Paper AS09, RC05

Implementing CFR 21 part 11 for SAS without tears or joins

David J. Garbutt, BSI AG, Baden-Dättwil, Switzerland

ABSTRACT

This paper argues that the best way to construct an easy-to-program in and CFR 21 part 11-compliant environment for SAS programming is by using a version control system, specifically the IBM Rational product ClearCase. It concentrates on architectural issues to do with protecting *data* and *outputs* as well as *programs*, and stresses the importance of linking an output with its constituent parts. The main reasons for using ClearCase are explained and illustrated with examples from a successful implementation. Enough information about ClearCase is included to show why it is uniquely suited, amongst all Source Code Management Systems, for this task. Although this paper focuses on SAS the environment can also be used for other programs with little or no changes.

INTRODUCTION

This paper discusses why and how a version control system can be used to make CFR 21 part 11 compliance a daily reality for SAS programmers.

To do this successfully we must combine *ease-of-development* with *rock-solid auditing*. Any system we build must also lend itself to integration with other tools and future changes, for example, in SAS itself, in business processes, or in new methods of automation. The advantages of using programs like *make* for controlling large SAS projects are discussed and the problem of the tedious and error-prone manual listing of dependencies for SAS programs is solved.

This paper is not a project report, a guide to make or ClearCase, nor is it a marketing gambit: its aim is outline how and why using ClearCase for this problem works. A secondary aim is to illustrate the way in which the pillars of the design fit together to support a powerful and comprehensive solution allowing SAS programming to be done in an optimal way.

Some screen shots from a successfully implemented solution are included.

BACKGROUND OVERVIEW

Many Pharmaceutical companies have independently developed their SAS environments for Clinical Reporting but those now extant tend to share the same characteristics:

- We have separated the Clinical Data Management System(s) from the system for creating table output. Typically they are separate parts of the organization
- We copy from the CDMS to SAS datasets in some separate file area or normally server.
- We are using SAS programs as ASCII files (not catalogs).
- We keep lots of copies of things 'just in case', and certainly copies of everything delivered outside the organisation. To fill this need we typically have a complicated directory structure for each project, including sub-trees for each interim analysis.
- For an average clinical study we might need 250 tables or,
- As many as 2000 for some trials
- The programming team size ranges from 2 to 20 or 30 and is often geographically dispersed.

IS VERSION CONTROL FOR SAS PROGRAMS OPTIONAL?

In a recent text discussing the position and strategic use of Source Code Management (SCM) systems in the software industry the author, Brian White [1] says:

> 'Version control is deemed mandatory for any software project with 3 or more developers'

He derives the version control system needs from the size and complexity of a project. This throws interesting light on the typical practice in the Pharmaceutical industry. White would classify the typical SAS project mentioned above as 'medium to major' and adding geographic dispersion of team members would make it 'major to extreme'. He suggests for such 'major to extreme' projects only a large-scale modern SCM tool will be sufficient. For small teams (2-5 developers working together, at one site, on a discrete (software) product with no dependencies on other projects) he believes early SCM tools (such as CVS or RCS) would be adequate. For the smallest projects that he classifies as 'Individual projects' he says

> 'In general, the individual does not require an SCM tool to solve his or her problems. However, an SCM tool can offer such benefits as better organization and security, thereby freeing the individual from doing version control via directory and file copies.'

To summarise: the benefits of version control systems are accepted by all in the software industry. The benefits are many but include solutions to the following scenarios (as specific examples) :

- Files deleted accidentally
- Test data accidentally used for submission
- Wrong treatment codes used for some outputs (which?)
- Program tweaks after validation introduce a bug
- Two programmers fix different problems in a program. Neither has a version that is completely correct.
- Serious bug found in a last month's released version of a standard macro - what is affected?
- Home directories used for creating deliverables, and the team not informed.
- SAS viewer changes modification date of data file (V8.2) so we have to re-run programs to make the dates consistent for audit.
- We changed a couple of titles, but with 2000 outputs to make we can't easily tell what should be re-run. Let's just re-run everything. Oh, yes I suppose we will have to re-check the outputs….

I am sure the reader will be able to think of many examples from their own experience of the kind of issues that occur beyond the control of formal procedures which can add up to a disaster that could be avoided, or at least ameliorated, if a version control system had been in place.

White's conclusion is that for all but trivial projects some form of SCM is essential. This is for normal software development not even for software in regulated environments. Surely using SCM can give all those benefits to our SAS software projects and at the same time cover the legal requirements of 21CFR11?

WHY DON'T ALL SAS PROGRAMMERS USE VERSION CONTROL?

SAS is different is the main reason I have heard as a response to this question. And it is true.

SAS *is* different from C++ - because no compiled version is normally kept it is effectively a scripting language. So, for example the program can easily be modified without being necessarily re-run. It is also possible to edit an output, for example to fix a typo in a title.

But it is not just that SAS is different. More important is that our *product* is different - it is not a compiled program that can be put on a CD and sent complete-as-created to a customer. It is an *output* of some type (html, pdf, rtf, cgm…) and it is different in content (and, therefore, meaning) according to the **data** used to generate it.

It is also true that the dependencies of SAS programs in our environment are more complex and dynamic than a typical small C^{++} application.

Therefore we need to control **program**, the **data** *and* the **outputs** to ensure they were created together.

KEY REQUIREMENTS OF CFR 21 PART 11

1. A log is kept of changes to programs independent of the program itself and
2. Outputs can be traced back to their source(s)

Considered together these two requirements imply that previous versions of data, outputs and programs must still be available somewhere, and must also be linked in some way so they can be located together and accessed again. For diagnosis of problems it is also very useful to have past data and program versions.

For example: Two submitted report have different numbers of AEs or patients. In cases like this it is usually most important to find out why the problem has occurred so we can correct outputs or find a reason (perhaps the two output use different populations because of an undocumented request, or perhaps definitions of AEs have changed, etc.).

Logically also other software (macros, third party products,…) should also be available because they can also affect the validity of the result.

All these requirements mean one thing, keeping metadata about:

- changes
- contributing components
- deliverables
- processes that have happened to each program

How to add CFR 21 Part 11 support to an application

There are two approaches that I am aware of- the first one, almost universally used, is to reprogram an application or build a customized wrapper for each application. A smarter of variety of this that could be used (on Windows, at least) is to add support in the application for the SCC API that is supported by several versioning tools. The second is to run ordinary (i.e. unmodified) applications within a general encapsulating environment; this approach has many *prima facie* advantages, and is the one advocated and exemplified here.

This paper only refers to SAS but in fact the environment discussed here also supports NONMEM, S-plus and even integration of S-plus with SAS via clearmake.

OUTPUTS, DATA, PROGRAMS AND THEIR DEPENDENCIES

Tracking SAS program dependencies is not trivial because of:

- The high number and complexity of the dependencies
- The use of macro variables in libnames giving dynamic dependencies, difficult to resolve except at run-time
- Compiled macro catalogs in libraries
- The autoexec call-hierarchy means that the same program can be have results dependent on the directory the program was run from. Exploiting this mechanism gives a brittle set-up easy to misapply.

- Command line options can affect which macro versions are used, which macro version is called as well as which autoexec is called. They can also be used to pass arguments (seen as macro variables by the program). Command line options are not tracked in the saslog.
- Libnames point at directories, so their meaning is time-dependent. If they contain relative paths they will point to different places depending where they are run from. On Windows they can contain a disk letter and therefore point to the same or a different path depending on the set up the user has made, and independent of the letter used in the libname.

Fig. 1 shows an example of a SAS output and some typical dependencies. In this case they are traced back to the data extraction from a CDMS and the following step of de-normalisation which creates the analysis datasets is included specifically.

We don't want to *pre-define* all dependencies because that is not robust against human error. It is also tedious (have you ever listed all nested macro calls when you have a system for analysing tables? When the macros called are data dependent?). It is not a good idea to solve the problem by using some arbitrary rule to make border around what is 'our stuff' and what is not, meaning we do no tracking of 'outside' objects. If we do that we can be sure (by Murphy's Law) that one day beyond the border is exactly where the problem will be. And of course we will be asked 'why didn't you include these elements, or such-and-such a directory?'.

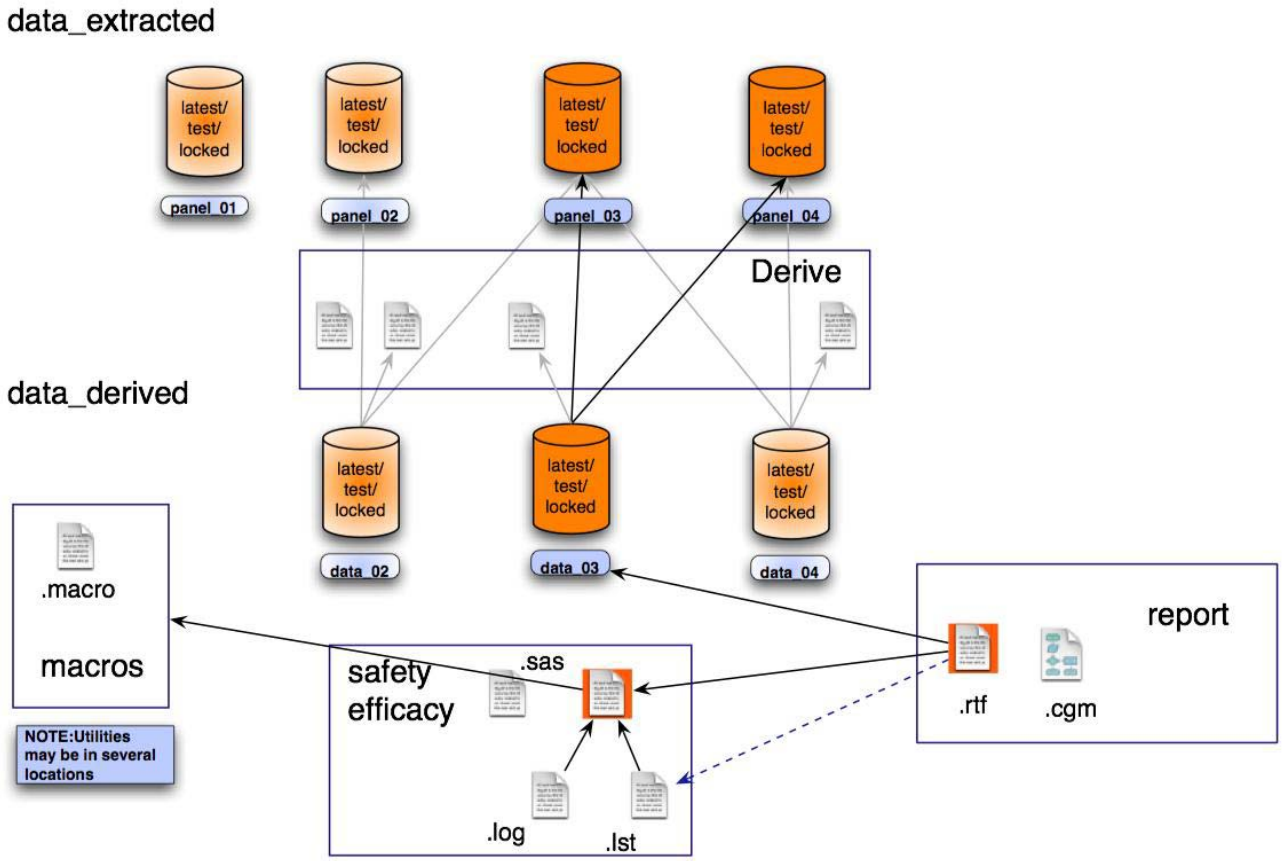


Figure 1. An example of an output and its dependencies

Programs are files, they exist in a file system; they have permissions, modify dates, sizes… but no other metadata (comments, attributes, modifying user…). So, how can we connect additional information indelibly with a file? And how can we associate an *output* with the both the correct *data* and the correct *program*? How can we ensure the link is not broken, by future work or by human errors? How can we track this without relying on a human maintained database which would be too prone to errors of omission? How we can provide for a client to view this information from another system or client, perhaps viewing files across the web.

Is there a component of the computer system that knows about dependencies? No, not directly. But, the operating system sees this information because all file open commands pass through it. Therefore, that is where our SCM must live!

THE WORLD OF VERSION CONTROL AND CONFIGURATION MANAGEMENT

Keeping metadata means having a place to store it, so we need a database holding details of all SAS programs and outputs. This is a big part of what version control systems do.

There are many systems available and they have been used and refined for the last 40 years they range in price and complexity from CVS and Subversion (open source, free) to major proprietary products like Perforce, Star Team, and Rational ClearCase.

As mentioned before our problem is non-standard because we are keeping program, output and data input as well. This requirement is out of the scope of almost all SCM systems. Many do not support storing binary files in repositories

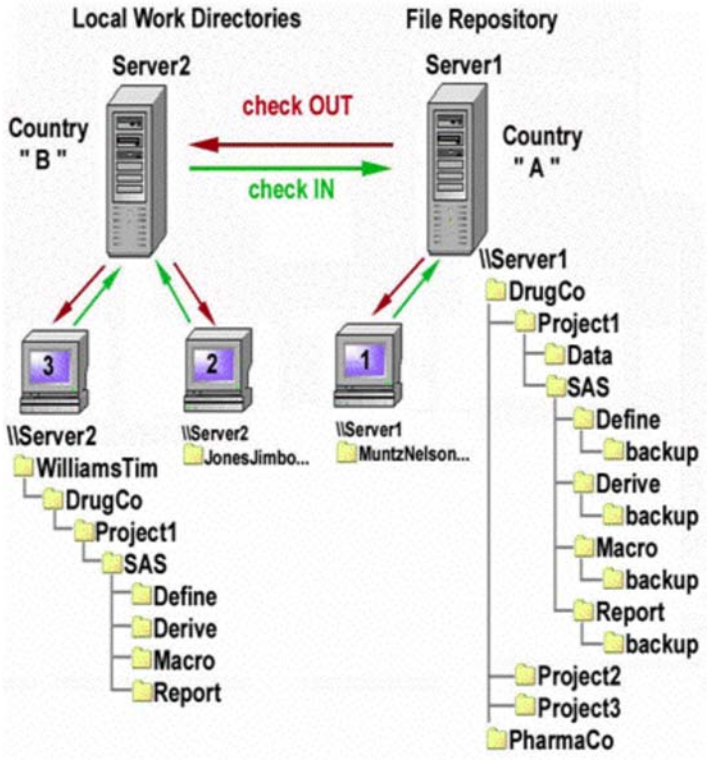


Figure 2 How a user's workspace relates to the version repository

Most version control systems function by creating a workspace or work area per user. A workspace is a separate (partial) copy from the repository. Figure 2 shows how a central directory structure for *project_1* on *server_1* in *country_A* can be replicated and copied to *server_2* under the directory *WilliamsTim*. This copying process is called called *check-out*. The original may now be locked in some way so other users cannot also make a check-out as well. SCM systems differ significantly in both their default and the set of available policies to deal with this scenario. Putting the program back to the repository is generally referred to as *check-in*.

This paradigm, (See **Figure 2** from Williams, SUGI27 [2]), gives the user a sandbox to modify the part of the system they are responsible for without affecting, or being affected by, the work done simultaneously by other programmers. So it is possible to do unit testing without the risk of breaking an already working application.

This implementation solves two problems for the programmer:

1. a central repository of programs and data giving increased security
2. information can be replicated across distant sites

It is less obvious but in this implementation check-out is necessary for read access to be possible for an object; in fact, if an item is not checked out from the repository then it does not exist at all as far as a program running in the workspace can see. For example, a libname statement might work correctly, but accessing a particular dataset could fail.

There are several disadvantages of this method for our desired usage:

- many programs only need read access to a data set, giving more permission just opens us to problems
- If a program is changed such that it needs another file to be checked out then the user then has to remember to perform the check-out before running the program. Forgetting to do that will cause the program crash. This error may be easy, or hard, to diagnose from the saslog. Delays to development can therefore result from this issue. Human nature being what it is such mistakes happen most under pressure, precisely when they are most damaging to all concerned.
- Disk space demands are proportional to the number of programmers times the number of delivery copies kept because every programmer needs access to the data. Disk space is also wasted because a programmer may have to generate a copy of another programmers output so cross-checking can be performed.
- Cross-checking of outputs (Integration testing) is more difficult because you need to keep track of where the latest copy is, and then request access to another workspace. Or a whole workspace is dedicated for cross-checking, and further increasing the disk-space demand.
- If you only need the latest copy of an item, especially data, then you must frequently refresh your workspace just in case a new version has been made.
- It is not very obvious how to implement a safe area for unblinding outputs or lab data.

GOAL SUMMARY: WHERE DO WE WANT TO BE?

- Preservation of work done (and delivered)
- Easy program development : No distinction between production and development
    - in location of files or
    - The methods for running programs
- Validation process support (do not deliver outputs created by unvalidated programs)
- Using automation more
- Improved robustness of the processes
- Assurance that unknown changes have not been made
- Trackability of programs and outputs
- For tracing problems. Both in study specific programs and general utilities.
- For CFR 11 part 21 - why were changes made? By whom?
- No re-running programs just to be sure dates of log and list files match and are consistent with the data date and time.
- Cross platform (Unix plus Windows)

One way of looking at the robustness of our development processes is to examine the kinds of excuse we make when things have gone wrong. We would like to move from the 'Oh, X happened and it failed' to excuses like 'Oh NO! X and Y happened and because Z and Q were also true we had a problem.' I term the first type of excuse a first order error (it is also often called a unit test) whereas the second is a fourth order error. Doing enough testing to prevent first order errors is usually feasible. But the more complicated the conditions the harder it is to detect them (and the more border conditions and special cases there are to program for).

The underlying truth here is that if it takes an effort of N man days of testing to ensure that if condition X occurs the program performs correctly then every other factor we add to the testing (Y, Z and Q) increases the testing effort to test all combinations exponentially. So we can never get to 100% assurance without an effectively infinite effort.

But all is not lost because automation and careful systems development can give us a lot of protection. And trackability can give us the confidence we won't be hit twice by the same problem.

WHICH VERSION CONTROL SYSTEM SHOULD WE USE?

Our key reasons to use ClearCase for this problem are:

- ClearCase from IBM Rational is a mature product on market for over 10 yrs
- the seamless integration of data, programs and output (it supports binary files in the repository)
- Data, programs and outputs can be put into databases (VOBS, equivalent to a repository in other systems)
- Triggers that are fully scriptable allow a customized work flow to be created
- it can enforce the workflow related to these objects
- it has the ability to create audits of outputs built within the system
- it can actually detect unnoticed dependencies when creating outputs
- it does not use the normal workspace paradigm
    - flexible views provide read access without a check-out
    - the efficiency and power of views that avoid the need for many working copies with the attendant loss of auditing control
- the possibility to include the outputs of various windows programs without Unix batch interfaces
- Cross platform (GUI on Unix and on Windows and with a web client)
- Very flexible set-up as to number of servers and location of ClearCase components
- The MVFS is a version-aware file system supporting
    - Fully customisable meta-data
    - And run-time auditing of created objects
- MS-Office integration
- Versioning of XML files (interesting with CDISC coming)

A VERY QUICK TOUR OF CLEARCASE

To connect you to your files ClearCase provides a GUI interface that looks much like Windows explorer.

Figure 3 shows a screen shot of the CC explorer, looking much like Windows explorer but with several notable differences.

First version information is visible (circled in red) and icons in the left margin show the status of files. The tick marks icon indicates we are seeing a checked out version. Top level directories under 'daves_view' are actually separate version databases (VOBs) not sub-directories. They could be located locally on the client or on the same server.

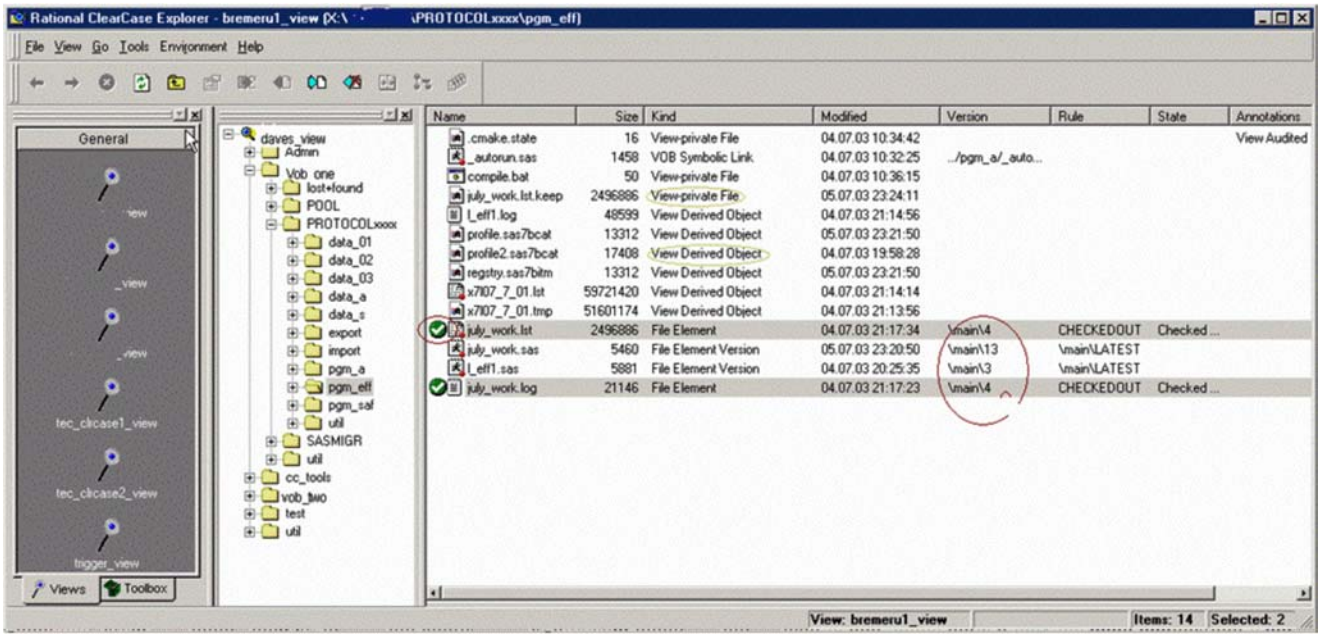
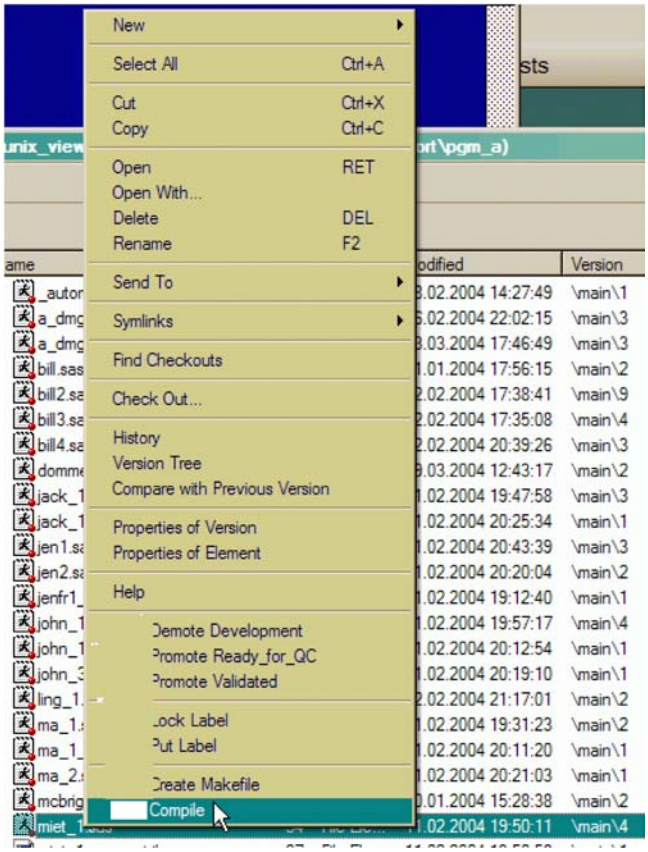


Figure 3 ClearCase Explorer (Windows GUI)



Many ClearCase commands are available from the menus or from context menus. When we right-click on an item (file, folder, VOB, View) the menu that pops up is context-, object- and status- sensitive. For example the context menu for july_work.log would contain *Check-in* as an option; For the file above it the same position would have *Check-out*. In Figure 4 we see it with the standard list of check-out, version tree, compare with previous version, etc., as well as customized (workflow) actions:

- Demote to status Development (will need re-validation)
- Promote status to Validated
- Promote Ready_for_QC
- Put a label
- Lock a label
- Create makefile (from a simple CSV format text file)
- Compile (means execute makefile on Unix host)

There is of course a lot more to ClearCase than this, but I have tried to restrict what is in this paper to the pieces that are vital to our problem. They are not necessarily given the same prominence in the ClearCase documentation and marketing material because our application is somewhat unique as previously discussed.

Figure 4: The Customized right-click menu

The two alternatives are part of the custom workflow and actually depend on what the validation level for the file is: critical, most-critical or uncritical (!). It is also possible to cancel the check-out of a file.

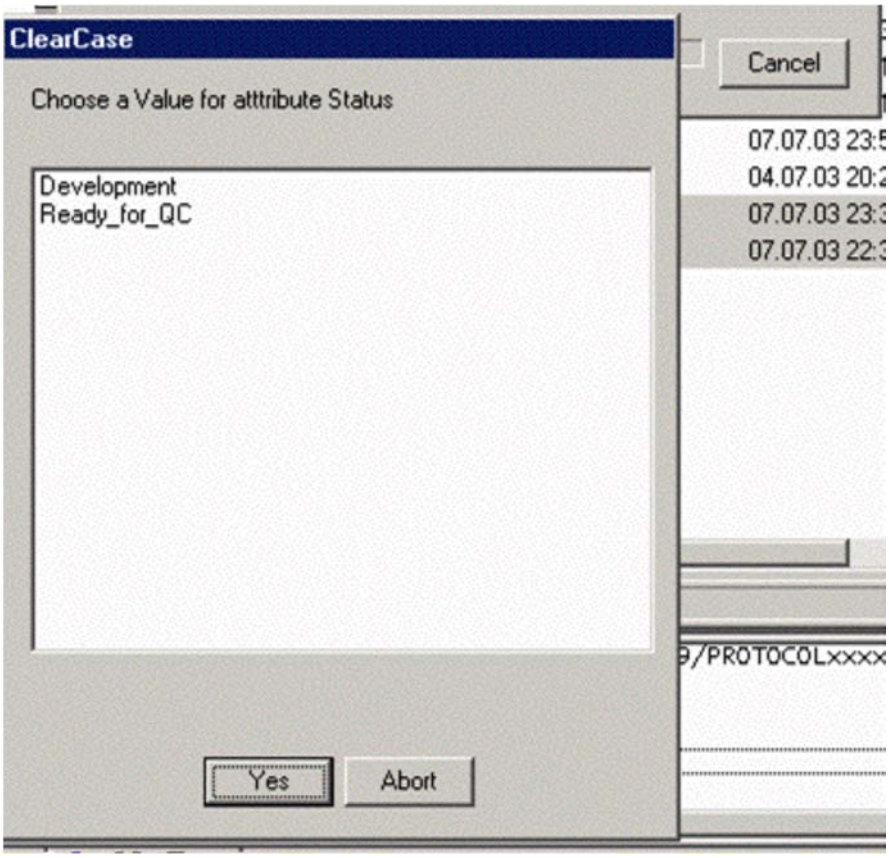


Figure 5 Choose a status for the SAS program as it is checked-in after it has been modified

ARCHITECTURE OF THE SOLUTION

These are the crucial parts of the architecture:

- ClearCase has a *customisable interface*
- Views give *efficient access* to SAS data
- MVFS enables clearmake to create *derived objects*
  Derived objects know what files and which versions were used to make them, and what command and options were used.
- User defined triggers allow fully *customised workflow*
  Creating and maintaining custom attributes per object
- Make a sandbox to allow development with as few constraints as possible by:
  - Keeping all objects in View and VOB databases (with caching for efficiency)
  - Policing the exit:
    Only allowing correctly generated outputs out into the world means the system can be simpler for SAS developers

We will now look at each part of the design in more detail.

PILLAR 1: CUSTOMISABLE INTERFACE

The interface can be extended and more importantly this is relatively simple because ClearCase is event-driven and has built-in hooks whose user-written code can be added. It is open sourced in the sense that the source just a text file. It becomes easy to see, for example, where the list of attributes in Figure 5. is stored. When an event such as check-out happens a trigger is called and this trigger can be quickly customized (usually with a Perl script, capable of running from windows or from Unix). This system is secure and cannot be subverted by the user. It can of course be by-passed by the administrator when necessary.

PILLAR 2 - VIEWS ARE POWERFUL AND EFFICIENT

A ClearCase view reveals file and directory elements via the MVFS. It appears to the user like a network drive (Windows) or part of the file system (Unix). Views themselves can actually be located anywhere and still be viewed the same way. All elements visible in the view can be read by a program, if they are not checked out their access is set to read only.

For example, see Figure 3, a typical address might be:

```
V:\viewname\vob-B\proj_a
P:\vob-B\proj-a
```

Or, on Unix:

```
/view/viewname/vob-B/proj_a/
/vob/vob-B/proj-a
```

Such paths can be put directly into a Libname statement with no problems.

More about views

Views can be private, or shared, named for tasks, or for users; they are light-weight and fast to create. Files are served (and cached) only as the users read them, just browsing a list of files does not cause their content to be generated.

Therefore to run a program you do *not* need a private copy of all the data it uses, neither do you need to know which files will be accessed. (The exception is that all VOBs accessed must be *mounted*). When a program runs the listing and log files are created in the view. In this sense it is like having your own home directory with a copy of everything in it. Except that your directory is organized exactly like the main project, synchronised with it, and without you making a copy. Also others can see that you have a particular file either by browsing directly in your view or, by searching for checkouts.

Views are efficient on disk space because they only take the space needed for your own files. Otherwise with, for example, 20 programmers and 10Gb of files in a project we have a high demand for work areas, and we rely on people remembering to check work back in if versions are changed.

The version that will be visible in a view is dynamically re-configurable. The normal default is to see the latest version, or the checked out copy if there is one. But Views can be configured in many ways. If say all versions used for a given ISS are labeled such as 'ISS-oct-05' it is easy to set up a view to only see those versions current at that time-point. This makes it trivial for ClearCase to fulfill the requirement that outputs can be recreated. If labels were not applied then views can also be configured to correspond to a particular date and time.

Views are the *only* way to access information in a ClearCase VOB. A VOB is a special database built for this function and very stable and reliable.

VOBS and views may be on clients or servers or mixed the storage unit is both the VOB and the view.

A version tree for a file is shown in Figure 6 it is available from a right click, the eye icon indicates the version visible in the current view. The version tree browser provides a simple way to explore version differences via the compare versions tool, and also to see what label and attributes apply to each version.

View private files

View Private Files are a uniquely ClearCase concept and they are exactly that - they only exist in your view, unless you add them to source control. This means you can check out a program, run it 30 times and then just check it in at the end of the day. The log and listing files will be view private files and you can delete them or keep them as you decide. Note that this concept means ClearCase easily supports parallel development of a program. The two versions and their products can co-exist in different views. They can even be compared using standard system tools like diff.

PILLAR 3: CLEARMAKE AND THE DO

Standard make

There is not space here to discuss details the details of the make program and how it helps solve our problem. The senior-programmer summary is that at the expense of writing down a list of programs and data needed to build an output we get a system where running one command (most likely as a batch job) will prepare and update all the outputs that are out of date, *but no others*. So if only panel04 is re-extracted (see Figure 1) then data_03 and data_04 will be re-built but data_02 will not. Subsequently any reports using data_03 or data_04 will also be re-run. Make examines the tree of dependencies implied by the information provided in the make file and runs the necessary commands in a correct order. A part of the example makefile for the RTF file in Figure 1 looks like this:

```
Report.rtf: data03.sas7bdat   std_macros.sas
     sas report.sas
     checklogs report.sas

Data03.sas7bdat: build03.sas panel03.sas7bdat  panel04.sas7bdat
     sas build03.sas
```

The rest of the file has entries for each *produced* object in that diagram. The objects being produced are called *targets*, the objects required to build a given target are its *dependencies*. The syntax of the make file is

```
target : dependency ….
<tab>command
```

A dependency can be a file (path) or another target. This allows the same make file to run a whole series of tasks, or just one part, therefore giving the user more control over what is run at any one time, while at the same time forcing necessary steps to be completed also.

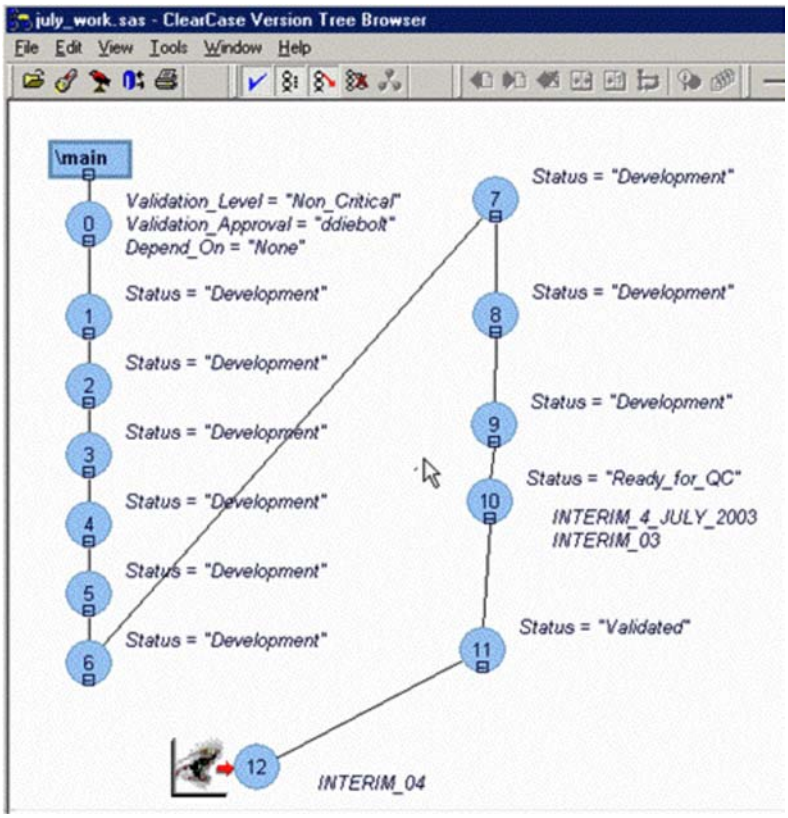


Figure 6 Version tree for a SAS program showing attributes and labels (in capitals)

There are various extensions and short cuts to help to build the make file with less typing, but that is the essence. Useful as this approach is (and the benefits of never having to do unnecessary re-runs and always having an up to date set of results files readily at hand should not be underestimated) it does have some practical problems. Such as

- the effort needed to maintain the make file and
- ensuring all dependencies are listed (if one is missed then some reports will not be re-run when they should be)
- make relies on modify dates to see if a file was changed. These are not reliable for SAS datasets and catalogs because the SAS viewer changes the modify date when only browsing the file.

Clearmake

The ClearCase version of make is called clearmake and it has some vital extensions, which allow us to avoid many of the above problems with standard make.

Derived objects (DOs) are created if the command creating them is run by clearmake or clearaudit. They have a configuration record (CR) that holds details of how, and with what, they were created. This is true even if not all those item are included in the make file. DOs can be checked in and their CR preserved and once checked in they must be checked out before being over-written.

Derived objects form the core of the audit trail for the traceability requirement and also free us from the standard make problems: incompleteness of the make file and reliance on dates for build decisions. Clearmake re-builds only if the version number has changed since the previous build.

This all sounds like a magic trick – but it isn't, it works.

An actual configuration record (CR) can be generated using cleartool, the cross-platform command line interface of ClearCase, like this:

```
cleartool> catcr "X:\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398"
```

It could look like this (note the date and time of the building is given, this means you can tell from this list what objects were output and which were read only):

```
Derived object: \proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398
Target m:/djg_view/proj_a/PROTOCOLxxxx/export/pgm_eff/x7107_7_01.rtf built by
ccadmin.CLEARCASE_Users
Host "hhtga-dt-42" running NT 5.0 (i586)

Reference Time 04-Jul-03.10:20:02, this audit started 04-Jul-03.10:20:02

View was hhtga-dt-42:d:\ClearCase_Storage\views\ch\ccadmin\djg_view.vws
Initial working directory was Y:\proj_a\PROTOCOLxxxx\export
----------------------------
```

```
MVFS objects:
----------------------------
\proj_a\PROTOCOLxxxx\data_01\diawky.sas7bdat@@\main\1 <26-Jun-03.15:20:38>
\proj_a\PROTOCOLxxxx\data_01\formats.sas7bcat@@\main\1 <03-Jul-03.10:43:59>
\proj_a\PROTOCOLxxxx\data_a\_vnslock.sas7bdat@@03-Jul.17:53.2377
\proj_a\PROTOCOLxxxx\data_a\a_ident.sas7bdat@@\main\2 <03-Jul-03.15:17:21>
\proj_a\PROTOCOLxxxx\data_a\formats.sas7bcat@@\main\1 <03-Jul-03.17:19:40>
\proj_a\PROTOCOLxxxx\data_a\TrakData.csv@@\main\2   <03-Jul-03.15:17:16>
\proj_a\PROTOCOLxxxx\data_a\xcontrol.sas7bdat@@\main\1 <01-Jul-03.17:37:26>
\proj_a\PROTOCOLxxxx\export\lst2rtf.bat@@\main\2    <03-Jul-03.17:12:27>
\proj_a\PROTOCOLxxxx\export\pgm_eff\x7l07_7_01.rtf@@04-Jul.10:20.2402
\proj_a\PROTOCOLxxxx\pgm_a\_autorun.sas@@\main\4    <02-Jul-03.17:23:40>
\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398
\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.sas@@\main\2    <03-Jul-03.15:19:07>
\proj_a\PROTOCOLxxxx\pgm_eff\x7l07_7_01.lst@@04-Jul.10:20.2401
\proj_a\PROTOCOLxxxx\pgm_eff\x7l07_7_01.tmp@@04-Jul.10:20.2400
\proj_a\PROTOCOLxxxx\util\_dropsuffix.sas@@\main\1  <30-Jun-03.18:06:35>
\proj_a\PROTOCOLxxxx\util\_last_in_path.sas@@\main\1 <30-Jun-03.18:07:45>
\proj_a\PROTOCOLxxxx\util\_setup.sas@@\main\2       <03-Jul-03.15:19:19>
\proj_a\PROTOCOLxxxx\util\_superscript.sas@@\main\1 <30-Jun-03.18:08:38>
\proj_a\PROTOCOLxxxx\util\fmtadlib.sas@@\main\1     <30-Jun-03.18:09:15>
\proj_a\PROTOCOLxxxx\util\formats.sas@@\main\1      <30-Jun-03.18:10:04>
\proj_a\PROTOCOLxxxx\util\make.csv@@\main\1         <02-Jul-03.16:31:52>
\proj_a\PROTOCOLxxxx\util\xcontrol.rmt.log@@04-Jul.10:20.2399
\proj_a\util\pgm\macro\ispgmvns.sas@@\main\1        <02-Jul-03.14:33:34>
\proj_a\util\pgm\macro\postrep.sas@@\main\2         <03-Jul-03.15:19:17>
\proj_a\util\pgm\macro\prerep.sas@@\main\2          <03-Jul-03.15:19:18>
\proj_a\util\pgm\macro\xcontrol.sas@@\main\1        <03-Jul-03.15:25:18>
\proj_a\util\pgm\macro\xgetctl.sas@@\main\1         <03-Jul-03.11:41:45>
\cc_tools\tools\bin\awk.exe@@\main\1                 <02-Jul-03.09:56:16>
\cc_tools\tools\bin\cygwin1.dll@@\main\1             <02-Jul-03.09:49:39>
\util\vns\data\template.sas7bitm@@\main\1            <27-Jun-03.11:13:06>
\util\vns\sasmacro\deldset.sas@@\main\1              <22-Jan-01.13:40:42>
\util\vns\sasmacro\dsetnobs.sas@@\main\1             <23-Nov-00.13:06:18>
\util\vns\sasmacro\lowcase.sas@@\main\1              <08-Aug-01.08:23:08>
\util\vns\sasmacro\try___r.sas@@\main\1              <08-Aug-01.07:03:10>
\util\vns\vnsmacro\maclist.sas@@\main\1              <24-Oct-01.12:37:20>
\util\vns\vnsmacro\patches.sas@@\main\1              <15-Feb-02.14:05:26>
\util\vns\usermacro\cnv2rtf@@\main\2               <03-Jul-03.17:13:29>
\util\vns\usermacro\vnsinit.sas@@\main\3             <02-Jul-03.17:23:34>
----------------------------
Variables and Options:
----------------------------
CMD_PAR=cd m:/djg_view/proj_a/PROTOCOLxxxx/pgm_eff; sas.exe l_eff1.sas ; echo -
----------------------------
Build Script:
----------------------------
        cd m:/djg_view/proj_a/PROTOCOLxxxx/pgm_eff; sas.exe l_eff1.sas ; echo -
----------------------------
cleartool>
```

There are lots of things to notice in this CR, for example,

- for every object its version number and modify time is included. (The version follows the @@ in the extended path name)
- the version of the awk utility used in the build is included and also
- all the macros called.
- The line labelled Build script is the SAS command call made to generate the output.

Also to be noted is the fact that only elements in a VOB can be included in the CR. This has advantages in that we can exclude some elements from consideration (the operating system version for example), but it does mean we have to include all the elements relevant to the deliverables in a VOB. This includes SAS datasets.

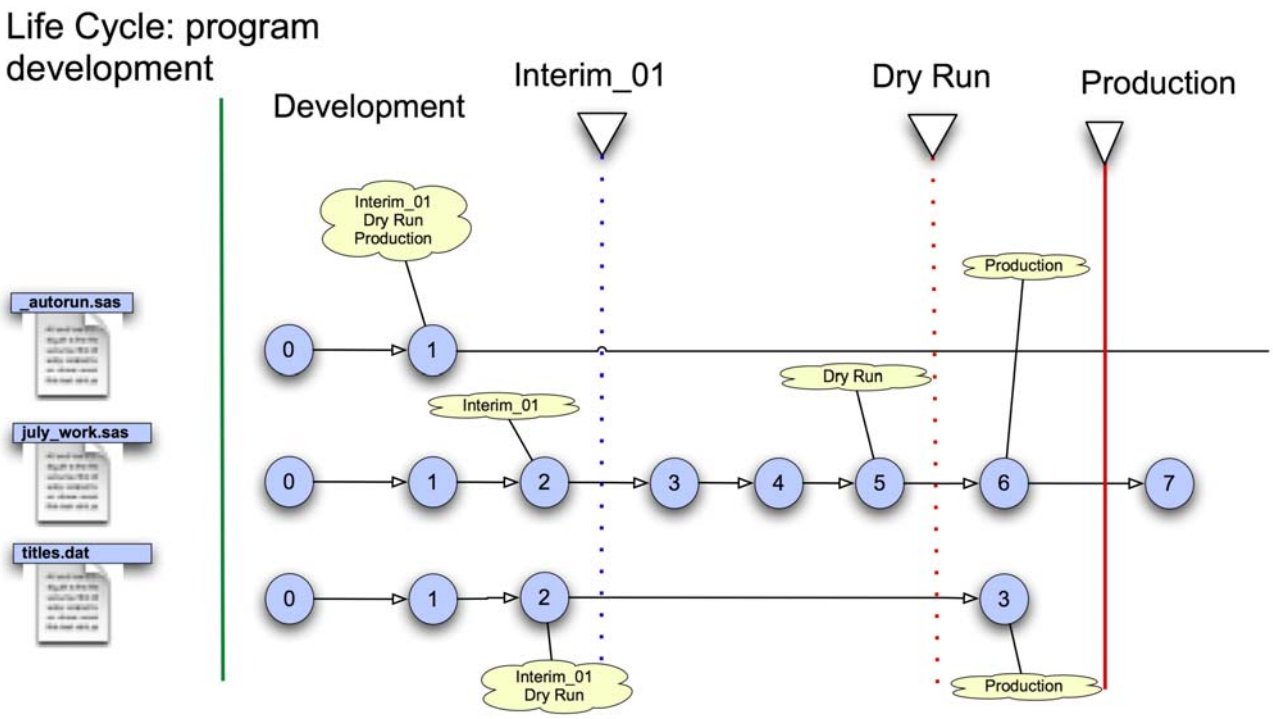


Figure 7 The life cycle of program development for three SAS objects

The development tree of Figure 7 shows how the development of the program july_work.sas might proceed, with time (and versions) along the x axis. The clouds represent labels as applied by the user to all objects at different stages of development (marked with inverted triangles) leading up to almost final production. Autorun.sas only has one version, which therefore has three labels. The main program july_work.sas has 7 versions. Given a default view at the maximum time-point we see version 1 of autorun.sas, version 7 of july_work.sas and version 3 of titles.dat.

Metadata in ClearCase

Metadata in ClearCase is Flexible, Powerful, and Configurable.

It is visible to a user on the Version Tree browser, the history browser, the explorer and can be queried by scripts. There are two main types of metadata in ClearCase:

1. Labels:
   are movable (unless locked), exist once per element, and are globally defined within a VOB
2. Attributes:
   Exist separately on every version and their values can be different for each instance.

Attributes

We need to control the Work flow so that we can know a given output really is 'releasable'

This is what we can use the attributes for. As an example we can record the validation status of a program with these states:

Program status: `draft, validation signed off, tested`

And the data status is kept as : `draft, locked, test_only`

Then, if our system sets the output status such that it depends on the program & data status, e.g.

```
if  statusOf(Program )= ='validation signed off'
and statusOf(Data ) == 'locked'
then
   setStatus(output) = 'releasable'
else
   setStatus(output) = 'incomplete'
fi
```

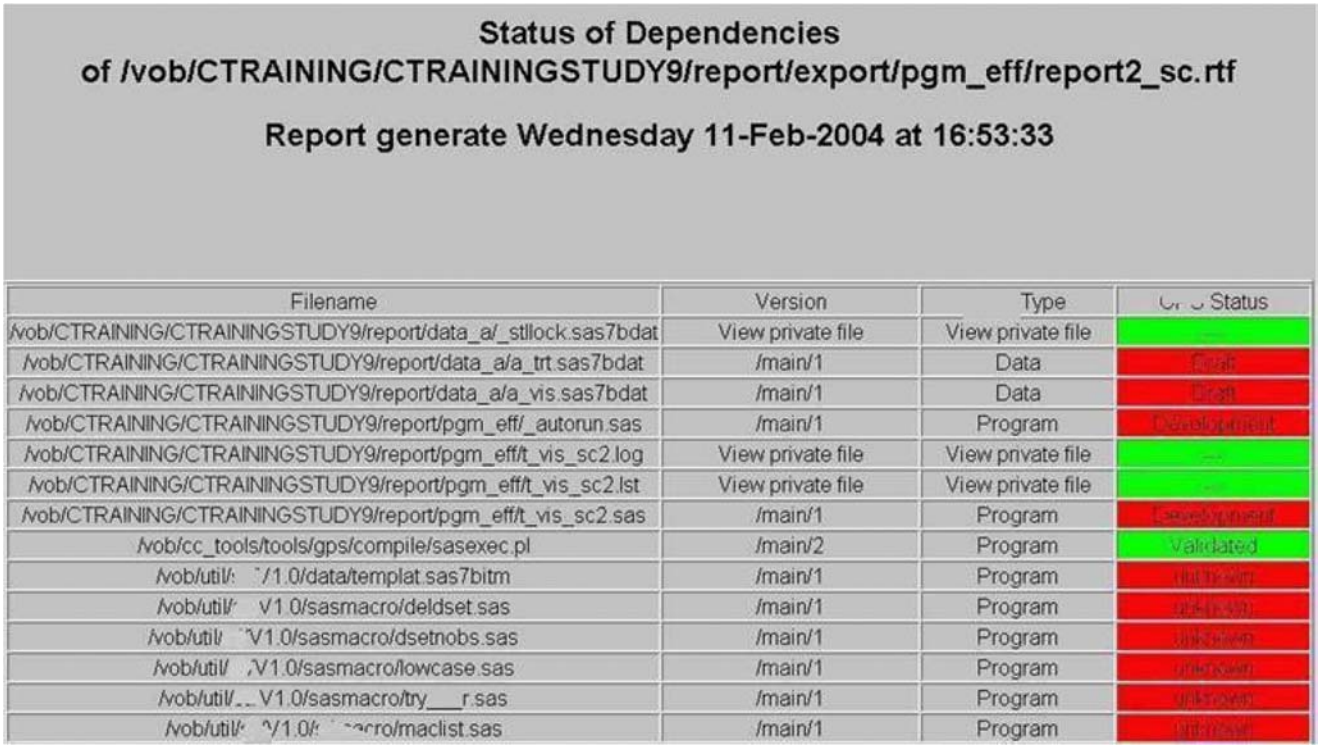
Then the status (and therefore uses) of the output can be fixed when created to be read later by any systems/ triggers that need to know the status.

Labels

In contrast labels naturally correspond to standard tasks, or milestones, eg '*Interim analysis 1*', '*first ISS*'.

Given the ability to set the status on all relevant objects in the VOB (see for example Figure 5) and the ability to query

that status we can make a utility to show the status and how it was determined for the user. This is shown in Figure 8, where the cells are coloured according to status. Green indicates *production/final/validated* while red indicates the object is at another step of the workflow.



Status of Dependencies
of /vob/CTRAINING/CTRAININGSTUDY9/report/export/pgm_eff/report2_sc.rtf

Report generate Wednesday 11-Feb-2004 at 16:53:33

Filename	Version	Type	Status
/vob/CTRAINING/CTRAININGSTUDY9/report/data_a/_stllock.sas7bdat	View private file	View private file	
/vob/CTRAINING/CTRAININGSTUDY9/report/data_a/a_trt.sas7bdat	/main/1	Data	
/vob/CTRAINING/CTRAININGSTUDY9/report/data_a/a_vis.sas7bdat	/main/1	Data	
/vob/CTRAINING/CTRAININGSTUDY9/report/pgm_eff/_autorun.sas	/main/1	Program	
/vob/CTRAINING/CTRAININGSTUDY9/report/pgm_eff/t_vis_sc2.log	View private file	View private file	
/vob/CTRAINING/CTRAININGSTUDY9/report/pgm_eff/t_vis_sc2.lst	View private file	View private file	
/vob/CTRAINING/CTRAININGSTUDY9/report/pgm_eff/t_vis_sc2.sas	/main/1	Program	
/vob/cc_tools/tools/gps/compile/sasexec.pl	/main/2	Program	Validated
/vob/util/ /1.0/data/templat.sas7bitm	/main/1	Program	
/vob/util/ V1.0/sasmacro/deldset.sas	/main/1	Program	
/vob/util/ V1.0/sasmacro/dsetnobs.sas	/main/1	Program	
/vob/util/ /V1.0/sasmacro/lowcase.sas	/main/1	Program	
/vob/util/ V1.0/sasmacro/try___r.sas	/main/1	Program	
/vob/util/ /1.0/ acro/maclist.sas	/main/1	Program	

Figure 8 Dependencies status report for report2_sc.rtf (HTML file)

It is worth noting that the attribute does not necessarily have to be locked. Why? Because all status changes are audited in the element's history and because the status can be re-calculated automatically at the point the element is delivered outside. In which case subverting the status will not allow a user to subvert the program lifecycle.

PILLAR 5: BUILD AN IMPENETRABLE FENCE AROUND THE SAND BOX

Security versus freedom

Generally there is a trade off between security and freedom. The more secure a system is, the more rigid it is and therefore the less responsive to unforeseen, or new, needs; Conversely, the freer the users are to create directories, rename directories and generally work how they prefer, the harder it is to be sure the deliverables are the correct ones and that validation has been correctly followed. For example, users may be able to work around the system (perhaps by adding outputs generated somewhere else, or by editing an output to fix a title).

Version control systems allow CFR21 part 11 requirements to be implemented so the user has lots of freedom but also cannot deliver anything but the genuine article. If the delivery mechanism is securely implemented then a lot of the controls and safeguards people consider necessary for this kind of environment (such as secure servers for executing code with their own copies of everything) are, in fact, not needed. The system as described is much easier to work with for the programmer and, in fact, to implement.

The workflow and work processes are kept in the background rather than being the focus of the programmers' daily life.

Delivering Safely via secured versioned interfaces

Assume there is a process whereby finished reports are released to another server or system, perhaps Documentum. Then this process must be customised either

- so it only works from a view where the configuration record is fixed so only the desired type of object is visible to the program, or
- By adding calls to the command for status checking.

Documentum has a Java API that can be used to submit material directly to a server and this interface is fast and stable. Such customization can guarantee that only outputs created from properly validated programs combined with data from locked databases can be delivered.

Similar customization considerations apply to extraction of data from CDMS systems.

IFS, BUTS, AND SURELY'S

OH NO! It is all so much more complicated. Why can't we just get on with programming!

SCM systems help with the process when looked at as a whole. Our craft is not only programming clever tricks with character functions. It is making safe reliable error free deliveries of outputs. Many kinds of

errors and problems are less frequent with a system like this. That means a lot of the time spent checking outputs by hand (etc.) can be converted back to programming time.

So now I suppose we need more directories for al our versions and so on.

Not at all, using labels sensibly can mean that all the work stages (ISS, Interim reports) that previously needed whole directory sub-trees can be set up with labels and the files all sharing the standard directories. This simplifies the SAS set-up and libnames needed.

Keeping so many copies wastes disk space!

No. For text files VC systems keep only the differences between versions. The efficiency is variable and depends on how many changes are made between versions, but it can easily be very high (e.g. 100 versions = space of two copies).

Well, maybe, but you can't difference SAS datasets and they use most of the space!

Binary files are compressed by default using the zip algorithm by ClearCase (and expanded on the fly behind the scenes). In fact, on the several projects I examined with this issue in mind the SAS datasets took about the same space as all the listings plus saslogs. Additionally, ClearCase has hooks so you can write your own binary file compression routines.

Humph! Why keep for ever 40 versions of a SAS program when only the production one is required to be kept. What a complicated waste of time, space,… and everything!

There are benefits from saving the programs in stages as they are developed. If you want to you can write a script to remove unlabelled versions of datasets, or programs after a study is closed.

Well, then using these views must slow SAS down, it depends on fast disks!

I haven't seen extensive tests on this but if there is a problem it is certainly not major.

What if we want to change our directory structure for new study standards, aren't we stuck?

No. Directories are versioned objects, just like files. New directories can be created and files moved into them and these changes are tracked in the same way as changes to files are. Labels can be applied to directories also.

Now I have to do months of ClearCase training in Timbuktu, when I could be SAS programming. Groan.

Not months, a few days of tailored presentations should be enough, with a follow up course perhaps after a few months. There is no Rational training currently scheduled in Timbuktu. Rational does, however schedule courses in many European countries, USA and India. Curse can also be tailored for in-house presentation.

We could go to Mars with what this is going to cost!

It is not free. But it is *much* cheaper than that, and arguably more useful.

SUMMARY:

- Customisable interface
- Views give efficient (read-only) access to SAS data
- The version aware files system MVFS services Clearmake which creates derived objects
- Derived objects know what objects and versions were used to make them, and what command
- ClearCase's customisable triggers allow fully customised workflow
- Creating and maintaining custom attributes per object
- Police the gateway to the publishing system
- And that will enforce the workflow on the objects that matter

I believe with ClearCase we can finally solve the problem of providing a flexible SCM and Release management environment for SAS programming, especially in a 21CFR11 context. A system built this way can form basis of a compliant environment with many components. It obviates the need to convert many applications because it hides inside the OS and apart from being less work it also makes it future-proof.

It uses a make program to automate running efficiently whole studies and allows us to move from a 're-run it all finally after DB-lock' to an 'always have the current best version available' paradigm.

REFERENCES

[1] White, Brian A, 'Software Configuration Management Strategies and Rational ClearCase';

Addison-Wesley, 2000, ISBN 0-201-60478-7

[2] Williams, Tim, 'A version control Kludge for SAS'; http://www2.sas.com/proceedings/sugi27/p095-27.pdf

ACKNOWLEDGMENTS

I would like to thank a client (who prefers to remain anonymous) for the chance to build such a ground breaking system and for the permission to write about it. I would also like to thank the Business team and the IT team involved for the team work that took us, boldly, where few had been before, and who came along despite the fact that all previous maps said 'here be dragons'. IT team members:

- Daniel Diebolt, IBM Rational
- Rudolf Bremer, Transnet
- Erika Thier, Theracon
- DJ Garbutt, BSI AG
- Norbert Spengler
- Ulrich Knappich

RECOMMENDED READING

White [1]. For more information about version control in software there are many resources on the web. All the main commercial packages have web sites but find out useful technical information there can be a slow job. For a discussion of CVS Cederquist see (http://developer.apple.com/darwin/tools/cvs/cederquist/)is very useful and for make the Unix Power Tools has this article about using make for things besides programs (http://www.unix.org.ua/orelly/unix/upt/ch28_13.htm). Excellent reference material for make is in the GNU make guide available from the GNU documentation project (see http://www.gnu.org/).

CONTACT INFORMATION

I would value your comments and questions on this paper. Please contact the author at:

David J Garbutt
BSI AG
Täfernstrasse 16A
CH – 5405 Baden-Dättwil
Work Phone: +41 56 484 1920
Fax: +41 56 484 1930
Email: d.garbutt@bsiag.com
Web: www.bsiag.com