

# Overview of Subversion for openEHR

---

## Introduction

To all users: you can get into trouble without at least a basic understanding of what Configuration Management is, and what Subversion does. If you are new to these kinds of systems, please read these pages, and if possible the [openEHR Change Management Plan](http://www.openehr.org/svn/specification/TRUNK/publishing/CM/CM_plan.pdf)<sup>1</sup>, and the basic chapters of the [online subversion book](http://svnbook.red-bean.com/)<sup>2</sup>.

## What is Version Control (VC)?

We don't have the space here to provide a proper lesson on this complex topic. But, from the user point of view, it is conceptually simple. The main points to understand are as follows.

Version control in its simplest form (such as UNIX SCCS or RCS) enables successive changes to a file to be captured and remembered. The state of the file at previous points in its history can be retrieved if necessary. To effect this, the user will not just edit the file, but will perform the following logical steps (which may have different names in actual version control systems):

- check out the file
- make changes
- check in the file

The astute user will have realised that 'check out' and 'check in' imply that something must exist to check out of, and in to. This is a version control file - a little repository whose job it is to store file contents, and changes to contents in a smart way, so that the versions are remembered.

## What is Configuration Management (CM)?

Now, in reality, we are never interested in changing just one file, but in creating and changing an entire collection of information, typically corresponding to a project repository, usually represented as a directory hierarchy in a file system. Collections of files (of whatever kind) in a structured file system, or part thereof, can be thought of as a 'configuration'. Configuration Management is about managing changes to entire configurations, not just single files. The kinds of changes we can do are more involved than for a single file, and include:

- create and modify files
- delete a file
- rename a file
- move a file
- create, delete, remove, rename a directory

The focus of our mental model of change is now the configuration as a whole (think in terms of a project directory hierarchy if that is more comfortable), *not* a single file. Any change you do to the project repository may be any combination of the above kinds of changes, to as many or as few files as you like. When you commit the changes, you are committing what is often called a change-set to the repository, bringing it into a new state. The question is now: what combinations of changes are sensible? This is

---

1. [http://www.openehr.org/svn/specification/TRUNK/publishing/CM/CM\\_plan.pdf](http://www.openehr.org/svn/specification/TRUNK/publishing/CM/CM_plan.pdf)

2. <http://svnbook.red-bean.com/>

what Change Management is about. The basic criterion is that ideally, you want your repository to be in a consistent state at all times. That means that any given change-set should bring the repository to a new consistent state. The definition of 'consistent' is often project-specific, but usually includes criteria like:

- the source code compiles and runs
- the documentation is up to date
- every change-set is documented by a Change Request (CR) or similar document

However, such idealism is not always possible: consider that in the early stages of a project, the state of the repository is like a house just in the initial stages of construction. Everyone is working frantically, and the worksite seems like chaos. The repository is unlikely to be 'consistent' until some point down the track when the code first compiles or runs. At such a point, the group may decide to go into a disciplined development mode, where they require the above criteria to be met for every change. In particular, they will start using CRs.

There are many facets to CM and version control, which we will not discuss here.. The ones relevant to *openEHR* are documented in the [openEHR Change Management Plan](#)<sup>3</sup>. Subversion implements many of the recognised features of version control and configuration management. These pages provide concrete help on using the Subversion tools.

References on CM:

- Anne Hass. [Configuration Management Principles and Practise](#)<sup>4</sup>. Addison Wesley.
- Technical University of Eindhoven, Information Systems Section. [Online software engineering notes](#)<sup>5</sup>.

## Subversion for *openEHR*

Subversion is a version management tool in which each repository is a version-controlled piece of a file system sitting on a server, and accessible using a URL. *openEHR* has a number of subversion repositories, including:

- <http://www.openehr.org/svn/specification>
- <http://www.openehr.org/svn/knowledge>
- [http://www.openehr.org/svn/knowledge\\_tools\\_dotnet](http://www.openehr.org/svn/knowledge_tools_dotnet)
- [http://www.openehr.org/svn/ref\\_impl\\_eiffel](http://www.openehr.org/svn/ref_impl_eiffel)
- [http://www.openehr.org/svn/ref\\_impl\\_java](http://www.openehr.org/svn/ref_impl_java)

The web view of each repository is provided by Apache 2.0, but the URL is also the way of referencing the repository for checkouts and commits etc, since the server side is actually implemented as an Apache module. Pretty web views are provided by the Websvn PHP interface at <http://www.openehr.org/wsvn>.

One Subversion "repository" is a versioned file tree, with versioning applied across the whole tree, not on individual files. This has a number of consequences:

- "revisions" in Subversion always mean "the revision of the entire repository"
- distinct projects will usually want separate Subversion repositories, since otherwise changes to unrelated work in the same repository will increment the revision of your work as well
- any changes you make when committed will take the repository to a new state.

To manage tags and branches, Subversion does not provide any inbuilt features in the repository

directory structure, but its lazy copy methodology does support representation of tags and branches

3. <http://www.openehr.org/svn/specification/TRUNK/publicities/CM/CM-plan.pdf>

4. <http://www.awprofessional.com/articles/article.asp?p=31451&rl=1>

5. <http://wwwis.win.tue.nl:8080/2R690/cm.html>

once created. For this reason, Subversion repository administrators normally include a top level directory hierarchy like this in the repository:

- trunk
- tags
- branches

In *openEHR*, we have opted to capitalise these names, to distinguish them from normal directories, and add a directory for release branches. There are basic rules for how to work in these top-level directories, which are shown against the directories used in most *openEHR* repositories.

- TRUNK - normal development
- BRANCHES - lazy copy of some existing revision in TRUNK - any modifications allowed
- RELEASES - lazy copy of some existing revision in TRUNK - bugfix modifications only allowed
- TAGS - lazy copy of some existing revision in TRUNK - not to be modified!

Most users will just work by checking out the contents of the TRUNK directory from the server, making changes, and committing them.

The online [Subversion book](#)<sup>12</sup> contains chapters which all new Subversion users should read; the [Subversion FAQ](#)<sup>13</sup> is also useful.

## If you only remember three things...

### 1. **Don't do deletions, moves or renames in the filesystem - always use the subversion commands to do this.**

Subversion can correctly track not only changes to content, but also to names of files and directories, to locations of files and entire directories, and deletions. However, to do any of these things, you **MUST** use the rename/move/delete facilities in the client side subversion tools! Simply using command line mv/cp/rm or the Windows Explorer (or unix GUI equivalent) to move and rename things **WILL NOT WORK**, and **MAY CAUSE SERIOUS PROBLEMS** in a repository. Please remember, every repository is shared by a team, and potentially read by hundreds of people, so please take care.

### 2. **Never commit changes without first doing an 'update' .**

Others may have changed files while you were changing files; sometimes, they may be changing the same files. Doing an update before you commit brings your repository up to date; if the other user committed first, Subversion will detect the clashes, and will create appropriate copies, ensuring your changes (as yet uncommitted are not lost). You will have to do a merge or replace. See the subversion book for details, or the FAQ.

### 3. **Always provide a log message when committing changes.**

Everyone in the world can see the repository you are working on - including all the content and change history. For example, on [this page](#)<sup>14</sup> you can see the top of the specifications repository, with a log message "release 0.96". For non release changes, you are encouraged to include a message up to 3 lines or so.

---

12. <http://svnbook.red-bean.com/>

13. <http://subversion.tigris.org/faq.html>

14. <http://www.openehr.org/wsvn/specification/?sc=0>