Version Control the SourceSafe Way

SourceSafe by One Tree Software is generally described as a "Project Oriented Version Control System."

The phrase "Version Control System" describes many of the benefits that SourceSafe shares with other such programs. It stores and organizes your source code and other files; it mediates between different developers, preventing them from overwriting each other's code; it tracks old versions of your files; and it provides easy access to the latest version of all your source code.

But although it is built around well-established functionality and concepts, SourceSafe is not a carbon copy of other version control systems. The phrase "Project oriented" means that SourceSafe's internal architecture, the way it stores and organizes data files, is a completely new design that builds *project management* into the version control from the ground up. The most direct way to understand SourceSafe is therefore to understand its unique architecture on a technical level first, and then to extrapolate the powerful benefits which result from that model.

The SourceSafe Model

A standard version control system (such as the UNIX utility RCS) is *file oriented.* Such a system is essentially a collection of tools that operate on an individual file, controlling checkouts and updates and tracking old versions. To operate on a group of files, you write a batch file or specify wildcards on the command-line. Such a system has the advantage of being simple to implement and understand. But because it is based on individual files, it cannot track the relationships *between* those files: such tracking is left to the user.

SourceSafe follows a different pattern. In a SourceSafe installation, all your code files are stored in a centralized database on the network. This database is an unintelligible "black box" to the user--but when accessed through SourceSafe, it is seen to contain all your source files and their histories, organized into *projects.* Each project, like a DOS directory, is a container for a group of files. And each project, like a DOS directory, can be further organized into subprojects, creating a hierarchical tree for flexible organization.

Although the two models are very different, the way you use them is actually very similar. With a conventional version control system, you generally store your code in a directory tree on the network. A programmer checks out a file *from* the network to his hard drive, modifies and compiles there, and then updates it back *to* the network. SourceSafe is used the same way, except that you define a project tree inside the SourceSafe database *instead of* a directory tree on the network. Working with SourceSafe projects in this way is exactly like working with directories. In fact, the entire process is so easy, and so familiar, that you may begin to wonder where the advantage is. Are projects simply directories with a fancy name?

In fact, there is one critical difference: *because of its centralized database, SourceSafe knows exactly where all its files are at all times.* This gives SourceSafe the ability to track relationships on the project and system level as well as on the individual file level. This may seem like a minor point--but in fact, SourceSafe's detailed knowledge

of its own projects brings version control to an entirely new level, useful for any development and critical to large-scale projects. The following three examples, moving from the simple to the highly sophisticated, illustrate how SourceSafe combines version control with project management to provide truly global single-point control over your entire development system.

Preparing for a Build

Suppose you are about to do a complete build of a major system, consisting of many different programs. Before you do your build, you want to make sure nobody is revising the code at the last minute: or, in version control terms, that no files in the entire system are checked out.

A standard version control system provides you with a tool for asking if a file is checked out. Your job is to run this utility on every file in every directory that is being used for the build. Batch files and wildcards can make the task easier, but in a complicated system, it can still be quite a chore.

SourceSafe, like other systems, can determine if a file is checked out. But it can also provide, as a higher-level report, a list of all the checked-out files in a project. This feature becomes even more powerful if you use it *recursively;* that is, if you look into all the subprojects under the current project as well. SourceSafe looks through every file in every relevant project, and generates a list of every checked-out file--and immediately, you know whether you can do the build (or who to talk to if you can't!). So simply by executing a simple command on an organized collection of files, SourceSafe can automate a previously tedious, manual task.

History of a Project

A "file history" is a common report available in all version control systems (including SourceSafe). It lists each version of a file, from the most recent to the oldest, with information such as what happened to the file, who did it, when it was done, and what comment was left.

But despite their enormous usefulness, file histories have severe limitations. For instance, suppose a bug crept into your program last week, and you need to find the change that caused it. So you generate a history report on a likely-looking file, see if it changed recently, and look through the changes. If you don't find your bug, you pick another file to check...and so on. And you may go through every file in the system this way and never find the critical change--because the change was actually adding or deleting a file, which standard version control systems don't track at all!

SourceSafe, on the other hand, can generate a history report on the *project itself.* For instance, it might report that file 1 was just modified; before that, file 2 changed; before that, file 3 was added to the project; before that, another change was made to file 1; and so on. SourceSafe is doing the collating work you would otherwise have to do, enabling you to see what changed in the past week and when. This simple feature can save you hours, freeing you to immediately start hunting for the bug!

Another major advantage of project history tracking is that it enables SourceSafe to rebuild any past version of an *entire program*. For instance, suppose a client has version 2.03 of your program. That particular moment in the life of the project may have included version 10 of one file, version 15 of another, and so on; but you don't need to keep track of all that, because SourceSafe does so automatically. Ask SourceSafe for 2.03 and it reverses every change--every file you have updated, added, deleted, or renamed--to rebuild *exactly* what was in the project at that time.

Shared Code

If you write a number of programs around a common base of "core code"; if you customize one program for different customers' needs; or if you write in an object-oriented language such as C++; then you know the power of reusable code. You can add a feature to one shared module, and immediately get the benefit in five different programs. But you also know the organizational issues: you have to remember what programs are using this file, and propagate every change to all the necessary places! When five programs are using one reusable file, this may be a minor annoyance. But when twenty programs are mix-and-match sharing fifty reusable files in different combinations, keeping track of *which programs* use *which modules* can be virtually impossible!

A traditional version control system, which tracks only the contents of individual files, cannot help at all with this task. But SourceSafe, which tracks project relationships, can automate it completely with a single, simple feature: *one file can exist simultaneously in many different projects.* (As an implementation detail: inside SourceSafe's database, each file is stored only once. Each project that the file is "in" actually has a pointer to the file.)

To take a common example, suppose you have an error handling module which is common to many programs. In SourceSafe, every project that needs those routines would Share that file. If you find a bug, you update the file from any project--and the update is instantly propagated to every project that the file is in. Of course, you can always ask SourceSafe for a report of which projects the file is shared in, so you know which programs to recompile.

To take a very different scenario: suppose you have several different customized versions of the same program. Essentially, you have many different programs that share almost *all* their files between them. Using standard tools, tracking this situation can be a nightmare, requiring more time than the programming itself. Using SourceSafe, you can quickly and easily set up each customized program as its *own project,* indicating which files it shares with the other versions and which files are unique to this program. When you update a file, the change is reflected in all the appropriate programs, but not in the ones with their own specialized versions of that file!

The bottom line, as always, is that SourceSafe tracks exactly what files are used where--and can report the usage of any file to you on demand.

Beyond the Technical Report

This document is technical, specific, and detailed. It was written for people who

appreciate a well-structured system, who understand a sophisticated methodology, and who enjoy an elegant solution to a complicated problem.

But to make a decision, you have to take your technical hat off and put your practical hat on. You have to ask yourself--now that you understand the system--what would it do for you? How would it fit into the way you do development? What benefits would it offer you immediately? In the long run, how much actual time and trouble could you save with SourceSafe?

These are not easy questions, and they can rarely be answered by simply reading a white paper. In order to fully evaluate SourceSafe, you may want to discuss your particular situation with a sales representative at One Tree Software. Or you may want to talk to other SourceSafe users, and find out how the program is used in other real-world environments. In the end, you will almost certainly want to try SourceSafe yourself, and experiment to find what the product has to offer you, and your work, today.