



Stream-Based Architecture for SCM

This paper focuses on how the stream-based architecture works, and provides a technical comparison with file based branch and label SCM tools.

Author:
Damon Poole, Chief Technical Officer
John Posner, Director of Documentation

AccuRev



Table of Contents

The AccuRev Stream Based Architecture	3
Elements, Versions and Configurations	3
AccuRev Streams as Configurations	4
Upward information flow (Promotion on versions)	5
Downward Information Flow (Inheritance of Versions)	6
Contents of a Stream	7
The AccuRev TimeSafe Architecture	7
An Append-Only Database	8
Finding the Previous Contents of a Stream	9
Analyzing the Transactions: Implications and Applications	10
Comparing Two Releases	11
Defining and Moving Baselines	13
Conclusion	15

The AccuRev Stream Based Architecture

AccuRev is a software configuration management (SCM) system. It enables software development organizations to effectively manage the files that go into the production and maintenance of software deliverables. AccuRev is ideal for organizations in which a team of developers — programmers, QA engineers, tech writers, etc. — works with a code base of many hundreds or many thousands of files.

Architecturally, AccuRev differs from other powerful SCM systems in several fundamental respects. Other systems rely on metadata annotations to support basic system operations, such as computing the contents of software configurations. With AccuRev, configurations are first-class objects, called streams, whose contents aren't defined in terms of metadata annotations at all. On the other hand, AccuRev *does* rely on the chronological order in which users perform SCM operations. This enables AccuRev to leverage the fact that development is an incremental process — configurations whose creation times are close together differ only slightly. Other systems must rely on arduous metadata manipulations to compare configurations, no matter how slight the differences are.

This document describes the aspects of the AccuRev architecture that give it an “unfair advantage” over its rivals. You'll see that the AccuRev features that might initially seem to be “magic” (or if you're cynical, “smoke and mirrors”) are actually simple consequences of the revolutionary AccuRev architecture.

Before continuing ... if you are not familiar with AccuRev software, we recommend that you obtain a free trial copy of the product (www.accurev.com/download/index.htm) and work through the exercises in the *AccuRev Getting Acquainted Guide* (www.accurev.com/download/documentation).

Elements, Versions and Configurations

AccuRev calls each one of the code base's changing files and directories an element. Developers can create any number of versions of each element; typically, version N+1 of an element represents an incremental change from version N. A configuration of the code base consists of one version of every element:

<code>gizmo.java</code>	<code>version 45</code>
<code>frammis.java</code>	<code>version 39</code>
<code>base.java</code>	<code>version 8</code>
<code>release_number.txt</code>	<code>version 4</code>
<code>Gizmo_Overview.doc</code>	<code>version 19</code>
<code>Gizmo_Release_Notes.doc</code>	<code>version 3</code>
<code>... etc.</code>	

“Other Systems must rely on arduous metadata manipulations to compare configurations, no matter how slight the differences are.”

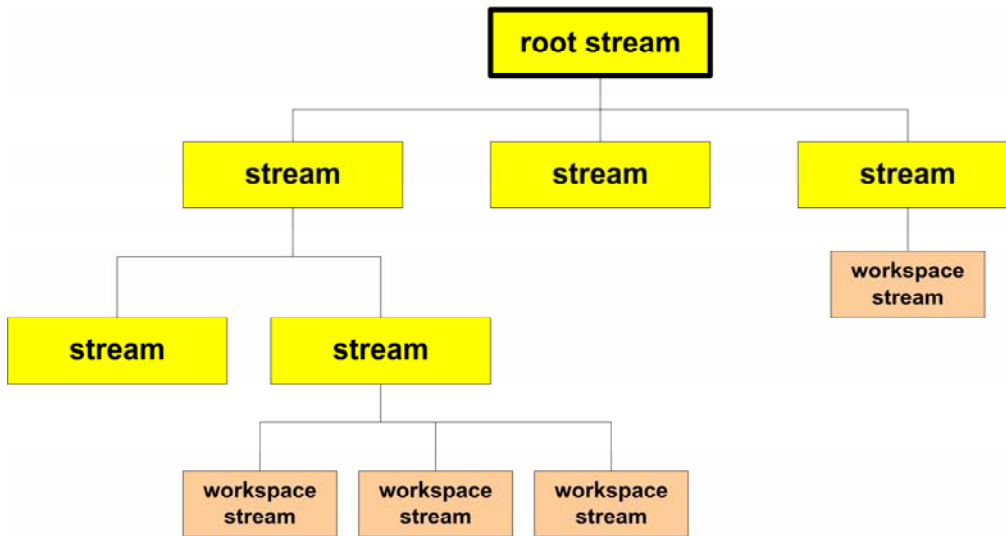
Stream Based
Architecture for
SCM
A White Paper

In most environments, each developer needs a complete configuration of the code base. Although he or she may be making changes to just a couple of files, the developer still needs *all* the files, in order to build the application and test the changes. At any given time, there may be a large number of configurations to be managed:

- Each programmer needs to work in isolation from other programmers, to eliminate instances of “your changes broke my build”. So each programmer needs a separate configuration of the code base.
- A QA engineer needs a configuration that combines the changes made by some or all of the programmers, in order to test all the new features in the next product release.
- To fix a bug in the previous product release, a maintenance engineer needs to start with the configuration that was used to build that release.

AccuRev Streams as Configurations

In the AccuRev architecture, the configurations of a software code base are implemented as a set of streams. Each stream is a distinct configuration, containing one version of each element in the code base. (The include/exclude facility enables a stream to configure a *subset* of elements, instead of the entire code base.) All the streams are organized into a single tree-structured hierarchy.



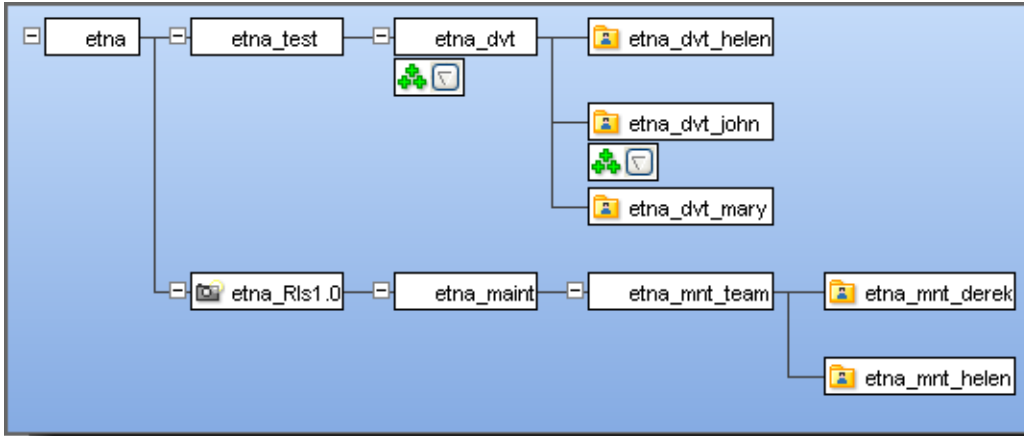
The stream hierarchy is much more than a mechanism for keeping track of the individual streams. The position of a stream in the hierarchy determines, to a large extent, the stream’s contents — that is, which versions of elements are in

“A stream’s position also determines the way in which AccuRev commands propagate development changes to other streams... Repositioning a stream automatically changes its contents.”

Stream Based
Architecture for
SCM
A White Paper

the stream's configuration. A stream's position also determines the way in which AccuRev commands propagate development changes (that is, new versions) to other streams. Users can change the position of streams, using the drag-and-drop StreamBrowser™ tool. Repositioning a stream automatically changes its contents.

Note: the diagram above illustrates the stream hierarchy in the conventional way for tree structures: root node at the top and node levels extending downward. The StreamBrowser displays the stream hierarchy left-to-right instead of top-to-bottom.

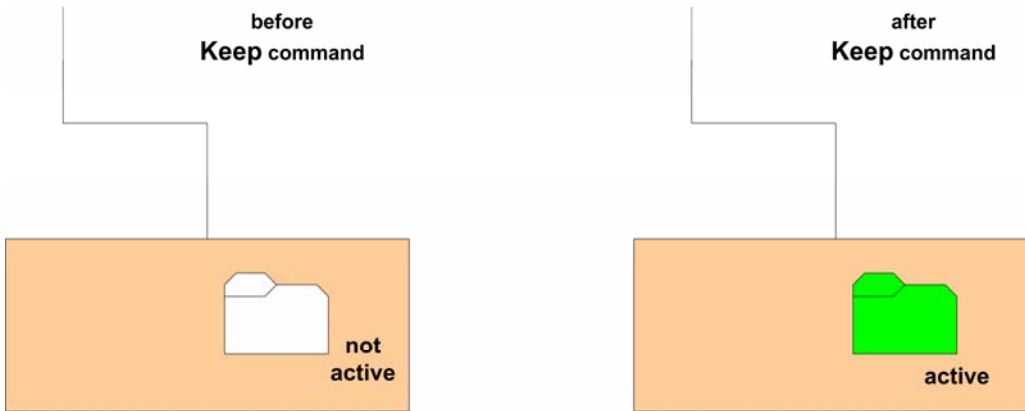


“A new version can represent a content change to the element or a namespace change. Creating the new version makes the element active in the workspace stream.”

Upward information flow (Promotion on versions)

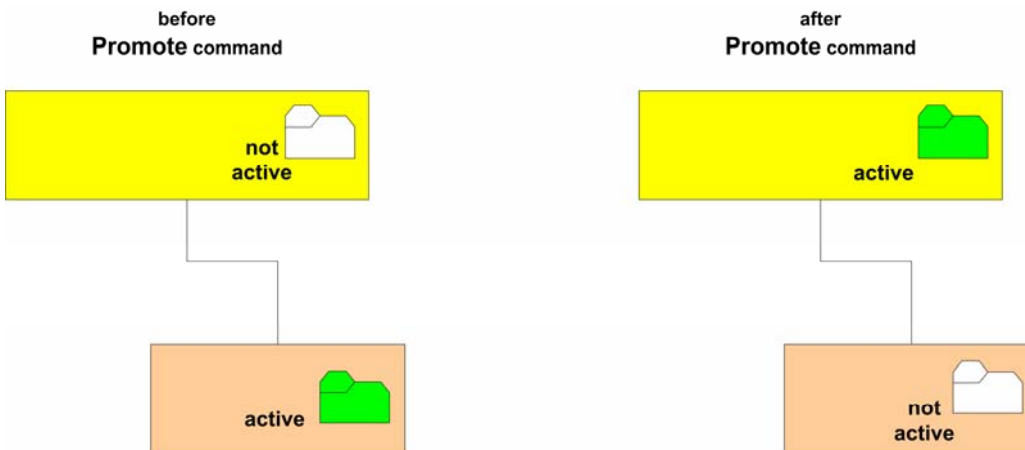
All new versions of elements are originally created in special workspace streams. Typically, each developer works in a separate workspace, with its own workspace stream. This provides developers with isolated, fully version-controlled work environments, equivalent to the “private branches” featured in other SCM systems.

A new version can represent a content change to the element (**Keep** command) or a namespace change (**Add**, **Rename/Move**, or **Defunct** command). Creating the new version makes the element active in the workspace stream.



Creating new version makes element **active** in workspace stream

The **Promote** command sends an active version of an element from a workspace stream to the parent stream — the stream immediately above it in the hierarchy. The element stops being active in the workspace stream, and becomes active in the parent stream.



Promoting version makes element **active** in "to" stream, and **not active** in "from" stream

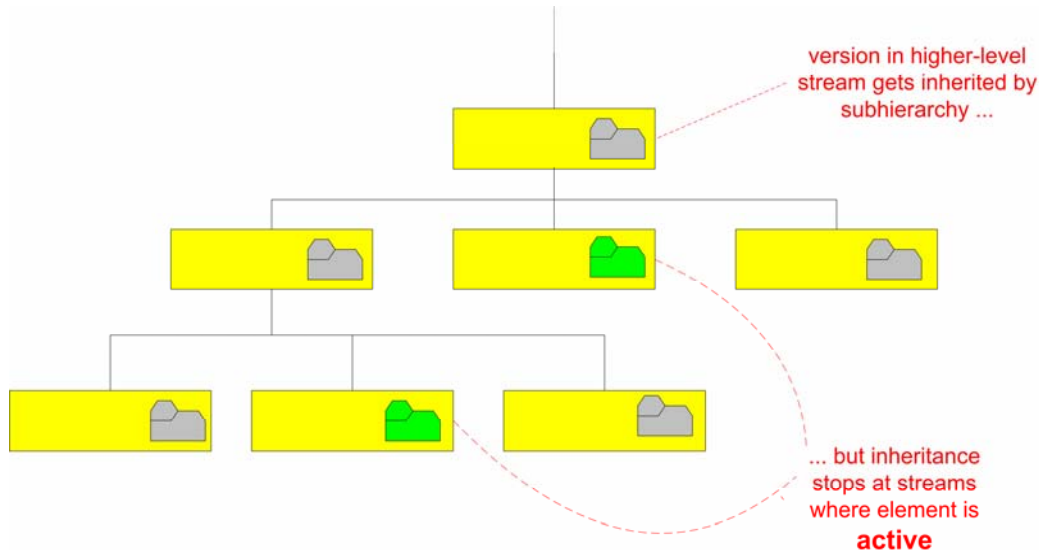
The version can be promoted up the stream hierarchy, all the way to the root stream. With each promotion, the element stops being active in the “from” stream, and becomes active in the “to” stream.

Downward Information Flow (Inheritance of Versions)

If an element is not currently active in a particular stream, as described in the preceding section, that stream uses the version that is currently in the parent stream. This “inheritance rule” is recursive: if the element is not currently active in the parent stream, either, the inherited version comes from the “grandparent” stream or the “great-grandparent” stream, etc. For some elements, the

“Each stream is a distinct configuration, containing one version of each element in the code base.”

inheritance will come all the way down from the root stream, where all versions are deemed active.



It's the automatic inheritance of versions that enables each stream to implement a configuration of the entire code base. The stream hierarchy itself — fully version-controlled and safely stored in the repository — controls the assembling of versions into complete configurations. In other systems, users must assemble configurations on their own, using a “some versions from this branch, other versions from that branch” procedure — perhaps a script, perhaps a ClearCase configuration specification.

Contents of a Stream

The principles outlined in the preceding sections imply the following recursive algorithm for determining the contents of a given stream, at a given time:

1. Start with all the versions of elements that were active in the stream at that time. (In the root stream, *all* versions are active.)
2. Add the versions that appeared in the parent stream at that time—except for elements already accounted for in Step 1.

Note that the algorithm calls for determining the contents of a stream at *any* time, not simply at the current time. The remainder of this white paper uses the task of finding the contents of a stream at a given time to illustrate the basic principles of the AccuRev implementation.

The AccuRev TimeSafe Architecture

The term TimeSafe describes the property of the AccuRev data repository that enables any past configuration of any stream to be determined with complete

Informally, TimeSafe simply means “You can't change the past”

Stream Based
Architecture for
SCM
A White Paper

reliability. Informally, TimeSafe simply means “you can’t change the past”. More formally, the AccuRev data repository is TimeSafe because:

- All queries on the database are timestamped. In practice, many queries are implicitly timestamped with the current time.
- The results of a timestamped query are the same no matter when the query is executed.

Thus, the query “what are the contents of stream Gizmo” becomes, for example, “what were the contents of stream Gizmo on June 6 2005 at 10:12:05 UTC”. And the results of the query will be the same on June 6, on Dec 1 2005, and any time in the future. Experienced AccuRev users will recognize this example as the specification of an immutable snapshot.

AccuRev converts a query’s timestamp into a database transaction number (see next section). This enables queries to be unambiguous, even if they are being submitted by developers distributed across many time zones.

An Append-Only Database

AccuRev accomplishes the TimeSafe-ness of its data repository by building the repository around a transaction-based, append-only database:

- Each change to the database is recorded as a transaction. Transactions are atomic (indivisible), so that the database is always in a consistent state. If a transaction is interrupted before it completes, the database remains unchanged.
- New transactions are appended to the database in chronological order. There cannot be any ambiguity as to the order of transactions, even in a distributed development environment.
- Existing transactions cannot be modified in any way, and they cannot be deleted.

(Some ancillary changes to the database are not recorded as transactions. All changes related to the contents of streams *are* recorded as transactions, making the database TimeSafe for the purposes of configuration management.)

Let’s look at an example. As described in section ***Contents of a Stream*** above, AccuRev must often determine which elements are active in a given stream. One of the database tables in the AccuRev repository tracks elements’ active status in streams. Here’s a simplified excerpt from this table:



Transaction	Stream-ID	Element-ID	Active?
1	10	201	T
2	10	202	T
3	11	202	T
4	10	201	F
5	10	201	T
6	11	202	F
7	11	202	T
8	10	202	F

Each transaction records the fact that a particular element — identified by its numeric ID — has become either active or inactive in a particular stream — also identified by its numeric ID.

(This simplified example clearly omits many transactions, including the ones that initially created the streams and elements.)

The query “what elements are currently active in stream Gizmo?” might translate to “what elements were active as of transaction #8 in stream #10?”. First, AccuRev applies a filter to extract the transactions related to the specified stream:

Transaction	Stream-ID	Element-ID	Active?
1	10	201	T
2	10	202	T
4	10	201	F
5	10	201	T
8	10	202	F

Then, AccuRev selects the most recent transaction for each element:

Transaction	Stream-ID	Element-ID	Active?
5	10	201	T
8	10	202	F

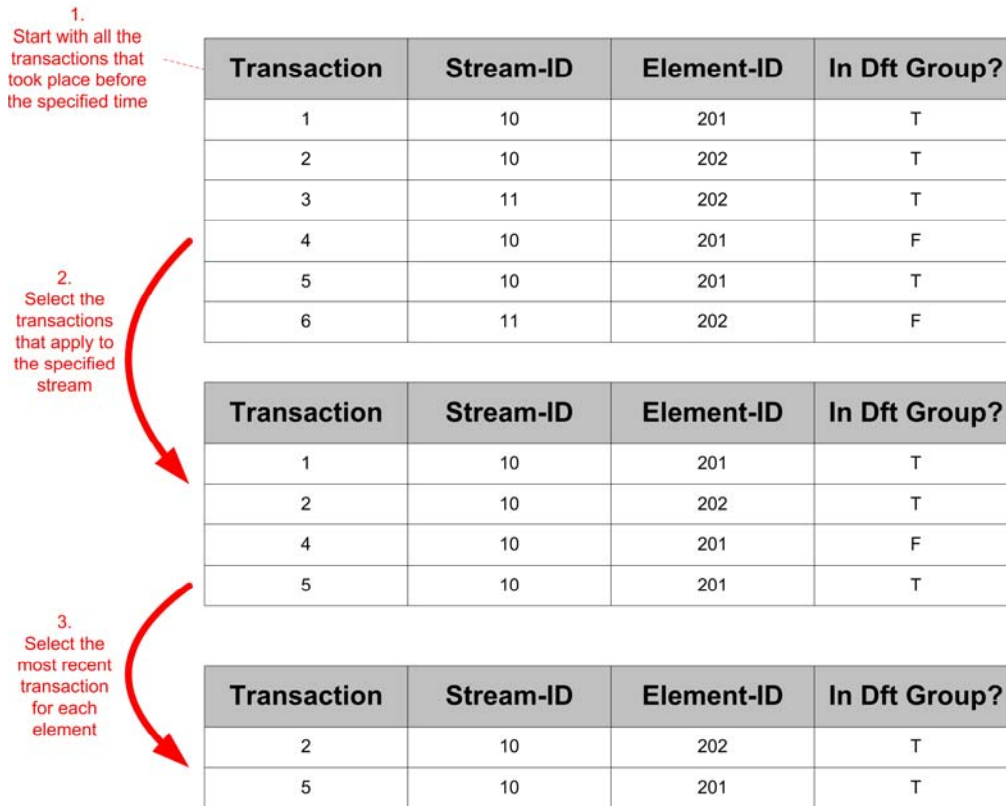
Query results: only element #201 is currently active in stream #10.

Finding the Previous Contents of a Stream

With just a slight change, this procedure can also determine the contents of a stream at any time in the past. If “at the current time” translated to “as of

“Contrast this with SCM systems such as CVS and ClearCase: to be able to answer this question, those systems need to attach a ‘version label’ to a version of each element. For a code base of 10,000 files, that means creating 10,000 metadata annotations, just to be able to reconstruct the code base for a single release.”

transaction #8”, then “at May 23 15:17:49 UTC” might translate to “as of transaction #6”. So the procedure plays out as follows:



Query results: at May 23 15:17:49 UTC, elements #201 and #202 were both active in stream #10.

Analyzing the Transactions: Implications and Applications

The examples above illustrate an architectural principle that gives AccuRev a tremendous advantage over other configuration management systems. AccuRev can determine the answer to questions such as “what versions went into Release 2.1.3?” simply by analyzing the transactions in the repository database. There is no need to add metadata annotations — which are both space-consuming and time-consuming — to enable this functionality.

Contrast this with SCM systems such as CVS and ClearCase: to be able to answer the above question, those systems need to attach a “version label” to a version of each element. For a code base of 10,000 files, that means creating 10,000 metadata annotations, just to be able to reconstruct the code base for a single release.

“The TimeSafe principle guarantees that the snapshot can be defined at any time.”

Stream Based
Architecture for
SCM
A White Paper

Think of these other systems as using metadata to precalculate and store for later use the answer to the “what versions” question. This approach is inefficient and scales poorly — 10,000 annotations for 10,000 files, 250,000 annotations for 250,000 files, etc. By contrast, AccuRev merely needs to store the specifications of the question. Defining the code base for Release 2.1.3 might come down to storing the specifications, “stream=Gizmo, transaction=13508”. AccuRev captures these specifications as a snapshot of the Gizmo stream — a special substream that is “frozen in time” at transaction #13508.

Moreover, the TimeSafe principle guarantees that the snapshot can be defined *at any time*. With AccuRev, it’s easy to say, “The point release will be based the state of the code base last Tuesday at 12:01AM”. With other systems, it’s all too common to be forced into saying, “We didn’t put down a label last Tuesday at 12:01AM. So we have no way of getting out of our current troubles by rolling back to that state”.

Comparing Two Releases

The architectural advantage of AccuRev is even more dramatic when it comes to comparing the versions in two releases — for example, Rls 2.1.3 and Rls 2.1.4. For the systems based on version labels, the comparison involves a great deal of overhead and a great deal of work:

- Define Rls 2.1.4 by attaching version labels — adding another 10,000 or 250,000 pieces of metadata to the repository database.
- For each element, determine whether the “2.1.3” and “2.1.4” labels are attached to the same version, or to different versions.

“An AccuRev snapshot serves to define a baseline.”

Stream Based
Architecture for
SCM
A White Paper

With AccuRev, there are no labels to deal with — just the transactions (probably a relatively small number) that took place between Rls 2.1.3 and Rls 2.1.4.

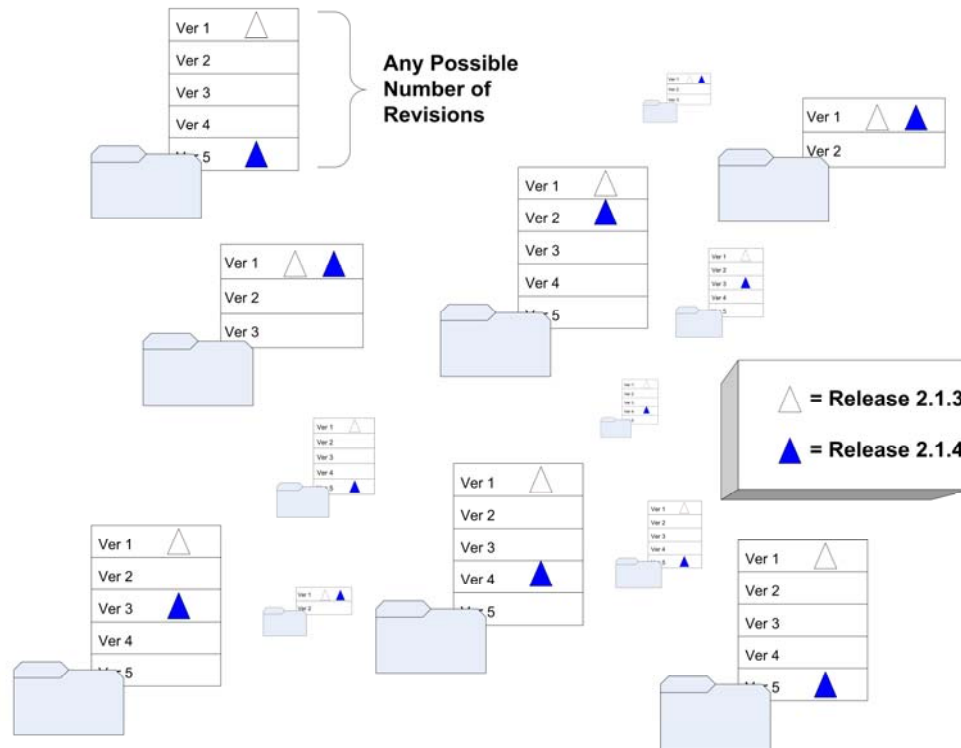
	Transaction	Stream-ID
	13504	33
	13505	47
	13506	47
	13507	39
	13508	47
Rls 2.1.3 Snapshot of stream #47	13509	55
	13510	39
	13511	47
	13512	33
	13513	33
	13514	55
	13515	55
Rls 2.1.4 Snapshot of stream #47	13516	47
	13517	55
	13518	39
	13519	55
	13520	39

In this illustration, the only differences between the two releases were recorded in transactions #13509 through #13516. And only two of those transactions, #13511 and #13516, affected the Gizmo stream (#47) that was used for the release. So the comparison of the two releases is reduced to an analysis of a very few transactions.

By contrast, it takes a great deal of work to find the difference between two releases in an SCM system that depends on version labels. This illustration suggests the scale of the task:

“Other Systems must rely on arduous metadata manipulations to compare configurations, no matter how slight the differences are.”

Stream Based
Architecture for
SCM
A White Paper



Comparing the releases amounts to finding the elements for the red triangle, which “labels” Rls 2.1.4, and the black triangle, which “labels” Rls 2.1.5, are on different versions. This task is increasingly time-consuming as the code base grows, even if the releases are nearly identical.

Defining and Moving Baselines

When embarking on a long-lived development project, a programmer typically begins with a stable configuration of the code base. The configuration might represent a major release, a point release, or even just a snapshot that builds successfully and passes a “smoke test”. The developer usually wishes to remain isolated from the day-to-day changes to the code base, in order to concentrate on the particular development task at hand, without having to worry about change-integration tasks. But it’s a winning strategy to integrate periodically, rather than postpone all the integration work until the last minute.

Example: the developer might start a project with the code base for point release 3.4.5. That is, Rls 3.4.5 acts as the baseline for the developer’s work. After working in isolation for days, weeks, or months, the developer learns that the maintenance team has put together another point release, 3.4.6. At this point, he can choose to incorporate the changes made in Rls 3.4.6 into his own work. That is, he can rebase his project, so that Rls 3.4.6 is the new baseline for his work.

An AccuRev snapshot serves to define a baseline. As discussed above, it takes just a couple of parameters to define a snapshot of any code base, no matter how

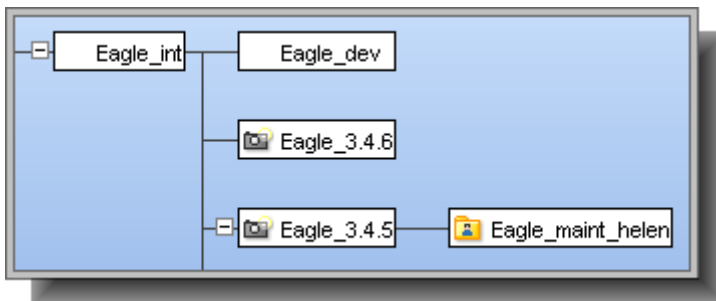
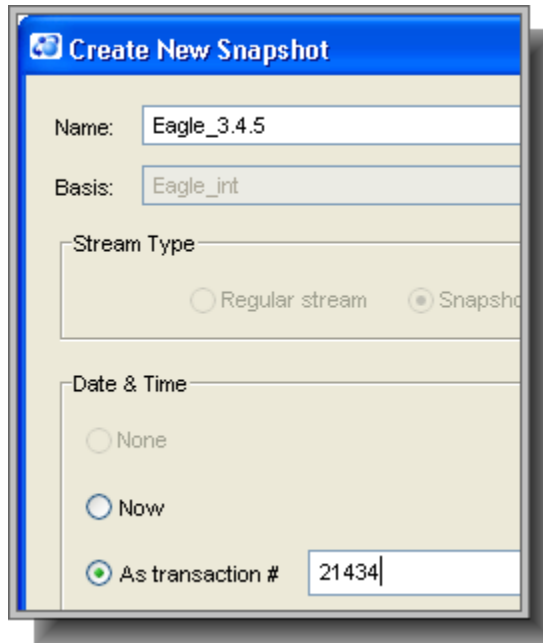
“It takes a great deal of work to find the difference between two releases in an SCM system that depends on version labels.”

large. This screen shot shows how the Rls 3.4.5 baseline can be defined as a snapshot named **Eagle_3.4.5**, capturing the state of the stream **Eagle_int** as of transaction #21434.

The AccuRev StreamBrowser shows snapshots as part of the stream hierarchy. This makes sense from the user's viewpoint: a snapshot is a configuration that inherits a certain set of versions from its parent stream, and can act as the basis for lower-level streams and workspaces.

It also makes sense from an implementation viewpoint: a snapshot is actually a stream ("static stream"), one whose contents cannot change, due to the TimeSafe property of the AccuRev repository.

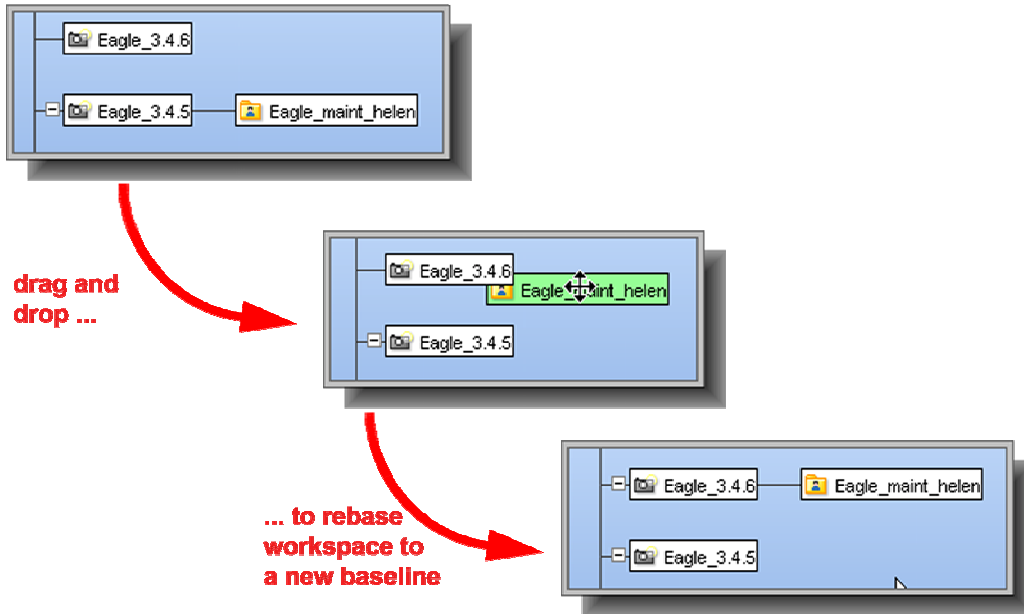
This screen shot below shows how the snapshots for the Rls 3.4.5 and Rls 3.4.6 baselines appear in the StreamBrowser. It also shows a workspace for user Helen, who is using the Rls 3.4.5 baseline as the basis for her project work.



"The AccuRev StreamBrowser makes sense from the user's viewpoint."

Stream Based
Architecture for
SCM
A White Paper

When Helen decides to rebase her work to the Rls 3.4.6 baseline, she need only perform a drag-and-drop operation in the StreamBrowser:



Conclusion

The AccuRev architecture begins with an append-only database, ensuring that the past can be reconstructed with absolute reliability. This foundation, along with the fact that development events have a strict chronological order, enables AccuRev to maintain software configurations as a hierarchical system of streams. We've shown that these simple principles provide the basis of the AccuRev revolution — modeling and managing sophisticated software development processes with speed, power, and flexibility.

For more on AccuRev, contact us at damon.poole@accurev.com.

To obtain a free trial copy of AccuRev software visit: www.accurev.com

Stream Based
Architecture for
SCM
A White Paper