AccuRev Concepts Manual

Version 4.9

October 2010

Copyright

Table of Contents

AccuRev Concepts

The chapters in this manual describe the main concepts and facilities of the AccuRev® software configuration management system:

- *The AccuRev Data Repository*
- *What is a Software Configuration?*
- *AccuRev Software Configurations: The Stream Hierarchy*
- *AccuRev Workspaces and Reference Trees*
- *AccuRev Transactions*
- *AccuRev/AccuWork Change Packages*

The manual concludes with a cross-referenced *AccuRev Glossary*.

The AccuRev Data Repository

As a data management product, AccuRev's foremost job is to provide a secure data repository for long-term storage of your organization's development data. AccuRev's implementation of the repository is straightforward and flexible; a repository can grow gracefully to span multiple disks, possibly on multiple machines. And key product features make it easy to protect the repository from accidental or malicious damage.

AccuRev has a simple client-server architecture. A single program, the AccuRev Server (**accurev_server**), is the only program that accesses the data repository directly. This "single point of entry" to the repository makes it easy to enforce tight security at the operating system level.

The data repository is built around a unique database technology, which is both transaction-based and append-only. This makes the repository extremely resistant to accidental damage. Using "atomic" transactions means that the database won't become corrupted, even if a power failure occurs while the database is being modified. The append-only feature enables "live backup" of the repository, without having to interrupt developers' work. This means that backups can be made as often as desired — even continually; and the more recent the backup, the less data is lost in the event of a catastrophic hardware failure.

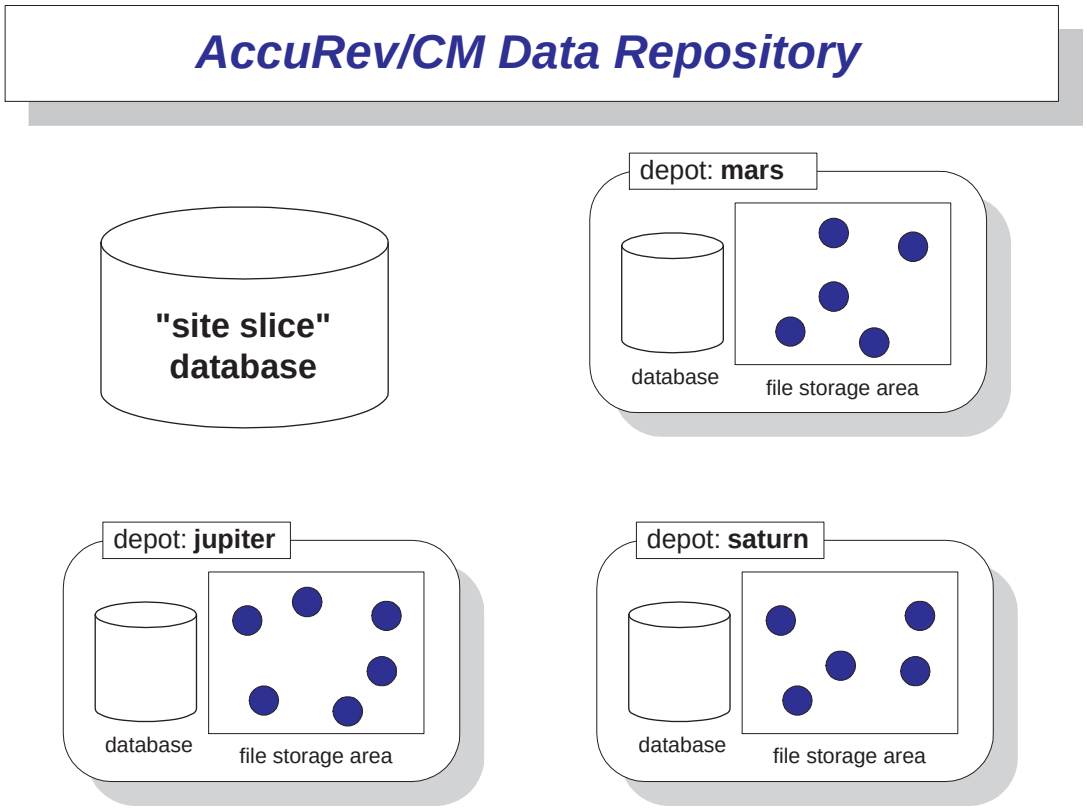
Organization of the Repository: Storage Depots

An AccuRev data repository consists of:

- The site slice, a central database that contains repository-wide information. (It's the "slice" of the repository that contains data pertaining to the entire AccuRev "site".) This includes a user registry and lists of global data structures.
- Any number of depots — short for "storage depots" — which contain separate sub-repositories. Each depot implements a version-controlled directory tree. It provides protected, permanent storage for all the versions of the files in the tree; it also includes a database that tracks the changes to the files themselves, their names, and their organization into directories.

  Alternatively (or additionally), a depot's database can store issue records. Typically, a depot's issue records hold bug reports relating to the depot's files. (For licensing purposes, AccuRev's issue management capability is termed "AccuWork".)

The illustration below shows the modular structure of the AccuRev data repository. Logically, the entire repository is located in a single directory tree on the machine where the AccuRev Server program runs. But only the various databases must physically reside on the server machine. The file storage areas — which typically are far larger than the databases and grow far faster — can be located elsewhere. For example, the file storage area of depot **jupiter** might be located on another disk on the AccuRev server machine, and the file storage area of depot **saturn** might reside within the local area network's disk farm.



Single Depot vs. Multiple Depots

You can place all version-controlled files in a single depot, or split them among multiple depots. In general, we advise storing all files for a given project in the same depot. By "project", we mean all the programs and other software deliverables that share the same development/test/release procedures and the same release cycle. The procedures determine how a depot's stream hierarchy will be structured; the release cycle determines how the stream hierarchy will be used.

If Project_X and Project_Y have completely different release cycles, then put their source files in different depots. Likewise, if Project_A requires stringent in-house regression testing and two levels of beta-testing, whereas Project_B is mandated to "ship yesterday", use different depots.

> Note: if you use the include/exclude facility, you can have a single depot serve multiple partially-independent or totally independent projects. See *The Include/Exclude Facility* on page 19.

AccuRev has no scalability limits, so there is no problem in storing thousands, tens of thousands, or even hundreds of thousands of files in a single depot.

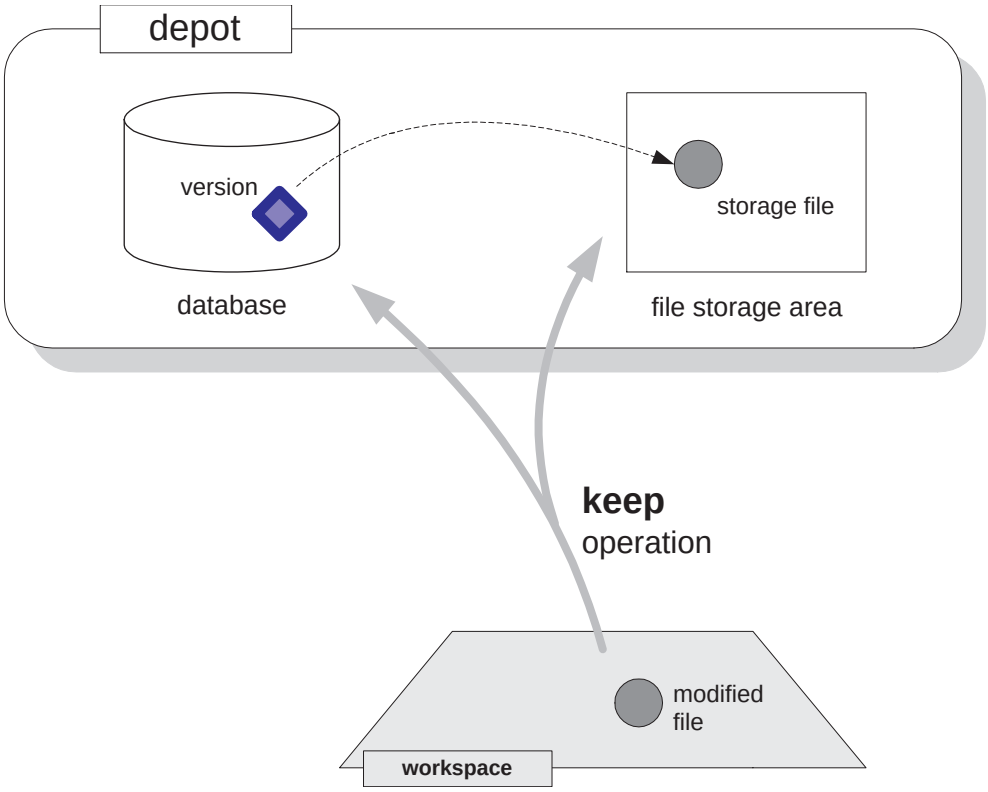
Inside a Depot: Versions and Files

Let's look inside a depot, to examine its database/file-storage-area architecture. This will help explain how AccuRev works, and will illuminate some of its most important, and unique, features.

Developers working on their files — that's the principal activity in any software development environment. With AccuRev, a developer's files are stored in an ordinary directory structure — perhaps on the hard drive of a personal computer or laptop, perhaps in a designated area of a well-backed-up disk farm, etc. The only thing special about such a "developer's work area" is that AccuRev keeps track of its association with a particular depot. (The work area is termed a workspace — for more information, see *AccuRev Workspaces and Reference Trees* on page 21.)

A developer can use any software tools to create and edit files, compile and build modules and applications. AccuRev doesn't get involved in these operations at all, so there's no performance penalty. Every so often, the developer tells AccuRev to save the current contents of a file (or a group of files). This operation, called a keep, does two things:

- Copies the current contents of the file to a container file in the depot's file storage area.
- Creates an associated version object in the depot's database.



This association is permanent: no matter what happens in the future, the contents of the file will always be available, through a reference to the version object. (For now, we'll skip the details of how to specify a version — it's just a bit more complicated than saying "version 45 of file gizmo.c".)

In addition to providing access to the actual file contents, the version object stores additional information relating to the "keep" operation: a timestamp, the user who performed the operation, a user-supplied comment, etc. This kind of information is often termed "metadata".

In general, version objects are much smaller than the corresponding container files. (Developers often work with large source files; they also work with audio, image, and multimedia files, which can be *really* big.) As developers create more and more versions, the depot's file storage area may

grow to many gigabytes, requiring it to be split among multiple disk drives. But since the depot's database stores the relatively small version objects, it grows much more slowly. Most likely, it will never outgrow its original storage location.

Versions of Non-File Objects

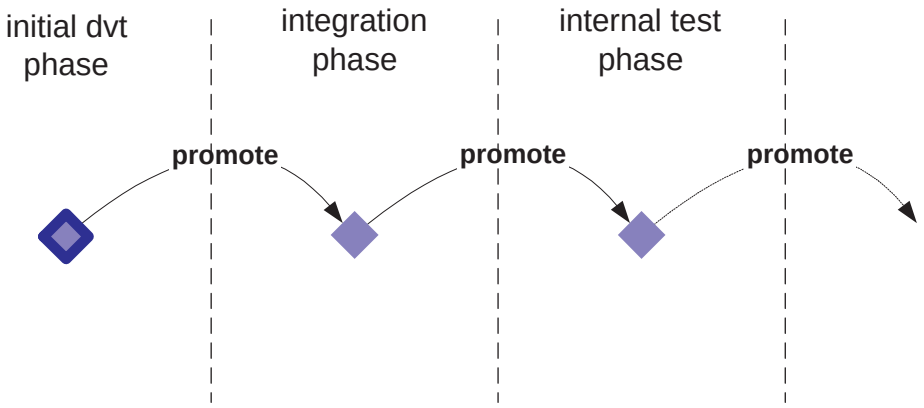
In all modern operating systems, files are organized into directories (or folders). Some operating systems also support additional kinds of file system objects: symbolic links, hard links, device files, named pipes, etc.

AccuRev provides full version-control of directories. A new version of a directory records the renaming of the directory or the moving of the directory to another location in the depot's directory hierarchy.

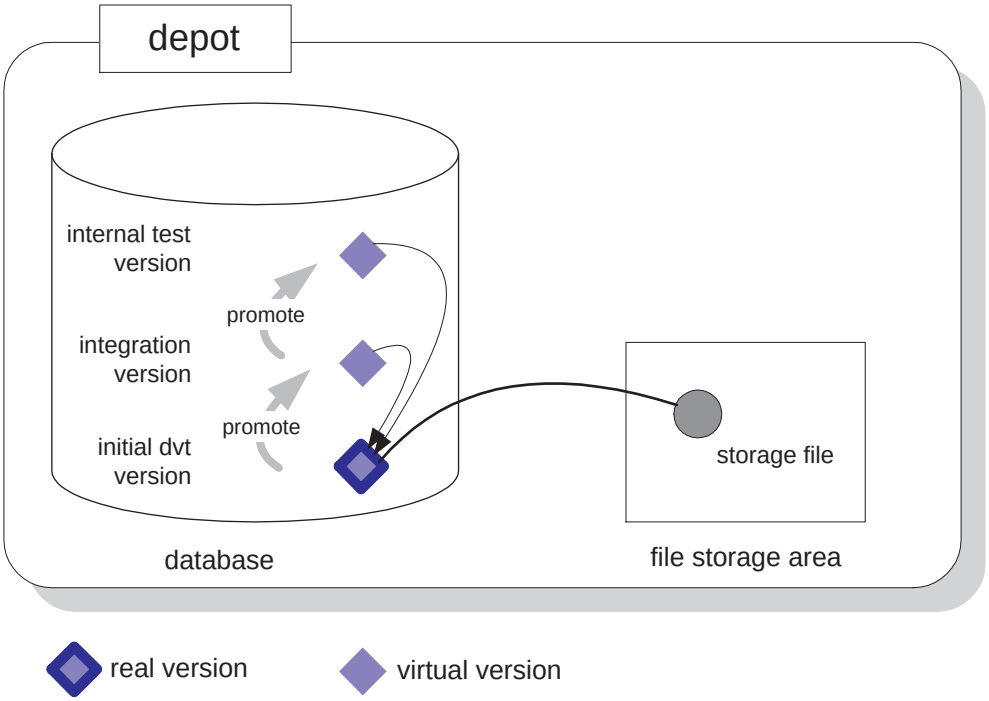
AccuRev provides file-link and directory-link objects, which can be used to version-control hard links and symbolic links / junction points.

Promotion: Real Versions and Virtual Versions

Software development is much more than just creating and modifying files. A typical development project involves many phases, possibly including initial development, integration of work done independently, internal system testing, external testing, and final production. AccuRev uses a "promotion model" to manage files in these multiple development phases. Files progress through the phases, one by one: when a file passes the tests (if any) mandated for a particular phase, a user working on that phase promotes it to the next phase.



AccuRev keeps track of each promotion by creating a new version of the file. But promotion doesn't change the contents of a file; it only changes the file's "approval level". Thus, each new version object created by promotion is merely an additional reference to (or "alias for") the same file in the depot's file storage area.



AccuRev distinguishes between the original version, created by a **keep** operation, and all the additional versions created by a **promote** operation:

- A real version is created by a **keep** (or an **add**, which places a new file in the depot). The operation creates a new version object in the depot's database, and also places a new file in the depot's file storage area.
- A virtual version is created by a **promote**. It creates a new version object in the depot's database, which provides an additional reference to an existing file in the file storage area.

Replication of the Repository

The AccuRev repository can be replicated at multiple sites, to support distributed development organizations. It can also be replicated at a single site, to provide better performance. Each replica repository can store the data for a selected subset of the master repository's depots (or for all the depots).

Replication is discussed in the *AccuRev Administrator's Manual*.

Archiving Data and Removing Data

In a perfect world, no one makes mistakes and there's an infinite amount of disk storage. But in this world, you sometimes save data by mistake and your repository sometimes outstrips your storage capacity. In general, AccuRev's TimeSafe principle (its ability to reproduce previous configurations) does not allow data to be removed from the repository once it has been placed there. There's just one exception: you can remove an entire depot from the repository.

Removing depots that were created by mistake (or, perhaps, for practice) can help to reclaim valuable disk space. Another strategy is to "get rid" of old versions of elements, ones that you anticipate won't need to be used again.

AccuRev allows you to archive old versions, moving their space-consuming container files out of the repository to offline storage. And if it turns out that you *do* need the versions, after all, you can restore them from offline storage to the repository.

Both depot removal and archiving of versions are discussed in the *AccuRev Administrator's Manual*.

What is a Software Configuration?

AccuRev is a software configuration management (SCM) product. So what's a software configuration? Software developers (programmers, QE engineers, tech writers, etc.) work with information stored in files. The contents of the files change over time, as developers work on them. The developers save the changes in new versions of the files. The organization of the files changes, too: new files are created, old files are deleted, some files get renamed, and directory structures get reorganized.

Take a particular set of files — for example, the files required to build and deliver an application named **Gizmo**. At any given moment, this set of files is in a particular state, which can be described in terms of version numbers:

```
gizmo.c                     version 45
frammis.c                   version 39
base.h                      verion 8
release_number.txt          version 4
Gizmo_Overview.doc          version 19
Gizmo_Release_Notes.doc     version 3
```

... or in terms of time:

```
gizmo.c                     last modified 2004/11/18 14:15:03
frammis.c                   last modified 2004/11/18 14:15:19
base.h                      last modified 2004/10/08 09:09:44
release_number.txt          last modified 2004/11/17 21:59:34
Gizmo_Overview.doc          last modified 2004/11/20 17:25:00
Gizmo_Release_Notes.doc     last modified 2004/11/21 19:29:57
```

That's a software configuration — or more simply, a configuration: a particular set of versions of a particular set of files. (AccuRev's naming scheme for versions is slightly more complicated than "version 45 of file gizmo.c".)

> Note: unlike some other SCM products, AccuRev keeps track of changes to both files and directories. In this discussion, though, we'll concentrate on files.

Suppose one of the files changes:

```
   ...
release_number.txt          last modified 2004/11/24 07:19:18 (version 5)
   ...
```

(Somebody forgot to modify the release number; we're sure that has never happened at your organization.) You can think of this change as producing a new software configuration. But in many situations, it's more useful to think of this as an incremental change to an existing, long-lived configuration — the one called "Gizmo source base" or, perhaps more precisely, "Gizmo Version 2.5 source base".

So in the end, is a software configuration just "a bunch of files"? Almost, but not quite. It's important to keep in mind that a software configuration does not contain the files themselves, but

only a *description* or listing of the files and their versions. Think of the difference between an entire book (big) and its table of contents (small). This crucial distinction makes it possible for AccuRev to keep track of hundreds or thousands of software configurations, without needing an infinite amount of disk storage.

The change described above to file **release_number.txt** illustrates the distinction between files and configurations of files. The change to the contents of the file is something like this:

> replace text line “RELEASE=2.5” with text line “RELEASE=2.5.1”

The change to the software configuration is something like this:

> replace version 4 of file “release_number.txt” with version 5

For another example of the distinction, recall that a configuration takes into account filenames and directory structures, too. Consider this configuration:

src/gizmo.c	version 45
src/frammis.c	version 39
src/base.h	verion 8
src/release_number.txt	version 4
doc/Gizmo_Overview.doc	version 19
doc/Gizmo_Relnotes.doc	version 3

Boldface shows the differences from the first configuration listed above. The file contents are exactly the same; but one filename has changed, and the files have been organized into subdirectories. So this is a different software configuration, even though there has been no change to the *contents* of the files.

Software Configurations and Development Tasks

In most modern software development organizations, many tasks are under way concurrently. At the beginning of this section, we listed a few: new products, new releases of existing products, ports to different platforms, and bugfixes. In addition, consider the fact that each one of the above tasks is often several coordinated efforts: initial development, unit testing, internal system testing, external system (“beta”) testing, final production.

To enable all the tasks to progress smoothly at the same time, each person gets her own software configuration — her own set of versions of the files in the repository. (A small, close-knit team might choose to share a single configuration.)

It’s the job of the software configuration management system, such as AccuRev, to help the organization:

- Keep track of the various configurations.
- Manage, preserve, and protect changes to the files.
- Detect conflicting changes that take place in different configurations (for example, two people modify the same section of the same file).
- Resolve such conflicting changes.

AccuRev Software Configurations: The Stream Hierarchy

This section discusses the AccuRev implementation of software configurations. Be sure to read the section "What is a Software Configuration?" before this section. First, we set the scene and introduce some necessary terminology.

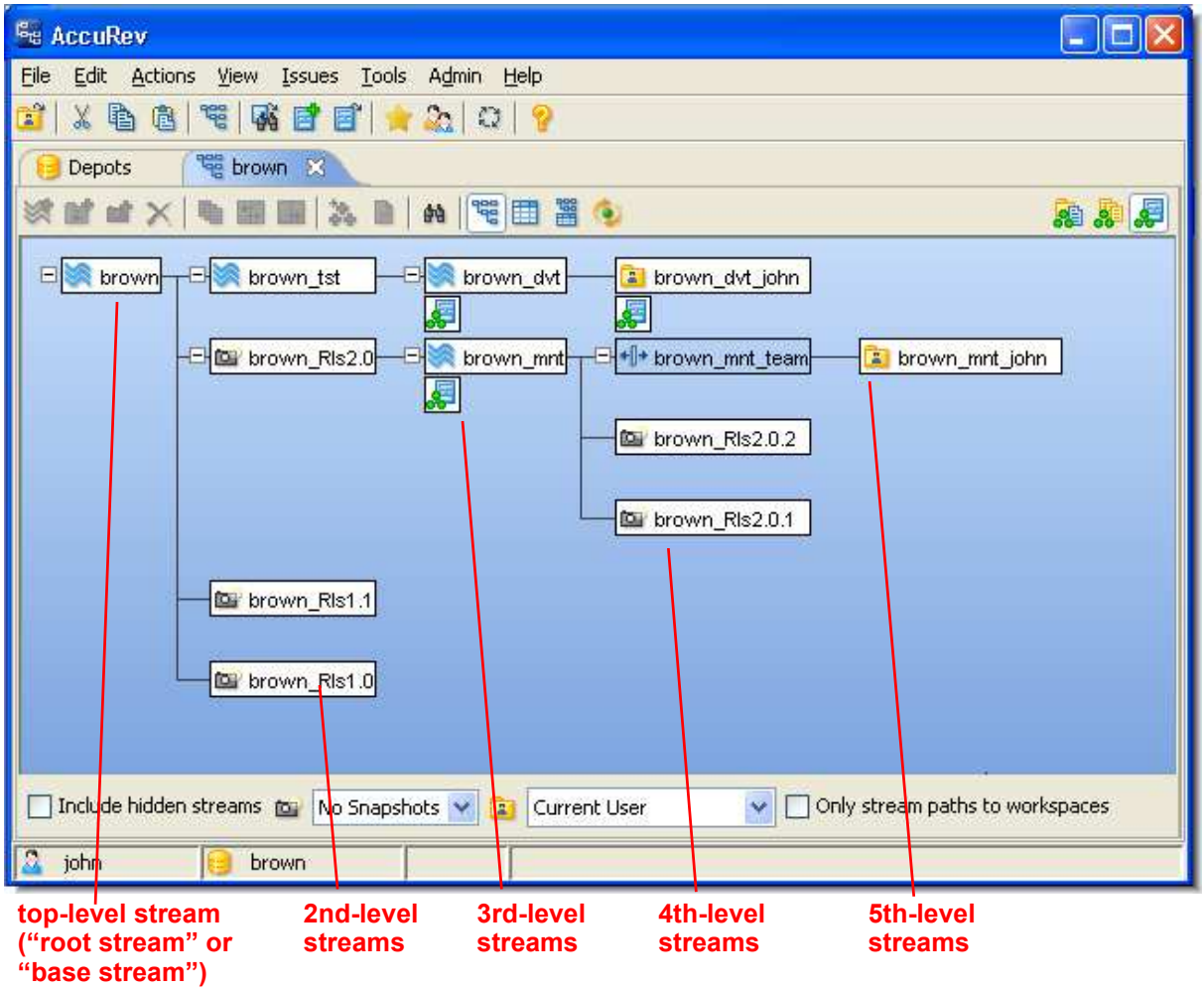
AccuRev's basic job is to keep track of the changes that a development team makes to a set of files. That's called version control. A file under version control is called an element; developers can create any number of versions of each element. AccuRev saves all the versions permanently in a database called a depot.

> Note: we're oversimplifying here. AccuRev version-controls directories as well as files; and there can be multiple depots, each one storing a separate directory tree. But the above paragraph is enough to get us into a discussion of software configurations. For more on depots and version-controlled files and directories, see section *The AccuRev Data Repository* on page 3.

AccuRev can manage any number of configurations of a depot's elements. Each configuration contains one version of every element in the depot — or perhaps, just some of the elements. Here are the basic data structures:

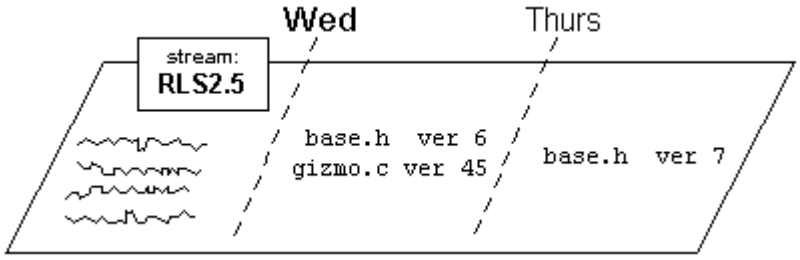
- A stream is a configuration of the depot that changes over time.
- A snapshot is a configuration of the depot that never changes.
- A depot's streams and snapshots are organized into a stream hierarchy: each stream or snapshot has one "parent", and can have any number of "children".

The stream hierarchy can be changed at any time: move a child to a different parent, interpose a new stream between a child and its parent, etc. Using these structures, it's easy and intuitive to model many aspects of the software development process.

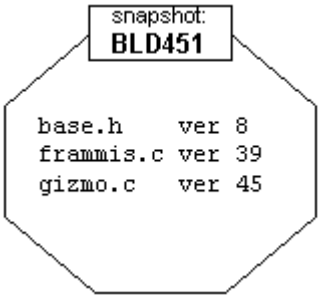


The main idea is to enable multiple development tasks to take place concurrently, and to manage when (and if) work done for one task is shared with other tasks. For example:

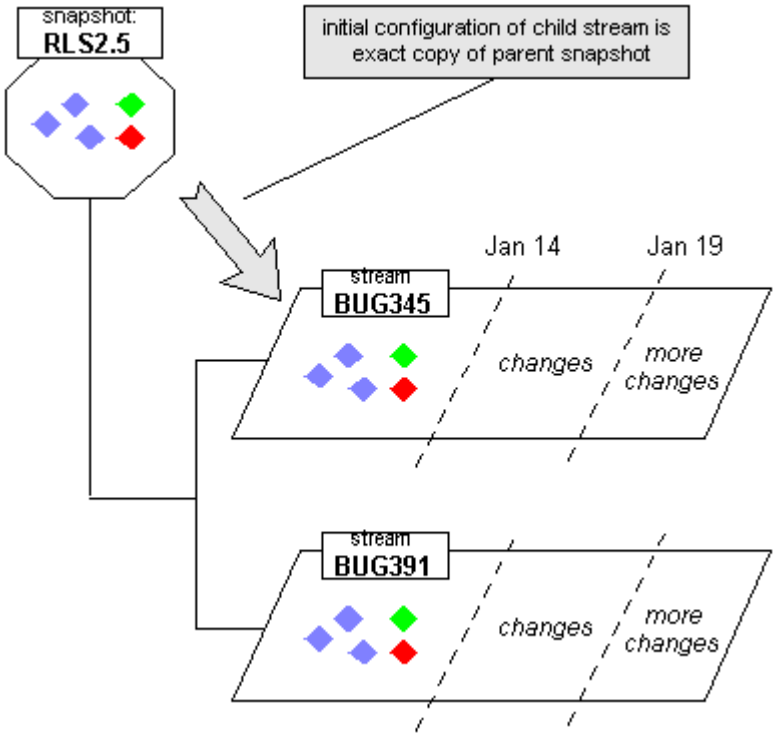
- A stream corresponds to a development task. It might be a long-lived project, such as "the Release 2.5 development effort"; or it might be a quickie, such as "fix error message ERR037". When a developer modifies an element, the new version is recorded as a change to the configuration of a particular stream.



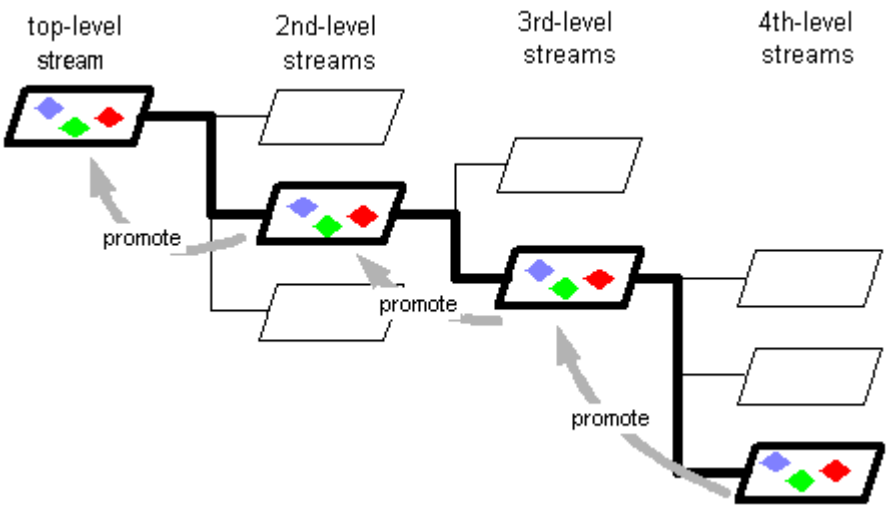
- A snapshot corresponds to a project milestone, such as “Build 451” or “Release 2.5 final build”. It’s vitally important to be able to tell exactly which versions of which files went into Build 451, no matter what changes were made subsequently. A snapshot answers this need precisely and completely reliably, because it’s a never-changing configuration.



- A “parent” snapshot acts as a stable starting point for any number of “child” streams. No matter when a new child is created, its initial configuration is an exact copy of the parent snapshot. This structure is appropriate for managing multiple bugfixes to an old release. Each bugfix stream starts with the versions that were used to build the original release — say, the versions in snapshot “Release 2.5 final build”.



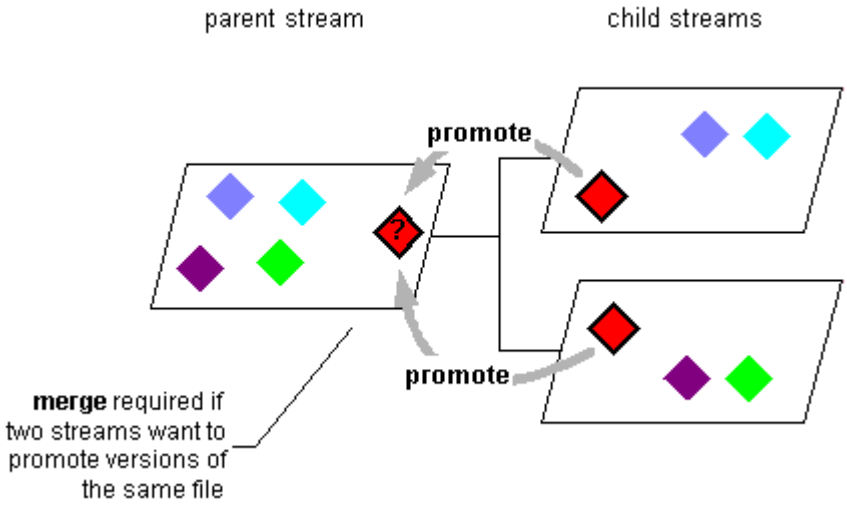
- Versions created at the bottom of the stream hierarchy rise up through the hierarchy by being promoted from stream to stream — from child to parent, then from parent to grandparent, etc. Promotion is one of AccuRev’s most important operations, enabling you to intuitively model a project’s workflow.



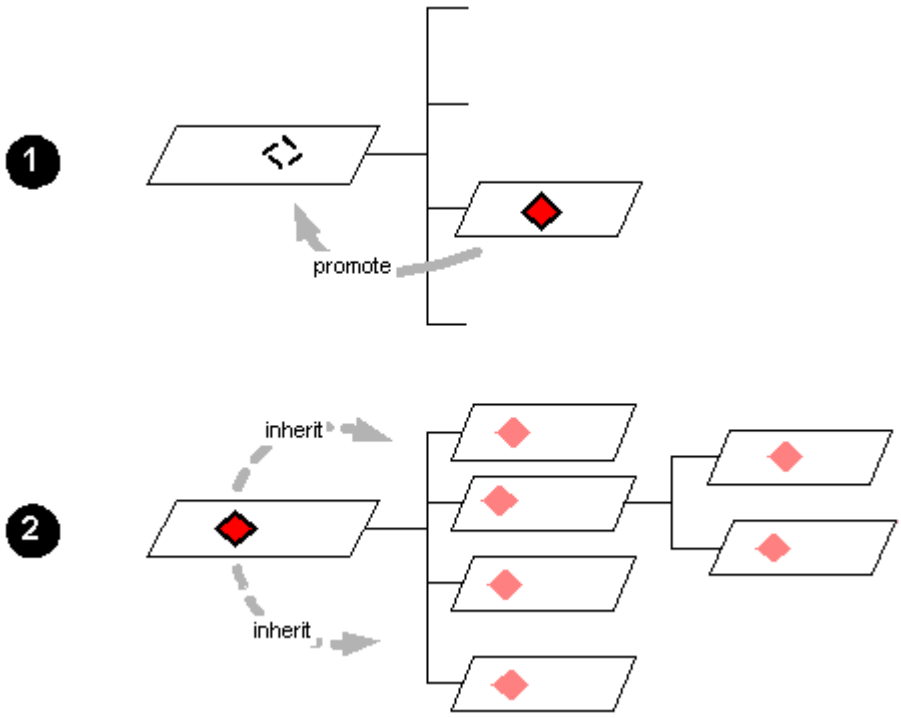
For example, after initial development work on a set of files is completed, the files are submitted to unit testing, then to internal system testing, then to external system ("beta") testing, then to final production. If this workflow is too elaborate for your organization, or not elaborate enough, just design your stream hierarchy differently. You can redesign a project's workflow at any time by changing the stream hierarchy.

- A parent stream provides an integration point for any number of child streams. This structure is appropriate for a development effort that is divided into multiple tasks, to be undertaken concurrently by different developers. As developers complete their changes, they promote the changes to the parent "integration stream".

If two or more developers happen to change the same file, AccuRev makes sure that the changes are merged together. This ensures that one person's work is not overwritten accidentally by another person's.



- Each stream provides a change scope for the subhierarchy beneath it: child streams, grandchild streams, etc. Once a version has been promoted to a stream, that version becomes available to the stream's entire subhierarchy. In many cases, the newly promoted version will appear automatically in ("be inherited by") all the descendant streams. This auto-integration mechanism complements the explicit integration of merging, described in the preceding paragraph.



  For example, suppose a new corporate logo has been designed and saved in a new version of file **corp_logo.png**. Promoting this version to a high-level stream makes it appear instantly in many lower-level streams where Web pages are being developed and updated.

It may be worthwhile to study the above scenarios a bit more, and to consider how your organization might use AccuRev's streams and snapshots in your own development environment. As you do so, keep these two important points in mind:

- A stream is a software configuration, a specification of particular versions of particular elements. A stream doesn't contain copies of files stored in the depot's file storage area; it just contains a "matched set" of versions, selected from all the versions recorded in the depot's database.
- A depot's files are organized into a directory tree; a depot's streams are organized into a tree-structured hierarchy. These two tree structures are different and independent of each other. In a sense, the directory tree is a "picture" of a software application, and the stream hierarchy is a "picture" of the software development process that creates and maintains the application.

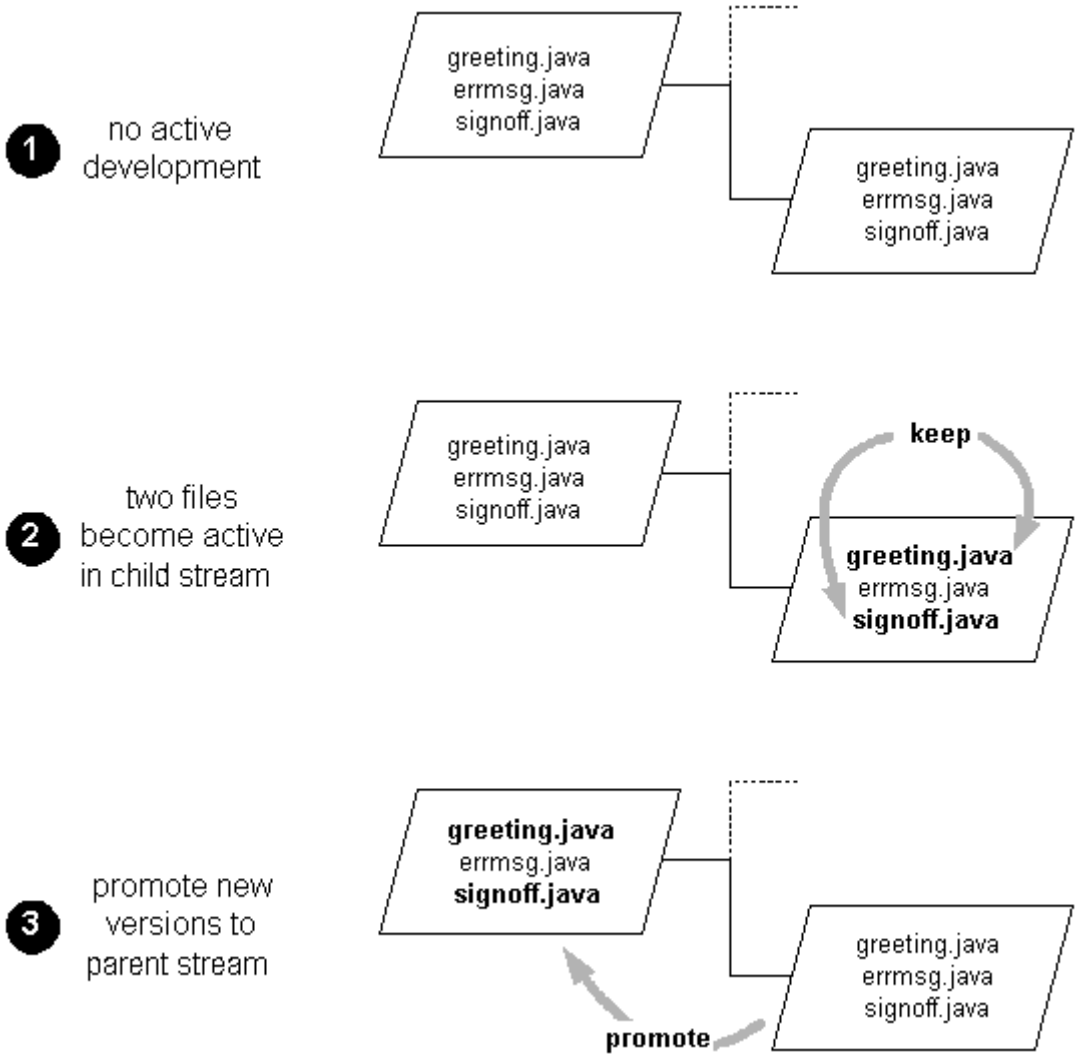
How Changes Migrate Through the Stream Hierarchy

AccuRev provides configuration-management capabilities that are sophisticated and robust, without sacrificing ease of use. What's the secret? One main reason is that AccuRev sees the development environment in the same way as a typical development team:

- Many development tasks are active concurrently, all using the same source base.

- Tasks are often interrelated; they must share their changes with each other ("integration") and weed out inconsistencies; some tasks cannot be completed until one or more others have been completed.
- Most tasks are accomplished by making changes to relatively few files.
- A task is completed by "delivering" a set of changes to another task. For example, a development task might deliver its changes to an integration task, or to a testing task.
- A developer's next task may involve changing a completely different set of files from the previous task.

AccuRev streams neatly model all these aspects of development tasks. The (relatively few) files that a developer changes for a task become active in a particular stream. Typically, this occurs when the developer records new versions of the files, using the **keep** command. To complete the task, or to mark an intermediate milestone, the developer delivers the changes to the parent stream, using the **promote** command. The files become active in the parent stream, and they revert to being inactive (not under active development) in the child stream.

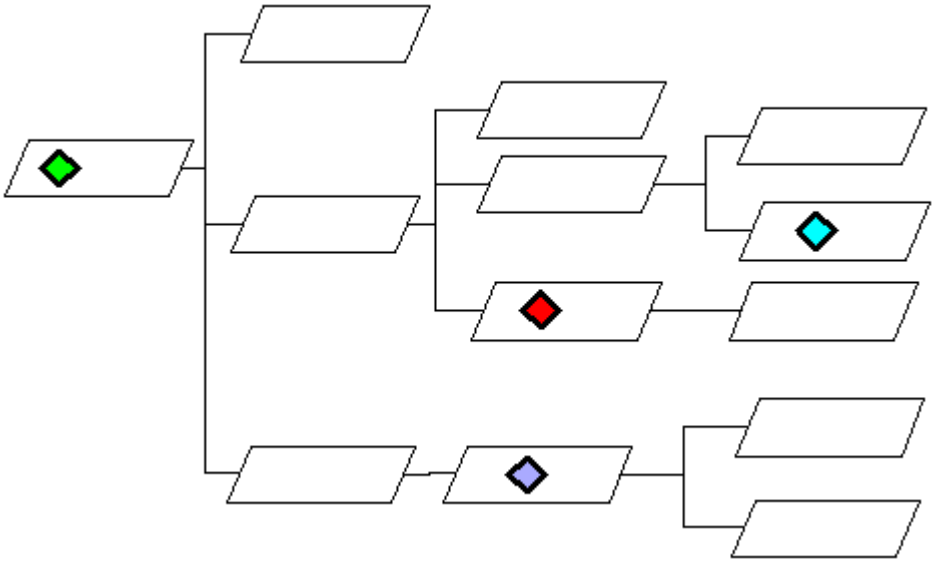


In a multiple-level stream hierarchy, several promotions are required to propagate a set of changes all the way to the top level. Each promotion causes the file(s) to become active in the "to" stream, and inactive in the "from" stream.

> AccuRev terminology: the set of elements (files and directories) that are currently active in a particular stream constitute the default group of that stream.

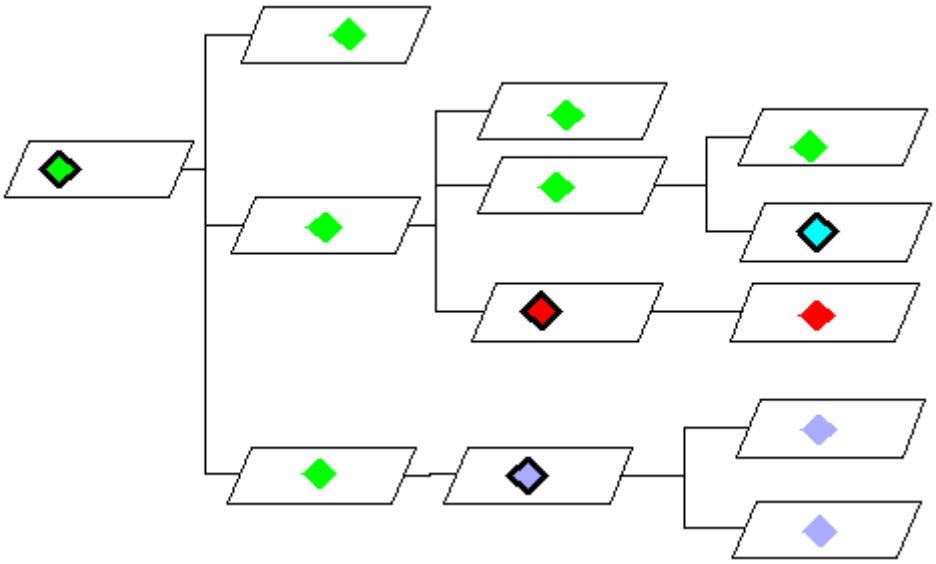
You may have gotten the impression that a given file can be active in only one stream at a time. Not so — that would mean only one development task at a time could be actively working on the file. AccuRev allows each file to be active in any number of streams — even *all* of the streams at once. Typically, though, a file is active in just a few streams at any particular moment.

The diagram below uses contrasting colors to show how a particular file might be active in four different streams. That is, four different versions of the same file are in use at the same time, for various development tasks.



Inheriting Versions From Higher-Level Streams

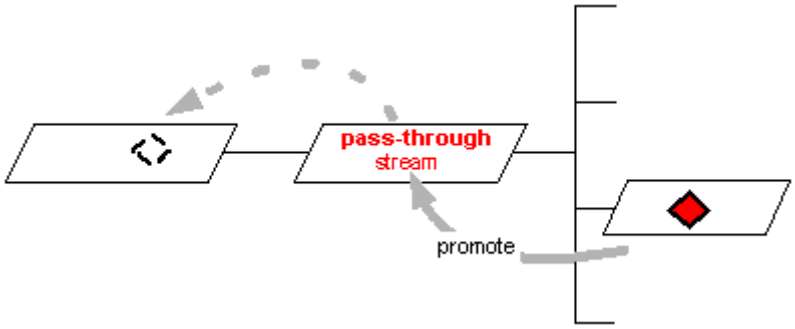
What about the other streams? Each stream in the hierarchy contains *some* version of the file; if a file is not active in a particular stream, the stream automatically inherits an active version from a higher-level stream. The diagram below shows how the four active versions fill out the entire stream hierarchy:



This scheme makes it easy for an organization to manage many development tasks concurrently, each with its own software configuration in a separate stream. As changes are made for certain tasks, AccuRev takes care of automatically applying the changes to the software configurations used by other subsidiary tasks — except for the tasks that are actively working on the same file(s). Just a few **promote** operations can effectively propagate versions to tens or even hundreds of other streams.

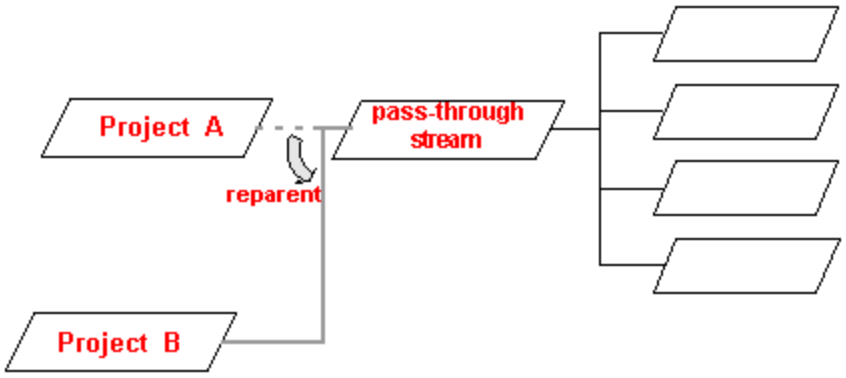
Pass-Through Streams

AccuRev features a special kind of stream, called a pass-through stream. A version that is promoted to such a stream automatically passes through to the parent stream. The file doesn't become active in the pass-through stream; it *does* become active in the parent of the pass-through stream.



Pass-through streams are useful for grouping lower-level streams. (Most commonly, the streams to be grouped are ones built into user workspaces. For a full discussion of workspaces, see *AccuRev Workspaces and Reference Trees* on page 21.) For example, suppose a "swat team" of four programmers often moves from project to project. AccuRev accomplishes the task of moving a programmer from Project A to Project B by reparenting the programmer's personal stream: making it a child of the Project-B stream instead of the Project-A stream.

Reparenting all four programmers' personal streams from the Project-A stream to the Project-B stream requires four separate operations. But suppose the programmers' streams were all children of the same pass-through stream; moving the team to a different project requires just a single operation: reparenting the pass-through stream.



The intermediate stream level doesn't impose any extra day-to-day work on the programmers. The versions they promote automatically pass through the intermediate stream to the project stream.

The Include/Exclude Facility

AccuRev has an advanced include/exclude facility, which vastly increases the flexibility of a depot's stream hierarchy:

- You can configure any dynamic stream (or workspace) to include just *some*, not all, of the elements from its parent stream; the subhierarchy below the stream inherits this configuration. This facility makes it easy to logically partition a source tree, so that different development projects can work on different parts of the source code, and so that different development groups cannot even see each other's work.
- In any dynamic stream or workspace, you can change the backing stream of individual elements or entire subtrees of elements. In effect, this special kind of include/exclude (termed cross-linking) provides a way to make the stream hierarchy look different for different elements. You can think of it as a way to draw "dotted lines" or "detours" in the stream hierarchy, for individual elements or sets of elements.

See *Getting Only the Files You Need: the Include/Exclude Facility* on page 29.

AccuRev Workspaces and Reference Trees

As described in *AccuRev Software Configurations: The Stream Hierarchy* on page 11, AccuRev uses streams to organize your development data, as any number of projects are under way concurrently. But streams are not the entire story:

- A stream is just a bookkeeping device, though a very sophisticated one! It's a database mechanism that records which versions of files are in use for a particular development task. But what about the actual files themselves, which developers edit and build software systems with?
- The **promote** command propagates an existing version of a file from a lower-level stream to a higher-level stream. But how are new versions of files created in the first place?

In other words, how do users access AccuRev-controlled files, in order to perform their day-to-day development tasks? The answer: through workspaces.

A workspace is an ordinary directory tree that instantiates a stream. That is, the workspace contains files that are copies of the versions in the stream. We say that the workspace is "attached to the stream" or "based on the stream". And the stream is said to be the backing stream for the workspace; we'll explain this term in *Updating a Workspace* on page 25.

For example, suppose a stream contains these versions of the elements in a (very small) depot:

```
src/gizmo.c                    version 45
src/frammis.c                  version 39
src/base.h                     verion 8
src/release_number.txt         version 4
doc/Gizmo_Overview.doc         version 19
doc/Gizmo_Relnotes.doc         version 3
```

A workspace attached to this stream is a directory tree containing:

- a **src** subdirectory, containing four files (**gizmo.c**, **frammis.c**, **base.h**, **release_number.txt**).
- a **doc** subdirectory, containing two files (**Gizmo_Overview.doc**, **Gizmo_Relnotes.doc**).

Another stream in the depot's stream hierarchy might contain different versions of some or all the files. So, for example, the contents of files **release_number.txt** and **Gizmo_Relnotes.doc** might be different in a workspace attached to another stream.

Any number of workspaces can be attached to the same stream. A typical scenario is for all the members of a project team to maintain workspaces attached to the stream that records the project's ongoing work. Conversely, a workspace can be attached to any stream. But typically, workspaces are created only at the "leaf level" of a depot's tree-structured stream hierarchy: if a stream acts as the backing stream for one or more workspaces, it generally doesn't have child streams, too.

Using a Workspace

As the name implies, a workspace provides a location for performing development tasks: editing source files, compiling, debugging, testing, creating web sites, etc. Since a workspace is a regular directory tree in the file system, there are no special issues involved with using software development tools with AccuRev data. Just do it.

Here are a few points that show how easy it is to do day-to-day work in a workspace:

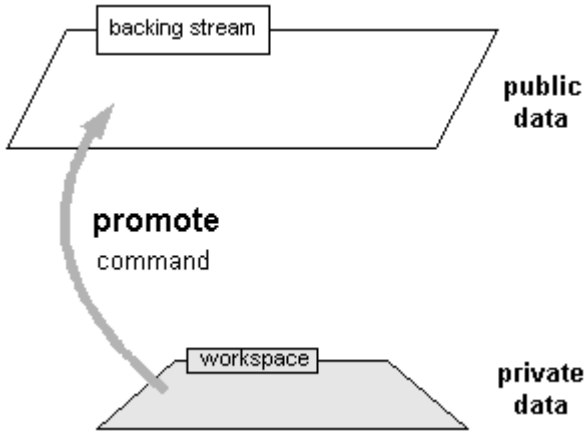
- A workspace need not be in any special file system location. Any place where you have permission to store data will do.
- If you decide you need more space, you can move a workspace to another location. And you don't have to worry about losing track of your workspaces — AccuRev keeps track of every workspace's location.
- You can modify any file in a workspace at any time. Some configuration management systems require you to perform a "check out" operation before working on a file, and keep most files in a read-only state — but not AccuRev.

The thing that's special about a workspace is that it provides a two-way portal to the AccuRev data repository: you put your own changes into the repository, and you draw out the changes that your colleagues have previously recorded there.

Putting Data Into the Repository

A workspace enables you to create new versions of the files in a particular depot. (Each workspace is attached to a particular stream, which belongs to a particular depot.) First, you use any development tools to work with the workspace's copies of existing versions; then you use AccuRev commands to store new versions in the depot. In addition to creating new versions of existing files (**keep** command), you can use the workspace to add new files and directories to the depot (**add** command), rename files and directories (**move** command), and even rearrange the depot's directory hierarchy (**move** command).

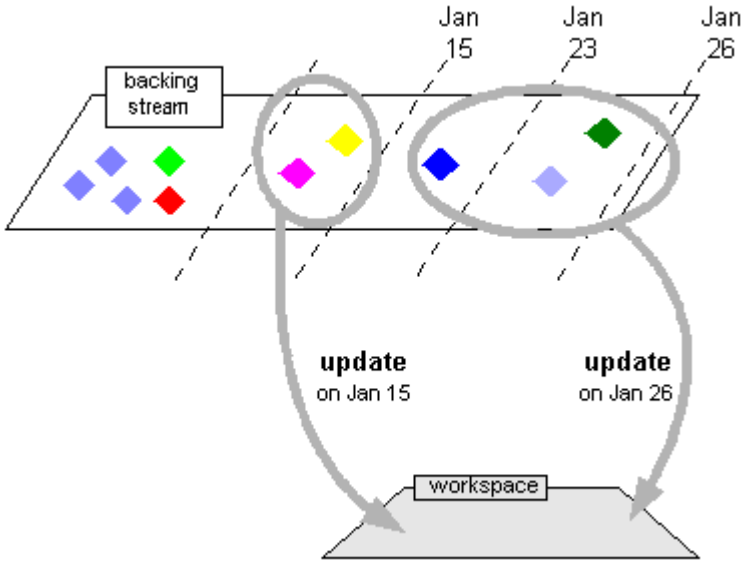
Because it's a separate directory tree, a workspace provides an isolated, private development environment. The changes you make become public only when you enter a **promote** command. This creates versions of one or more files in the attached stream. These versions are public: your changes are now available to be incorporated into other workspaces attached to the same stream. Subsequent promotions to higher-level streams will make the changes available to an even wider scope of workspaces.



Getting Data Out of the Repository

A stream is a changing software configuration of a depot. A typical stream has new versions entering it all the time. Some of the versions are promoted from the workspaces attached to them, as described just above; other versions are inherited automatically from higher-level streams. (See *Inheriting Versions From Higher-Level Streams* on page 17.)

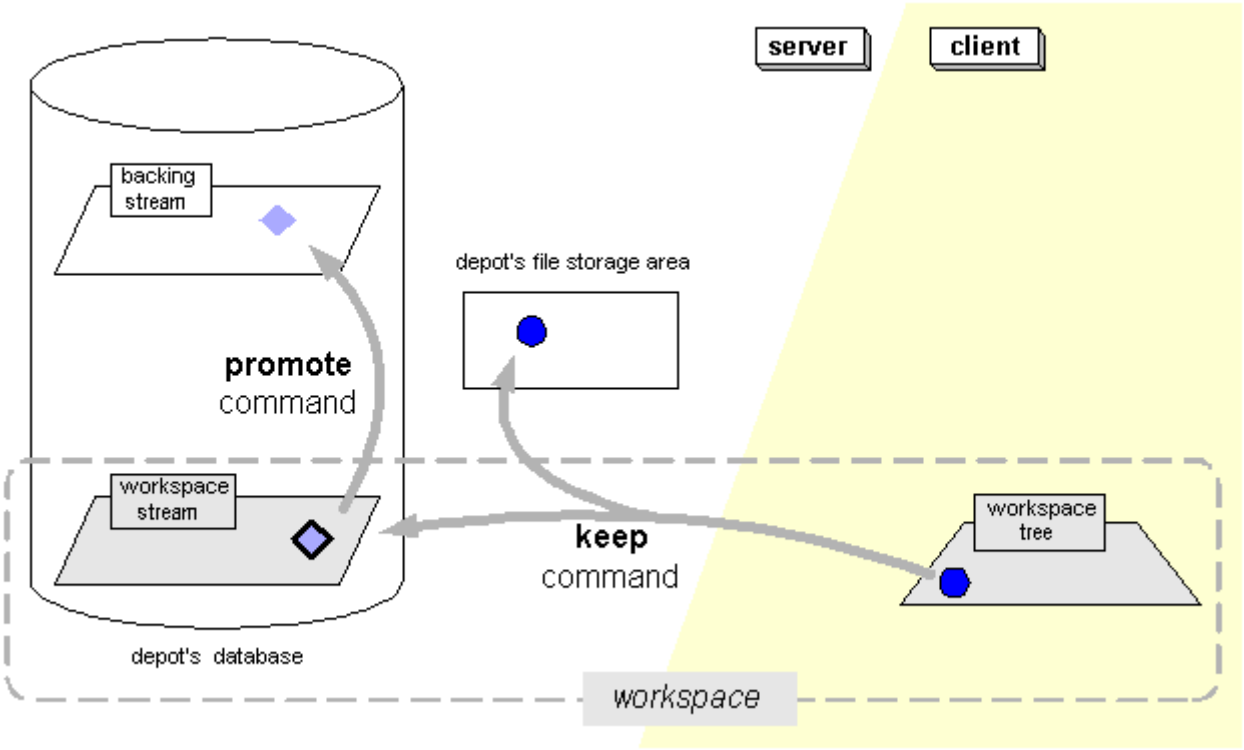
As new versions enter a stream, they become available to the workspace(s) attached to the stream. But AccuRev never copies a new version of a file into your workspace automatically. Instead, you periodically use AccuRev commands to **update** the workspace. This replaces existing files (or adds new ones), so that the files in the workspace accurately reflect the backing stream's current contents, including any recently-arrived versions. AccuRev takes care not to "clobber" files that you're working on when it copies new versions to the workspace.



The Workspace's Built-In Stream

The diagram above, showing how data flows from a workspace into the repository, is an oversimplification. Changes that you make in your workspace don't actually go directly into the backing stream. Long experience with configuration management systems has shown that users sometimes enter changes into the repository before they're truly ready to be shared with others — for example, code that's never been tested. But a delaying strategy also has its drawbacks — for example, it increases the chances of mistakenly deleting several weeks worth of changes without ever preserving them in the repository.

Some other version control systems use "private branches" to address these issues. AccuRev solves the problem by building a private stream into each workspace. This built-in stream is separate from the backing stream. Here's a more detailed diagram showing how data flows from a workspace into the repository; this one includes the workspace's built-in stream.



This diagram shows that in AccuRev's client-server world, a workspace has one foot on each side of the divide:

- The ordinary directory tree that we discussed above (the workspace tree) lives on the client side. The development data you work with on a day-to-day basis lives entirely in the workspace tree; it's "just a bunch of files".
- The built-in stream (the workspace stream) lives on the server side, in the data repository managed by the AccuRev Server. It contains all of the workspace's configuration management information. And it resides, as all streams do, entirely within the database of a particular depot.

The diagram also shows that recording a new version of a file in the backing stream is a two-step process:

1. The **keep** command creates a new version in the workspace stream. Think of **keep** as moving data from the client side to the server side. This command also copies the file in your workspace tree, storing the copy in the depot's file storage area. The data stays within the workspace, remaining private.
2. The **promote** command propagates the version from the workspace's built-in stream to the backing stream. This command operates totally within the depot's database. No data file is copied to the file storage area during a **promote**.

Why the extra stream and the extra step? Isn't it redundant? No, because the workspace stream and backing stream play different roles. The whole idea of the workspace is to provide a degree of isolation from the changes that others are making concurrently. The workspace stream makes the isolation more flexible. It enables you to **keep** any number of intermediate versions of a file in

your workspace, before "going public" by **promote**'ing the most recent version. If you decide that you've headed off in the wrong direction, you can revert a file to any of those intermediate versions and promote that version instead. No one else needs to know. You can even **purge** *all* the work you've done on a file, reverting the workspace to using the version in the backing stream.

All the intermediate versions that you **keep** are stored permanently in the depot, even the versions you never **promote** to the public stream hierarchy. Thus, the **keep** command provides a data-backup capability: "save a copy of this file, just in case I ever want to restore it to its current state". It also means you can change your mind as many times as you like about which version of a file should be shared with the rest of the world.

Real Versions and Virtual Versions

The difference between **keep** and **promote** highlights an important aspect of the way that AccuRev organizes and manages development data. It also highlights the difference between backing streams and workspace streams.

All "real" development takes place in workspaces, because that's where the files are. The **keep** command preserves the changes you've made to a file (Java source file, Perl script, MPEG audio file, etc.) Accordingly, versions created by the **keep** command are called real versions. Real versions live in workspaces — more precisely, every real version is created in the built-in workspace stream of some workspace.

By contrast, the **promote** command does not record a new change to any file. Rather, it changes the approval level and availability of a change that was previously recorded with **keep**. The version that **promote** creates in a higher-level stream is called a virtual version; each virtual version is just a pointer to, or alias for, an existing real version in some workspace stream.

Active Files and the Default Group

AccuRev keeps track of which files you're actively working on in your workspace. This set of files is called the workspace's default group. It includes all the files for which you've recorded changes in the repository. Typically, most of the changes are new versions, created with **keep**. The default group also includes renamed or relocated files (**move** command) and deleted files (**defunct** command).

When you **promote** a file's changes from your workspace to the backing stream, the file is removed from the workspace's default group. This reflects the fact that you're done working on that file — at least for now! Similarly, a **purge** of your work on a file removes the file from the workspace's default group.

Updating a Workspace

The workspace's two-part structure — workspace tree on the client side, workspace stream on the server side — plays an important role in how AccuRev keeps a workspace synchronized with the stream to which it's attached.

At any given time, a workspace should contain:

- the files you're actively working on (that is, the members of the workspace's default group)

- for each other version-controlled file in the depot, a copy of the backing stream's version

(You can think of the active files as being in the "foreground" of the workspace, and the non-active files as being in the "background". Those "background files" are copies of versions in the stream to which the workspace is attached. That's why it's officially called the workspace's backing stream.)

But a workspace often gets out of date with respect to its backing stream. Typically, each member of a development team has his own workspace, and all the workspaces are based on the same backing stream. For files that you're not working on, your workspace continues to have copies of old backing stream versions, even as your colleagues are promoting new versions of those files to the backing stream. If the backing stream contains a file more recent than one in your workspace, that file's status in your workspace is stale. Updating the workspace clears the "stale" status, restoring it to backed.

It's the job of the **update** command to synchronize the workspace and its backing stream in this way. To determine which files you're actively working on, **update** looks in the workspace stream; it considers a file to be active if you've created one or more new versions of it in the workspace stream. Then, **update** makes sure that the workspace tree contains a copy of the backing-stream version of each non-active file. Typically, this involves replacing old files with new files. But it can also involve renaming, relocating, and removing files — if those kinds of changes have recently been recorded in the backing stream.

Variation #1: Workspace Based on a Snapshot

A workspace can be based on a snapshot, instead of a stream. Initially, this might not seem to make sense; after all, a snapshot is an *unchanging* software configuration, and a workspace is a tool for getting changes in and out of the data repository (a "two-way portal"). But a snapshot-based workspace is quite useful — for example, for performing maintenance work on a previous product release.

When you create a snapshot-based workspace, AccuRev copies the versions in the snapshot to the new workspace tree. (This step is just like the creation of a stream-based workspace.) For example, you might create a workspace containing exactly the source versions that were used to build Release 6.1 of your product. That's the only time development data flows from the repository to the workspace. It doesn't make sense to **update** the workspace, because there's guaranteed to be nothing new in the snapshot. It's a configuration that never changes.

You can make changes to the files in a snapshot-based workspace, saving the changes in the workspace stream with the **keep** command. You can't **promote** the changes to the snapshot, though, because — once again — the snapshot is a configuration that never changes. In some cases, there won't be any need for such promotions. For example, some of the bugfixes to a previous product release never need to be propagated elsewhere. You can just build the maintenance release(s) in the maintenance workspace where you've fixed the bugs.

In other cases, you'll want to incorporate bugfixes into ongoing development work — perhaps Release 6.2 or 7.0 of your product. AccuRev has special facilities, including the Change Palette, which enable you to propagate changes from a maintenance workspace (or any snapshot-based workspace) to any stream.

Variation #2: Reference Tree

Let's go back to our original definition of a workspace: an ordinary directory tree that instantiates a stream (or a snapshot). We expanded that definition, showing that a workspace also includes mechanisms for creating new versions in the stream. Sometimes, though, you don't need to create any new versions — you just need the files. For example, you might want a complete set of your product's source files in order to test the speed of a new C++ compiler.

For such "just the files" purposes, you can create a reference tree instead of a workspace. A reference tree instantiates a stream or snapshot, but doesn't provide any mechanism for creating new versions. Thus, you can't use the **keep** or **promote** commands when working in a reference tree. You can use the **update** command, though. Here's a typical scenario:

- Create a reference tree named **nightly**, based on stream **gizmo_dvt**.
- Each night, perform an **update** of the reference tree. This retrieves new copies of the files for which new versions appeared in the **gizmo_dvt** stream that day.
- After the update is complete, build the Gizmo software application using the updated sources.

You can think of a reference tree as a 1-way portal to the AccuRev data repository (in contrast to a workspace, which is 2-way).

Parallel and Serial Development

Like other advanced configuration management systems, AccuRev supports parallel development:

- **Edit Stage.** Two or more users start with the same data: a particular backing-stream version of a file. Each user works on a copy of the file in his own workspace. He can keep as many (private, intermediate) versions as he wishes in his workspace stream.
- **Merge Stage.** The merge stage begins when one of the developers promotes his private version of the file to the backing stream. After that, each other developer must merge the current version in the backing stream into his own work, then promote this merged, private version. In the end, all users' changes are incorporated into the backing stream; conflicting changes to the file, if any, are both detected and resolved.

If two developers work on a file concurrently, a single merge-and-promote is required. If *N* developers work on a file concurrently, then *N–1* merge-and-promotes are required.

Serial Development through Exclusive File Locking

Parallel development is flexible and powerful, but it is not appropriate for every situation. Some organizations don't like the extra steps involved in merging, even though merging is largely automated. Some files cannot be merged, because they are in binary format. (The merge algorithm handles text files only, not binary files such as bitmap images and office-automation documents.)

Accordingly, AccuRev supports serial development through its exclusive file locking feature. Each workspace is in parallel-development mode (exclusive file locking disabled) or is in serial-

development mode (exclusive file locking enabled). You can also set file locks on individual elements.

The serial development model places more restrictions on users in the edit stage, but it eliminates the merge stage altogether. Here's the standard scenario, in which all the workspaces are in serial-development mode:

1. A user starts working on a file by specifying it in a **co** ("checkout") or **anchor** command. The file changes from being read-only to writable.
2. AccuRev places an exclusive file lock on the file. This prevents the file from being processed with **co**, **anchor**, or **keep** in other workspaces.
3. The user can edit and **keep** any number of private versions of the file in his workspace. Then, the user **promote**s his most recently kept version to the backing stream. The exclusive file lock guarantees that no merge will be required before this promotion.
4. After **promote** records the new version in the backing stream, things return to the initial state: AccuRev releases the exclusive file lock, and the file returns to read-only status in the user's workspace.
5. A user in any workspace can now **co** or **anchor** the file, which starts the exclusive-file-locking cycle again.

For more details, see *Exclusive File Locking and Anchor-Required Workspaces* on page 4 of the *AccuRev CLI User's Manual*.

The Limited Effect of an Exclusive File Lock

Exclusive file locking does not freeze an element completely:

- The lock applies only within the scope of a particular backing stream. It doesn't affect other backing streams and the workspaces based on them.
- The lock acquired through workspace-level or depot-level exclusive file locking applies only to workspaces in serial-development mode. Users in parallel-development-mode workspaces can make changes and promote the changes to the backing stream.

  A lock placed on an individual file element in a workspace applies to all sibling workspaces.
- The lock doesn't prevent the current version in the backing stream from being promoted to higher-level streams.

Exclusive file locking does not prevent any user from modifying any file with a text editor or IDE. AccuRev encourages users in serial-development-mode workspaces to "ask permission first": it maintains files in a read-only state, and makes a file writable when a user executes a **co** or **anchor** command on it. But users can modify a file "without asking permission", by changing the access mode (Unix/Linux: **chmod** command, Windows: **attrib** command or **Properties** window) and then editing it. Such "unauthorized" changes can't be sent to the AccuRev depot, though: the exclusive file lock disallows a **co**, **anchor**, or **keep**.

Anchor-Required Workspaces

AccuRev also offers a less-restrictive variant of exclusive file locking. Anchor-required workspaces allow parallel development, with multiple users modifying the same file at the same time (in their own workspaces). But as in the exclusive file locking environment, files are read-only by default, and must be anchored ("checked out") before they can be modified.

Getting Only the Files You Need: the Include/Exclude Facility

In some development situations, it makes sense to configure your workspace to contain a specified subset of the depot's elements, rather than all the elements. The benefits can be quite significant:

- less clutter, allowing you to concentrate on the files that are important to you
- less disk space required to store your workspace on your machine
- faster backups of your workspace
- faster AccuRev processing of your workspace, especially during the **Update** command

AccuRev's include/exclude mode implements this capability. You use include and exclude rules to specify which elements in the backing stream become part of a workspace:

- An include rule can specify an individual file, an individual directory's contents, or the contents of an entire directory tree.
- An exclude rule can specify an individual file or an entire directory tree.
- Rules are inherited by lower-level streams and workspaces, in much the same way that versions are inherited.
- Rules at lower levels of the directory hierarchy can refine rules at a higher level. For example, a graphic artist might add a rule to exclude everything below the directory **src**, but then add another rule to include the single subdirectory **src/gui/images/icons**.
- An include rule can implement a cross-link, specifying a different backing stream for an individual element or an entire subtree. In effect, this enables different elements to have different stream hierarchies, even though they are in the same depot.

Include/exclude rules are not restricted to workspaces — they work throughout a depot's stream hierarchy. Rules set in higher-level streams are inherited by lower-level streams. Using rules in streams is a powerful way to partition a depot's elements — for example, to restrict different groups of developers to different parts of the source tree. See *The Include/Exclude Facility* on page 19.

Historical Note: Sparse Workspaces

Prior to Version 3.5, AccuRev supported a feature similar to include/exclude mode, called sparse workspaces. A sparse workspace started out empty; you added certain elements to the workspace using the **Populate** command. Those elements were maintained in the regular manner, using **Keep**, **Promote**, and **Update**. Other elements in the depot were ignored.

This scheme was satisfactory for many purposes, but there were some drawbacks. For example, an **Update** would not bring newly created elements into your workspace, just new versions of the elements that you had already **Populate**'d.

The include/exclude facility has significant advantages over sparse workspaces:

- Include rules and exclude rules are official attributes of the workspace, maintained in the AccuRev repository. Since AccuRev knows that a directory is included in the workspace, it "remembers" to bring newly created elements in that directory into the workspace during updates.
- You can use include rules and exclude rules to configure streams as well as workspaces. The rules are inherited down the stream hierarchy. For example, to make the **marketing** directory invisible to all developers, you can exclude that directory from a stream that all developers' workspaces are based on (either directly or through multiple stream levels).

AccuRev no longer supports the creation of sparse workspaces, but you can continue to use existing sparse workspaces that were created in older versions of AccuRev.

AccuRev Transactions

The AccuRev data repository is organized into a set of depots, each of which stores the complete revision history of a particular directory tree. Each depot has its own database. Changes to a depot's database are structured as a series of transactions, each of which captures all the information involved in a particular change to the database. Thus, the entire story of how a depot's directory tree has evolved is contained in its transaction history.

Transactions are a well-established database technology, helping to guarantee that the database is always in a self-consistent state. But for AccuRev, transactions are not just a low-level mechanism for achieving database integrity. They play an essential role in organizing the user environment. Two aspects of AccuRev transactions make this possible: atomicity and immutability.

Transactions are Atomic

A user command that modifies elements is recorded as a single transaction in the depot's database, no matter how many elements are involved. For example, if a user enters a **keep** command to create new versions of 12 files, a single transaction records all 12 versions. What if something goes wrong (for example, a network failure) while AccuRev is processing those 12 files? The entire transaction is cancelled, and no new version is created of any file. We use the term atomic to describe this "all or nothing" aspect of AccuRev transactions.

The atomicity of transactions makes life simpler for the user. He never needs to worry about how to finish up the work of a partially-successful command. If a command fails, he just fixes the problem that caused the failure and enters exactly the same command again. Atomicity also means that AccuRev's view of the various changes applied to the repository matches the user's view.

> Note: AccuRev does not record *every* change in a transaction, only changes to your development data. Thus, **keep**ing a new version is recorded in a transaction, as is **promote**'ing an existing version to a higher-level stream. But no transaction is recorded when you create a new stream or change the location of a workspace.

Transactions are Immutable

Once a transaction is recorded in a depot's database, it's there permanently. There is no way to revise or delete an existing transaction — we describe the transaction as immutable. (And we describe the depot's database as being "append-only".) This property is essential to successful configuration management. Users must be able to recreate previous configurations with absolute reliability. The immutability of transactions means that users can reproduce *any* previous configuration, not just a few configurations that they happened to designate with a "save" or "label" command.

AccuRev does make it easy to undo the *effect* of a transaction. For example, the **revert** command reinstates an old version of one or more files. But this is accomplished by recording an additional transaction, not by removing any existing transaction.

Transactions and Workspaces

This section describes how AccuRev uses a depot's transaction history to efficiently manage the contents of the depot's workspaces.

Over time, the version-controlled files in a workspace change in two ways: you modify certain files yourself, using text editors and other development tools; and you periodically use the **update** command to get copies of the files that your colleagues have modified. Accordingly, at any given moment the version-controlled files in a workspace fall into two categories:

- **Files placed in the workspace by the 'update' command.** All of these files are unmodified copies of the versions in the workspace's backing stream at the time of its most recent **update**. Some of these files may have been placed in the workspace during previous updates. Typically, some files are copied into the workspace when it is originally created and are never touched thereafter, because no new versions of the files are ever created in the backing stream.

  AccuRev records the fact that the workspace is up-to-date as of the transaction that most recently precedes the time of the update. (This is completely accurate — no new versions could have been created between that transaction and the update.) This transaction is called the current update level of the workspace.

- **Files that you've worked on in the workspace.** These are files that you've modified (or newly created), and whose changes you've preserved with the **keep** (or **add**) command. You may also have **promote**'d the latest version you created to the workspace's backing stream.

AccuRev can quickly fulfill a request to **update** the workspace, because it doesn't need to consider every file in the depot. Instead, it needs to process only the files that have gotten new versions since the workspace's last update. It accesses these versions by examining the set of transactions between the workspace's current update level and the most recent transaction. When the update is complete, the most recent transaction becomes the workspace's new update level.

Transactions and Issue Management

The atomicity of transactions makes it efficient to implement the integration between AccuRev's basic version-control facility and its issue-management facility (AccuWork). Suppose a particular AccuWork issue record contains a bug report. When you fix the bug by modifying five files, you'll want to note this fact in the issue record. AccuRev can simply note the single **promote** transaction that placed the fixed versions of the five files in the backing stream. Alternatively, you can have AccuRev keep track of the individual versions in the issue record; in this case, the issue record acts as a change package, recording all the versions that were created to implement a particular bugfix or new feature.

AccuRev/AccuWork Change Packages

Any version-control system must be able to keep track of the changes that developers make to individual files. A full-fledged configuration management system, like AccuRev, should be able to handle questions like these:

> "What were all the changes made to source files in order to fix bug #457?"
>
> "Have all the changes made to fix bug #457 been handed off to the QA Group" (That is, have the appropriate versions been promoted to the QA stream?)

AccuRev can handle such questions through its change package facility. (Change packages are available in the AccuRev Enterprise product only.)

Structure of a Change Package

A change package is a collection of element versions; for example:

> version **kestrel_dvt_jjp/13** of element **/./src/brass.c**
> version **kestrel_dvt_jjp/14** of element **/./src/brass.h**
> version **kestrel_dvt_jjp/16** of element **/./src/commands.c**

The basic idea is that this set (or "package") of versions contains all the changes required to implement a certain development project. But we need to refine this idea. Consider that version 14 of **brass.h** probably contains *more* than just the changes for that development project. For example:

- Versions 1-7 might have been created years ago, when the product was first developed
- Versions 8 and 9 might have been minor tweaks, performed last month
- Versions 10-14 are the only versions with changes for the development project in question

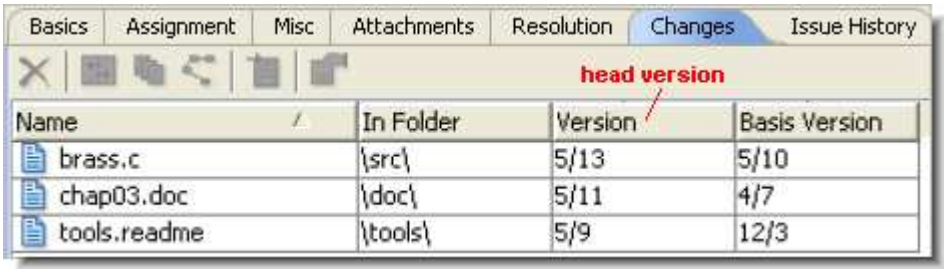
So we need a way to express the idea that only the "recent changes" to **brass.h**, those in versions 10-14, are to be included in the change package. AccuRev accomplishes this by defining each change package entry using two versions: a user-specified head version and an older, automatically-determined basis version The "recent changes" to be included in the change package were made by starting with the basis version (version 9 in this example) and **Keep**'ing one or more new versions (versions 10, 11, 12, 13, and 14 in this example).

In the AccuRev GUI, the head version of a change package entry is usually identified simply as the "Version".

> Note: the **Patch** command uses the same "recent changes" analysis to determine which changes in the "from" version are to be incorporated into the "to" version.

Where should the change package entry for **brass.h** be recorded? AccuRev already provides a mechanism for keeping track of development activities: the AccuWork issue-management facility. Each task — fixing a bug, creating a new feature, etc. — is tracked by a particular AccuWork issue record. So it makes sense to implement change packages using issue records.

Each issue record includes a **Changes** section that acts as an "accumulator" for versions' changes. Here's how the above example of a change package would appear in an issue record's edit form:

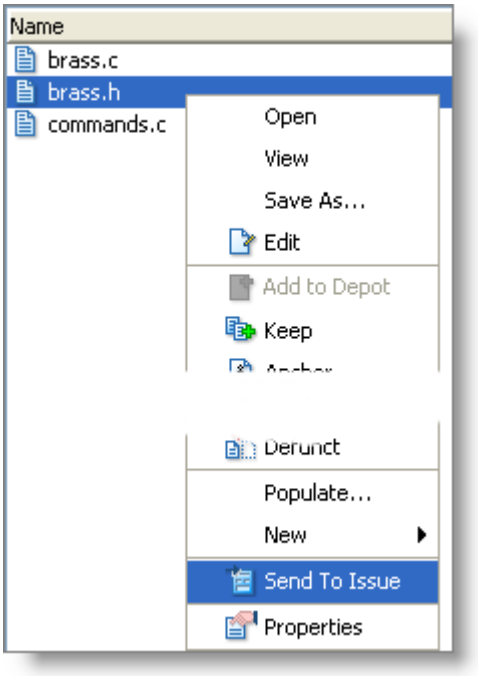


This change package has entries for three elements:

- **brass.c**: The basis version, 5/10 was created in the user's own workspace. This indicates that the user promoted 5/10 to the backing stream. AccuRev assumes that this change was for another task, not the one covered by this issue record. Then, the user turned his attention to the current task, creating additional versions up to and including 5/13, the head version.
- **chap03.doc**: This change began when the user updated his workspace, bringing in version 4/7 of the element (which had originally been created in another workspace, then was promoted to the backing stream). Then, the user created one or more versions in his own workspace, up to and including version 5/11, the head version.
- **tools.readme**: Similarly, this change began when the user updated his workspace, bringing in version 12/3, originally created in another workspace. The user created one or more versions in his workspace, ending with version 5/9, the head version.

Each change package can include at most one entry for a given element. This rule helps to ensure that the changes in a given change package are consistent with each other. See *Updating Change Package Entries* on page 37.

Creating Change Package Entries



You can add entries to a change package manually: right-click a version in the File Browser, Version Browser, or History Browser, and then select the **Send to Issue** command from the context menu. The selected version becomes the head version of the change package entry; AccuRev automatically determines the corresponding basis version. As the examples above suggest, AccuRev uses an algorithm that determines the set of "recent changes" to the element, made in a single workspace.

In the Version Browser, a variant command, **Send to Issue (specifying basis)**, enables you to pick the basis version, rather than allowing AccuRev to determine it automatically.

You can also invoke the **Send to Issue** command on the Changes tab of an issue record. This copies an existing change package entry to a different change package (issue record).

AccuRev can record change package entries automatically, whenever the **Promote** command is invoked in a workspace. For example, suppose issue record #3 represents a particular bug (and its fix). Whenever a developer promotes one or more versions whose changes address that bug, he specifies issue #3 at a prompt. AccuRev automatically creates a change package entry in issue #3 for each promoted version.

Automatic recording of change package entries is enabled through the change-package-level integration between AccuRev configuration management and AccuWork issue management. For more on both these integrations, *Integrations Between AccuRev and AccuWork* on page 81 of the *AccuRev Administrator's Guide*.

Complex Change Package Entries

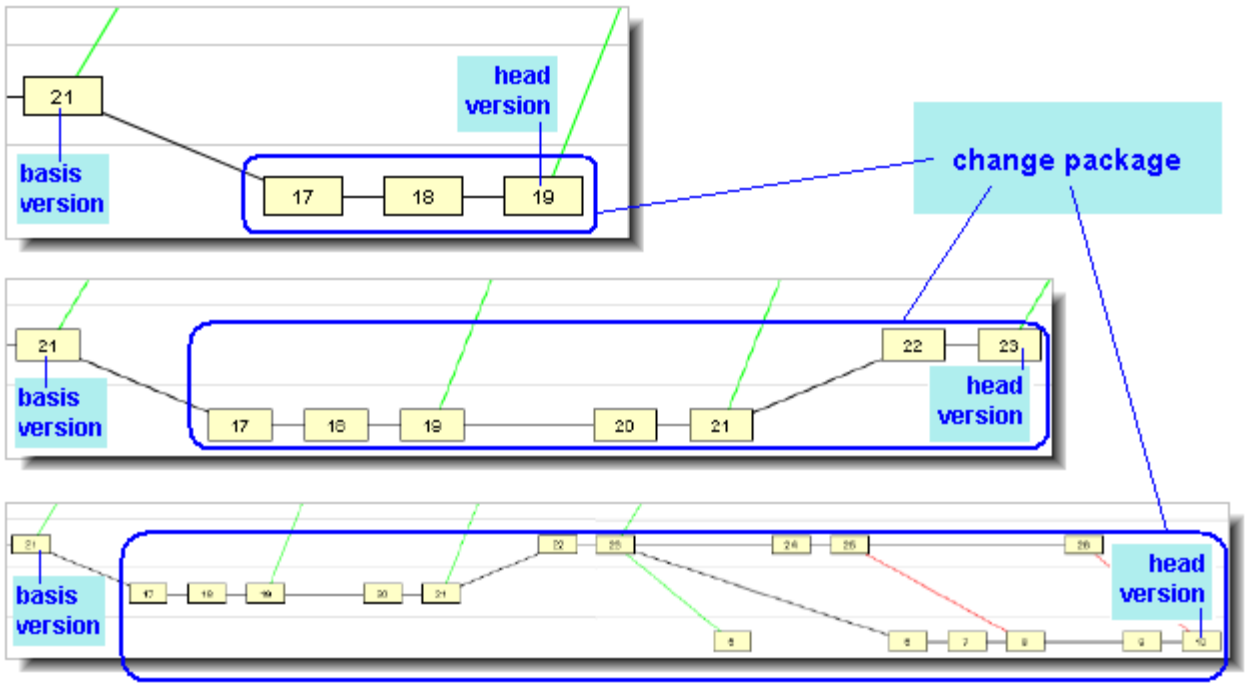
All change package entries are recorded in terms of real versions (those created in users' workspaces), even though there may be corresponding virtual versions (created by promoting the real versions from workspaces to dynamic streams). In all the examples shown above, each change package entry is a series of consecutive real versions created in the same workspace — that is, each change package entry records a particular patch to the element.

But the change package facility can also track *ongoing* changes to elements — changes made at different times, and in different workspaces. To support this capability, AccuRev defines a change package entry in a more general way than a patch:

> A change package entry for an element consists of all the real versions in the element's version graph between a specified basis version and a specified head version. Between-ness is determined both by direct predecessor-successor connections (created, for example, by **Keep**)

and by merge connections (created by **Merge**). Patch connections are *not* considered in this determination; the basis version itself is not part of the change package entry.

The following Version Browser excerpts show the range of complexity that a change package entry can have. In fact, these excerpts show how the same change package entry can change over time, becoming more complex.



These illustrations suggest the following definition for a change package entry, which is equivalent to the definition above:

> A change package entry for an element consists of the element's entire version graph up to the specified head version, *minus* the entire version graph up to the specified basis version. For these purposes, the version graph includes direct predecessor-successor connections and merge connections, but not patch connections.

Updating Change Package Entries

When you want to create change package entry for a particular element, but an entry for that element already exists in the change package, AccuRev attempts to combine the new entry with the existing one. (Recall that there can be at most one entry for a given element in a given change package.) This produces an updated entry that includes all the changes.

A Little Bit of Notation

To help explain how AccuRev performs "change package arithmetic" to combine and update entries, we'll use a simple notation. Suppose a change package entry contains the set of an element's versions defined by these specifications:

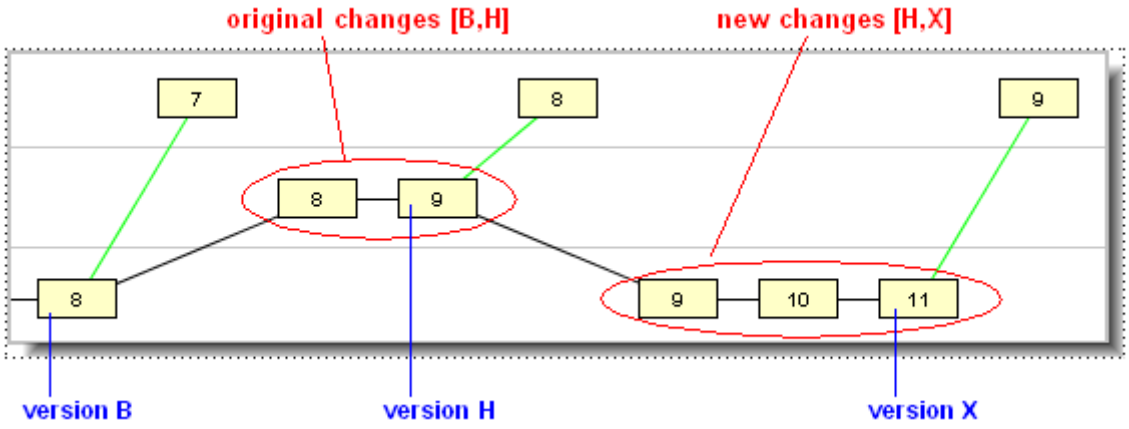
the head version is **H**
the basis version is **B**

We'll use the ordered pair **[B,H]** to indicate this change package entry.

Combining Two Change Package Entries

Now, suppose a new change is to be combined with the existing change package entry **[B,H]**. There are several cases, each handled differently by AccuRev:

- **Case 1: [B,H] + [H,X]** — This simple case typically arises when you think you're done with a task and record your work as change package entry **[B,H]** — but it turns out that more work on the same element is required. So you (or a colleague) start where you left off, with version **H**, and make changes up to version **X**. Then, you want to incorporate the new set of changes **[H,X]** into the same change package.

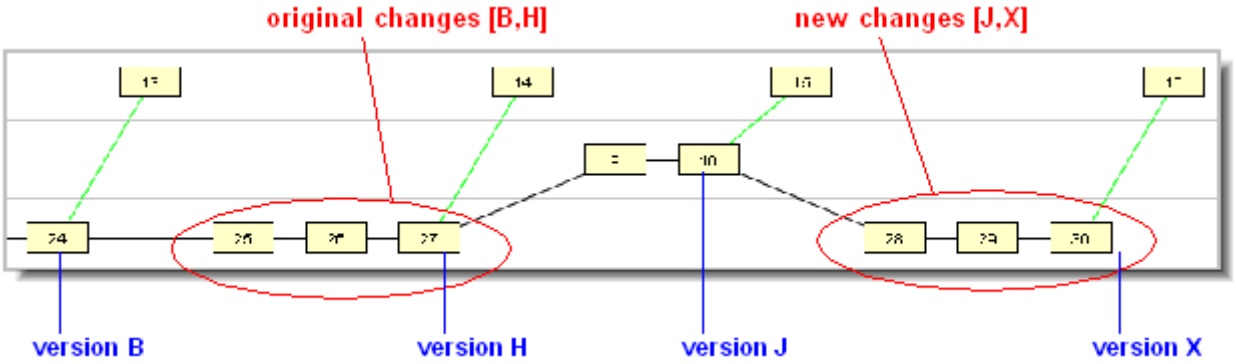


In this case, it's clear that the two series of changes can be viewed a single, uninterrupted series — starting at version B and ending with version X. That is:

[B,H] + [H,X] = [B,X]

Accordingly, AccuRev updates the change package entry automatically — keeping **B** as the "Basis Version" and changing the "Version" from **H** to **X**.

- **Case 2: [B,H] + [J,X]** (where H is an ancestor of J: “change package gap”) — This case typically arises when you do work on a task at two different times, and someone else has worked on the same element in between.



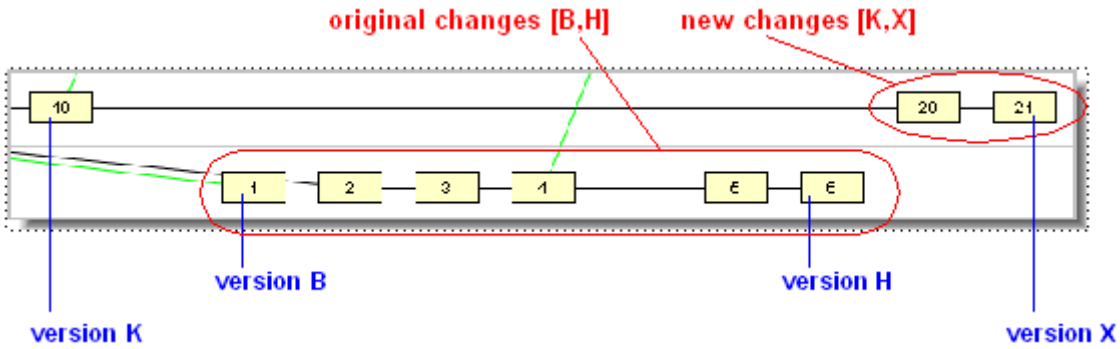
In this example, a colleague updated her workspace to bring in your original changes, created versions 9 and 10 in her workspace, and promoted her changes. You then updated your workspace to bring in her changes, and made a new set of changes.

When AccuRev tries to combine the change **[B,H]** and the change [**J,X]** into a single change package entry, it detects that version **H** and version **J** are not the same, but that **H** is a direct ancestor of **J**. Thus, there is a simple “gap” in the potential combined change package entry (in this example, consisting of your colleague’s versions 9 and 10).

Probably, your colleague was not working on the same task when she made her changes. (If she had been, she would have added her changes to the same change package, as in Case 1.) On the other hand, it’s probably OK to include the entire, uninterrupted series of versions **[B,X]** in your change set — this includes both your original changes and your new changes (and, harmlessly, your colleague’s changes, too).

Accordingly, AccuRev can “span the gap” between the two change set entries, in order to create a single, combined entry.

- **Case 3: [B,H] + [K,X]** (where H is *not* an ancestor of K: “change package merge required”) — This case typically arises when developers in workspaces that do not share the same backing stream try to use the same change package. There is no simple “gap” between the existing change package entry and the new one — which means there is no way to combine them into a single change package entry, according to definitions in *Complex Change Package Entries* on page 35.



AccuRev signals this situation with a “change package merge required” message, and cancels the current operation. You can remedy this situation by performing a merge at the element level. (There is no merge operation defined at the change package level.) In the example above, merging version H and version X would create a new version; a change package entry with the new version as its head can be combined with the existing entry.

AccuRev Glossary

3-way merge

The kind of algorithm that AccuRev uses to combine the contents of two *version*s (*contributor*s) of a text-file *element*: it compares the two files line-by-line with a third version, the *closest common ancestor* of the *contributor*s.

access control list

A data structure that controls the rights of one or more users, or groups of users, to access the data within a particular *depot* or *stream*.

access mode

(Unix/Linux only) The standard set of permissions (user/group/others, read/write/execute), as they apply to a particular file *element*.

AccuRev home directory

A subdirectory, named **.accurev**, of your operating-system home directory (or of the directory specified by environment variable ACCUREV_HOME). This subdirectory stores your *preferences file* and other AccuRev configuration files.

AccuRev Server

The program that manages the AccuRev *repository* and handles commands issued by AccuRev *client program*s.

ACL

A set of entries ("permissions") that controls the rights of individual users or user groups to access the data within a particular *depot* or *stream*.

active

An *element* is said to be active in a *workspace* or *stream* if a new *version* of the element has been created there, and that version has not been either (1) *promote*d to the *parent stream* or (2) purged from the workspace or stream. An *issue record* is said to be active in a workspace or stream if the *head version* of one or more of its *change package* entries is in the stream's default group. See *default group*, *backed*, *passive*.

add

The operation that places a file or directory, located in a user's *workspace tree*, under *version control*.

ancestor

In the *version* graph of an *element*, version A is an ancestor of version B is there is a direct line of descent (possibly including *merge*s) from A to B. See *predecessor* (or *direct ancestor*). "A is an ancestor of B" is equivalent to "B is a descendant of A".

ancestry

The entire set of *version*s of an *element*. See *version graph*.

anchor

A "checkout"-type operation, which declares that a file *element* is under development in the current *workspace*. AccuRev records the fact that the element is "active" by adding it to the workspace's default group. With exclusive file locking, anchoring a file in one workspace prevents it from being made active in sibling workspaces.

anchor-required

An optional setting on a *workspace*, specifying that the workspace's file *element*s are to be read-only until the user performs a *checkout* operation (GUI: **Anchor** command; CLI: **anchor** or **co** command).

anyuser

A security-related keyword: describes the set of users who do not have passwords. See *authuser*.

archive

An operation that transfers the *storage file*s for one or more *version*s from a *depot*'s *file storage area* to its *gateway area*. After the archived storage files are copied to off-line storage, the disk storage within the gateway area can be *reclaim*ed.

atomic

An important characteristic of AccuRev *transaction*s: the entire transaction (including all specified files) must be performed successfully; if not, the entire transaction is cancelled, as if it were never attempted.

authuser

A security-related keyword: describes the set of users who have passwords. See *anyuser*.

backed

An *element* has "backed" status in a *workspace* or *stream* if it is not currently *active* there. This means that the workspace/stream *inherit*s the *version* of the element that is currently in the workspace/stream's *parent stream* (also called the *backing stream*).

backing chain

The "path" (sequence of *stream*s) through the *depot*'s *stream hierarchy*, leading from a particular *workspace* or stream up to the depot's *root stream*.

backing stream

("parent stream", "basis stream") The *stream* that is just above a given *workspace* or stream in a *depot*'s stream hierarchy. The given *workspace*/stream inherits *version*s from the backing stream.

base rule

The *include/exclude* rule that makes the top-level directory of a *depot* appear in the depot's root stream.

base stream

("root stream") The top-level *stream* in a *depot*'s stream hierarchy.

basis stream

("parent stream", "backing stream") The *stream* that is just above a given *workspace* or stream in a *depot*'s stream hierarchy. The given *workspace*/stream inherits *version*s from the basis stream.

basis time

A date-timestamp setting for a *stream*, affecting which *version*s the stream inherits from its *parent stream*: for each *element*, the version inherited is the one that was in the parent stream at the basis time. See *snapshot*.

basis version

A particular *ancestor* of the *version* specified in a **patch**, **revert**, **diff**, or **co** command. The series of versions between the basis version and the specified version constitute the "recent changes" to be patched into (or removed from) the target. Similarly, a *change package* entry consists of all the versions between a specified basis version and a specified head version.

binary

See *element type*.

change package

A set of entries, each in the form of a *basis version*/*head version* pair, recorded on the Changes tab of an *issue record*. The change package records the changes to one or more *element*s, made to implement the feature or bugfix described in that issue record. Each entry in the change package describes changes to one element: the changes between the basis version and the head version. See *patch*.

change package dependency

A relationship between the *change package* of an *issue record* (A) and the change packages of one or more other issue records (B,C,D, ...), expressing the fact that *promote*'ing A would also cause some or all of the changes in B,C,D, ... to be promoted.

Change Palette

The AccuRev GUI tool that enables users to perform *merge* and *promote* operations involving any *stream*s, not just a *workspace* and its *parent stream*.

change section

In a text-file *merge* (or *patch*) operation, a location where the two *contributor*s being *merge*d differ from each other. The Merge tool highlights and counts the change sections. It also tracks

the conflicting changes (*conflict*s) — the subset of change sections in which both *contributor*s differ from the *closest common ancestor*. Conflicts must be resolved by human intervention. See *difference section*.

checkout

An operation that makes a file *active* in a *workspace*, without recording any new changes to the file in the *repository*. In an *exclusive file locking* or *anchor-required* workspace, a checkout transitions the file from read-only to writable.

checkpoint

Stopping to save a *version* of an *element*, then proceeding to make additional changes to the element.

client program

An AccuRev CLI or GUI program through which users submit commands to be executed by the *AccuRev Server*.

closest common ancestor

(of two *version*s of an *element*) The most recent version that is an *ancestor* of two specified versions. Used in a *merge* operation to minimize the amount of work required to combine the contents of the two specified versions. See *merge*, *version graph*.

coalesce

If a promote-by-issue operation (a standard child-to-parent promote, not a cross-promote) involves multiple issues whose change packages include the same element, AccuRev attempts to combine those entries into a single, valid change package entry. If the element's change package entries cannot be coalesced (caused, for example, by a "gap"), the promote operation fails.

If this occurs, proceed as described in section "Cross-Promoting Issues to a Non-Parent Stream — Patch Required" in *AccuRev Technical Notes*.

concurrent development

("parallel development") The practice of having two or more users concurrently work on the same project — modifying the same version-controlled *element*s. See *serial development*.

configuration

A set of *element* versions — one *version* of each element. Typically, the set of versions currently in a particular *workspace* or *stream*.

conflict

See *conflicting change*.

conflicting change

The situation in which both *contributor*s to a *merge* operation differ from the *closest common ancestor* at the same text line (or set of lines). Also, the situation in which both contributors have pathnames that differ from the closest common ancestor, and from each other.

container file

The ordinary file, located in the file storage area of the AccuRev *repository*, that contains the permanent copy of a *version* created in a *workspace* with the **keep** command.

content change

A change to the contents of a file *element*, recorded in a new *version* created with the **keep** command. For a symbolic-link element, a change to the target pathname is a content change. For an element-link element, a change to the target element is *not* considered a content change to the link. See *namespace change*.

contributor

Either of two *version*s of an *element*, which are to be combined in a *merge* operation, producing a new version of the element. This can involve both content changes and *namespace change*s.

cross-link

An include/exclude mode operation ("Include from Stream" or **incl –b**) that includes an *element* in a *workspace* or *stream*, specifying an alternative *backing stream* for that element. Cross-linking a directory also cross-links the entire subtree below it. Cross-linked elements have **(xlinked)** status.

cross-promotion

A *promote* operation that propagates one or more *version*s from a *dynamic stream* to another *stream* that is not its direct parent. See *parent stream*.

current change

See *current difference*.

current depot

CLI: the *depot* associated with the *workspace* that contains current working directory. GUI: the depot whose data appears in the currently visible GUI tab. The current depot's name is displayed in the status bar at the bottom of the GUI window.

current difference

The currently highlighted difference section (Diff tool) or change section (Merge tool).

current version

The version of an *element* that currently appears in a particular *workspace* or *stream*. (It's also possible that a given workspace/stream might not contain any version of a given element.)

The current version can be directly active in the workspace/stream; if not, it is inherited from the *parent stream*. See *passive*.

current workspace

The *workspace* whose data is displayed in the current tab; or the workspace from which the current tab was invoked.

cyclical

In a change-package dependency display, refers to the situation in which issue A depends on issue B *and* issue B depends on issue A.

deep overlap, deep underlap

An *overlap* or *underlap* that is not in the current *workspace* or *stream*, but in the *parent stream* or another stream higher up in the *stream hierarchy*.

default group

The set of *element*s that are currently active in a particular *workspace* or *stream*.

default query

An AccuWork query that you've designated to be executed automatically in certain situations: when you open a new Queries tab; when AccuRev prompts you to specify one or more *issue record*s in a **co** command; when you execute **promote** and an AccuRev/AccuWork integration is enabled.

defunct

A particular kind of change to an *element* in a *workspace* or *stream*: that the element is to be deleted. The element disappears from the workspace or stream. Somewhat counter-intuitively, it also becomes active in the workspace or stream, because defuncting is a change that can be *promote*d to the *parent stream* (or can be undone with a purge operation). A defunct operation is originally recorded as a new *version* of the element in some workspace. Promoting this version up the stream hierarchy causes the element to disappear from the higher-level streams.

dependency

See *change package dependency*.

depot

The portion of the AccuRev *repository* that stores the entire history of a particular directory tree. See *element*, *version*.

depot-relative pathname

A pathname that begins with /./ (Unix/Linux) or \.\ (Windows), indicating the path from the top-level directory of a *depot* to a particular *element*.

descendant

See *ancestor*.

diff

An operation that compares the contents of two *version*s of a text-file *element*.

difference section

In a text-file-comparison operation, a location where the two files (or two *version*s of the same file) differ from each other. The Diff tool highlights and counts the difference sections. See *change section*.

direct ancestor

See *predecessor*.

direct

An issue record is "in" a stream *indirectly* if its versions were propagated to the stream with a promote-by-issue operation (a cross-promote, not a standard child-to-parent promote).

Any other kind of promotion causes an issue record to be "in" a stream *directly*.

directory

("folder") A file system object that can contain files and other directories. Each *version* of a directory records a change to its name and/or pathname location in the *depot*'s directory hierarchy.

directory link

This term is no longer used. See *element link* and *symbolic link*.

double vision

The appearance of two or more *version*s of an *element* in the same *workspace* or *stream*, each version at a different pathname. This is a possible side-effect of *cross-link*ing the element (or a higher-level directory).

dynamic stream

A *stream* whose configuration changes over time, with new *version*s *promote*d from child *workspace*s and/or from other *dynamic stream*s. It also inherits versions from its *parent stream*.

edit-by-diff

The Diff tool feature enables you to edit your *workspace*'s *version* of an *element* while you're comparing it with another version.

edit form

(AccuWork) A fill-in-the-blanks form for displaying and changing the field values of *issue record*s.

EID

See *element-ID*.

element

A file or directory that is under AccuRev *version control*. See *version*.

element-ID

The unique, immutable integer identifier by which AccuRev tracks the changes to a particular file *element* or directory element. An element's name or pathname can change, but its element-ID never changes.

element link

(element-link element) An *element* whose contents is a pointer to another element, which must be in the same depot. The target element can be a *directory* element, a file element, another element link, or a *symbolic link*.

element type

The kind of data stored in *version*s of a file *element*. Different versions of the same element can have different element types. Three element types exist: text, ptext, and binary. Text and binary are relatively self-explanatory, but ptext is a special case. When AccuRev copies a text file from the repository to a workspace (such as through an **update** or **pop** command), it gives it line terminators appropriate for the machine where the workspace exists. Binaries are copied exactly as they exist in the repository. However, if you declare a text file to be a ptext file, it will be copied to and from the repository with no line termination changes, just like a binary. For more information, see the description of the **add** command in the *AccuRev CLI User's Guide*.

exclude rule

See *include rule*.

exclusive file locking

An AccuRev feature that enforces *serial development*: when a file becomes *active* in one *workspace*, an exclusive file lock prevents the file from becoming active in sibling workspaces.

executable bits

(Unix/Linux only) The data items in a file's access mode that controls the ability of users to invoke the file as an executable program.

external

A file or directory that is located within a *workspace tree* but has not been placed under *version control* has "external" status.

File Browser

The Explorer-like tool in the AccuRev GUI that shows the contents of a *workspace* or *stream*.

file link

This term is no longer used. See *element link* and *symbolic link*.

file storage area

The portion of a *depot* in which AccuRev maintains a permanent copy ("storage file") of each newly created file *version*. See *metadata*.

filter

Same as an AccuRev *search*. See also *stream filter*, *user/group filter*.

folder

("directory") A file system object that can contain files and other folders. Each *version* of a folder records a change to its name and/or pathname location in the *depot*'s folder hierarchy.

from version

One of the *contributor version*s in a *merge* operation. In a typical merge, it's the version in the *parent stream* that is to be combined with the version in the user's *workspace*.

gateway area

A directory with a *depot*'s slice, but outside the depot's *file storage area*, where *version* container files are staged for offline archiving. The gateway area is also used to restore archived versions' *storage file*s.

group

A named set of AccuRev *user*s. Each user can belong to multiple groups, and groups can be nested.

head version

The version of an *element* that, along with a basis version, specifies that element's entry in a *change package*. Equivalently, the head-version/basis-version pair specifies a patch to that element.

header section

(AccuWork) The section of a multiple-page edit form that always remains visible as you switch from page to page.

History Browser

The AccuRev GUI tool that displays the set of *transaction*s that affect a particular data structure: *depot*, *stream*, file, etc.

immutable

The "permanence" property of an AccuRev transaction: the transaction cannot be deleted or modified in any way.

include rule

User-defined *include rule*s and *exclude rule*s specify which *element*s are to appear in a given *stream* or *workspace*. Rules can apply to individual files or directories, or to entire directory

trees. Rules for a stream are inherited by its subsidiary streams and workspaces, but can be overridden at lower levels.

include/exclude

The facility of *stream*s and *workspace*s that enables users to specify which *element*s are to appear. See *include rule*.

incomplete

This term has the same meaning in a change-package dependency situation as in the Stream Issues tab: some, but not all, of an issue's change package entries are "in" the stream.

indicator

See *status indicator*.

indirect

See *direct*.

inherit

The facility by which *version*s in higher-level *stream*s automatically propagate to lower-level streams. If an *element* is not currently active in a stream or *workspace*, the stream/workspace inherits the version of the element that appears in its *parent stream*.

invisible

Describes a data structure that has been deactivated (**remove** command), and so does not appear in default GUI displays or CLI listings.

issue record

(AccuWork) A set of data, consisting of fields and values, which represents one AccuWork issue in the current *depot*. Each issue record implements a bug report, feature description, etc.

issue schema

(AccuWork) The set of specifications that define the structure of issue records in a *depot*: data fields and their value types/ranges, edit-form layout, field validations.

keep

The operation (**keep** command) that creates a new *version* of a file *element* in a *workspace*, permanently recording that version in the AccuRev *repository*.

kept

Refers to a *version* that has been created with a *keep* operation.

lock (dynamic stream)

A control on the ability to perform *promote* and *include/exclude* operations involving the *stream*.

lock (file element)

A control on the file *element*, requiring (1) users must *anchor* the file before editing it, and (2) if a user has anchored the file, users in sibling *workspace*s cannot anchor or edit the file.

login

Establishing a particular user identity ("username") with the **login** command. The username must have been created previously. AccuRev licenses specify a maximum number of currently-active usernames.

master repository

The primary data *repository* that logs all transaction activity processed by the AccuRev *master server*. All storage *depot*s are created in the master repository, from which they can later be replicated.

master server

The instance of the AccuRev Server process that handles all *transaction*s that change the status of *element*s in the master *repository*. Only the master server can write data to the repository. See *replica server*.

member

An *element* has member status in a *workspace* or *stream* if one of its *version*s is in the *default group* of that workspace or stream. An element with member status is said to be *active* in that workspace or stream; otherwise, it's *passive*.

merge

An operation that combines the contents of two *version*s (*contributor*s) of the same *element*. To merge the contents of text files, AccuRev uses a "3-way merge" algorithm: it compares the two files line-by-line with a third file, the version that is the *closest common ancestor* of the other two. Merging of *namespace change*s also takes into account the closest common ancestor.

metadata

Information stored in the AccuRev *repository* other than the contents of file *version*s. Metadata is stored in the *repository database*; file contents are stored in the *file storage area*.

modified

A file *element* has "modified" status in a *workspace* if the file's contents have changed since the last time the user kept a new *version* of the file or *update*d the entire workspace.

multiple-columns mode

The mode of a table displayed by the AccuRev GUI in which you can define a hierarchical sort order for the rows, using the values in two or more of the table's columns. See *single-column mode*.

namespace change

A change to the pathname of a file or directory *element*: either renaming the element in place or moving the element to a different location in the *depot*'s directory hierarchy.

non-conflicting change

In a *merge* operation, a change that occurs in just one *contributor* (not both of them). Such a change can be merged automatically, without requiring a decision from the user.

optimization

A heuristic algorithm that AccuRev uses to speed the performance of certain operations on users' *workspace*s. In the *timestamp optimization*, AccuRev ignores files created/modified before the workspace's most recent scan threshold. In the pathname optimization, AccuRev ignores files whose pathnames match a pattern specified in environment variable ACCUREV_IGNORE_ELEMS. See *search*.

overlap

Version X, in a *workspace* or *stream*, has "overlap" status if the *parent stream*'s current *version* of the *element* contains changes that are not reflected in version X. (That is, the parent stream's version is not an ancestor of version X.) Such a version cannot be *promote*d to the parent stream; the user must create a new version with a *merge* operation, combining version X with the parent stream's version. The new, merged version can then be *promote*d. Similarly, an overlap can exist between the versions in two dynamic streams. See *deep overlap, deep underlap*, *underlap*.

parallel development

("concurrent development") The practice of having two or more users concurrently work on the same project — modifying the same version-controlled *element*s. See *serial development*.

parent stream

("backing stream", "basis stream") The *stream* that is just above a given *workspace* or stream in a *depot*'s *stream hierarchy*. The given workspace/stream inherits *version*s from the parent stream.

pass-through stream

When a *version* is nominally *promote*d to pass-through stream X, the version automatically "passes through" X: it is actually *promote*d to the *parent stream*.

passive

An *element* that is not active in a *workspace* or *stream* is said to be passive in that workspace or stream. Passive *version*s can be overwritten by an *update* operation.

patch

A set of *version*s of a text-file *element* -- typically, containing the "recent changes" made in one *workspace*. Also, the *merge*-like operation that incorporates those changes into another

version of the same element. See *merge*, *basis version*, *head version*, *change package*, *reverse patch*.

pathname optimization

One of AccuRev's optimizations, which improves the performance of *workspace* searches to determine the status of *element*s.

pending

An *element* has "pending" status in a *workspace* if the *version* in the workspace has changes that have not yet been *promote*d to the *parent stream*. The set of pending elements includes both *kept* elements and *modified* elements.

permission

See *ACL*.

predecessor

(direct ancestor) The *real version* from which a given *version* was derived. A version and its predecessor are not necessarily located in the same *workspace stream*. In the Version Browser, a version and its predecessor are connected by a black line. (Exception: a version created by the **revert** command is connected to its predecessor by a dashed blue line.)

preferences file

An XML-format file, named **preferences.xml**, stored in the **.accurev** subdirectory of your AccuRev home directory.

principal-name

The username of an AccuRev *user*, recorded in the AccuRev *repository*.

private query

(AccuWork) A *query* that appears in the Queries pane only for the user who created it. See *public query*.

promote

The operation (**promote** command) that transitions a *version* from being active in one *workspace* or *stream* to being active in the *parent stream* (or some other stream). This operation creates a new *virtual version* in the parent stream; the virtual version provides an alias for the *real version*, which was originally created in some user's *workspace*. See *version*.

ptext

See *element type*.

public query

(AccuWork) A query that appears in the Queries pane for all *user*s. See *private query*.

purge

The operation (CLI: **purge** command; GUI: **Revert to Backed** command) that discards the changes made to an *element* in a given *workspace* or *stream*.

query

A set of search criteria that selects AccuWork issue records, based on the records' field values. Within each depot containing issue records, one of the queries designated as the *default query*, to be invoked automatically in certain situations calling for the user to specify one or more *issue record*s.

real version

A *version* of an *element*, created in some user's *workspace*, recording a change to the contents, type, and/or pathname of the element. See *version*, *virtual version*.

recent changes

A set of *version*s of a particular file *element*; representing the changes made to accomplish some task (or any set of related changes). The recent changes start with the current version (or another selected version), termed the *head version*; they extended backward to (but do not include) the corresponding *basis version*.

reclaim

An operation that deletes archived *storage file* from a *depot*'s gateway area, to reduce the amount of disk storage required for the depot's *slice*.

reference tree

A directory tree in users' disk storage that instantiates a particular *dynamic stream* or *snapshot*. It contains a copy of the current *version* of each *element* in the stream or *snapshot*. A reference tree based on a dynamic stream can be updated, to incorporate the stream's recent changes.

reparent

The operation that changes the *parent stream* of a particular *workspace* or *stream*.

replica repository

A copy of part or all of the contents of the *master repository* that must be resynchronized regularly to remain current. New transaction records are written to the master repository only, making resynchronization necessary.

replica server

The instance of the *AccuRev Server* process that is associated with a *replica repository* on the same machine. It can directly service *client program*s' repository-read requests, but forwards repository-write requests to the *master server*.

repository

The directory tree and database, which together store all software configuration management data managed by AccuRev. This data is maintained by the AccuRev Server, responding to requests made through AccuRev client programs. Users never manipulate the repository directly.

repository database

The portion of the *repository* that stores data other than the contents of *element version*s. See *file storage area*.

reverse patch

An operation that removes a selected set of changes from the current *version* of a text-file *element*. See patch, *change package*.

revert

An operation that "removes" a selected set of changes from a specified *version*, by creating a new version that does not contain the change.

root stream

("base stream") The top-level *stream* in a *depot*'s stream hierarchy.

scan threshold

The time at which a *workspace*'s most recent *search* for *modified* files was initiated. Such searches are performed by the **update** and **files** commands, and by certain forms of the **stat** command. In the GUI, several of the File Browser searches include a search for a workspace's modified files. See *update level*.

SCM

Acronym for software configuration management.

search

An operation that determines all the *element*s in a *workspace* or *stream* that have a particular *status*.

serial development

The practice of ensuring that multiple *user*s do not work concurrently on the same file under *version control*. See *parallel development*.

server

See *AccuRev Server*.

session file

A file in the **.accurev** subdirectory of your home directory, which establishes your user identity for a particular AccuRev Server.

sibling

Two or more *workspace*s or *stream*s that have the same *parent stream*. Pass-through streams "don't count" -- that is, all workspaces that *promote version*s to the same stream are considered siblings, even if some of them are direct children of the stream, while others are children of an intervening pass-through stream.

single-column mode

The mode of a table displayed by the AccuRev GUI in which the rows are sorted on the values in one of the table's columns. See *multiple-columns mode*.

site slice

The subdirectory tree within the AccuRev *repository* that stores repository-wide information (such as AccuWork configuration data, workflow configuration data, server preferences, and triggers). By default, the top-level directory of this subtree is named **site_slice**. See *repository database*.

slice

The subdirectory tree that contains the depot's *file storage area*. By default, the top-level directory of this subtree has the same name as the depot itself.

snapshot

An immutable ("frozen", "static") *stream* that captures the *configuration* of another stream at a particular time. A snapshot cannot be renamed or modified in any way.

stage

See *workflow stage*.

stale

An *element* has "stale" status in a *workspace* if it is not currently *active* in the workspace, but a new *version* of the element has entered the *parent stream*. An update operation will overwrite the stale version with the parent stream's new version.

static stream

An AccuRev *snapshot*. The term "static stream" emphasizes the fact that snapshots are part of a *depot*'s *stream hierarchy*.

status

The state of an *element*, from a *version control* perspective, in a particular *workspace* or *stream*.

status indicator

A keyword (usually enclosed in parentheses) that reports the AccuRev-level *status* of a particular *element* in a particular *workspace* or *stream*. Commonly, multiple status indicators apply to an element.

storage file

See *container file*.

stranded

An *element* has **(stranded)** status in a *workspace* or *stream* if it's currently *active*, but cannot be accessed through the file system. This can occur in several situations:

- There is no pathname to the element, because the element's directory (or a higher-level directory) was removed from the workspace or stream by the *defunct* command or an *exclude rule*.
- (*dynamic stream* only) There are one or more defunct elements at a given pathname, along with one non-defunct element. The defunct element(s) have **(stranded)** status.
- The element's directory (or a higher-level directory) is *cross-link*'ed, making another version appear at the pathname of the active version.

stream

The AccuRev data structure that implements a *configuration* of the *element*s in a particular *depot*. The configuration of a *dynamic stream* changes over time; the configuration of a *snapshot* (static stream) never changes. Each *workspace* has its own private *workspace stream*. See *workspace*, *stream hierarchy*, *stream path*.

stream-ID

An integer that uniquely identifies a *stream*, *snapshot*, or *workspace* with its *depot*. Changing the name of a stream or workspace does not affect its stream-ID.

stream filter

A list of *stream*s, which is used to define a subset of a depot's *stream hierarchy* that includes the *stream path* and all children of each stream in the filter. This subset is used to build the StreamBrowser display and populate lists of streams in the AccuRev GUI as long as stream filtering is in effect.

stream hierarchy

The tree-structured collection of *stream*s — including *snapshot*s and *workspace stream*s — for a particular *depot*.

stream lock

See *lock (dynamic stream)*.

stream path

A sequence of streams that starts at the *root stream* and ends at the stream being referenced. See also *stream*.

StreamBrowser

The GUI tool that provides both graphical and tabular view of a *depot*'s *stream* hierarchy. It has commands for comparing streams, *promote*'ing *version*s between streams, and other stream-based operations.

symbolic link

(symbolic-link element) An *element* whose contents is a pathname. The pathname can point to AccuRev data (that is, a location inside a workspace) or non-AccuRev data.

target

element link, *symbolic link*: the file system location that the link points to.

target transaction

The most recent *transaction* at the time of a *workspace*'s most recent update. The **update** command attempts to load *version*s created in transactions up to and including the target transaction.

text

See *element type*.

time-based stream

A *stream* that has a *basis time*, affecting which *version*s it inherits from its *parent stream*. Unlike a *snapshot*, the basis time can be changed.

time-spec

A specification of a particular date/time combination, used in various contexts: creating *snapshot*s, viewing portions of the history of an *element*, etc.

TimeSafe

The aspect of AccuRev's architecture that guarantees the reproducibility of any previous configuration of a *stream*, a *depot*, or the entire *repository*.

timestamp optimization

One of AccuRev's *optimization*s, which improves the performance of *workspace* searches to determine the status of *element*s.

timewarp

A situation in which the discrepancy between a client machine's system clock and the *AccuRev Server* machine's system clock exceeds the allowable limit.

to version

One of the *contributor version*s in a *merge* operation. In a typical merge, it's the version in a user's *workspace* that is to be combined with the version in the *parent stream*.

transaction

A record in the AccuRev *repository database* that indicates a particular change: promoting of a set of *version*s, changing the name of a *stream*, modification to an *issue record*, etc. Each transaction has an integer transaction number, which is unique within the *depot*.

transaction-level integration

The AccuRev facility that records the *transaction* number of a *promote* operation in a user-specified AccuWork *issue record*. This facility is enabled on a *depot*-by-depot basis by a *trigger*.

transaction history

The set of transactions related to a particular *depot*, *stream*, *element*, or other AccuRev data structure that changes over time.

transaction level

The number of the most recently completed *transaction* for a particular *depot*. See *update level*.

transition

See *workflow transition*.

trigger

The AccuRev facility that enables user-defined procedures (trigger scripts) to be performed automatically before or after certain operations take place.

trigger script

The executable program that implements a user-defined procedure, to be invoked when a *trigger* fires. Also called a trigger program.

twins, evil twins

Two or more *element*s with the same pathname in a *dynamic stream*. Example: (1) an element is defuncted in a *workspace*, (2) the element is *promote*'d to the *backing stream*, (3) another element is created at the same pathname in the same workspace or a *sibling* workspace, (4) the new element is promoted to the backing stream.

undefunct

The operation (**undefunct** command) that undoes the effect of a previous *defunct* operation, restoring a previously removed *element* back to a *workspace*. (The element must then be *promote*'d to be visible to other streams.)

underlap

Similar to *overlap*: for both underlap and overlap, the version in the *parent stream* is not an ancestor of your version. With an underlap (but not an overlap), your version is an ancestor of the parent stream's version; that is, the parent-stream *version* already contains all the changes

in your version. Deep underlaps can occur in the stream hierarchy, just like deep overlaps. See *deep overlap, deep underlap*.

update

The operation (**update** command) that copies new *version*s of *element*s into a *workspace* from its *parent stream*.

update level

The most recent (highest-numbered) *transaction* whose changes have been incorporated into a *workspace*, through an *update* operation. See *scan threshold*, *transaction level*, *target transaction*.

user

A person who uses an AccuRev client program to access (read and/or change) the data in the AccuRev *repository*. Access is granted only to those who *login* with a "username" that was previously registered in the AccuRev repository.

user/group filter

A subset of AccuRev *user*s or *group*s, which is used to limit the data displayed in parts of the AccuRev GUI.

validation

(AccuWork) A rule, specified on the Validations subtab of the Schema Editor, that controls a particular edit-form field. This can take various forms, including specifying a default value, making a field required, and modifying the list of choices in a multiple-choice listbox. See *issue schema*.

version

A particular revision of an *element*, reflecting a content change (files only) or a *namespace change* (files and directories). All versions are originally created in *workspace*s, and can subsequently be *promote*d to *dynamic stream*s. The original (workspace) version is termed a *real version*. Each promotion to a dynamic stream creates a *virtual version*, which serves as an alias for (pointer to) the original real version.

version-ID

The unique identifier for a *version*, consisting of two parts: (1) the name or number of the *workspace* or *stream* in which the version was created; (2) an integer. Examples: **talon_dvt_mary/14**, **245\19**.

Version Browser

The AccuRev GUI tool that displays the *version graph* of an *element*.

version control

The discipline of keeping track of the changes made over time to a file or directory.

version graph

The directed-graph data structure that shows the *ancestry* of an *element*. The nodes are all the *version*s of an element, and whose lines indicate how later versions were derived from earlier versions. The Version Browser displays the *version graph* of an element.

version specification

Identifies a particular *version* of one or more *element*s. It can be a *version-ID*; in many contexts, it can be a *stream* or *workspace* name/number, which indicates the version currently in that stream/workspace.

version tools

AccuRev GUI tools that provide access to historical *version*s of *element*s. The Version Browser provides easy access to all versions of an element. The History Browser provides access to versions through the *transaction*s in which they were created. The Stream Version Browser provides easy access to the version that currently appears in a given *stream*.

virtual version

In a *dynamic stream*, a *version* of an *element*, created by the **promote** command, which serves as an alias for (reference to) a previously created *real version*. In a *workspace stream*, a version created by the **co** or **anchor** command, referring to the real version that the command "checked out". See *checkout*.

workflow

A directed graph, defined in the Workflow subtab of the AccuWork Schema Editor. The graph's nodes are the *workflow stage*s that an AccuWork issue record can pass through. The graph's arrows are the *workflow transition*s that users invoke to migrate issue records from stage to stage.

workflow query

An AccuWork *query*, automatically composed in the Stream Browser or a Queries tab, and then executed in order to determine which *issue record*s are "in" a particular *workflow stage*.

workflow stage

A node in an AccuWork *workflow*, representing one of the steps in the "lifetime" of an *issue record*. See *workflow transition*.

workflow transition

An arrow in an AccuWork *workflow*, pointing to a particular *workflow stage*. This represents one step that an *issue record* can take through the workflow. A transition has two components: a transition action (such as "Finish Dvt") and a workflow stage (such as "Implemented") that is the arrow's destination. Each workflow transition can be configured to start from any number of stages.

workspace

A location in which one or more *user*s perform their work, using files under *version control*. Each workspace consists of a *workspace stream* in the *repository* and a *workspace tree* in the user's disk storage.

workspace name

The name by which users refer to a *workspace*. A workspace name always ends with **_<principal_name>**, to indicate the user who own its. The default workspace naming convention, used by the **mkws** command, is **<backing_stream_name>_<principal_name>**.

workspace stream

The private *stream* that is built into a *workspace*. All new *version*s of *element*s are originally created in workspaces; AccuRev records these versions in *workspace stream*s.

workspace tree

The ordinary directory tree, located in the *user*'s disk storage, in which the user performs development tasks and executes AccuRev commands.