that new releases are at least smoke tested by an experienced technician before being deployed.

Updates are made available at: http://www.tortoisecvs.org/download.shtml but check the changelog first: http://www.tortoisecvs.org/stablechangelog.shtml

2.2.2 Documentation

TortoiseCVS ships with a CHM (Microsoft Help) format manual, or it can be downloaded at: http://www.tortoisecvs.org/UserGuide_en.chm

Again, Google can be helpful at times.

2.2.3 Support

As with CVSNT; however, as TortoiseCVS is on SourceForge, the process is a little cleaner.

Register at SourceForge (http://www.sourceforge.net) and then check the outstanding bugs: http://sourceforge.net/tracker/?group_id=48103&atid=451972

If you don't find the issue, you can create a new issue and this will be responded to fairly quickly. However, the response can be hostile if you repeatedly raise errors which are (a) already registered (b) not TortoiseCVS's fault (e.g. server issues, windows configuration) or (c) RTFM[1].

[1] Read the "friendly" manual.

2.3 Other

2.3.1 Textpad

Textpad is an excellent text editor with superlative search capabilities, worth it if only for the fact that it adds a "Textpad" option to the explorer context menu on every file type, including extensionless files: http://www.textpad.com

2.3.2 WinMerge

By far the best merge and diffing tool for use with TortoiseCVS.
http://www.winmerge.com

2.3.3 WinCVS

TortoiseCVS was derived from this relatively heavyweight CVS client. These days we rarely need to use it although it can be very handy when TortoiseCVS's performance gets irritating.

3. Configuration and deployment

3.1 Server

3.1.1 File system

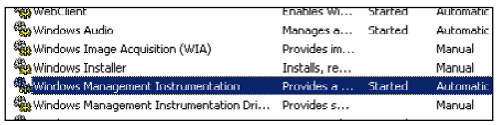
First of all, designate a directory on your server to be the repository. You can in fact have several repositories, but the single-repository model matched our original SourceSafe setup most closely so it made migration easier. **Don't share this folder over the network** – except as directed in 3.1.9, below. CVSNT is a client-server system and does not use Windows file shares (SMB) to transfer files.

3.1.2 Installation/updates

The installer for CVNT runs as a simple wizard. This should run with no problems for the first time.

However, on Windows 2003 Server, we have encountered a problem when updating an existing installation with the new version. Installation pauses with an error informing you that **cvslock.exe** is locked (sometimes it can be a different file).

At this point, the installer will wait for you to specify Retry, Cancel, Abort.

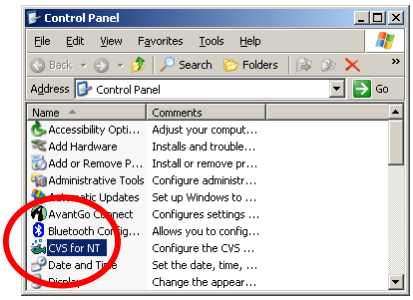


The problem is caused by the WMI (**Windows Management Instrumentation**) service. We need to stop this service temporarily in order to continue.

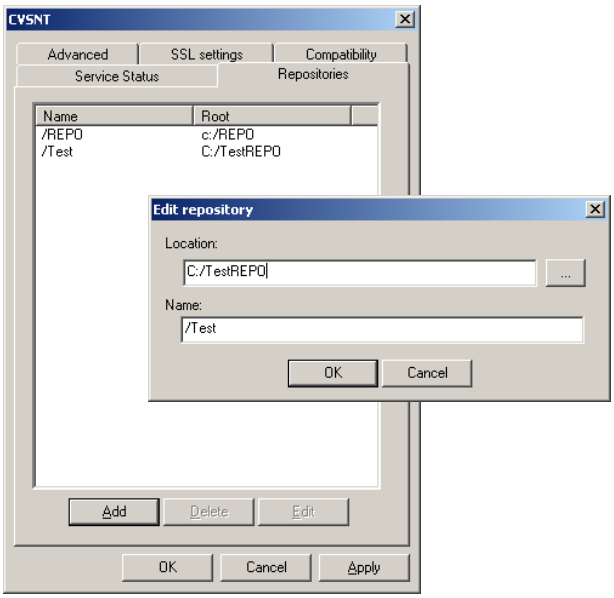
- Leave it waiting for the moment and open the Services Management Console Applet (Right-Click **My Computer > Manage > Services**).
- Locate the **Windows Management Instrumentation** service.
- Check that this service is not required by any mission-critical systems, or wait for scheduled downtime.
- Stop the service.
- Return to the installer and hit **Retry**. CVSNT should now install.
- Restart the WMI service.

3.1.3 Basic setup

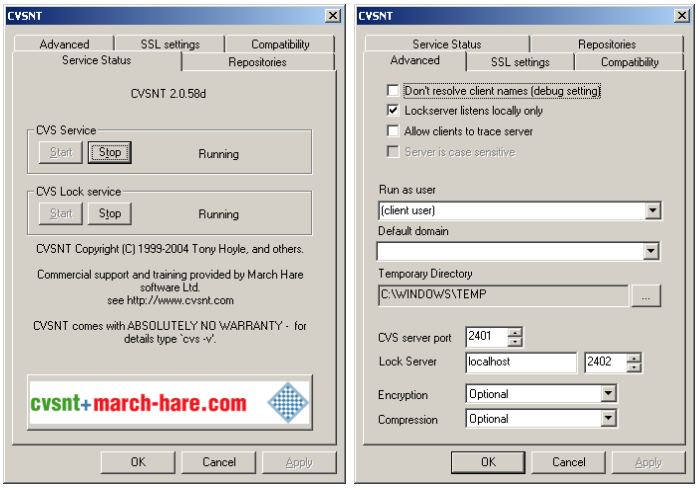
Once installed, CVSNT can be set up from the control panel applet:



First, you need to configure the repository directories. We only use one repository, split into modules, but this is a matter of taste.

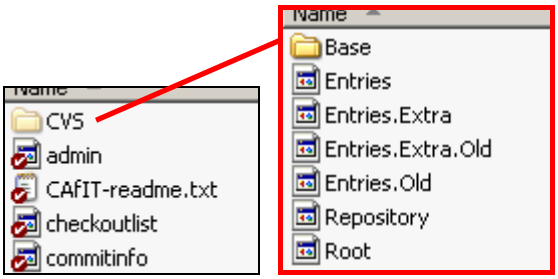


Starting and stopping the server is simple, just remember to start both the main service and the lock service.



3.1.4 Anatomy of a sandbox

In addition to the well-documented **.cvsignore** file (see the TortoiseCVS documentation), if you **show hidden files** in explorer (no doubt you always do – although you will find that turning this off makes working with CVS much neater!) you will find a hidden directory called **CVS** in every sandbox directory, at every level.



A great deal of server information is cached here, to minimise communication with the server. Common maintenance issues occur if this data gets out of sync with the server.

Base – this directory contains gzipped copies of any edited files, taken at edit time. They are used to restore a file back to its previous state if it is unedited instead of committed.

Entries - This file contains a list of all the files and directories within this directory, which are under version control. If a file is in the directory but is not on this list, TortoiseCVS will mark it as not under version control (and give the option to add it, unless you have it in an ignore list). Equally, if, at update time, a file in this list is missing from the directory, it will be replaced from the repository. Alongside the file names are stored the last modified date for the "sticky" repository version, and that version's number. If the modified date of the file is later than this date, it can be marked as "modified" by TortoiseCVS.

Entries.Extra, Entries.Extra.Old and **Entries.Old** – We're not quite sure what these are for, but when meddling by hand with **Entries**, we have always made the effort to make analogous changes to these files too.

Repository – This gives the relative path of the directory within the repository.

Root – This gives the server address and repository path. Note that if you change the address of the server, all the sandboxes will be out of date and will not work. See 6.1.6 for information on how to resolve this.

3.1.5 Anatomy of the repository

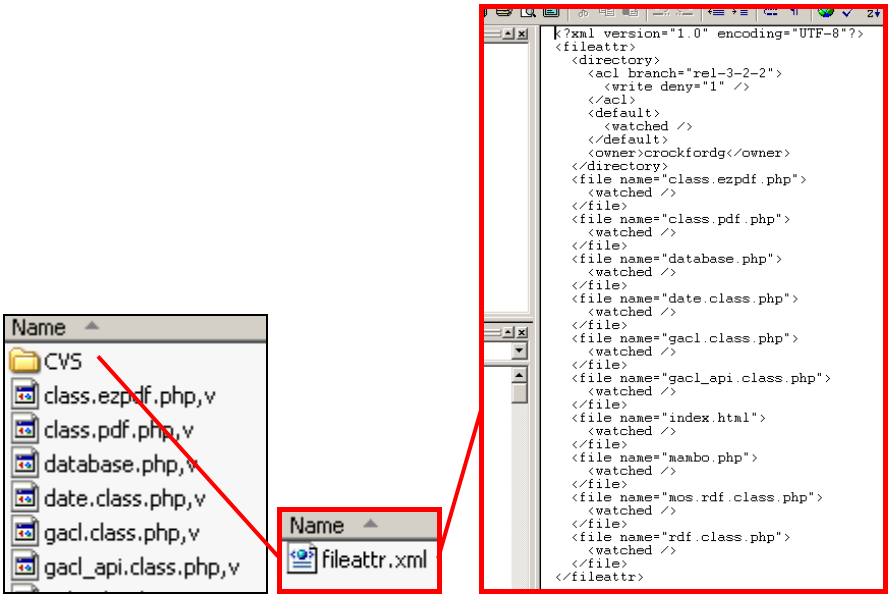
Once you have created the directory tree in your sandbox and added it, you will note that a mirror structure has appeared in the repository on the server. This simply consists of the directory tree, with each file in it (with ",v" appended to the filename) and lots of files called **fileattr.xml** in directories called **CVS**.

The ,v files are RCS repository files. CVS is designed as a wrapper around RCS, which maintains a repository of individual version-controlled files. RCS was perfectly adequate as a single machine, single file version control system, but CVS adds all the features required to make it a project level system, including concurrency and a client-server model.

If you open the files you may be able to decipher what you see, which will be a list of version numbers, tags and branches, with dates, comments, permissions etc. After these will be a full copy of the text of the latest version of the file followed by a series of reverse diffs to take you progressively back to the initial version. This has been shown over time to be the most space efficient storage method when balanced against speed – checkouts of the latest version (the most common form of checkout) are very quick, and the process gets slower the further up the version tree you go, although diffing is a relatively cheap operation on text files.

We have never seen corruption of RCS files – it is at least 14 years old at time of writing and has been in mission critical use for most of that time! As a result it is rare that you will need to "hack" these files.

The **fileattr.xml** files are less trustworthy and are the most common source of repository corruption, although such corruptions are rarely damaging – as usual CVS has a tendency to "fail-safe" and stop any CVS operations working at all..



fileattr.files are well-formed XML listing file and folder level settings. This is where "watch mode on" for a directory is actually stored, as well as branch level access controls and file edits. Reading them is fairly self-explanatory, but try editing and unediting files in your sandbox and watching the file change.

3.1.6 A major bug

This deserves a special mention; although it may be fixed by the time you read this. With CVSNT 2.0.58d and a few previous versions, file or directory names with an ampersand (**&**) will cause a corruption of the fileattr.xml file belonging to the parent directory. Examining the XML may result in you missing the problem several times since it is fairly obscure – **the ampersand is a control character in XML**. In order to put ampersands into XML one should write **&**.

CVSNT (which is used on both client side – by TortoiseCVS itself – and on the server side) just writes the ampersand and thus causes the file to be corrupted.

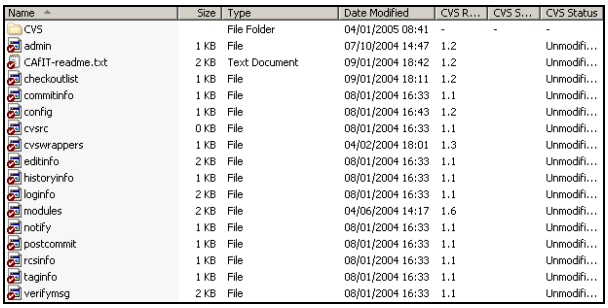
In short, **do not allow the ampersand to be used in file or directory names in CVS**. This should be a matter of policy until the bug is fixed.

See 6.1.3 for some advice on recovery if you end up with a well-populated directory with an ampersand in the name, containing several files, which are being edited by several people before this bug is uncovered...

3.1.7 Configuring the CVS server – "CVSROOT"

Most of the configuration of CVS can be carried out in real-time without resetting the server. A special module exists in each repository called CVSROOT. Watch should always be off for this module.

CVSROOT contains a number of config files, which can be edited on a local sandbox and then committed to apply them immediately. A useful side effect is that this means a full history of configuration changes are kept, right from installation. This makes it easy, for example, in this case, where I want to double check exactly what changes *we* made to get everything to work!



Name	Size	Type	Date Modified	CVS R...	CVS S...	CVS Status
CVS		File Folder	04/01/2005 08:41	-	-	-
admin	1 KB	File	07/10/2004 14:47	1.2		Unmodifi...
CAfIT-readme.txt	2 KB	Text Document	09/01/2004 18:42	1.2		Unmodifi...
checkoutlist	1 KB	File	09/01/2004 18:11	1.2		Unmodifi...
commitinfo	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
config	1 KB	File	08/01/2004 16:43	1.2		Unmodifi...
cvsrc	0 KB	File	08/01/2004 16:33	1.1		Unmodifi...
cvswrappers	1 KB	File	04/02/2004 18:01	1.3		Unmodifi...
editinfo	2 KB	File	08/01/2004 16:33	1.1		Unmodifi...
historyinfo	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
loginfo	2 KB	File	08/01/2004 16:33	1.1		Unmodifi...
modules	2 KB	File	04/06/2004 14:17	1.6		Unmodifi...
notify	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
postcommit	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
rcsinfo	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
taginfo	1 KB	File	08/01/2004 16:33	1.1		Unmodifi...
verifymsg	2 KB	File	08/01/2004 16:33	1.1		Unmodifi...

A full explanation of all configuration options can be found in the CVSNT documentation. Here is a list of changes we made that may prove useful:

config – We just set **LockServer=localhost:2402** – by default CVS stores lock files in the repository, which means that users need NTFS read and write access to a directory just to view it. Lockserver enables file locking using an external service provided in the CVSNT package instead, so we can give certain users only read access at the file level.

admin – This file is just a text list of the user names which will be allowed to carry out administration tasks, such as calling **cvs admin** or changing any files in CVSROOT. Note that this file is normally an exception in that it is not under version control – a user checking out CVSROOT will not get a copy. To update it, you need access directly to the repository.

checkoutlist – Following on from **admin**, the only change we made here was to add the **admin** file into the list of additional files in CVSROOT to put under version control. This is not recommended behaviour – it is considered more secure for no-one other than those which direct access to the repository to be able to see this file, but we have sufficient confidence in NTFS permissions to put this under version control and make it visible in CVSROOT.

modules - We added two lines:

```
All -a !Archive !CVSROOT .
Build -a Project1 Project2 Project3
```

The first line creates an alias module called "All" which when checked out actually checks out every module in the repository at once. This is useful when first creating a sandbox. Note that it works by just checking out everything *except* Archive and CVSROOT, which are both system modules.

The second line is similar but is instead inclusive, containing as it does a list of the modules to check out. In this case it contains everything necessary to do a build. Again useful for our automated build process.

3.1.8 Decide policy

CVS works well when used in the way it was designed – with no notion of a "lock" on a file such as with Microsoft SourceSafe's "Check Out". Many users can use the same files simultaneously and commit their changes in any order, as long as they are capable of merging in their changes at commit time.

However, this is not an option, which is available with binary (non-text) files. There is no (consistent) way of comparing two binary files for changes and displaying them in such a way that update anomalies can be resolved and changes merged. For files created by individual applications such as Microsoft Word, partial solutions exist, but these are obviously specific to these files and not to binary files as a whole.

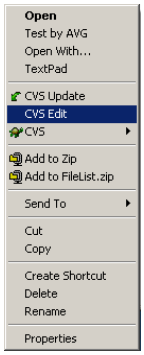
Our big problem was hit with Visual Basic. It stores binary data attached to Forms and User Controls as .frx and .ctx files respectively, which very much require version control. The CVS community tends to be unsympathetic in this case – technologies such as XML have made even Microsoft switch to text for its .NET platform, and use of binary files is viewed as increasingly restrictive. However, this doesn't help us!

The only way to prevent update anomalies is to enable reserved checkouts ("watch mode") on CVS. This changes the edit workflow from:

- Change file
- CVS Update
- CVS Commit

To

- CVS Edit
- Change file
- CVS Update
- CVS Commit
- CVS Unedit



The "Edit" and "Unedit" operations being calls to the CVS server to indicate that you are changing the file or have finished changing the file respectively. In the case of text files, CVS will simply warn another user if they try and edit the same file. However, in the case of binary files, CVS will refuse to issue an Edit on any file which is already Edited.

On CVS modules where this "watch mode" is turned on, the client (such as TortoiseCVS) can be configured to check out sandbox files read-only, and mark them as writable when the user carries out an edit.

This appears to fix the problem, but has some drawbacks.

1. **A user does not have to ask for an edit.** The commit operation is completely independent of the edit/unedit process. Therefore the user can just mark the file as writable, make the changes and commit, regardless of whether another user has an edit.

2. **Edits can be tricky to maintain**. Users have a tendency to abandon sandboxes, or delete and re-checkout, leaving their edits hanging, and start on new sandboxes, which appear to contain unedited files. Until recently, a bug existed in TortoiseCVS where files did not automatically get unedited after commits, leading to the same effect. In addition, it often happens that hanging edits are impossible to remove using the usual admin commands, necessitating hard editing of CVS's repository files (more on this shortly).

The majority of problems arising due to these issues can be avoided. (1) can be avoided via strict company policy. Much of (2) just comes down to periodic clearups of edits and regular requests from users to clear them as required.

In our experience, with proper procedures in place, most problems have "failed safe", in that they have created maintenance tasks, rather than errors.

If you have decided to use watch mode, you can enable it across the entire repository by CD-ing into the root of a sandbox folder at a command prompt and typing:

```
cvs watch on -R
```

This will turn on watch mode recursively for every folder in the module. Omitting the **–R** will just act on the files in the folder and not recurse subdirectories.

Users' sandboxes will need to be re-checked out to pick up this change.

3.1.9 NTFS permissions

To use NTFS permissions (which is assumed throughout this document) ensure that all clients connect to the CVS server using **SSPI** authentication. Turning off all other types during installation can enforce this.

Logon to CVS will become transparent and NT usernames will then be used throughout. File permissions will apply to all server threads as they are opened for a specific user's request, so you can use NTFS permissions to grant or deny access to certain areas of the repository based on username/group.

Put simply, as long as you enable lockserver (see 3.1.6) setting NTFS permissions is fairly intuitive. We created a group, "CVS Admins" and gave this group full read/write access to the repository. In addition we shared the repository folder and denied everyone access to the share *except* the CVS Admins group. This is essential for some fault resolution.

Beyond this, you might grant R/W access to some projects and read-only access to others on a person-by-person or group basis. As long as everyone has read access to Archive and CVSROOT, there should be no problems with this.

For permissions at branch level, see 5.1.3.

3.1.10 Backups

Since all CVS configuration and data is stored in a single directory tree, and locks are managed by an external service, there should be no danger in simply backing up the repository directory overnight. Indeed we have been doing this for some time with no difficulties encountered.

We have not looked into the dangers of backing up whilst the repository is active, e.g. during the working day. Atomic commits are, to our knowledge, not implemented in CVS (this is trumpeted as one of **SVN**'s major features – http://subversion.tigris.org) so there is a theoretical risk.

3.2 Clients

3.2.1 Installation

Our recommended setup consists of installing:

- TortoiseCVS
- WinMerge

Both have simple install wizards.

3.2.2 Backup simplicity vs. Performance

It is recommended by both the CVSNT and TortoiseCVS communities that you don't store your CVS sandboxes on a network drive.

From the point of view of the CVSNT server, it greatly increases bidirectional network traffic, causing communication to slow down (and therefore CVS operations to take more time).

From the point of view of TortoiseCVS, it suffers from a feature of Windows Explorer. Files updated by the local machine on local drives just cause windows messages to be sent to explorer, so it can just update its displays on request. However, no such messaging system works for network shares, so Explorer "polls" any open directories on a frequent schedule to check for updated files. A large number of sequential operations on an open folder cause an unbearable slowdown, to the extent that closing the explorer window and leaving a long operation (for example, a large directory copy) processing results in a many, many fold increase in speed. This obviously hits TortoiseCVS, which is regularly called upon to carry out recursive update or edit/unedit operations on open folders full of files.

Despite all this, we still went for storing sandboxes on network drives. When the security of data was weighed up, with a RAID array server backed up to tape versus a room full of budget desktop PCs, it was decided that we couldn't run the risk of running a distributed backup process.

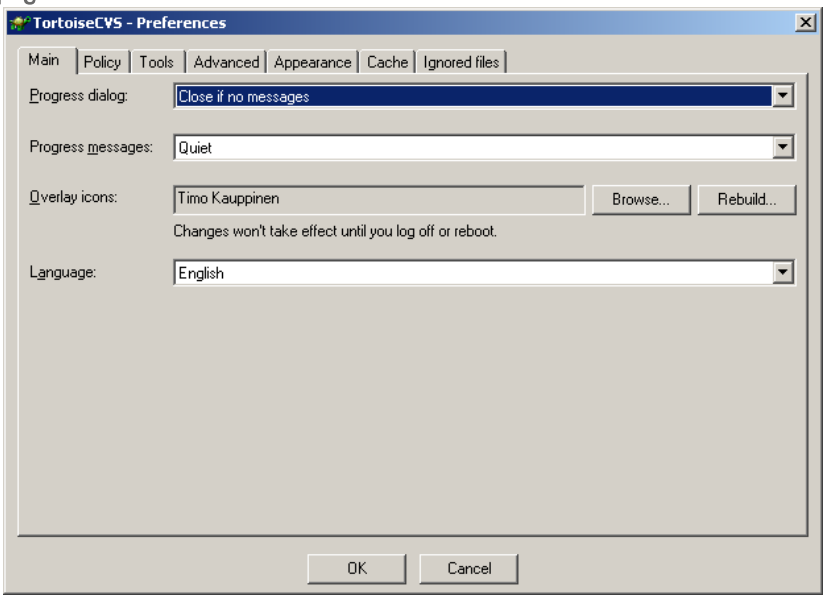
This is a decision, which will need to be made. If you have a distributed backup process/are still planning your process/have no process at all then use local sandboxes – the performance difference is enormous.

3.2.3 Designing a standard configuration

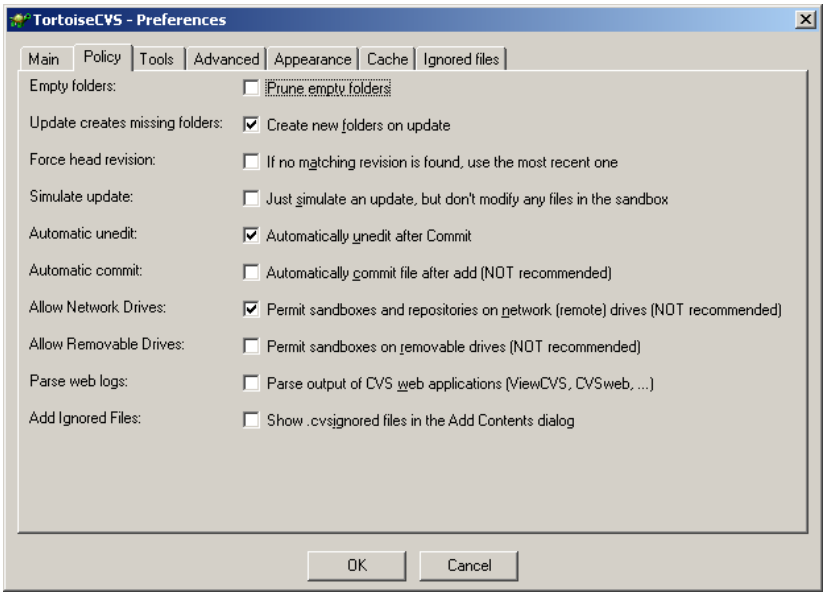
WinMerge will work from scratch (although we recommend turning the view font size down a little – 11pt or 10pt allow for greater readability).

However, TortoiseCVS requires configuration, much of which is linked to policy decisions.

The following screens show our recommended settings, with attached notes where the reasoning or meaning isn't clear.

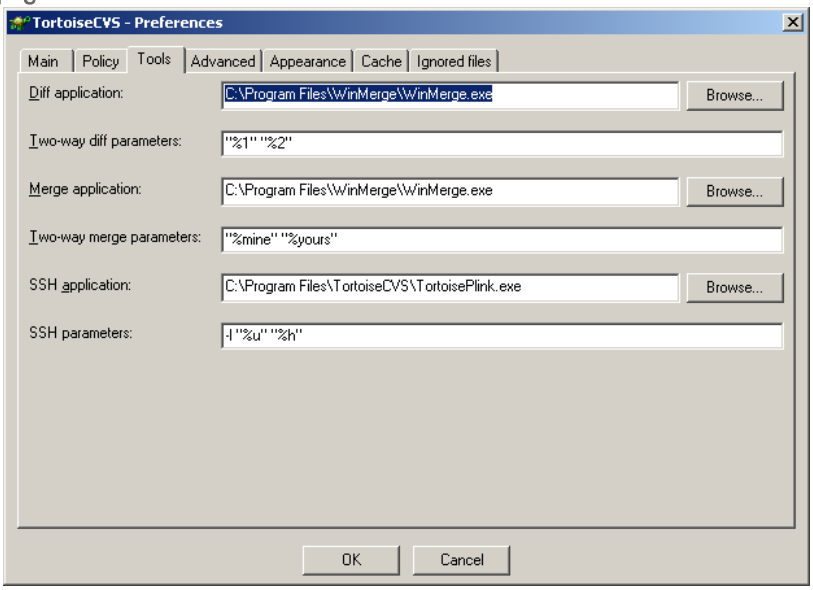


We find the **Timo Kauppinen** icons the ones that fit most neatly with the standard Windows icon set. **Do not set the progress dialog to just close** – this is dangerous if it means users ignoring CVS errors.

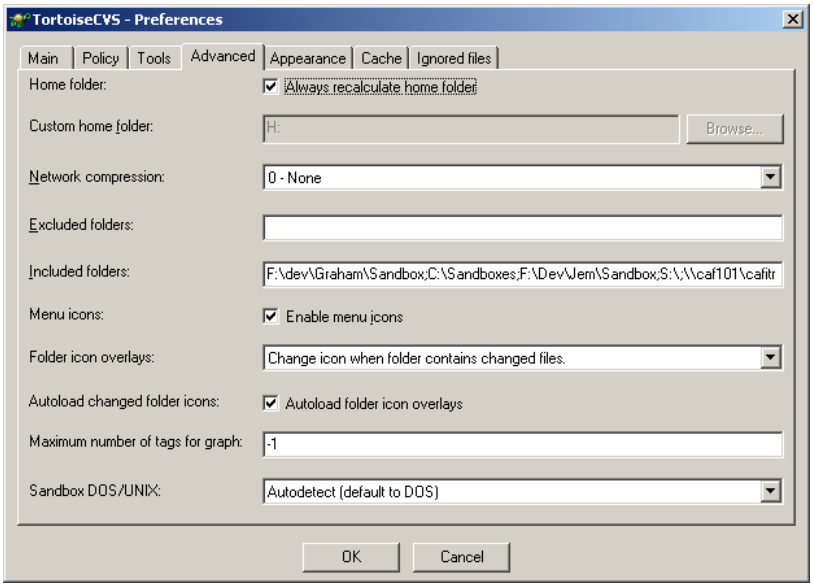


Automatic Unedit after commit is, in our opinion, essential if you are using watch mode, and as explained, we decided to make it a matter of policy to *force* users to use network drives for their sandboxes, for reasons of security. If you have a distributed backup strategy this may be unnecessary.

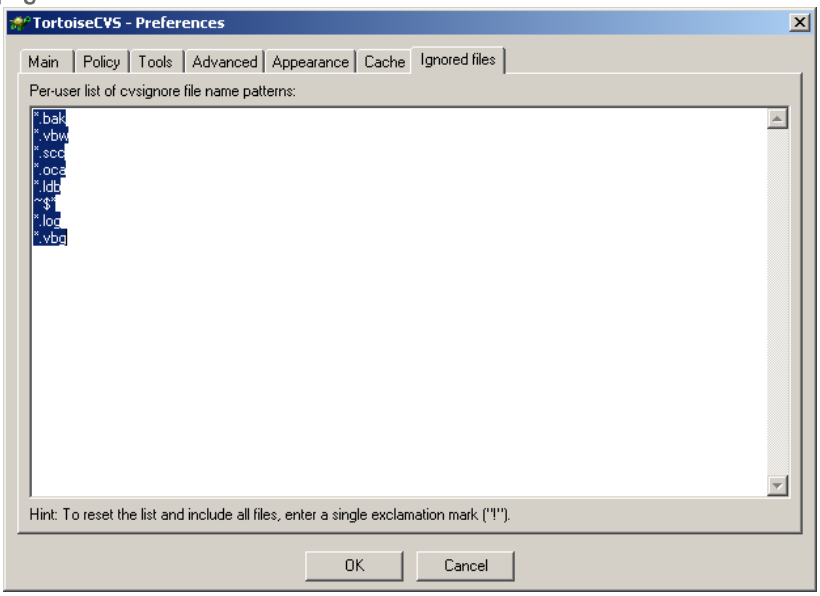
Prune empty folders is a nice option to make sandboxes tidier – without it, sandbox directories contain all repository directories even if they are empty on the selected branch. Some find they prefer this option off, in order to see the full structure of the repository at all times.



We use **WinMerge** for diff and merge, and make it a practice to install it to the same location on every machine so we can reuse these settings across machines.



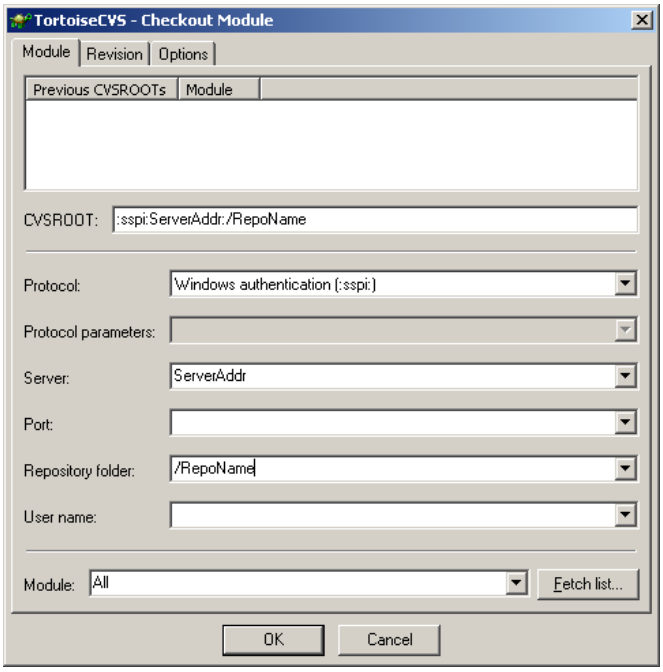
Specifying the directories in which you actually have sandboxes can speed up browsing performance, as can disabling menu icons - you don't actually need them if you have set up the explorer detail columns (see 3.2.6). That said, they make using the system so much nicer! A good compromise is to not make the folder icons recursive, so they will only indicate changed files directly within them, rather than in any of their children, grandchildren, great grandchildren etc.



Setup the global ignore list here. We have a standard list:

- ***.bak –** Winmerge backup files
- ***.vbw –** Visual Basic group settings
- ***.log –** Visual basic error log files
- ***.scc –** Sourcesafe files (we still had a few after transitioning)
- ***.oca –** Control TypeLib Caches (created by Windows when OCX objects are in use)
- ***.ldb –** Access database lock files
- ***.obj –** Object build files
- **~$* -** Word lock files

3.2.4 Setting up some useful defaults



TODO

3.2.5 Deploying config and defaults across multiple machines

TortoiseCVS stores most of its configuration in a registry key:

[HKEY_CURRENT_USER\Software\TortoiseCVS]

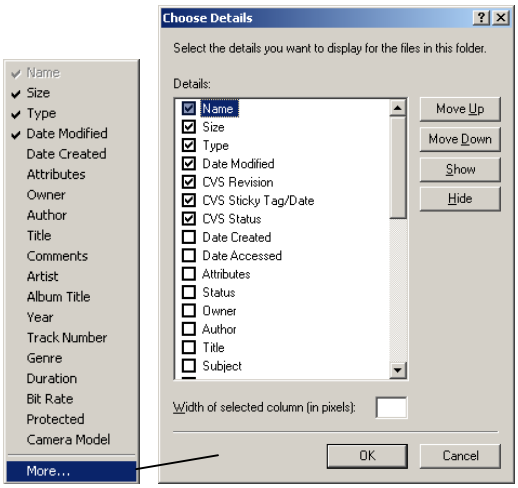
We handle multiple deployments by getting TortoiseCVS set up on a single PC, and then using **regedit** to export this registry key to a file. To apply it to any machine, you just need to run this .reg file and then log off/log on.

The only other source of configuration information is the global ignore list, which is stored in the user's home directory. We take a copy of this file, and put it in a well-publicised network share, along with the registry data, the latest version of TortoiseCVS and the latest version of WinMerge. This makes deployment as quick as possible.

Name	Size	Type	Date Modified
Optional		File Folder	02/12/2004 15:37
.cvsignore	1 KB	CVSIGNORE File	02/02/2004 11:37
cafit-tortoisecvs.reg	6 KB	Registration Entries	13/02/2004 16:39
CSDiff40.exe	1,236 KB	Application	11/02/2003 17:46
No CVS.flt	1 KB	Adobe Illustrator Filter	04/06/2004 14:39
TortoiseCVS-1.8.8.exe	3,858 KB	Application	02/12/2004 15:38
WinMergeSetup202.exe	2,005 KB	Application	24/03/2004 09:54

3.2.6 Explorer settings

We consider it essential to have Windows Explorer set into Detail mode (**View > Details**) and to then to turn on the CVS columns, by right-clicking on the column headings and selecting **More...**



Once these have been applied, use **Tools > Folder Options > View > Apply to All Folders** to ensure these columns are always there.

Name	Size	Type	Date Modified	CVS Revision	CVS Sticky Tag/Date	CVS Status
CVS		File Folder	15/12/2004 12:55	-	-	-
cafcompPersistorXML.vbp	2 KB	Visual Basic ...	13/12/2004 17:30	1.24		Unmodified r/o
compPersistorCls.cls	42 KB	Visual Basic ...	13/12/2004 17:30	1.20		Unmodified r/o
sortedFileList.cls	7 KB	Visual Basic ...	07/04/2003 17:27	1.3		Unmodified r/o

4. In use

This section covers some useful advice on CVSNT/TortoiseCVS in use and the best practice to avoid problems. Hopefully it will grow over time.

4.1 Safe practices

4.1.1 Merging

When you attempt to commit a file and another user has changed it since you last did an update, you will receive an out-of-date error and be told to update. At this point, the differences between the version you changed and the version the other user committed are merged into your working file. If CVS can't do this automatically, it will open you diff/merge program and ask you to resolve the conflict.

In order to second-guess CVS' automatic merging, and indeed to double-check one's own code changes, we have found the following procedure beneficial. It assumes you are using WinMerge as your diff/merge tool.

- Take any bits of code from the left hand side (LHS), which are missing from the right hand side (RHS) and add them to the RHS. This will often mean a mangled file with repeating sections of code. Don't worry about this for the time being.
- Save and quit WinMerge.
- Diff the file against CVS. You should find that the only differences are the changes that you made. Anything else should be fixed/removed, since you now have the other user's changes in your working file.
- In the directory with the merged file, find the CVS backup file for the file it automatically merged (usually the file name with a version number appended). Load this as the LHS into WinMerge and diff with your working file. Since you are now comparing your file pre merge on the LHS and post merge on the RHS, you can see exactly what the merge process did. Consult and modify as required.
- Commit.

5. Maintenance

5.1.1 The command line

It is important when maintaining a CVS deployment that the support engineer becomes familiar with using CVS from the command line. The command line can be used interchangeably with TortoiseCVS or indeed any other CVS client.

Full documentation can be found on the main CVSNT website.

5.1.2 Software updates and deployment

Checking

Our procedure is to try and keep the versions of CVSNT used by the installed version of TortoiseCVS and the version installed on the server the same.

We regularly check for updates to the stable TortoiseCVS (which can appear almost daily at times) and run this version on one or more machines (usually the machines of those responsible for maintaining CVS).

Once this version has been working with no issues and no significant bug fix releases for a couple of weeks, we mark it as stable. If it introduces bug fixes, which, if left unrepaired, could affect us, or introduces useful new features, we will deploy it to existing client machines.

Deployment

The install procedure is sufficiently simple to just e-mail around a link to the installer and ask all users to run it. As long as the user just runs the setup and doesn't uninstall TortoiseCVS first, the configuration should be kept and that should be all that is involved. This may be unsuitable for larger organisations, however.

CVSNT versioning

If an update to TortoiseCVS includes a new version of CVSNT, we usually take this opportunity to upgrade the server. Apart from being a suitable opportunity, CVSNT often introduces compatibility issues with older client versions as updates appear – it is easiest to avoid these by keeping the client and server versions in-line.

Updating the server is normally straightforward (stop services, run setup, start services), except on Windows 2003 Server (see 3.1.2)

5.1.3 Permissions at branch level

Granting or denying access by branch is not possible using NTFS permissions since it is at a higher granularity, within individual files. This needs to be done using CVSNT's own permissions system using the **cvs chacl** command.

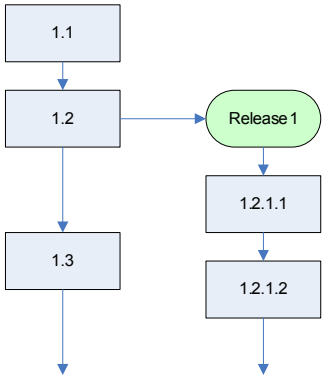
Detailed documentation can be found on the CVSNT website (a worked example can be found here: http://www.cvsnt.org/wiki/SetAcl).

Note that the **chacl** command has undergone several changes recently, apparently ahead of the documentation, and it may take some time and experimentation to find the right syntax.

5.1.4 Merging

An Example

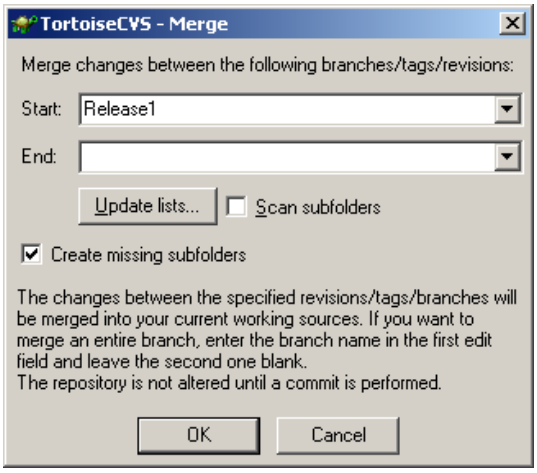
If you have created a branch called **Release1**, to create an effective code freeze prior to a release, whilst continuing future development on the **HEAD**, after a short while a file's revision graph might look as follows:



Let us say that changes **1.2 – 1.2.1.1** and **1.2.1.1 to 1.2.1.2** were bug fixes made in the weeks leading up to the release. It is necessary that these bug fixes make their way back onto the HEAD, since the **Release1** branch will effectively be closed once support for this release is abandoned.

Start and End

Assuming just this one file, we would at this point carry out a merge on the HEAD branch as follows:

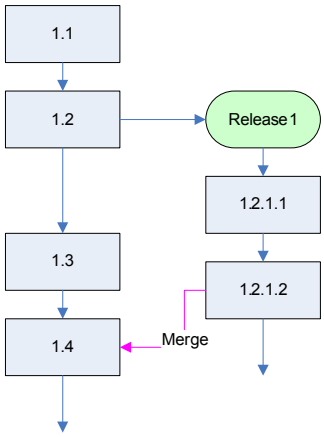


Or "merge **Release1** into this branch (HEAD)". This is a shortcut for:

The Start and End for a merge indicate the two versions between which we want to find the changes to apply. If the End is omitted, it will be taken to be the latest entry on the same branch as the Start.

Since in the first case we are saying "take all the changes made on the **Release1** branch since it branched from this branch (HEAD), and apply them to this branch", the latter case is identical since we have just picked out those version numbers by hand.

The merge points would look like this:



Multiple files

In practice, going through this process for every file individually across a large project, or several projects, is a tedious maintenance task. Therefore, the first notation (just using Start) really comes into its own – we can just use

Start=Release1 across a large number of directories and files and the merge should work as expected, where required.

However, there are many issues that arise from this. Firstly, CVS cannot be trusted to carry out merging across a large number of files unsupervised. There are always cases that arise where a programmer has moved a function to a different position in a file (which causes two copies to turn up in the merged result), or lots of similar functions cause mistakes in diffing – we have ended up with identical error handlers in consecutive functions causing problems for example.

Secondly, if merges are left for some time and then handled as a batch job by a single developer, it is likely that (s)he will not be familiar enough with every single change to either check CVS has done a good job, or resolve conflict cases.

Best practice
In addition to the safe merging practices described in 4.1.1 (this process can be applied equally well to branch merging), we have come to the conclusion that the easiest and safest way to maintain two branches is for every developer to merge every time they commit. This way, they will only be applying the changes they have made, which will usually be compact and familiar enough to resolve any conflicts.

6. Troubleshooting

6.1 Q&A

The following are common problems we have encountered and how to resolve them.

6.1.1 "No such tag <tag>" errors

We have encountered cases where the val-tags file in the repository's CVSROOT directory (not under version control) has become corrupted. This file caches the tag names in use so corruptions can lead to actions on a specific tag failing.

There are several mailing list items relating to these problems, for example:

http://www.cvsnt.org/pipermail/cvsnt/2004-April/012274.html

Here is an example of a corruption that we have encountered:

```
build-4-0-0-4 y
build-3-2-4-4 y
build-3-2-3-2 y
build-4-0-0-11 y
build-3-2-9-1 y
y ld-3-2-9-1
build-4-0-0-27 y
build-3-2-12-4 y
build-3-2-12-6 y
build-4-0-0-26p3 y
```

This file can be fairly easily corrected or you can just delete the contents, or indeed delete the file, but **be warned** – if the default permissions are such that any CVS user gets denied access to the file, you will encounter the error below.

6.1.2 "Permission" denied writing "val-tags" file

Sometimes operations such as Branch, Tag, Checkout (with a tag) and Update Special (with a tag) fail with an error like this:

cvs [server aborted]: cannot write D:/CVSRepository/Bonds/CVSROOT/val-tags: Permission denied

Error, CVS operation failed

This is usually caused by an NTFS permissions problem, almost always due to a kindly soul fixing the error in 6.1.1 by deleting the val-tags file. Ensure that any users with write access to any part of the repository also have write access to the val-tags file.

6.1.3 Invalid characters in fileattr.xml

See 3.1.6 for background information. If you end up with a well-populated directory with an ampersand in the name, containing several files, which are being edited by several people before this bug is uncovered, you have a major problem.

Just correcting the XML is insufficient since it will quickly get corrupted again. It is necessary to physically rename the files in the repository. This is a messy operation – if no-one were editing the files, you could just fix the repository and then delete everyone's sandbox directories and do an update. Not so if they are.

1. First, remove all read/write access to the parent directory or ensure that no users are likely to use the contents in any way.
2. Then rename the file(s) to remove the ampersand.
3. If you have renamed any actual RCS files (,v extension) then you will also need to open them and (carefully!) change any references in the file itself.
4. Change all the references to the badly named file in the fileattr.xml file. This should also fix the corruption. It is also advisable to take a note of any edits (so you remember to talk to the person concerned), and remove them.
5. Re-enable access to the folder.
6. For each CVS user, check if they had any files edited in the directories you have changed. If so, take a copy of these files from their sandbox.
7. Once you have copies of the edited files, **delete** the directory in their sandbox and do an update to get a fresh copy. This is much less effort than editing the Entries, Entries.Extra etc.
8. If any of the edited files were binary and you are using watch mode, get the edit back on the clean file(s).
9. Copy the edited files back in over the top of the fresh ones.
10. Repeat for all users (!)

This is a mistake you only let happen once... the kind of problem which one can shrug about in an open source repository with hundreds of users and no watch mode, since the clients are generally savvy enough to fix their own sandboxes if told to do so, but in an enterprise scenario it generally falls to the support engineer to fix both server and all clients.

6.1.4 Watch mode gets turned off

We have our suspicions that this may be happening spontaneously, but haven't been able to confirm anything yet. It has occurred several times due to careless editing of fileattr.xml files when fixing error 6.1.3.

Just re-enable watch mode for the directory (**cvs watch mode on**) and then mark the user(s) sandbox files as read-only.

6.1.5 Hanging edits

This is the bane of anyone who decides to use watch mode. In many cases, after a user has finished with a file, their edit may be left open. This completely locks out other users from editing binary files, and is annoying for text files.

Reasons include:

- Deleting sandboxes with edits in them and checking out a new one
- Users having multiple sandboxes on the same branch and forgetting about one of them
- Users just forgetting they have edits
- CVS bugs

The latter appears to be the most common – although as I write there claims to be a fix in the latest version of TortoiseCVS (1.8.11).

In many cases, you can just ask the user concerned to unedit the file. If their file is not showing as edited, they may be able to force an unedit by marking the file as writable and choosing **CVS > Unedit** from the explorer menu.

If that doesn't work, you will need to force an unedit as a CVS administrator. From the command line, **cd** into a sandbox directory containing the file and type:

```
cvs unedit –u <username> <filename>
```

Where **<username>** is the name of the user whose edit you wish to remove.

This often doesn't work either. In this final case the only way we have found to remove the edit is to locate the offending fileattr.xml file in the repository and remove the watch entry by hand:

```
<file name="frmMain.frm">
  <editor name="DOMAIN\username">
    <hostname>ServerAddress</hostname>
    <pathname>D:\Sandbox\MyVBProject</pathname>
    <tag>HEAD</tag>
    <time>Thu Jan  6 12:44:57 2005 GMT</time>
  </editor>
  <watched />
  <watcher name="DOMAIN\username ">
    <temp_commit />
    <temp_edit />
    <temp_unedit />
  </watcher>
</file>
```

Would become:

```
<file name="frmMain.frm">
   <watched />
</file>
```

6.1.6 Moving servers causing disconnected sandboxes

If the address and/or DNS name by which the users' sandboxes are referencing the CVS server changes, all the users' sandboxes' **Root** files will be pointing to the wrong server – in the case of a large sandbox a big operation to fix.

We wrote a quick and dirty script in VB6 to fix a sandbox, the "meat" of which is here. We are fully aware how ugly the code is!

```
Private Sub Command1_Click()
    'On Error GoTo stdErr

    Dim fol As Folder
    Dim fso As FileSystemObject

    Set fso = New Scripting.FileSystemObject

    Set fol = fso.GetFolder(Text1.Text)
    RecurseDir fso, fol

    RichTextBox1.Text = "DONE" & vbNewLine & RichTextBox1.Text
    Beep

    Exit Sub
stdErr:
    MsgBox "Error: " & Err.Description, _
            vbExclamation + vbOKOnly, "Error"
End Sub

Private Sub RecurseDir(ByRef fso As FileSystemObject, _
                       ByRef fol As Folder)
    Dim fil As File
    Dim subFol As Folder
    Dim ts As TextStream

    If fol.Name = "CVS" Then
        For Each fil In fol.Files
            If fil.Name = "Root" Then
                Set ts = fso.OpenTextFile(fil.Path, _
                                ForWriting, True)
                ts.WriteLine Text2.Text
                ts.Close
                Set ts = Nothing
                RichTextBox1.Text = "Fixed: " & fil.Path & _
                          vbNewLine & RichTextBox1.Text
                DoEvents
            End If
        Next fil
    Else
        For Each subFol In fol.SubFolders
            RecurseDir fso, subFol
        Next subFol
    End If
End Sub
```

The program expects a TextBox called "Text1" containing the path of the folder to fix, and writes a status log to a RichTextBox control. It requires the Microsoft Scripting Runtime.