



The TimeSafe[®] Configuration Management System

AccuRev Administrator's Guide

Version 4.0.1

February, 2006

AccuRev Administrator's Guide

February, 2006

Copyright © AccuRev, Inc. 1995–2006
ALL RIGHTS RESERVED

TimeSafe and **AccuRev** are registered trademarks of AccuRev, Inc.

AccuBridge, **AccuReplica**, **AccuWork**, and **StreamBrowser** are trademarks of AccuRev, Inc.

All other trade names, trademarks, and service marks used in this document are the property of their respective owners.

Table of Contents

The AccuRev Repository	1
Repository Access Permissions	1
READ ME NOW: Assuring the Integrity of the AccuRev Repository.....	1
Backing Up the Repository	3
Restoring the Repository.....	4
Archiving Portions of the Repository	4
Moving a Workspace or Reference Tree.....	4
Moving a Depot	5
Removing a Depot	5
A Word of Caution on Windows Zip Utilities	5
Storage Layout	5
The AccuRev Server	7
User Identity of the Server Process.....	7
Unix Systems Only: Administrative User Identities, 7	
Starting the AccuRev Server.....	8
Running the Server Automatically at Operating System Startup, 8	
Starting the Server Manually, 8	
Server Configuration File.....	8
Controlling Server Log Verbosity, 9	
Unix Systems Only: Controlling the Server's User Identity, 9	
Controlling Multithreading of the AccuRev Server, 9	
Server Watchdog	10
Server Logging.....	10
Watchdog Logging, 11	
Verbose Logging, 11	
Verbose Log Messages for Failed Commands, 12	
Controlling Server Operation.....	12
Unix: 'acservctl' Utility, 12	
Windows: 'Services' Console, 14	
Server-Control Files, 14	
Open Filehandle Limits and the AccuRev Server.....	15
Changing the Per-Process Open File Descriptor Limit, 16	
Linux, 16	
Solaris, 17	
HP-UX, 17	
Server Subtask Codes, 19	
Archiving of Version Container Files	21
The 'archive' Command	22
Determining Which Versions to Archive, 22	
Archiving the Versions, 22	
The 'reclaim' Command.....	23
Attempts to Access Archived Versions.....	24
Using 'hist' to Research Previous 'archive' Commands	24

Restoring Archived Versions — The ‘unarchive’ Command24

Replication of the AccuRev Repository27

Master and Replica.....27
AccuRev Licensing in a Replication Environment.....29
Installation Procedure: Assumptions29
Setting Up the Master Server29
Setting Up the Replica Server.....30
 Install AccuRev, 30
 Revise the Server Configuration File, 30
 Start the AccuRev Server Process, 31
 Synchronize the Site Slice, 31
 Configure the Replica Server to Include the Desired Depots, 31
Using the Same Host as Both Master Server and Replica Server.....32
Setting Up a Client Machine to Use a Replica Server32
Using a Replica Server.....32
 The Update Command, 33
Creating New Depots33
Adding and Removing Depots from a Replica Repository.....33
Synchronizing a Replica Manually34
 On-Demand Downloading of a Version’s Storage File, 34
Automating Replica Synchronization34
Synchronization Security35
The replica_site.xml File.....35

Moving the AccuRev Server and Repository to Another Machine .37

Procedure for Moving the Repository.....37
On the Destination Machine37
On the Source Machine.....38
On the Destination Machine39

AccuRev Triggers41

Pre-Operation Triggers.....41
 Client-Side Triggers, 41
 Server-Side Triggers, 41
Post-Operation Triggers42
Transaction-Level Integration Trigger42
Preparing to Use an AccuRev-Provided Trigger Script.....43
Enabling a Trigger.....44
 pre-create-trig, pre-keep-trig, pre-promote-trig, server-post-promote-trig, 44
 server_admin_trig, 44
 server_preop_trig, 44
 server_dispatch_post, 45
 Notes on Triggers in Multiple-Platform Environments, 45
The Trigger Parameters File.....45
 Format of the “pre-create-trig” Trigger Parameters File, 46
 Overwriting the ‘pre-create-trig’ Trigger Parameters File, 47
 Format of the “pre-keep-trig” Trigger Parameters File, 48
 Format of the “pre-promote-trig” Trigger Parameters File, 49

Overwriting the ‘pre-promote-trig’ Trigger Parameters File, 50	
Manipulating the ‘promote’ Comment String, 51	
Format of the “server-post-promote-trig” Trigger Parameters File, 51	
Format of the “server_preop_trig” Trigger Parameters File, 52	
Format of the “server_admin_trig” Trigger Parameters File, 53	
Format of the “server_dispatch_post” Trigger Parameters File, 54	
Encoding of Element Lists, 54	
Encoding of Command Comments, 55	
Trigger Script Contents, 55	
Trigger Script Exit Status, 56	
File Handling by Trigger Scripts, 56	
Trigger Script Execution and User Identities.....	57
‘Administrative Users’ in Trigger Scripts, 57	
The Trigger Log File.....	57
The ‘maintain’ Utility	59
‘maintain’ Command Reference	59
Backup/Restore of the AccuRev Repository	61
Removing a Depot from the AccuRev Repository	62
Before You Begin, 62	
Depot Removal Procedure, 62	

The AccuRev Repository

The AccuRev Server program manages a data repository, which provides long-term storage for your organization's development data — for example, all versions of all source files. (Among other things, AccuRev is a special-purpose database management system; the files in the repository — thousands of them — are part of this database.) By default, the repository resides in subdirectory **storage** of the AccuRev installation directory. The repository consists of:

- **site_slice** directory: implements a database that contains a user registry, list of depots, list of workspaces, and other repository-wide information.
- **depots** directory: contains a set of subdirectories, each storing an individual depot. A depot subdirectory stores one or both of:
 - A version-controlled directory tree: all the versions of a set of files and directories, along with a database that keeps track of the versions.
 - A database of AccuWork issue records.

When it starts, the Server program determines the location of the **site_slice** directory by looking at the `SITE_SLICE_LOC` setting in configuration file **acserver.cnf**. This file must reside in the same directory as the Server program (**accurev_server**) itself.

Repository Access Permissions

The user identity of the AccuRev server process — **acserver** (Unix) or **System** (Windows) — must have full access to all the files and directories within the data repository. For maximum security, this should be the *only* user identity with permission to access the repository. The only exception to this might be an **acadmin** AccuRev administrator account, as suggested in *Unix Systems Only: Administrative User Identities* on page 7.

This user identity must also have access to the **bin** directory where the AccuRev executables are stored.

READ ME NOW: Assuring the Integrity of the AccuRev Repository

The integrity of the AccuRev data repository is critically important. If information in the repository is lost or corrupted, your organization's ability to do business may be severely compromised. The integrity of the data repository relies on the integrity of underlying software (the file system, including the device drivers for data storage devices) and underlying hardware (the data storage devices themselves). Certain practices will enhance the safety and reliability of these underlying facilities. We strongly recommend the following:

- Use high-quality disk drives and disk controllers.
- Reduce the impact of a hard-disk failure by using disk mirroring (for example, using a RAID system) or other fault-tolerant disk subsystems.

- Power the AccuRev server machine with an uninterruptible power supply (UPS), with automatic shutdown of the server machine if the UPS is running out of power. This reduces the likelihood of interrupted data transfers to disk.
- Establish a good data-backup regimen, and make sure your backups are reliable by doing test restores on a regular basis. (See *Backing Up the Repository* on page 3.)

This section focuses on one aspect of data integrity: guaranteeing “write” operations to the repository. The AccuRev Server process does not, itself, perform the act of writing data on the disk. Like all application programs, it makes a “write” request to the operating system (Unix, Windows). In turn, the operating system performs a “write” operation to the disk itself. (On some larger systems, there may be additional links in this chain of write operations.)

Operating systems and disk subsystems often use special techniques that boost the performance of write operations, but can compromise data integrity. For example, when an application program makes a write request, the operating system might:

- Acknowledge the request immediately — good, because the application program can then proceed to its next operation.
- Delay actually sending the data to the disk (“write-behind”) — bad, because a system failure at this point might result in the data never being stored on the disk.

It is essential that such techniques *not* be used when the AccuRev Server process sends information to the disk containing the AccuRev data repository. The Server always follows each write request with a “synchronize the disk” request. Sometimes, this ensures that data is safely on disk before the Server proceeds to its next task. For example, this is typically the case if the repository is stored on a disk that is local to the machine on which the Server is executing.

But in some situations delayed-write techniques may be used even when the AccuRev Server makes “synchronize the disk” requests. This is typically the case if the repository is located on a network shared file system. In such situations, the Server’s “synchronize the disk” requests are effectively ignored, so that successful completion of write operations to the AccuRev repository cannot be guaranteed. (Some disk subsystems implement such a guarantee by having their own battery backup; buffered data is flushed to disk when the power fails.)

In an attempt to avoid such unsafe situations, the AccuRev Server process attempts to determine whether the file system where the repository is stored guarantees the successful completion of write operations. If it decides “no”, the Server refuses to use the repository. This determination is not foolproof — both “false positives” and “false negatives” are possible.

There’s a workaround in the “false negative” case — where the AccuRev Server process decides that the file system does not guarantee write operations, but *you* know that writes are, in fact, guaranteed. In this case, set environment variable `AC_FS_WRITE_GUARANTEED` to the value 1 in the environment in which the Server process runs; then restart the Server process.

If you have any question about the safety of your data-storage system, please contact AccuRev Support.

Backing Up the Repository

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

AccuRev supports live backup of the data repository: making copies of the data repository files while the AccuRev Server is running. The **backup** command takes just a few seconds to make checkpoint copies of certain **site_slice** files in a subdirectory named **backup**. It also records a “high water mark” file, **valid_sizes_backup**, in each depot directory, noting the depot’s current transaction level. Transactions that are underway at the time the **backup** command executes are not included.

During **backup** command execution, clients can continue to work, but may notice a slight delay: transactions arriving at the AccuRev Server are queued for execution after completion of the **backup** command.

After executing the **backup** command, you can make a complete copy of the repository (the **storage** directory tree), without worrying about synchronization or time-skew. The append-only nature of AccuRev’s databases makes this simple scheme possible. No matter when you make the backup copies, you’ll be able to restore the repository to its state at the time you executed the **backup** command.

Note: the live-backup scheme relies on the ability to copy files that are currently open to the AccuRev Server process. Your backup utility must be able to copy files that are currently open at the operating system level. If you have any doubts or questions, contact AccuRev support.

Thus, the repository backup procedure is:

1. “Checkpoint” the database portion of the repository:

```
accurev backup mark
```

2. If your backup utility cannot copy files that are currently open at the operating system level, stop the **accurev_server** program. (See *Controlling Server Operation* on page 12.)
3. Use standard operating system backup/restore tools to create a backup copy of the entire directory tree below the **storage** directory. This backup can be all-at-once or piecemeal; for example, you can back up the **site_slice** directory and the individual subdirectories within the **depots** directory with separate commands.

Good candidates for backup/restore tools are **tar** (Unix), **xcopy /s** (Windows), and **zip** (both).

Note: if your site slice is in a non-standard location (as specified by the **SITE_SLICE_LOC** setting in the **acserver.cnf** file — see *Server Configuration File* on page 8), or if some depots are in non-standard locations (perhaps moved with the **chslice** command), then your job in backing up the entire repository is more complicated than simply to copy the **storage** directory.

4. If you stopped the **accurev_server** program in Step 2, start it again. (See *Controlling Server Operation* on page 12.)

Restoring the Repository

If you have backed up the repository according to the directions above, you can easily restore the repository to the time at which you executed the **backup** command:

1. Stop the **accurev_server** program. (See *Controlling Server Operation* on page 12.)
2. Restore the backup copies of the **site_slice** and individual depot directories, using the standard backup/restore tools.
3. Go to the **site_slice** directory.
4. Overwrite database files with the checkpoint files in the **backup** subdirectory:

```
cp backup/* . (Unix)
```

```
copy backup\*. * . (Windows)
```

5. Reindex the site slice:

```
maintain reindex
```

6. Restore and reindex each depot:

```
maintain restore <depot-name>
```

```
maintain reindex <depot-name>
```

7. Restart the **accurev_server** program. (See *Controlling Server Operation* on page 12.)

Note: suppose a particular depot's files were not backed up for several hours after the **backup** command was executed. During that interval, several new versions of file **gizmo.c** were created with the **keep** command. All of those versions will officially be lost when the repository is restored to its state at the time the **backup** command was executed. But you can still retrieve a copy of the last-created lost version of file **gizmo.c** from the backup medium.

Archiving Portions of the Repository

The container files that store the contents of individual file versions can now be move to offline storage, in order to save online storage space for the repository. For details, see *Archiving of Version Container Files* on page 21.

Moving a Workspace or Reference Tree

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

First, move the physical contents of the workspace tree or reference tree with standard operating system tools (e.g. **tar**, **zip**, **xcopy /s**). Then, let AccuRev know about the move:

```
accurev chws -w <workspace-name> -l <new-location>
```

```
accurev chref -r <reftree-name> -l <new-location>
```

Moving a Depot

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

First, move the physical contents of the depot with standard operating system tools (e.g. **tar**, **zip**, **xcopy** /s). Then, let AccuRev know about the move with this command:

```
accurev chslice -s <slice-number> -l <new-location>
```

(Use **accurev show depots** to determine the slice number of the depot.)

Removing a Depot

A depot can be removed completely from the repository with the **maintain rmdepot** command. This operation is irreversible! For details, see *Removing a Depot from the AccuRev Repository* on page 62.

A Word of Caution on Windows Zip Utilities

Be careful when using WinZip[®] or PKZIP[®] on a Windows machine to perform the tasks described above: backup/restore of the entire repository, or moving a workspace, reference tree, or depot. You may want to use **tar** on a Unix machine to “pack up” a directory tree, and then use the Zip utility on a Windows machine to “unpack” it.

- When moving the entire repository or an individual depot, be sure to disable conversion of line-terminators during the “unpack” step:
 - In WinZip, make sure the option “TAR file smart CR/LF conversion” is not selected (**Options > Configuration > Miscellaneous**).
 - In PKZIP, make sure the “CR/LF conversion” setting is “None -- No conversion” (**Options > Extract**).

Enabling conversion of line-terminators during the “unpack” step will corrupt the text files in a depot's file storage area (see *File Storage Area* below). The AccuRev Server always expects lines in these text files to be terminated with a single LF character, no matter what kind of machine the server is running on.

- Conversely, when moving a workspace or reference tree, you may wish to enable “TAR file smart CR/LF conversion”. The files in a workspace or reference tree are handled directly by text-editors, compilers, testing tools, etc. Many Windows text-editors are incapable of handling text files whose lines are terminated with a single LF character.

Storage Layout

Each AccuRev depot is stored in a separate directory tree under the installation area's **storage** directory. The **storage** directory is a sibling of the executables (“bin”) directory. For example, if AccuRev is installed at **/usr/accurev** and depots named **moe**, **larry**, and **curly** are created, the

directory layout would be:

```
/usr/accurev
  bin
  storage
    site_slice
    depots
      moe
      larry
      curly
```

A depot consists of three parts:

Configuration Files

The **mktrig** command creates a one-line configuration file that names the script to be executed when the trigger fires for transactions involving this particular depot. For example, making a trigger of type “pre-keep-trig” creates a configuration file in the depot named **pre-keep-trig**. (This file might contain the pathname **/usr/local/bin/accurev_prekeep.pl**.)

Metadata Area

The metadata area stores information about versions, times, and source file storage locations within the source file storage area of the depot. The metadata is stored in files ending with **.ndb** and **.ndx**.

The metadata area must be physically located on the machine where **accurev_server** is running. This guarantees the integrity of physical disk writes. Moving the metadata area to a remote file system compromises data integrity and is not supported by AccuRev, Inc.

File Storage Area

Whenever a user creates a new real version of a file with the **keep** command, the AccuRev Server copies the file from the user’s workspace to the depot’s file storage area. The newly created storage file is permanently associated with the real version-ID in the workspace stream (e.g. 25/13), and also with subsequently created virtual version-IDs in higher-level streams (7/4, 3/9, 1/4).

Storage files are located in subdirectory tree **data** within the depot directory. The files may be in compressed or uncompressed form. Compressed files may correspond to more than one real version. Conceptually, storage files are numbered sequentially starting with 1 and going up to 2**64. (That should be enough.) Within the **data** directory, they’re arranged in a hierarchy for faster access. For example, storage file #123456 would be stored as **data/12/34/56.sto**. Recovery information for the storage file is stored in a like-named **.rrf** file (e.g. **data/12/34/56.rrf**).

You can relocate a depot’s file storage area onto other disk partitions or even onto remote disks. The cautions about storing data locally do not apply to files in the data directories. However, exercise extreme caution when relocating storage in this area. Make sure you have first done a full backup and have shut down the **accurev_server** program.

The AccuRev Server

The AccuRev data repository is managed by a single program, the AccuRev Server (**accurev_server**). This program must be started prior to running any AccuRev client commands. The server program should be the only process that directly manipulates the AccuRev repository. No person should attempt to work directly with the repository, unless it is an emergency.

User Identity of the Server Process

The AccuRev Server process, named **accurev_server**, has a user identity at the operating system level. This process should run with a special user identity, to help ensure that no other user or process modifies the data repository:

- Unix: create a user named **acserver**.

You may be tempted to simply let the AccuRev Server process run as **root**. But we strongly recommend against this, as it would open a large security hole. The Server can run user-supplied trigger scripts (see *AccuRev Triggers* on page 41). In general, having user-supplied scripts run as the **root** user is very dangerous!

- Windows: don't create a new user; the AccuRev Server process runs as the built-in local user named **System**.

This user identity must have access to the AccuRev executables (**bin**) directory and to the data repository (see *Repository Access Permissions* on page 1).

Note: **acserver** and **System** are user accounts at the operating system level. You don't need to — and should not — create a principal-name (AccuRev user name) “acserver” or “System”. On the other hand, the AccuRev Server might need to take on an AccuRev user identity when it executes a server-side trigger script. For more on this topic, see *Trigger Script Execution and User Identities* on page 57.

Unix Systems Only: Administrative User Identities

You can control the user identity under which the Server runs with the server configuration file. See *Server Configuration File* on page 8.

Not even Unix administrators should run as **acserver** unless it is an emergency. If you want to have an AccuRev administrator account, we recommend creating a separate account named **acadmin**. Place both **acserver** and **acadmin** in a group named **acgroup**, and record these names in the server configuration file:

```
USER = acserver
GROUP = acgroup
```

With this setup, the **acadmin** user will be able to access the repository. You can configure Unix-level auditing and place other appropriate controls on this account; this leaves the **acserver** account (and thus, the AccuRev Server process) unencumbered by such controls.

Starting the AccuRev Server

The following sections describe how to start the AccuRev Server program, either automatically at operating system bootstrap time, or manually at a command prompt. (You can also perform a “manual” startup with a Unix shell script or a Windows batch file.)

Running the Server Automatically at Operating System Startup

Typically, the Server program is started automatically when the operating system boots on the server machine. On Unix systems, an “rc” or “init.d” startup script starts the **accurev_server** program. The AccuRev installation program does not install the startup script automatically, however. You must customize and install the sample startup script located in the **extras/unix_init** subdirectory of the AccuRev installation directory. See the **README** file for complete instructions.

On Windows NT/2000/XP systems, the AccuRev installation program automatically configures the **accurev_server.exe** program as a Windows service. Use the standard **Services** applet on the Windows Control Panel to control the Server program.

Starting the Server Manually

The AccuRev Server must be started manually in the following environments:

On Windows 95/98/ME systems

Start the Server with the **AccuRev Demo Server** shortcut on the desktop. Alternatively, run the **server_start.bat** script, located in the AccuRev executables (**bin**) directory.

On Windows NT/2000/XP systems

If you’ve changed the startup type of the AccuRev service to “Manual”, you can start the service from the **Services** applet. Alternatively, run the **server_start.bat** script, located in the AccuRev executables (**bin**) directory.

On Unix systems

Start the Server with the **acserverctl** utility:

```
<AccuRev-executables-dir>/acserverctl start
```

Server Configuration File

When it starts, the Server program reads configuration file **acserver.cnf**, located in the AccuRev executables directory. This configuration file is generated during installation, but can be edited manually thereafter.

Here is a sample **acserver.cnf**:

```
MASTER_SERVER = accurev_server_machine.company.com
SITE_SLICE_LOC = /partition0/site_slice
```

IMPORTANT NOTE: the white space surrounding the equals sign (=) in configuration files is mandatory.

The name of the server machine should be the fully-qualified server name, including a domain name and Internet extension. Using just the server name may work in most situations, but fully-qualified is preferred. Alternatively, you can use the IP address of the server machine.

The `SITE_SLICE_LOC` setting points to the directory that the server uses for storing information about the site such as the user registry, depot information, etc. This directory should be owned by the **acserver** account (Unix) or the System account (Windows) and must be physically located on the server machine. The `SITE_SLICE_LOC` location must not be within a remotely mounted file system (Unix) or within a shared directory (Windows) on a remote machine.

Controlling Server Log Verbosity

The AccuRev Server maintains a log file, **acserver.log**, in subdirectory **logs** of the **site_slice** directory. To enable verbose logging, add this line to the **acserver.cnf** file:

```
LOG_LEVEL = 1
```

Unix Systems Only: Controlling the Server's User Identity

Note: this section applies to Unix machines only. On Windows machines, the user identity of the AccuRev Server is always the local **System** account.

If you perform a server installation as the **root** user, the following specifications are stored in the **acserver.cnf** file:

```
USER = nobody  
GROUP = nobody
```

When the Server starts automatically at system bootstrap time, it reads these specifications and assumes the identity: user **nobody** in group **nobody**.

You can edit these settings to change the AccuRev Server's user/group identity. This change takes effect the next time the Server is started by the **root** user — either automatically at system bootstrap time or manually, using the **acserverctl** utility. (See *Controlling Server Operation* below.)

When the Server is started manually, it runs under the identity of the user who started it.

Controlling Multithreading of the AccuRev Server

The AccuRev Server is a multi-threaded program, architected to support a maximum of 256 concurrent threads. To conserve system resources, you can specify a lower maximum in the **acserver.cnf** file:

```
MAX_THREADS = 25
```

As it's running, the Server may reduce the maximum even further than the specified `MAX_THREADS` level, depending on the available computing resources.

Server Watchdog

The AccuRev Server is designed for high reliability, to ensure the integrity of the repository and its availability to AccuRev users. But even the robust software systems are occasionally compromised; the AccuRev Server can be brought down by a bad disk sector or an administrator's mistaken command.

The reliability of the AccuRev Server is further enhanced by a companion program, termed the Watchdog, which runs on the same machine. The sole function of the Watchdog is to monitor the Server and restart it in the event of a failure. The effect of the Watchdog on Server performance is insignificant.

Note: both the Server and Watchdog show up in the operating system's process table with the same name: **accurev_server**.

Every 10 seconds, the Watchdog sends a simple command to the Server. If the Watchdog detects that the Server is not responding or is not functioning properly, the Watchdog restarts the Server. If the Watchdog detects 5 such failures within a 3-minute timespan, it doesn't restart the Server; such a situation indicates the need for server reconfiguration or investigation by the AccuRev support team.

For the most part, the Watchdog is "transparent", making administration simple:

- The Watchdog process starts automatically when the Server process is started (typically, at operating system bootstrap time).
- The administrative commands for stopping the Server process cause both the Watchdog and Server to stop. These commands have been reworked to terminate the Watchdog directly; before it exits, the Watchdog terminates the Server.

Tools that control the execution of the Server and Watchdog are in described in section [Controlling Server Operation](#) on page 12.

Server Logging

The AccuRev Server maintains a log file in subdirectory **logs** of the **site_slice** directory:

- The name of the log file is **acserver.log**. (In previous releases, the log file was named **accurev_server.log** and was located in the **site_slice** directory itself.)
- Each log message includes a timestamp, a client-server connection ID, the user's principal-name, the name of the server subtask being performed, the ID of the thread performing the server subtask, and the IP address of the client machine. (Server subtasks are described in section [Verbose Logging](#) below.) For example:

```
2003/05/09 12:30:36 1002 jjp cur_wspace 0x803 127.0.0.1
```

For more on server subtasks, see [acserver.command.tasklist](#) on page 17 and [Server Subtask Codes](#) on page 19.

- "Log file rotation" keeps the log file from growing too large. Periodically, the AccuRev Server timestamps the current log file and moves it to subdirectory **logs** of the **site_slice** directory.

For example, the log file might be renamed **acserver-2002-01-23-04-47-29.log**. The Server then creates a new **acserver.log** file.

The log file is rotated weekly; it is also rotated whenever the AccuRev Server is restarted.

Watchdog Logging

The Watchdog also maintains a simple log file, **acwatchdog.log**, in the **logs** directory. It sends a message to the log file each time it “probes” the Server to determine whether it’s still running:

```
[2002\03\26 11:06:05] -Info- watchdog: connection testing
```

The Watchdog log file is rotated in the same way as the Server log file.

Verbose Logging

As of Release 3.1, you can enable verbose Server logging with a setting in the **acserver.cnf** file:

```
LOG_LEVEL = 1
```

A typical AccuRev client command causes the AccuRev Server to execute a set of server subtasks. The Server is a multi-threaded program, so it can handle several client commands concurrently. For each client command, the Server’s “master thread” creates a new “worker thread” to perform the set of subtasks for that particular command. When the worker thread has performed all the subtasks, it exits. At the verbose logging level, the log messages indicate many of the details of server subtask execution.

For example, a single create-new-workspace command generates a set of log messages like this:

```
[2002/03/07 01:58:10] connection 1021 on 24 started
[2002/03/07 01:58:10] connection 1021 on 24 task 192.168.1.43 dgold add_ws
[2002/03/07 01:58:10] connection 1021 on 24 task 192.168.1.43 dgold end
[2002/03/07 01:58:10] connection 1021 on 24 success 0.454 0 0 0 192.168.1.43 dgold
```

These messages may or may not appear on consecutive lines of the log file. If multiple client commands are being executed concurrently by different worker threads, the log messages that the threads generate will be interleaved in the log file.

Let’s examine each message in the above example:

```
[2002/03/07 01:58:10] connection 1021 on 24 started
```

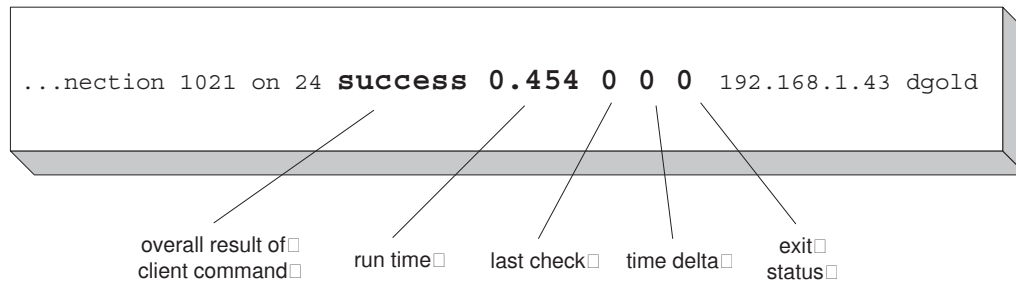
The first message is generated at the time ([**2002/03/07 01:58:10**]) a client request is accepted by the Server’s master thread (**connection 1021** between the client and the server). The master thread creates a new worker thread (worker thread # **24 started**) and hands the request off to it. Note that this message does not identify the client command (create-new-workspace).

```
[2002/03/07 01:58:10] connection 1021 on 24 task 192.168.1.43 dgold add_ws
[2002/03/07 01:58:10] connection 1021 on 24 task 192.168.1.43 dgold end
```

Each time the worker thread begins a particular subtask, it sends a message to the log. In this example, the client command translates to server subtasks **add_ws** and **end**. Each message includes the IP address of the client machine (**192.168.1.43**) and the principal-name of the AccuRev user (**dgold**).

```
[2002/03/07 01:58:10] conn... 1021 on 24 success 0.454 0 0 0 192.168.1.43 dgold
```

The last message is generated by the worker thread after it has completed all subtasks and is about to exit. In addition to the data included in the preceding messages, this message reports:



- **success / failure:** the overall result of the attempt to execute the client command.
- **run time:** the total time, in seconds, that the worker thread took to process the entire client command.
- **last check:** the time, in seconds, elapsed since last progress update from worker thread. In a success message, this value is 0. In a failure message, this value is non-zero.
- **time delta:** the time difference between the clocks on the client and server machines
- **exit status:** the exit code for thread: 0 = success, non-zero = error code

Verbose Log Messages for Failed Commands

The following set of log messages document a failed add-file-to-depot command:

```
[2002/03/07 01:58:36] connection 1027 on 30 started
[2002/03/07 01:58:37] connection 1027 on 30 task 192.168.1.43 dgold cur_wspace
[2002/03/07 01:58:37] connection 1027 on 30 task 192.168.1.43 dgold show_trigger
[2002/03/07 01:58:37] connection 1027 on 30 task 192.168.1.43 dgold create
[2002/03/07 01:58:37] -Error- from 30: 1 OS error - db.c:732 - unable to
    open file for reading: '/u2/test1/storage/depots/gizmo/valid_sizes'
[2002/03/07 01:58:37] connection 1027 on 30 failure 1.231 1 0 1 192.168.1.43 dgold
```

Controlling Server Operation

AccuRev includes facilities for controlling the operation of the AccuRev Server and the new Watchdog. The user interface varies by platform:

- Unix: a special command-line utility
- Windows: the standard **Services** console

Unix: 'acservctl' Utility

If the AccuRev Server is running on a Unix machine, you can control its operation with the **acservctl** program. This is a Bourne-shell script, located in the AccuRev **bin** directory. (It is based on the control script for the Apache Web server.)

Note: by default, **acservctl** assumes that AccuRev is installed at **/opt/accurev**. If this is not the case, you must run **acservctl** in an environment where **ACCUREV_BIN** is set to the pathname of the AccuRev **bin** directory. For example:

```
env ACCUREV_BIN=/var/accurev/bin acservctl ...
```

acservctl provides a set of non-interactive commands. The format of each command is:

```
acservctl <command-name>
```

(Omitting *<command-name>* is equivalent to executing **acservctl help**.) The commands are:

start

Start the Server and Watchdog processes.

stop

Tell the Server and Watchdog processes to stop gracefully.

status

Report whether the Server is running or not.

pause

Tell the Server to stop accepting new requests from AccuRev clients.

resume

Tell the Server to start accepting new requests from AccuRev clients again.

restart

Tell the Server process to stop gracefully; this allows the Watchdog to restart it. If the Watchdog is not running, a **start** or **hardrestart** is performed.

kill

Forcibly stop the Server and Watchdog processes. This is accomplished by sending a TERM signal to each process. The script gets the process-IDs from files **acserv.pid** and **acwatchdog.pid**, located in the **site_slice** directory. These files are written automatically when the processes are started.

hardrestart

Perform a **kill**, followed by a **start**.

help

Display an **acservctl** command summary.

The various “tell a process” capabilities are implemented through server-control files. (See *Server-Control Files* below.)

Windows: ‘Services’ Console

If the AccuRev Server is running on a Windows machine, you can control its operation from the standard Windows **Services** console. The **Services** console is located in the Windows **Control Panel**; in some versions of Windows, it’s in a subfolder called **Administrative Tools**.

The context (right-click) menu of the AccuRev service includes these commands:

- start**
- stop**
- pause**
- resume**
- restart**

For descriptions of these commands, see *Unix: ‘acsreverctl’ Utility* above. On Windows, the **restart** command brings down both the Server and the Watchdog, by performing a **stop** followed by a **start**.

Server-Control Files

On all platforms, the AccuRev Server and Watchdog processes check, once per second, for the existence of several “server-control files” in the **site_slice** directory. The existence of the server-control file causes the process to perform a particular action. In most cases, the contents of the file are irrelevant; a zero-length file will do.

acsrever.command.pause

(used by the **pause** server-control command) Tells the Server to stop accepting new requests from AccuRev clients. The Server completes transactions that are already in progress and logs its “paused” status to the log file. Then, it continues to run, but the only thing it does is monitor the **acsrever.command.pause** file. When this server-control file is removed, the Server resumes normal operation.

This server-control file is *not* removed when a new Server process starts up. If the file exists, the Server starts up in the paused state.

acsrever.command.shutdown

(used by the **stop** and **restart** server-control commands) Tells the Server to “finish up quickly” and exit. The Server immediately stops accepting new requests from AccuRev clients. It continues to work on transactions that are already in progress, but it aborts any transactions that are not completed within 10 seconds. Then, the Server exits.

If the Watchdog is running, it detects the Server’s shutdown and starts up a new Server immediately. Thus, this server-control file typically causes a Server restart. In any event, this file is automatically removed whenever a new Server process starts up.

If 10 seconds is not the appropriate interval for “finishing up quickly”, place another integer (such as 120) in the **acsrever.command.shutdown** file. The Server exits when there are no more transactions to work on, or when the timeout interval has passed, whichever comes first.

acwatchdog.command.shutdown

(used by the **stop** server-control command) Tells the Watchdog to exit cleanly. When the Server detects that the Watchdog has exited, it exits also, just as if it had found an **acserver.command.shutdown** file (see above). In this case, however, there is no longer a Watchdog process, so no restart of the Server takes place.

This server-control file is automatically removed when a new Server process starts up.

Open Filehandle Limits and the AccuRev Server

The AccuRev Server is designed to handle multiple client commands concurrently: any number of requests that “read” data, along with one command that “writes” data. Accomplishing such concurrency typically requires that the AccuRev Server have many files open at the same time. Each operating system imposes limits on how many files can be open simultaneously. There may be an “open file descriptor” limit for each user process, or an overall limit for all user processes, or both. If the AccuRev Server hits the open file descriptor limit, additional client requests will be queued until file descriptors become available. (No client command is cancelled, and no data is lost. Hitting the open file descriptor limit just slows AccuRev Server performance.)

The table below indicates the default open file descriptor limits for the various AccuRev-supported operating systems. Following the table are instructions for increasing these limits.

Operating System	Default Limit on Open Filehandles	Command to Change Limit on Open Filehandles
Windows NT, Windows 2000, Windows 2003, Windows XP Pro	2048 per system	none
Windows 98, Windows ME, Windows XP Home	2048 per system	none
Solaris 7	64 per process	no command; must reconfigure Unix kernel (see below)
Linux (Red Hat Fedora 3)	about 3500 per system	/sbin/sysctl -w fs.file-max=10000
Linux (Power PC 2.4)	1024 per process	
AIX 5.1	2000 per process	
HP-UX 11	360 per process	no command; must rebuild Unix kernel (see below)
Tru64 5.1	4096 per process	
IRIX 6.2	1024 per process	
SCO UnixWare 7.1.1, 7.1.4	60 per process	no command; must rebuild Unix kernel (see below)

Note: If you are performing a pre-purchase evaluation of AccuRev in an environment with a limited number of users and a limited amount of data, there is no need to make any changes. The default limits will be more than adequate.

Changing the Per-Process Open File Descriptor Limit

The procedure for increasing a process's maximum number of open files varies from operating system to operating system.

Note: in all cases, be sure to remove file **acservr.handle.limit**, located in the AccuRev **site_slice** directory, before restarting the AccuRev Server or rebooting the operating system. This file caches the current value of the open-files limit.

Linux

You must be the **root** user to perform the following procedure.

1. Change the overall limit on the number of open file descriptors each process can have (e.g. to 10,000):

```
> /sbin/sysctl -w fs.file-max=10000
```

The number you specify is stored in file **/proc/sys/fs/file-max**.

2. Add this line to file **/etc/pam.d/login**:

```
session    required    /lib/security/pam_limits.so
```

3. Change the capabilities of the Bash shell command **ulimit**, by creating or editing the “nofile” (number of open files) lines in file **/etc/security/limits.conf**. Example:

```
*    soft    nofile    1024
*    hard    nofile    10000
```

These lines specify that:

- By default, a Bash shell (and its subprocesses) can have as many as 1024 open file descriptors.
- A Bash shell can execute the command **ulimit -n *number***, with 65535 as the maximum value of ***number***. This enables that particular shell (and its subprocesses) to have up to ***number*** open files. (The person executing the **ulimit** command doesn't need to know what the “hard limit” specified in **/etc/security/limits.conf** is — he can just enter the command as **ulimit -n unlimited** to get the maximum value.)

4. Test your change, by entering a **ulimit** command in a Bash shell, setting the limit somewhere in between the **soft** and **hard** specifications you made in Step 3. For example:

```
> /bin/bash  
  
$ ulimit -n  
1024  
  
$ ulimit -n 5000  
  
$ ulimit -n  
5000
```

5. Restart the AccuRev Server process from a Bash shell:

```
> <AccuRev-bin-directory>/acserverctl stop(stop the AccuRev Server)  
  
> /bin/bash(start a Bash shell)  
  
$ <AccuRev-bin-directory>/acserverctl start(restart the AccuRev Server)
```

Solaris

You must be the **root** user to perform the following procedure.

1. Change the overall limit on the number of open file descriptors each process can have (e.g. to 5,000), by adding or changing this line in file **/etc/system**:

```
set rlim_fd_max=5000
```

2. Reboot the operating system.

HP-UX

You must be the **root** user to perform the following procedure.

1. Enter this command to start the System Administration Manager utility:

```
> sam
```

2. Select **Kernel Configuration**, then **Configurable Parameters**.
3. Select the **maxusers** parameter.
4. Increase the value of this parameter, for example to 128.
5. Invoke the **Actions > Process New Kernel** command, to create a new HP-UX kernel.
6. Exit the System Administration Manager utility.
7. Reboot the operating system.

acserver.command.tasklist

Enables the periodic writing of file **acserver.tasks**, which reports on the current set of “worker threads” in the multi-threaded AccuRev Server. At any given moment, each worker

thread is executing one of the server subtasks that implement a particular client command. (Each client command translates to at least two server subtasks.)

If you create **acservr.command.tasklist** as an empty file, the server updates file **acservr.tasks** every 10 seconds. To specify a different update interval, place a single text line in file **acservr.command.tasklist** (e.g. **30**, to indicate a 30-second update interval).

Each time the server updates the **acservr.tasks** file, it replaces the entire contents with a new thread-status table, which looks like this:

```
### AccuRev Server Tasklist ###
# time [2002/03/07 01:58:48]
# active_tasks 6
#
# connid runtime threadid checktime state user_ip user task
#-----
    1034      3      52      1      1 192.168.1.43 dgold create
      0      0      0      0      0 null
    1039      0      55      0      2 192.168.1.43 dgold create
    1031     122      41      0      1 192.168.1.43 dgold create
    1036      0      53      0      1 192.168.1.43 dgold create
      0      0      0      0      0 null
    1027     422      30     390      1 192.168.1.43 dgold create
      0      0      0      0      0 null
    1038      0      54      0      0 192.168.1.43 dgold create
      0      0      0      0      0 null
```

The thread-status table always has 256 rows, one for each “thread slot”. (The AccuRev Server is architected to support a maximum of 256 concurrent threads. To tune server performance, you can lower the limit by setting the **MAX_THREADS** parameter in the **acservr.cnf** configuration file. The current limit is reported as the **max threads** value in the status table’s header.) Typically, only the first few rows of the table are non-null, indicating the status of an active thread.

Each row includes the following fields:

connid

The ID number of the client-server connection. Each client command establishes a separate connection with the server.

runtime

The time (in seconds) that this worker thread has been running.

threadid

The ID number of this worker thread.

checktime

The time (in seconds) since this worker thread last issued a progress update.

state

The current state of this worker thread:

- 0 = slot assigned to worker thread, but thread not active yet (set by master thread).
- 1 = worker thread active (set by worker thread).
- 2 = worker thread completed, but master thread has not yet cleared the slot for reuse (set by worker thread).

user_ip

IP address of client machine.

user

AR principal-name of the user who issued the client command.

task

Name of the server subtask being executed by this worker thread. For the correspondence between these names and subtask numeric codes, see *Server Subtask Codes* on page 19.

Let's look at the status of two threads, each of which is executing a create subtask:

```
1031      122      41      0      1      192.168.1.43      dgold      create
```

The worker thread reported in the above line has been active a long time (122 seconds), but it has recently issued a progress update to the master thread (checktime = 0). It's likely that everything is OK. Perhaps this is a very large transaction, such as converting thousands of external files to elements.

```
1027      422      30      390      1      192.168.1.43      dgold      create
```

This worker thread is in trouble. It hasn't issued a progress update in 390 seconds, so it's probably hung.

acserver.command.taskkill

Use this capability only as a "last-resort". Typically, it's preferable to restart the entire AccuRev Server process (which allows in-progress tasks to complete), rather than terminating just one of its worker threads.

To terminate a particular worker thread:

- Go to the **site_slice** directory.
- Determine the ID number of the worker thread by examining the thread-status table in file **acserver.tasks** (see **acserver.command.tasklist** above).
- Place the thread's ID number (e.g. **42**) in the flag file:

```
echo 42 > tempfile
mv tempfile acserver.command.taskkill      (Windows: use ren command)
```

Using the **mv** (or **move**) command instead of the **echo** command to create the **taskkill** file prevents a race condition that might cause the server to see **taskkill** as an empty file.

Important note: after terminating a thread, restart the AccuRev Server as soon as possible. This minimizes the likelihood that terminating the thread will cause a memory resource leak in the Server process, impairing overall system performance.

Server Subtask Codes

The following table lists the name and numeric code for each AccuRev Server subtask.

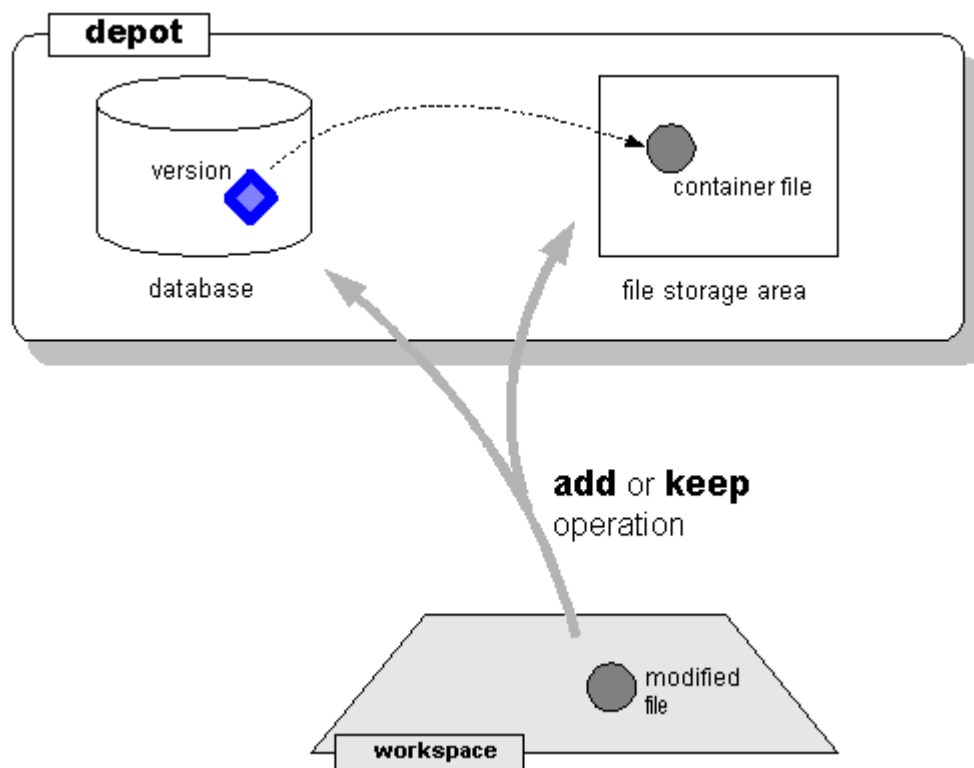
Name	Code	Name	Code	Name	Code
PROMOTE	1	CHWS	28	BACKUP	53
DEFUNCT	2	CHSLICE	29	ANC	54
UNDEFUNCT	3	ADD_PRINC	30	LSACL	55
MOVE	4	SHOW_PRINC	31	AUTHN	56
CHANGE_STREAM	5	POPULATE	32	SEXEC	57
CREATE	6	CHECK_TIME	33	TRANS_LOOKUP	58
CO	7	CHECK_PRINC	34	SHOW_GROUP	59
KEEP	8	ASSIGN	35	SHOW_MEMBERS	60
PURGE	9	SHOW_TRIGGER	36	ADD_MEMBER	61
WS_TYPE	10	MKTRIG	37	RM_MEMBER	62
CUR_LOC	11	RMTRIG	38	SETACL	63
STAT	12	PING	39	IS_MEMBER	64
SHOW_STREAMS	13	FETCH	40	DEFCOMP	65
HIST	14	TRANS_TIME	41	WIP	66
STREAM_TOP	15	SHOW_REF	42	NAME	67
ADD_STREAM	16	CHUSER	43	LSCOMP	68
SERVER_VERSION	17	CUR_WSPACE	44	PATCH_LIST	69
DIFF	18	REMOVE	45	TRANS_LIST	70
MERGE_PREP	19	UPGRADE	46	GETCONFIG	100
UPDATE	20	LOCK_SET	47	XML	101
ADD_PROJECT	21	LOCK_UNSET	48	PUTCONFIG	102
ADD_WS	22	LOCK_TEST	49	DISPATCH_READ	103
SHOW_PROJECTS	23	REACTIVATE	50	SHOW_DISPATCH	104
SHOW_WSPACES	24	CHANGE_PROJECT	51	MINIPING	105
SHOW_SLICES	25	SHOW	52	SECINFO	106
				END	999

Archiving of Version Container Files

As of AccuRev Version 4.0, only binary files can be archived. Text files cannot be archived.

Users execute a **keep** command to preserve the current contents of a version-controlled file (“file element”) in an AccuRev depot. Similarly, users execute an **add** command to place a file under version control. The **add** and **keep** commands:

- copy the current contents of the file to a container file, located in the depot’s file storage area.
- create an associated version object in the depot’s database.



In accordance with the TimeSafe principle, the version object can never be deleted from the database or modified in any way. The corresponding container file is always accounted for, and can be in either of these states:

- **normal** — the container file is located in the depot’s file storage area (the **data** subdirectory of the depot directory). AccuRev commands, such as **update**, **cat**, and **diff**, can access the contents of the version.
- **archived** — the container file has been moved to a gateway area outside the depot’s file storage area. AccuRev commands cannot access the contents of an archived version. After container files have been moved to the gateway area, an administrator can use standard operating system or third-party tools to transfer the container files to off-line storage: tape, CD-ROM, removable disk drive, Web-accessible storage, etc.

Only binary files can be in the archived state. Text files are always in the normal state.

The AccuRev CLI commands **archive** and **unarchive** shift container files back and forth between the normal and archived states. Before using **unarchive**, the administrator would transfer the appropriate container files from off-line storage back to the gateway area. Then, invoking **unarchive** moves the container files back into the depot's **data** directory.

The 'archive' Command

The command **accurev archive** processes one or more versions of file elements, shifting the versions' container files from **normal** status to **archived** status. The command has this format:

```
accurev archive [ -i ] [ -s <stream> ] [ -t <transaction-range> ]  
[ -c <comment> ] [ -R ] [ -E <elem-type> ] <element-list>
```

Determining Which Versions to Archive

archive determines the set of versions to archive as follows:

- It focuses on a particular stream. If you don't specify a stream with **-s <stream>**, it uses your current workspace's stream.
- It narrows the scope to a particular set of file elements, which you specify as command-line arguments in the **<element-list>**. You can include directories in this list; in this case, use the **-R** option to include the recursive contents of those directories.
- By default, all eligible versions of the specified elements in the specified stream are archived. You can use **-t** to limit the set to the versions of those elements created in a specified transaction, or range of transactions:

```
-t <number>single transaction  
-t <number>-<number>range of transactions
```

- You can also limit the set of versions to those of a particular element type, using the **-E** option.

Note: you *must* specify the **binary** element type: **-E binary**.

The **archive** command refuses to archive any version that is currently visible in any stream or snapshot. It also refuses to process versions in text format. Only binary-format versions will be archived.

Using the **-i** option (in addition to the other options described above) generates an XML-format listing of the desired versions, but does not perform any actual archiving work. It is highly recommended that you do this before actually archiving any versions.

Archiving the Versions

After determining which versions to process, the **archive** command moves a version's container file from a "normal" location under the **data** directory:

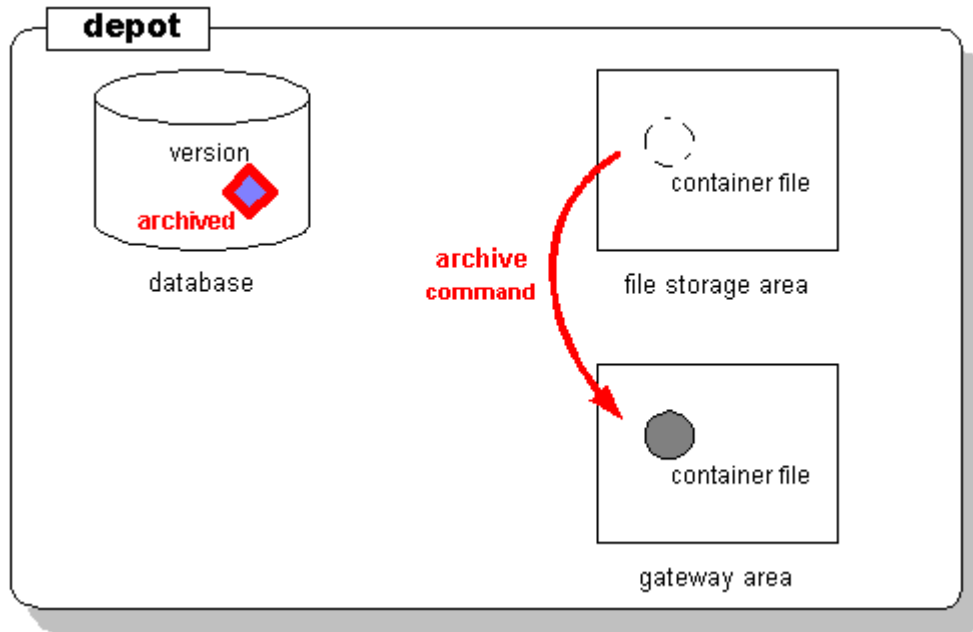
```
.../storage/depots/gizmo/data/25/07.sto
```

... to a corresponding “archived” location in the **archive_gateway/out** area:

```
.../storage/depots/gizmo/archive_gateway/out/data/25/07.sto
```

archive also marks the version as “archived” in the depot database.

Subsequent attempts by AccuRev commands to retrieve the contents of the archived version will fail.



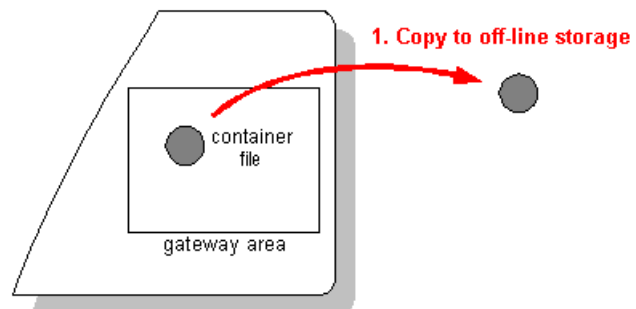
The changes made by this command are recorded in the depot database as a transaction of type **archive**. You can use the **-c** option to specify a comment string to be stored in this transaction. You can search for particular comment strings when using the **hist** command to locate previous **archive** transactions. See *Using 'hist' to Research Previous 'archive' Commands* on page 24.

The ‘reclaim’ Command

The **archive** command merely moves container files from one location (the depot’s **data** area) to another location (the depot’s **archive_gateway** area). To reduce the amount of disk space consumed by the archived versions, you must:

1. Copy the files from the **archive_gateway** directory tree to off-line storage. You can use operating system commands (**copy**, **xcopy**, **cp**, **tar**) and/or third-party data-backup utilities to accomplish this.

Be sure to use a tool that preserves the source data’s directory hierarchy in the copied data.

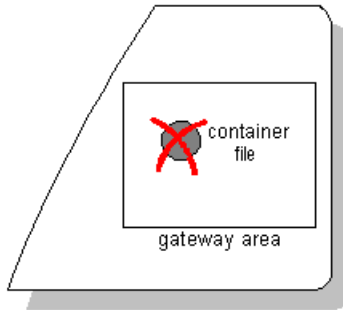


WARNING! AccuRev has no way of tracking which tool you use for this purpose, or what off-line storage medium you copy the files to. It's up to you to maintain good records of these activities!

2. Delete the files from the **archive_gateway** directory tree, using the **reclaim** command:

```
accurev reclaim [ -p <depot> ]  
                -t <archive-transaction>
```

You must specify a single transaction of type **archive**, created by previous **archive** command(s).



2. Reclaim disk space

Attempts to Access Archived Versions

The **archive** command affects depot storage only. It has no immediate effect on any workspace. But you might subsequently enter an AccuRev command that attempts to access a version that has been archived. For example, if version **gizmo_int/8** of file **floor_layout.gif** has been archived, then this command fails:

```
accurev cat -v gizmo_int/8 floor_layout.gif > old_layout.gif
```

In such cases, a message is sent to **stderr** and the command's exit status is 1.

Using 'hist' to Research Previous 'archive' Commands

Each depot's database contains a complete record of all version-archiving activity for that depot. Execution of the **archive** command is recorded as a transaction of kind **archive**. You can use the **hist** command to locate all such transactions:

```
accurev hist -a -k archive
```

You can also select just those **archive** transactions that were created with a particular comment string:

```
accurev hist -a -k archive -c "stadium images"
```

In a **reclaim** command, you must indicate the storage space to be reclaimed by specifying the number of an **archive** transaction.

Restoring Archived Versions — The 'unarchive' Command

After you have **archived** some versions and **reclaimed** the disk space:

- the versions' container files are no longer in the depot's **data** area.
- copies of the container files are no longer in the depot's **archive_gateway/out** area (since you've transferred them to off-line storage).

If you decide you need to restore some or all of the archived versions, you must first copy the container files from off-line storage back to the **archive_gateway** area. You must place the files under **archive_gateway/in**, at the same relative pathname as they were originally placed under **archive_gateway/out**. For example, if the **archive** command places a container file at:

```
.../storage/depots/gizmo/archive_gateway/out/data/25/07.sto
```

... you must restore the file from off-line storage to this pathname:

```
.../storage/depots/gizmo/archive_gateway/in/data/25/07.sto
```

After placing all the container files in the **archive_gateway/in** area, you can execute the **unarchive** command. This command has exactly the same format as **archive** — that is, you specify the versions to be restored in exactly the same way as you originally archived them. For example:

Archive all non-active versions of GIF image files in stream **gizmo_maint_4.3**:

```
accurev archive -s gizmo_maint_4.3 -E binary *.gif
```

Restore all those versions:

```
accurev unarchive -s gizmo_maint_4.3 *.gif
```


Replication of the AccuRev Repository

This chapter describes how to set up and use AccuRev's repository replication feature. One server machine stores the "master" copy of the AccuRev data repository; any number of additional server machines can store "replicas" of the repository. Each replica contains some or all of the repository's depots. Users can send commands to the AccuRev Server software running on any of these machines.

Note: use of the repository replication feature requires purchase of the *AccuReplica* product from AccuRev, Inc.

Master and Replica

One host in the network acts as the AccuRev server machine: it runs the AccuRev Server process and contains the AccuRev repository in its local disk storage. In a replication scenario, this original host (or more precisely, this instance of the AccuRev Server process) is termed the master server.

One or more additional hosts in the network can act as replica servers. Each such host runs its own instance of the AccuRev Server process; likewise, each such host has its own copy of the AccuRev repository. The diagram below shows the servers in a replication scenario, along with various client machines.

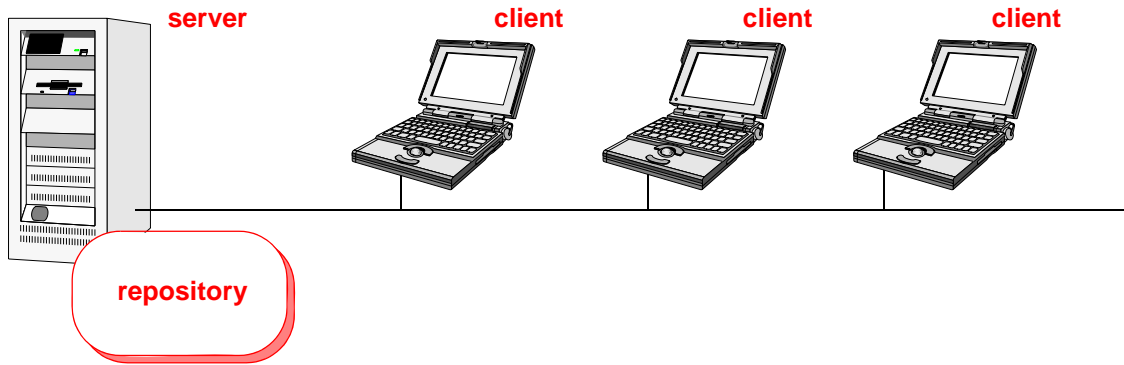
We use the terms master repository and replica repository to distinguish the multiple repositories in a replication scenario. The master repository is always complete and up-to-date; all transactions (operations that change the repository) are handled by the master server and are logged in the master repository.

By contrast, a replica repository can become out of date during day-to-day usage: it can be missing recent transactions initiated by clients using other replica servers or the master server. You can issue a simple synchronization command to download such missing transactions from the master repository to the replica repository. This makes the replica repository database into an exact copy (temporarily, at least) of the master repository database. Synchronization also occurs automatically whenever a transaction is initiated by a client using that replica server.

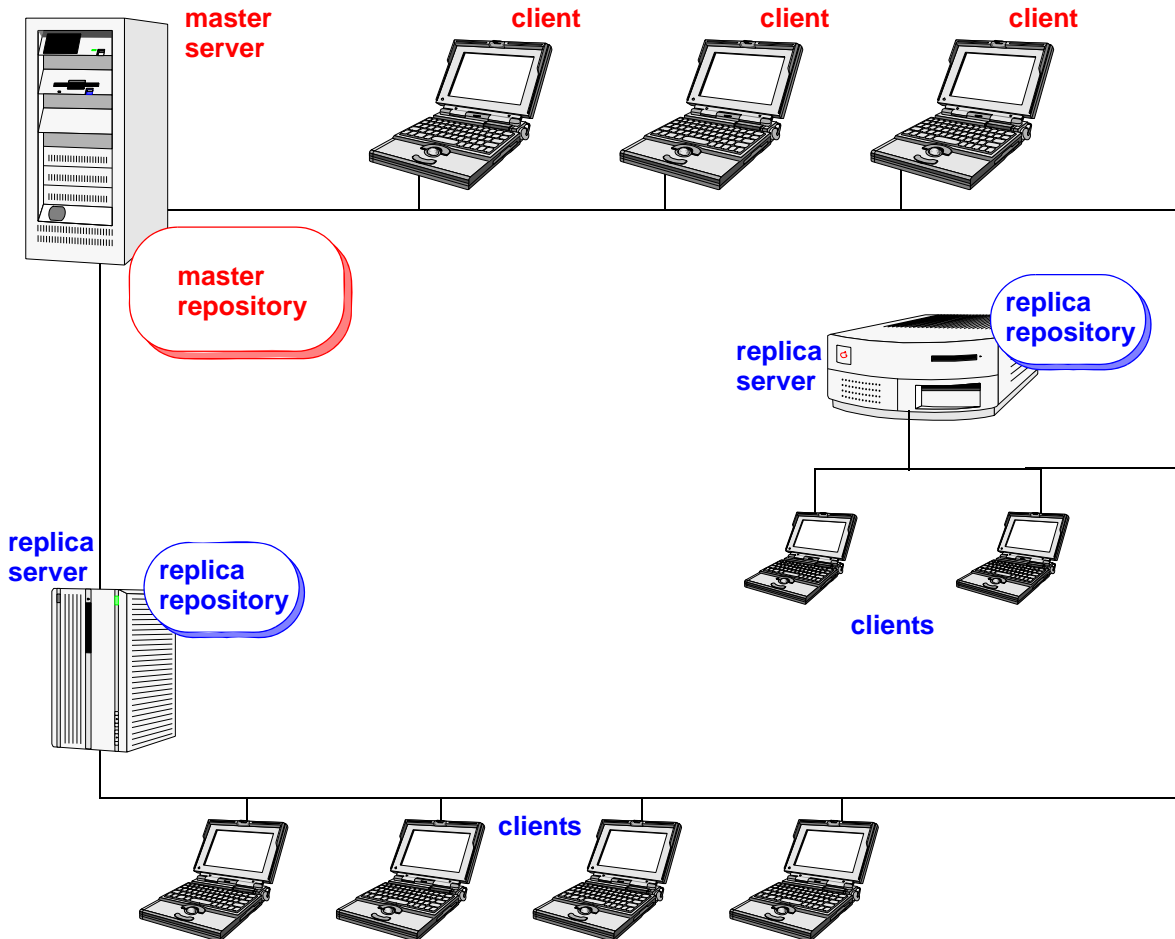
A replica repository can contain a selected subset of the depots in the master repository. If the master repository contains 10 depots, one replica repository might be configured to contain 4 of the depots, another replica repository might be configured to contain 7 of them, and a third replica repository might be configured to contain all 10 depots.

For more details on day-to-day operations involving master and replica repositories, see the sections starting with *Using a Replica Server* on page 32. First, we address licensing issues and describe the replica setup process.

Before Replication



After Replication



AccuRev Licensing in a Replication Environment

A standard AccuRev license (Professional Edition or Enterprise Edition) enables your organization to maintain a single data repository, managed by a single AccuRev Server process running on a particular machine. This license also enables a certain number of users to access the repository with AccuRev client software.

The most obvious (and easiest) way to add replication to this environment is to have the existing server machine become the master server. Let's say that you want to replicate the repository at two remote sites — one with 10 client users, the other with 25 client users. In this case, your organization needs to purchase:

- two copies of AccuRev Replication Sever
- ten individual Remote User Licenses
- one Remote Site License (for the 25 users at the second remote site)

Installation Procedure: Assumptions

The procedures in the following sections make the following assumptions about your AccuRev installation:

- The original AccuRev server machine is named **masthost**. If one or more replicas have already been created, then this machine is already the master server.
- The machine to be made into a replica server is named **replhost**.

Restriction: **masthost** and **replhost** must both have “big-endian” hardware architectures, or must both be “little-endian”. Little-endian architectures include Intel and AMD, running either Windows or Linux software. Big-endian architectures include Sparc (Sun) and PA-RISC (Hewlett-Packard), and PowerPC (AIX).

The same host can act both as the master server and as the replica server (or even as multiple replica servers). This can be convenient for performing testing of new releases, validating your organization's development procedures, etc. Before proceeding to the next section, read the notes in *Using the Same Host as Both Master Server and Replica Server* on page 32.

Setting Up the Master Server

The change described in this section needs to be made just once — when transitioning from a non-replicated setup to a replicated setup.

1. Stop the AccuRev Server process on **masthost**.
2. Edit the **acservr.cnf** file, which is located in the AccuRev **bin** directory. Add the following line:

```
REPLICATION_ENABLED = true
```

CAUTION: enabling replication poses a potential security risk. Before proceeding, be sure to read *Synchronization Security* on page 35.

3. Note the `MASTER_SERVER` and `PORT` settings in the `acserver.cnf` file. You'll need these settings in Step 8 below.
4. Restart the AccuRev Server process on `masthost`.

Setting Up the Replica Server

The following sections detail the steps for setting up `replhost` as an AccuRev replica server, using the repository data from `masthost`.

Install AccuRev

5. Obtain from AccuRev Customer Support a `keys.txt` file containing a license to run the AccuRev Server software on `replhost`. Most specifications, including the number of users, should match the license on `masthost`. The host name will differ — and the port number might differ, too.
6. Install the AccuRev software on `replhost`. During installation, choose both the Custom and Full options. If you've already installed AccuRev on this machine, choose a new location for the installation directory. The installation wizard will prompt you to specify the storage location for the new, local repository on `replhost` (“Customize: Choose a Folder for AccuRev Server Data Storage”).

In the following steps, we'll assume that the repository location is `/opt/accurev/storage` — adjust for your installation and operating system.

Revise the Server Configuration File

7. Stop the AccuRev Server process on `replhost`.
8. Edit the `acserver.cnf` file, which is located in the AccuRev `bin` directory.
 - Change the keyword `MASTER_SERVER` to `LOCAL_SERVER`, and change the keyword `PORT` to `LOCAL_PORT`. But don't change the value of either setting.
 - Add new `MASTER_SERVER` and `PORT` settings, using the values of these settings on `masthost`. (These are the settings you noted in Step 3.)

After these edits, the four lines might look like this:

```
MASTER_SERVER = masthost
PORT = 5050
...
LOCAL_SERVER = replhost
LOCAL_PORT = 5050
```

Note: there is no relationship between the `LOCAL_PORT` and `PORT` numbers. They can be the same or different.

Start the AccuRev Server Process

9. Make sure that the **replhost** operating system process in which the AccuRev Server will run has an AccuRev username (“principal-name”) identity with enough rights to access all the files in the **masthost** repository. AccuRev ACLs control access to depots and streams for specified AccuRev users and groups.
 - Unix: use environment variable ACCUREV_PRINCIPAL to establish the AccuRev principal-name of the operating system process. For example, in a Unix Bourne shell:

```
ACCUREV_PRINCIPAL=acadmin; export ACCUREV_PRINCIPAL
```
 - Windows: the AccuRev Server runs as a Windows service; reconfigure it to run as the desired AccuRev principal-name, instead of LocalSystem. In the Control Panel’s Services applet: open the Properties window for the AccuRev service, go to the Log On tab, select “This account”, and enter the AccuRev principal-name and Windows password.

On either Unix or Windows, make sure that the AccuRev-level password of the desired principal-name is set. (This password is independent of OS passwords.) The AccuRev-level password is stored in file **authn**, in the **.accurev** subdirectory of *principal-name*’s home directory.

10. Start the AccuRev Server process on **replhost**.

Synchronize the Site Slice

11. Run this client command on **replhost**:

```
accurev replica sync
```

This command copies data from **masthost**’s site slice to **replhost**’s site slice. In particular, it makes the AccuRev Server on **replhost** aware of all the depots in the master repository on **masthost**.

Configure the Replica Server to Include the Desired Depots

The AccuRev repository on **replhost** now has an up-to-date site slice, but the repository doesn’t yet contain detailed data on any depots.

12. List all the depots in the master repository, by executing this command on **replhost**:

```
accurev show -fix depots
```

In the XML-format output, the depots that exist in the master repository, but are not replicated on **replhost**, are listed with this attribute:

```
ReplStatus = "missing"
```

13. For each depot that is to be replicated on **replhost**, execute a **mkreplica** command on **replhost**. For example, if depots named **widget**, **gadget**, and **cust_support** are to be replicated:

```
accurev mkreplica -p widget
accurev mkreplica -p gadget
accurev mkreplica -p cust_support
```

Using the Same Host as Both Master Server and Replica Server

In a production environment, it doesn't make sense to have the same machine act as both the master server and a replica server. But it *does* make sense to use a single machine to test new software or work processes. You can use the same machine as both **masthost** and **replhost** in the procedures described in sections *Setting Up the Master Server* and *Setting Up the Replica Server* above. If you do so, keep these points in mind:

- There must be two separate AccuRev installations on the machine.
- Each installation has its own **acserver.cnf** configuration file, located in the AccuRev **bin** directory. The `SITE_SLICE_LOC` setting must be different in each **acserver.cnf** file. Similarly, the `DEPOTS_DEFAULT` setting must be different.
- There is no way to have a depot slice be associated with multiple repositories. A depot slice can be located outside the area specified by the `DEPOTS_DEFAULT` setting — with **mkdepot -l** on the master server; with **mkreplica -l** on a replica server; or with **chslice** on either kind of server. But don't use these techniques to make multiple instances of the AccuRev Server think that they are managing the same slice location.

Setting Up a Client Machine to Use a Replica Server

A machine on which the AccuRev client software is installed can use any server — either a replica server or the master server. As always, the `SERVERS` setting in the client configuration file — **acclient.cnf** in the AccuRev **bin** directory — specifies which AccuRev Server process is to be sent client command requests. Examples:

```
SERVERS = replhost:5050           (use replica server)
```

```
SERVERS = masthost:5050          (use master server)
```

You can switch a client back and forth among multiple replica servers (and possibly the master server, too). It's as simple as editing the client's **acclient.cnf** file.

Using a Replica Server

When your client machine is set up to use a replica server, you can issue all AccuRev commands in the usual way. In general:

- Configuration management commands that *read* data from the repository — such as **files**, **diff**, and **cat** — use the replica repository.

- Configuration management commands that *write* data to the repository — such as **keep**, **promote**, and **merge** — use the master repository. After the master repository has been modified, the local replica repository is automatically brought up to date. For details, see *Synchronizing a Replica Manually* on page 34, which describes how you can bring the local replica repository up to date when you are *not* writing data to the repository.
- All AccuWork issue management operations are handled by the master server. Thus, replication does not improve AccuWork performance.

The Update Command

The **update** operation works as follows when you execute it on a client that uses a replica server:

- A **stat** operation is performed on the replica server, to determine the state of the workspace stream and its backing stream.
- Database transactions are copied from the master repository to the replica repository, bringing the replica database completely up to date.
- Data files representing new versions of elements are copied from the file storage area in the master repository to the file storage area in the replica repository.
- Those data files are copied from the replica repository to your workspace tree.
- The transaction level of the workspace is set to the most recent transaction (or to the transaction specified with **update -t**).

Note: suppose a client machine is using the master server, and you update a workspace using that machine. If you then switch the client machine to using a replica server, you must perform a **replica sync** command before updating the workspace again.

Creating New Depots

New depots can be created only in the master repository, not in a replica repository. If a client using a replica repository issues a **mkdepot** command, an error occurs:

```
Cannot create a new depot on the replica server
```

After creating a new depot in the master repository, you can include it in a replica repository with this sequence of commands, issued on a client that uses the replica server:

```
accurev replica sync
accurev mkreplica -p <depot-name>
```

Adding and Removing Depots from a Replica Repository

After you have set up a replica repository, you can use the commands **mkreplica** and **rmreplica** to change which depots are included in the replica repository. These commands are described in the *AccuRev User's Guide (CLI Edition)*.

Synchronizing a Replica Manually

During the course of development, your local replica repository typically becomes out-of-date with respect to the master repository. This occurs when other users send commands to other replica servers or directly to the master server. In both such cases, new transactions are entered in the master repository, but are not entered in the your local replica repository.

At any time, you can enter this CLI command to bring your local replica repository up to date:

```
accurev replica sync
```

This transfers data from the master repository site slice to the replica repository site slice. It also transfers database transactions from the master repository to the replica repository — but only for the depots that are included in the local replica. It does not transfer the corresponding storage files for **keep** transactions. See *On-Demand Downloading of a Version's Storage File* below.

A **replica sync** command is performed automatically on the local replica after each operation that is initiated by a client of the local replica, and that makes a change to the repository. See *Using a Replica Server* on page 32.

Note: you never need to synchronize directly with other replicas; synchronizing with the master is sufficient to bring your replica up to date.

On-Demand Downloading of a Version's Storage File

As a performance optimization, AccuRev copies database transactions only — not storage files that hold the contents of **keep** versions — when it synchronizes the master repository with a replica repository ...

- ... during a **replica sync** command.
- ... during the automatic replica synchronization that follows an operation, invoked by a client using a replica server, that modifies the repository.

Storage files for versions are downloaded from the master repository to the local replica repository during an **update** (see *The Update Command* on page 33). The storage file for an individual version is downloaded when a client using a replica server explicitly references that version. Examples:

```
accurev cat -v talon_dvt/12 foo.c
accurev diff -v talon_dvt/12 foo.c
```

Both these commands cause the storage file for version **talon_dvt/12** of file **foo.c** to be downloaded to the local replica repository before the command itself is executed.

Automating Replica Synchronization

If a workgroup is much less active than other workgroups, its local replica repository can “fall behind” the master repository significantly. This can also occur if the workgroup uses the local replica repository mostly as a reference — for frequent read operations, but infrequent write operations. Falling behind in this way does no harm, but it can be bothersome. When some user finally does perform a write operation — keeping a new version of a file, or changing the location

of a workspace — the local replica repository automatically “catches up”, which might involve downloading tens or hundreds of transactions.

To prevent the local replica repository from falling too far behind, we recommend that you use operating system tools to perform an **accurev replica sync** command automatically, at regular intervals — say, every 15 minutes. On a Windows machine, use the Scheduled Tasks applet in the Control Panel. On a Unix/Linux host, set up a **cron** job to execute this command.

Synchronization Security

Note: this section describes a security risk that exists only for organizations using the *AccuRev Replication Server* product. This risk does not apply to organizations that use the standard AccuRev software, without the replication option.

The repository synchronization scheme poses a potential security risk: the **acserver.cnf** server configuration file on an AccuRev server machine can name *any* master server machine in a MASTER_SERVER setting. And by default, the targeted master server will comply with *any* synchronization request — even an **accurev replica sync** command executed on a completely unrelated client machine.

We strongly recommend using the **server_admin_trig** trigger on the master server machine to implement an authentication scheme, so that the master server will send repository data over the wire only to valid requestors. The following Perl code might be added to the sample **server_admin_trig** script included in the **examples** subdirectory of the AccuRev distribution:

```
if ($command eq "replica_sync") {
    if ($principal ne "rep01_acadmin" and $principal ne "rep02_acadmin") {
        print TIO "Repository synchronization disallowed:\n";
        print TIO "Authentication by the server_admin_trig script failed.\n";
        close TIO;
        exit(1);
    }
}
```

This code allows users **rep01_acadmin** and **rep02_acadmin** to perform repository synchronization, rejecting requests from all other user identities.

Note: a **server_admin_trig** script identifies the command as **replica_sync**, even though the actual CLI command is **replica sync**.

The replica_site.xml File

Each replica repository’s site slice directory contains an XML-format file, **replica_site.xml**. This file contains information about the depots that are replicated in that repository. The **mkreplica** and **rmreplica** commands maintain the contents of this file.

Moving the AccuRev Server and Repository to Another Machine

The AccuRev data repository should be physically located on the machine that runs the AccuRev Server process. (This is firm dictate, but not an absolute restriction — see *READ ME NOW: Assuring the Integrity of the AccuRev Repository* on page 1.) The repository consists of multiple slices: the site slice contains information that pertains to the entire repository, and each depot has its own slice. Each slice contains a database consisting of multiple files.

From time to time, you may want (or need) to have the AccuRev server process run on a different machine. To accomplish this, you must:

- Perform a “full” (client and server) installation of AccuRev on the new machine.
- Move the data repository to the new machine.

If the new machine has a different byte order than the old machine, you must migrate each slice to use the “opposite-endian” data format. This involves swapping the bytes in each machine-level word. The **maintain migrate** command performs this conversion. With no arguments, it migrates the site slice. With an argument, it migrates a depot slice. The results of the migration are stored in a new subdirectory named **swapped** within the site-slice or depot directory.

This procedure is safe: the original slice data is never modified, and there is no harm in running the **maintain migrate** multiple times on a slice.

Procedure for Moving the Repository

Make sure you perform each of the following steps on the appropriate server machine. We call them:

- The “source” machine — where the AccuRev server is currently running and the data repository is currently located.
- The “destination” machine — the machine to which you want to move the data repository.

Note: the steps below always show Unix pathname separators (/). When you’re executing commands on a Windows machine (either source or destination), be sure to use Windows pathname separators (\).

The procedure calls for multiple stops and starts of the AccuRev Server process. For details on how to accomplish this, see *Controlling Server Operation* on page 12.

On the Destination Machine

1. Perform that a “full” (that is, both client and server) installation of AccuRev on this machine. In the steps below, we’ll refer to the installation directory on the destination machine as *<dest-install-dir>*.

2. Run this command on the destination machine:

```
accurev hostinfo
```

3. Send the output of this command to AccuRev Customer Support, as part of a request for a new license key for the destination machine. When you get the new license key, install it in the **site_slice** directory, according to the supplied instructions.
4. Store an additional copy of the license key outside the AccuRev repository, for use in Step 14.
5. *Do the following only if you're moving the repository to an opposite-endian machine:*
Copy the following file to a secure location:

```
<dest-install-dir>/storage/site_slice/datadict.ndb
```

On the Source Machine

6. Execute the command **accurev show slices** and **accurev show depots**, and save the output for reference in the following steps.
7. Stop the AccuRev Server process. (Reminder: see *Controlling Server Operation* on page 12.)
8. Perform a full backup of the AccuRev repository, as described in *Backing Up the Repository* on page 3.
9. *Do the following only if you're moving the repository to an opposite-endian machine:*

9a) In the **site_slice** directory, make backup copies in a subdirectory of all the ***.ndb** files. For example, back up file **lock.ndb** by copying it to **save_original/lock.ndb**.

9b) Migrate the site slice, using the **maintain** program located in the AccuRev **bin** directory:

```
maintain migrate
```

This creates a **swapped** subdirectory under the **site_slice** directory.

9c) Move all ***.ndb** database files from the **site_slice/swapped** directory up to the **site_slice** directory. This overwrites the original ***.ndb** files in the **site_slice** directory. (These are the files that you backed up in Step 9a.)

9d) For each depot listed in the **show depots** output (see Step 6), make backup copies in a subdirectory of the depot's ***.ndb** files. For example, back up file **.../depots/<depot-name>/ancestry.ndb** by copying it to **.../depots/<depot-name>/save_original/ancestry.ndb**.

9e) For each depot listed in the **show depots** output, migrate the depot's slice:

```
maintain migrate <depot-name>
```

This creates a set of **swapped** subdirectories in the depot slices.

9f) Move all ***.ndb** files from the **<depot-name>/swapped** directories up to the parent **<depot-name>** directories. This overwrites the existing ***.ndb** files in the **<depot-name>** directories. (These are the files that you backed up in Step 9d.)

10. Copy the entire tree starting at directory **storage** within the AccuRev installation area. Be sure to use a method that preserves file ownership (e.g. **tar -cp**).

On the Destination Machine

11. Stop the AccuRev Server process.
12. Rename `<dest-install-dir>/storage` (e.g. to **storage.OLD**).
13. “Paste” (unpack, unzip, tar -x) the copy of the directory tree you made in Step 10, so that the pasted data becomes `<dest-install-dir>/storage`.
14. Overwrite file `<dest-install-dir>/storage/site_slice/keys.txt` with the copy of the license key that you saved in Step 4.
15. *Do the following only if you’re moving the repository to an opposite-endian machine:*
 - 15a) Restore the copy of `<dest-install-dir>/storage/site_slice/datadict.ndb` that you made in Step 5a.
 - 15b) Reindex the site slice:
`maintain reindex`
16. If the **storage** directory is located at a different pathname on the source and destination machines:
 - 16a) Start the AccuRev Server process.
 - 16b) For each depot, determine the corresponding slice number (see Step 6) and run this command:
`accurev chslice -s <slice-number>
-l <dest-install-dir>/storage/depots/<depot-name>`
 - 16c) Stop the AccuRev Server process.
Perform the following step only if you’re moving the repository to an opposite-endian machine:
 - 16d) Reindex each depot:
`maintain reindex <depot-name>`
17. Start the AccuRev Server process.

AccuRev Triggers

A trigger is a “code hook” or callback built into certain AccuRev commands. When a user enters the command, the corresponding trigger “fires”; this causes a user-defined or built-in procedure to be performed just before or after the command executes. Typically, a user-defined procedure is implemented as a script in the Perl scripting language. Sample Perl scripts are available in the **examples** subdirectory of the AccuRev installation directory.

Note: in this chapter, “trigger script” refers to any executable program, written in any language, that is executed when a trigger fires.

AccuRev supports both pre-operation triggers and post-operation triggers. In addition, there are triggers that integrate AccuRev’s configuration management facility with its issue management facility (AccuWork); these triggers have pre- and post-operation components.

Some triggers are set with the **mktrig** command; others are set by placing the script at a special location.

Pre-Operation Triggers

The following triggers execute a procedure before the user-requested command executes. Each of these triggers has the ability to cancel execution of the user’s command. Some of the triggers fire on the client machine, and others on the server machine. It’s possible for a single command (for example, **keep**) to cause triggers to fire both on the client and on the server.

Client-Side Triggers

The following pre-operation triggers fire on the client machine:

- **pre-create-trig**: fires on the client machine prior to execution of an **add** command. The trigger script must specify the element type of each element being created.
- **pre-keep-trig**: fires on the client machine prior to execution of a **keep** command.
- **pre-promote-trig**: fires on the client machine prior to execution of a **promote** command. The trigger script can transform (or completely replace) the user’s comment string.

Server-Side Triggers

The following pre-operation triggers fire on the server machine.

- **server_admin_trig**: fires on the server machine prior to execution of certain commands. This is a repository-wide trigger — it fires no matter what depot, if any, the user’s command applies to. The following commands cause **server_admin_trig** to fire:

<code>mkdepot</code>	<code>mktrig</code>	<code>mkuser</code>
<code>chdepot</code>	<code>rmtrig</code>	<code>chuser</code>
<code>chslice</code>	<code>lock</code>	<code>chpasswd</code>
<code>mkstream</code>	<code>unlock</code>	<code>lsacl</code>

```
chstream      defcomp      setacl
mkws          repl_sync   mkgroup
chws          write_schema addmember
chref                                     ismember
remove                                     rmmember
reactivate
```

The **defcomp** command is not user-visible; it's used in the implementation of the include/exclude facility CLI commands **incl**, **excl**, and **includo**. The **repl_sync** command recognized by the **server_admin_trig** trigger corresponds to the CLI command **replica sync**.

Note: prior to Version 3.5, the services now provided by **server_admin_trig** were provided by a trigger named **server_all_trig**. (In the AccuRev-provided sample trigger script, this includes allowing commands to be executed only by a specified list of AccuRev users.) For backward compatibility, the older trigger is still supported.

- **server_preop_trig**: fires on the server machine prior to execution of certain commands. This is a depot-specific trigger — it fires only for commands that operate on the depot(s) where the trigger has been activated. The following commands cause **server_preop_trig** to fire:

```
add           promote
keep          purge
```

The **server_admin_trig** and **server_preop_trig** triggers are independent of each other and are fired by different sets of commands — for a given command, only one of these triggers will fire.

Post-Operation Triggers

The following triggers execute a procedure after the user-requested command executes successfully. If the user's command fails, the post-operation trigger does not fire. A post-operation trigger always fires on the server machine.

- **server-post-promote-trig**: fires on the server machine subsequent to execution of a **promote** command.
- **server_dispatch_post**: fires on the server machine each time a AccuWork issue record is created or modified. This trigger is intended to enable email notification of A sample Perl script is available in the **examples/dispatch** subdirectory of the AccuRev installation directory.

Transaction-Level Integration Trigger

You can achieve tight coordination of your organization's configuration management and issue management capabilities by enabling one or both of the integrations between AccuRev's

configuration management and issue management facilities. The transaction-level integration is enabled by a trigger on a depot-by-depot basis:

```
accurev mktrig -p WidgetDepot pre-promote-trig client_dispatch_promote
```

The “client_dispatch_promote” integration routine is built into the AccuRev software — no scripts are required — and includes both pre-operation and post-operation components:

1. On the client machine, a user invokes the AccuRev **promote** command.
2. The pre-operation part of the trigger fires on the client machine, prompting the user to specify one AccuWork issue record. Alternatively, the user can specify the issue record on the command line, using the **-I** option. If this part of the trigger fails (for example, the user specifies a non-existent issue record), the **promote** command itself is cancelled.
3. The **promote** command completes, and is recorded in the AccuRev repository as a transaction.
4. The post-operation part of the trigger fires on the server machine, updating the issue record that the user specified by adding the number of the **promote** transaction to the **affectedFiles** field.

If you use the built-in “client_dispatch_promote” integration routine as the **pro-promote-trig** trigger, you must not also set a **server-post-promote-trig** trigger. Doing so would suppress the post-operation component of the “client_dispatch_promote” routine. For information on handling this situation and other aspects of customizing the transaction-level integration, see *Implementation and Customization of the Transaction-Level Integration* on page 58 of the *AccuWork Issue Management Manual*.

Note: AccuRev features another integration between configuration management and issue management, which works at the change-package level instead of the transaction level. See *Integrations Between Configuration Management and Issue Management* on page 54 of the *AccuWork Issue Management Manual*.

Preparing to Use an AccuRev-Provided Trigger Script

Sample trigger scripts are installed with AccuRev, in the **examples** subdirectory. These sample scripts are implemented in the platform-neutral Perl scripting language. Use the following procedure to install and use one of these scripts:

1. **Install Perl.** There are many source on the Web for Perl. We recommend the ActivePerl distribution from <http://www.activestate.com>. This distribution includes a conversion utility, **pl2bat**, which makes a Perl script executable under Windows, by embedding the Perl code in a Windows batch file (**.bat**).

Be sure to install Perl on all appropriate machines. Note that some pre-operation triggers run on the client machine, while others run on the server machine. All post-operation triggers run on the AccuRev server machine.

2. **Get a copy of the sample script.** Copy the sample script from the **examples** subdirectory of the AccuRev installation directory to an AccuRev workspace. Then use the **add** command to place the script under version control.

3. **Prepare the script.** Open the script in a text editor, and customize the script according to the instructions included as comment lines. Before embarking on heavy script customization, be sure to read *The Trigger Parameters File* on page 45.
4. **Enable the Trigger.** Enable the trigger, either with the **mktrig** command or by placing the script in the proper location. See the following section for details.

Enabling a Trigger

Depending on its type, an AccuRev trigger is enabled in one of these ways:

- Executing an **accurev mktrig** command, specifying the location of the script. AccuRev simply records the location you specify in the repository; it doesn't make a copy of the script. Make sure that no one moves it!
- Placing the executable script file in the location prescribed for that type of trigger.

For details, consult the appropriate subsection below:

pre-create-trig, pre-keep-trig, pre-promote-trig, server-post-promote-trig

Use the **mktrig** command to enable use of the script in a particular depot. For example:

```
accurev mktrig -p WidgetDepot pre-keep-trig /usr/ac_scripts/addheader
```

The **-p** option isn't necessary if your current directory is in a workspace associated with that depot. When the trigger fires, AccuRev will search for the script at the specified pathname (in the example above, **/usr/ac_scripts/addheader**).

We strongly suggest specifying an absolute pathname. Otherwise, when the trigger files AccuRev will use the search path of the user (**pre-create-trig**, **pre-keep-trig**, or **pre-promote-trig**) or the search path of the AccuRev Server (**server-post-promote-trig**) to find the specified script file.

server_admin_trig

Place an executable file in subdirectory **triggers** of the **site_slice** directory:

- Unix: the file must be named **server_admin_trig** or **server_admin_trig.pl**
- Windows: the file must be named **server_admin_trig.bat**

Example:

```
C:\Program Files\AccuRev\storage\site_slice\triggers\server_admin_trig.bat
```

Note: if the directory contains both a **server_admin_trig** executable and an old **server_all_trig** executable, only the **server_all_trig** script is executed.

server_preop_trig

Place an executable file in subdirectory **triggers** of the slice directory of one or more depots (**accurev show slices** displays slice directory locations):

- Unix: the file must be named **server_preop_trig** or **server_preop_trig.pl**

- Windows: the file must be named **server_preop_trig.bat**

Example:

```
/opt/accurev/storage/depots/talon_tests/triggers/server_preop_trig
```

server_dispatch_post

Place an executable file in the AccuRev executables (**bin**) directory on the AccuRev Server machine:

- Unix: the file must be named **server_dispatch_post** or **server_dispatch_post.pl**
- Windows: the file must be named **server_dispatch_post.bat**

Note: for compatibility with previous AccuRev releases, the script can also be named **dispatch_email**, with the appropriate suffix.

Example:

```
C:\Program Files\AccuRev\bin\server_dispatch_post.bat
```

Notes on Triggers in Multiple-Platform Environments

Observe these guidelines in a multiple-platform environment, where the trigger script will be accessed from both Windows machines and Unix machines:

- Don't even try to arrange for exactly the same script file to be accessed by all users, on all platforms. Instead, place scripts to be accessed by Unix users in one directory and equivalent scripts to be accessed by Windows users in another directory.
- Make sure the Windows script has a **.bat** suffix (for example, **check_for_comments.bat**), and that the Unix script has no suffix (**check_for_comments**).

On a Windows machine, AccuRev searches for a trigger script file named **check_for_comments** before it searches for a batch file named **check_for_comments.bat**. That's another reason to place Windows and Unix scripts in separate directories.

- Make sure the scripts run correctly on their respective platforms. And remember to revise *all* versions of the script when you revise any one of them!
- In the **mktrig** command, specify the script name without a suffix — for example:

```
accurev mktrig -p WidgetDepot pre-keep-trig check_for_comments
```

The Trigger Parameters File

When a trigger fires and executes a user-supplied script, AccuRev passes two arguments to the script:

- The first argument is the pathname of a flat-text file containing information about the transaction that is about to be performed (or was just completed).

- The second argument is the pathname of an XML-format file containing the same information. (In some cases, detailed below, the XML-format file contains a small amount of additional information that is not contained in the flat-text file.)

Exceptions: only one argument, the pathname of an XML-format file, is passed to a **server_preop_trig** script or a **server_admin_trig** script.

These files are called trigger parameters files. The flat-text file contains a series of values — usually one value per line — in a prescribed order. The XML-format file contains a set of elements below the top-level **<triggerInput>** element. Each such element contains the information for one parameter: the parameter name is the element tag, the parameter value is the element contents (sometimes encoded as a set of subelements). For example, here are two trigger parameters files generated by the same user command:

Flat-text trigger parameters file	XML-format trigger parameters file
pre-create	<triggerInput>
talon	<hook>pre-create</hook>
talon_dvt_john	<depot>talon</depot>
4	<stream1>talon_dvt_john</stream1>
adding some files	<changePackages></changePackages>
this multi-line	<comment>adding some files
comment has	this multi-line
four lines	comment has
C:/wks/talon/dvt_john	four lines</comment>
john	<topDir>C:/wks/talon/dvt_john</topDir>
/tools/cont.sh	<principal>john</principal>
/tools/end.sh	<elemList>
/tools/start.sh	<elem>/tools/cont.sh</elem>
	<elem>/tools/end.sh</elem>
	<elem>/tools/start.sh</elem>
	</elemList>
	</triggerInput>

The information contained in the trigger parameters file varies among the trigger types, as described in the following sections.

Format of the “pre-create-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-create-trig** script. This information describes the creation of one or more new elements to a depot (CLI: **add**, GUI: **Add to Depot**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: pre-create .
depot	Name of depot targeted by the command.
stream1	Name of the workspace stream in which the new elements are to be created.
changePackages	(XML-format parameters file only) The change packages affected by this command; currently defined to be empty for a pre-create-trig trigger.
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 55 below).
topDir	Pathname to the top-level directory of the user's workspace tree, as it is listed by the show wspaces command.
principal	AccuRev username of person invoking the command.
elemList	One or more files/directories to be added to the depot. For general notes, see <i>Encoding of Element Lists</i> on page 54 below.

In the flat-text trigger parameters file, the elements to be created are listed, one per line, at the end of the file:

```

/tools/cont.sh
/tools/end.sh
/tools/start.sh
      <-- end-of-file of trigger parameters file

```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as **<elem>** sub-elements of **<elemList>**:

```

<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>

```

Overwriting the 'pre-create-trig' Trigger Parameters File

A **pre-create-trig** script must overwrite its flat-text parameters file with data that indicates the type of each element to be created. Each line must describe one new element:

```

<element-pathname> <element-type>

```

... where *<element-pathname>* is a pathname from the input "elemList", and *<element-type>* is a numeric code:

- 1 directory
- 2 text file
- 3 binary file

For example, a command that adds two text files, two binary files, and a directory to the depot would replace its flat-text parameters file with this data:

```
/tools/end.sh 2
/tools/icons 1
/tools/icons/end.png 3
/tools/icons/start.png 3
/tools/start.sh 2
```

Note: there is currently no provision for the script to overwrite the XML-format trigger parameters file. The data to be passed to the AccuRev Server must be in flat-text format.

Format of the “pre-keep-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-keep-trig** script. This information describes the creation of a new versions of one or more existing elements in a depot (CLI: **keep**, GUI: **Keep**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: pre-keep .
depot	Name of depot targeted by the command.
stream1	Name of the workspace stream in which the new versions are to be created.
changePackages	(XML-format parameters file only) A set of <changePackageID> subelements, specifying the change packages affected by this command; currently defined to be empty for a pre-keep-trig trigger.
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 55 below).
topDir	Pathname to the top-level directory of the user’s workspace tree, as it is listed by the show wspaces command.
principal	AccuRev username of person invoking the command.
elemList	A specification for each new element version to be created. For general notes, see <i>Encoding of Element Lists</i> on page 54 below.

In the flat-text trigger parameters file, the versions to be created are listed, one per line, at the end of the file. Each line contains three specifications:

```
<element-pathname> <version-ID> <element-type>
```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list. For example:

```
/tools/icons/end.png talon_dvt_john/5 3
/tools/icons/end.sh talon_dvt_john/9 2
/tools/icons/start.png talon_dvt_john/2 3
/tools/icons/start.sh talon_dvt_john/13 2
```

In the XML-format trigger parameters file, each version to be created is encoded as an `<elem>` sub-element of `<elemList>`. The element's attributes specify the version-ID (**stream** and **version** attributes) and the element-type (**elemType** attribute). The element pathname is encoded as the contents of `<elem>`.

The following example contains the same data as the flat-text example above:

```
<elemList>
  <elem
    stream="talon_dvt_john"
    version="5"
    elemType="3"/>/tools/icons/end.png</elem>
  <elem
    stream="talon_dvt_john"
    version="9"
    elemType="2"/>/tools/icons/end.sh</elem>
  <elem
    stream="talon_dvt_john"
    version="2"
    elemType="3"/>/tools/icons/start.png</elem>
  <elem
    stream="talon_dvt_john"
    version="13"
    elemType="2"/>/tools/icons/start.sh</elem>
</elemList>
```

In either format, the element-type value can be either **2** (text file) or **3** (binary file). Note that different versions of an element can have different types.

Format of the “pre-promote-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-promote-trig** script. This information describes the creation of a new versions of one or more existing elements in a depot (CLI: **promote**, GUI: **Promote**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it's also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: pre-promote .

Parameter	Description
output_file	(XML-format parameters file only) Name of a file that the trigger script can overwrite with a comment string, which replaces the user-supplied comment string (if any).
depot	Name of depot targeted by the command.
stream1	Name of the workspace or stream that the versions are to be promoted from.
changePackages	(XML-format parameters file only) A set of <changePackageID> subelements, specifying the change packages affected by this command; currently defined to be empty for a pre-promote-trig trigger.
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 55 below).
topDir	Pathname to the top-level directory of the user’s workspace tree, as it is listed by the show wspaces command.
principal	AccuRev username of person invoking the command.
elemList	One or more files/directories to be promoted. For general notes, see <i>Encoding of Element Lists</i> on page 54 below.

In the flat-text trigger parameters file, the elements to be created are listed, one per line, at the end of the file:

```

/tools/cont.sh
/tools/end.sh
/tools/start.sh
      <-- end-of-file of trigger parameters file

```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as <elem> subelements of <elemList>:

```

<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>

```

Overwriting the ‘pre-promote-trig’ Trigger Parameters File

A **pre-promote-trig** script can work in tandem with a **server-post-promote-trig** script, providing customized “before and after” processing around the execution of **Promote** commands:

- The **pre-promote-trig** script overwrites its flat-text trigger parameters file.

- The *first line* of the overwritten parameters file becomes the value of the `<fromClientPromote>` parameter passed to the **server-post-promote-trig** script.

Note: there is currently no provision for a **pre-promote-trig** script to pass data to a **server-post-promote-trig** script by overwriting the XML-format trigger parameters file.

Manipulating the ‘promote’ Comment String

Each **promote** transaction includes a comment string (possibly null), supplied by the user who entered the **Promote** command. A **pre-promote-trig** script can modify or replace the user-supplied comment string. This is the *only* way to manipulate the comment string: once the **promote** transaction has been written to the AccuRev repository, it cannot be modified in any way.

The **pre-promote-trig** script can process the comment string using the following procedure. Note that the procedure must use the XML-format trigger parameters file, not the flat-text parameters file.

1. Extract the contents of the `<comment>` subelement from the trigger parameters file. This is the user-supplied comment string, which might include multiple text lines.
2. Extract the contents of the `<output_file>` subelement. This is the name of a temporary file. (The flat-text parameters file does not include this parameter; that’s why you must work with the XML-format file.)
3. Perform any processing to transform, or simply replace, the user-supplied comment. You might want to include, or at least consider, other values from the parameters file.
4. Write the new comment string to the temporary file named in Step 2, in the form of a `<comment>` element. For example:

```
<comment>
17 elements promoted from stream widget_dvt
</comment>
```

5. End the script with a zero exit status (success).

Format of the “server-post-promote-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **server-post-promote-trig** script. This information is generated by AccuRev, and describes the **Promote** command that has just executed. The first line of this file provides a mechanism for passing user-specified data from a **pre-promote-trig** script to a **server-post-promote-trig** script. See *Overwriting the ‘pre-promote-trig’ Trigger Parameters File* on page 50.

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: server-post-promote .

Parameter	Description
depot	Name of depot targeted by the command.
stream1	Name of the stream that the versions were promoted to.
fromClientPromote	A single text line (including the line-terminator): the first line of the trigger parameters file passed to the pre-promote-trig script (and possibly overwritten by that script).
changePackages	(XML-format parameters file only) A set of <changePackageID> subelements, specifying the change packages (that is, issue records) specified in the user's command.
transNum	The transaction number of the Promote transaction that just completed.
transTime	The time of the Promote transaction that just completed.
comment	Zero or more comment lines specified by the user in the Promote command (see <i>Encoding of Command Comments</i> on page 55 below).
topDir	Pathname to the top-level directory of the user's workspace tree, as it is listed by the show wspaces command.
principal	AccuRev username of person invoking the command.
elemList	A specification for each version that was promoted. For general notes, see <i>Encoding of Element Lists</i> on page 54 below.

Format of the “server_preop_trig” Trigger Parameters File

The parameters file passed to a **server_preop_trig** script is in XML format:

```
<triggerInput>
  <hook> ... </hook>
  <command> ... </command>
  <principal> ... </principal>
  <ip> ... </ip>
  ...
</triggerInput>
```

The set of subelements under the <**triggerInput**> element depends on the user's command. The following table provides a summary. For full details, see the sample **server_preop_trig** script in the **examples** directory in the AccuRev installation area.

Parameter	Description
hook	Type of trigger: server_preop_trig .
command	The user command: add , keep , promote , or purge .
principal	AccuRev username of person invoking the command.
ip	The IP address of the client machine.
stream1	The user's workspace stream.

Parameter	Description
stream2	The workspace's parent (backing) stream.
depot	Name of depot targeted by the command.
fromClientPromote	(Promote command only) The number of the AccuWork issue record entered by the user, when prompted by the transaction-based integration or the change-package-based integration.
changePackagePromote	(Promote command only) A set of <changePackageID> subelements, specifying the change packages (that is, issue records) specified in the user's command. These forms of the promote command generate a <changePackagePromote> element: <ul style="list-style-type: none"> • promote -I <issue-number> • promote (user prompted by issue-management integration to specify an issue record) • promote -Fx (user specifies a set of change packages with an XML file)
comment	Comment string specified by the user. If the comment spans multiple lines, line-terminators are embedded in the string, but the final line does not have a line-terminator.
elemList	A set of <elem> subelements, each specifying one element processed by the user's command.

Format of the “server_admin_trig” Trigger Parameters File

The parameters file passed to a **server_admin_trig** script is in XML format: The set of subelements under the <triggerInput> element depends on the user's command. The following table provides a summary. For full details, see the sample **server_preop_trig** script in the **examples** directory in the AccuRev installation area.

Parameter	Description
hook	Type of trigger: server_admin_trig .
command	The user command.
principal	AccuRev username of person invoking the command.
user	(chuser, chpasswd) AccuRev username being modified.
newName	(chuser) new AccuRev username.
newKind	(chuser) new user kind (dispatch or cm).
ip	The IP address of the client machine.
stream1	The stream targeted by the user command.
stream2	The parent (backing) stream of stream1 .

Parameter	Description
stream3	(chws, chstream) The new name of the workspace or stream.
objectName	(remove, reactivate) Name of object targeted by the command.
objectType	(remove, reactivate) Type of object targeted by the command: 1 =reference tree; 2 =workspace; 3 =stream; 5 =user; 6 =group

Format of the “server_dispatch_post” Trigger Parameters File

The parameters file passed to a **server_dispatch_post** script is in flat-text format. The order of the parameters in the table below is the order in which they appear in the file.

Parameter	Description
hook	Type of trigger: server-post-dispatch .
project	Name of depot in which the AccuWork issues database resides.
stream	blank line.
from_client_promote	Two SPACE-separated fields: <ul style="list-style-type: none"> The number of the transaction that created the previous version of this issue record. (The number 0 indicates that this is a newly created issue record.) The issue number
transaction_num	The number of the transaction that created this new version of the issue record.
transaction_time	The time at which transaction_num was created.
comment_lines	blank line.
author	The AccuRev principal-name of the user who saved the issue record.

Encoding of Element Lists

In both kinds of trigger parameters files, each element is listed by its path relative to the depot’s top-level directory:

```
/tools/cont.sh
```

The path begins with a slash in order to simplify constructing the element’s full pathname on the client machine: just append the given element pathname to the **topDir** pathname (the top-level directory of the user’s workspace tree).

In the flat-text trigger parameters file, the elements (or elements-to-be) to be processed by the user command are listed, one per line, at the end of the file:

```
/tools/cont.sh
/tools/end.sh
```

```
/tools/start.sh
        <-- end-of-file of trigger parameters file
```

Unlike the set of comment lines, there is no need to supply an element count; an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as **<elem>** sub-elements of the **<elemList>** element:

```
<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>
```

Encoding of Command Comments

In the flat-text trigger parameters file, the user's comment is indicated by a line-count (0 or greater), followed by the lines of the comment, if any:

```
4          <-- number of comment lines to follow
adding some files
this multi-line
comment has
four lines
```

In the XML-format trigger parameters file, the user's comment is encoded as the contents of the **<comment>** element: a single string. For a multi-line comment, this string has line-terminators embedded:

```
<comment>adding some files <-- embedded line-terminator
this multi-line          <-- embedded line-terminator
comment has              <-- embedded line-terminator
four lines</comment>
```

Note that the final line-terminator is automatically stripped from all comment strings.

The sample set of trigger scripts includes a Perl script for each kind of trigger. The script's comments include a detailed description of the layout of the parameters file for that kind of trigger.

Trigger Script Contents

A trigger script can send email, start a build, update a Web site, or perform many other tasks. In particular, you can run AccuRev commands to get more information. One common use of the **server-post-promote-trig** trigger is to run the **hist** command using the transaction number of the promotion, generating the list of promoted elements for inclusion in an email notification.

Trigger Script Exit Status

The exit status (return value) of a **pre-create-trig**, **pre-keep-trig**, or **pre-promote-trig** script is important:

- A zero exit status indicates success: the AccuRev command (**add**, **keep**, or **promote**) is allowed to proceed.
- A non-zero exit status indicates failure: the AccuRev command is canceled and the depot remains unchanged.

File Handling by Trigger Scripts

A trigger script can overwrite its parameters file (after reading it, presumably). This provides a way for the script to communicate with the AccuRev command or with a “downstream” script:

- The parameters file for a **pre-keep-trig** script ends with a series of lines, one per element to be kept:

```
<pathname-of-element> <version-ID> <element-type>
```

<pathname-of-element> is not a full file system pathname, but starts at the workspace’s top-level directory (which is included earlier in the parameters file). *<version-ID>* is the new version to be created for that element. *<element-type>* is the numeric code 1, 2, or 3, as described above. Note that different versions of an element can have different types.

See sample trigger script **addheader.pl** in the **examples** subdirectory of the AccuRev installation directory.

- The parameters file for a **pre-promote-trig** script ends with a series of lines, one per element to be promoted:

```
<pathname-of-element>
```

<pathname-of-element> is not a full fleshiest pathname, but starts at the workspace’s top-level directory (which is included earlier in the parameters file).

A **pre-promote-trig** script can overwrite its parameters file, in order to communicate with a **server-post-promote-trig** script: the *first line* of the overwritten parameters file becomes the value of the **from_client_promote** parameter in the **server-post-promote-trig** script.

See sample trigger script **client_dispatch_promote_custom.pl** in the **examples/dispatch** subdirectory of the AccuRev installation directory, along with **server_post_promote.pl** in the **examples** subdirectory.

A trigger script can also send data to STDOUT and STDERR. If the command for which the trigger fired was executed in the AccuRev CLI, this data appears in the user’s command window. If a GUI command caused the trigger to fire, the script’s exit status determines whether the user sees the STDOUT/STDERR data: in the “failure” case (non-zero exit status), the data is displayed in an error-message box; in the “success” (zero exit status) case, the data is discarded.

Trigger Script Execution and User Identities

When a trigger script executes on a client machine, it runs under the identity of the AccuRev user who entered the command. Since the user himself is registered (i.e. has a principal-name) in the AccuRev user registry, there won't be any authentication problems if the trigger script runs AccuRev commands that access the repository.

When a trigger script executes on the server machine, it runs under the user identity of the AccuRev Server itself. We recommend that the server run as user **acserver** or **System** (see *User Identity of the Server Process* on page 7), a user that should not be in AccuRev's user registry. If the trigger script runs AccuRev commands, it needs to have an AccuRev user identity. To accomplish this, set variable ACCUREV_PRINCIPAL to an appropriate principal-name (AccuRev username) in the environment of the **acserver** or **System** account. Alternatively, set this environment variable in the script itself. For example:

```
$ENV{ACCUREV_PRINCIPAL} = "jjp";
system("accurev hist ...");
```

'Administrative Users' in Trigger Scripts

The sample Perl trigger scripts supplied by AccuRev provide a very simple implementation of the "administrative user" concept: a user is permitted to perform certain operations only if his username is recorded in the **administrator** hash defined in the script:

```
$administrator{"derek"} = 1;
$administrator{"allison"} = 1;
...
if ( ! defined($administrator{$principal}) ) {
    print TIO "Execution of '$command' command disallowed:\n";
    ...
}
```

The Trigger Log File

When a trigger script runs on the AccuRev server machine — for a **server-post-promote-trig**, **server_preop_trig**, or **server_admin_trig** trigger — an invocation line is written to file **trigger.log** in the **logs** subdirectory of the repository's **site_slice** directory:

```
##### [2004/06/28 20:50:42] running: "C:\Program Files\AccuRev\bin\pst_pro.bat" ...
```

If the script produces console output (STDOUT and/or STDERR), this output is also sent to the **trigger.log** file.

As with other server log files, the **trigger.log** file is "rotated" periodically, to keep active logs from growing too large.

The ‘maintain’ Utility

This document describes AccuRev’s **maintain** utility, an administrative tool for occasional use under the guidance of an AccuRev, Inc. Product Support representative. The **maintain** utility is a non-interactive command-line tool, with a simple command structure. For example:

```
maintain reindex <depot-name>
```

The **maintain** program is located in the AccuRev **bin** directory. If the command-line client program, **accurev**, is on your search path, then so is **maintain**.

Each of the **maintain** commands is described in the next section. Following that are sections providing a more discussions of several of the commands.

‘maintain’ Command Reference

backup

Along with **reindex** and **restore**, provides a facility for backing up and restoring the AccuRev repository. See *Backup/Restore of the AccuRev Repository* on page 61.

chpasswd

```
maintain chpasswd <user> <new-password>
```

Changes the password stored in the AccuRev repository for an existing principal-name (named AccuRev user). To remove a user’s password, use two consecutive double-quote characters as the *<new-password>* parameter:

```
maintain chpasswd derek ""
```

For consistency, the user should invoke the **accurev setlocalpasswd** command to change his “local password” on each client machine that he uses. A user’s local password is stored in file **authn** in subdirectory **.accurev** of the user’s home directory.

chuser

```
maintain chuser <user-ID> <new-username>
```

Changes the principal-name (AccuRev username) of an existing user. You specify the user by the unique numeric user-ID, which is immutable. This command is similar to the **accurev chuser** command.

dbcheck

```
maintain dbcheck
```

Checks the consistency of the streams database (**streams.ndb**) in the **site_slice** directory, and checks the consistency of all the database (**.ndb**) files in all depots.

Note: be sure to stop the AccuRev Server process before running this command.

export

```
maintain export
maintain export <depot-name>
```

Produces an ASCII dump of the repository's site slice directory, or an ASCII dump of the specified depot.

loc128

```
maintain loc128 <depot-name>
```

Changes the maximum length of a pathname segment (the simple name of a file or directory) to 128 characters, for the specified depot.

Use this command to “upgrade” depots created with earlier versions of AccuRev; in later versions, all depots are created with the 128-character maximum for pathname segments.

migrate

```
maintain migrate
maintain migrate <depot-name>
```

Converts the specified depot from little-endian to big-endian format, or vice-versa. If you omit the *<depot-name>* argument, it converts the **site_slice** directory. In all cases, the conversion is non-destructive: the data structure itself is not modified or deleted; the converted data is written to a subdirectory named **swapped** within the **site_slice** directory or depot directory. Converting the “endian-ness” consists of reversing the order of the 8-bit bytes in each machine-level word. For a step-by-step conversion procedure, see *Moving the AccuRev Server and Repository to Another Machine* on page 37.

reindex, restore

Along with **backup**, provides a facility for backing up and restoring the AccuRev repository. See *Backup/Restore of the AccuRev Repository* on page 61.

rmdepot

```
maintain rmdepot <depot-name>
```

Removes a depot from the AccuRev repository. All streams, snapshots, and workspace streams are also removed from the repository. (Workspace trees are *not* removed.) For details, see *Removing a Depot from the AccuRev Repository* on page 62.

timeshift

```
maintain timeshift <depot-name> <earlier-trans-#> <later-trans-#> <seconds>
```

Adds a number of seconds to the timestamps of a range of transactions, for the specified depot. The *<seconds>* parameter must be an integer; use a negative integer to shift timestamps backward. All transactions in the specified transaction range (inclusive) are time shifted.

This command refuses to make any change if any *shifted* transaction would move past any *unshifted* transaction. That is, the command refuses to change the order of transactions in the repository.

vercheck

```
maintain vercheck [ -c | -q ] [ -e <eid> ] <depot-name>
```

Checks the storage containers in the specified depot's **data** directory tree, to verify that a storage container file (**.sto**) exists for each file version recorded in the depot database. It also reports occurrences of “crc mismatch” problems: the actual checksum (CRC) of a **.sto** file does not match the checksum recorded in the corresponding rapid recovery file (**.rrf**).

In addition, you can correct “crc mismatch” problems, using these options:

- **-q** option: for each file with a “crc mismatch”, (step 1) compute the checksum of the **.sto** file, and (step 2) replace the “c:” value in the **.rrf** file with this newly computed value.
- **-c** option: like **-q**, start with this step for each file: (step 0) change the **.sto** file by removing all its CR characters — that is, all bytes with the value **0x0D**.

You can restrict the processing to versions of a particular element with the **-e** option. **vercheck** fixes all the versions of the element that have a “crc mismatch” problem, leaving other versions as is.

Backup/Restore of the AccuRev Repository

An **accurev** command (**backup**) and two **maintain** commands (**restore** and **reindex**) are involved in the scheme for backing up and restoring the AccuRev repository with a minimum of disruption to development activities.

The command **accurev backup mark** declares a “checkpoint” of the entire AccuRev repository, by:

- Copying all the database files (**.ndb**) and the index file **stream.ndx** from the **site_slice** directory to a subdirectory named **backup**.
- Recording the current state of each depot's database files in file **valid_sizes_backup** in the depot directory.

This is an **accurev** command — it can be executed while the AccuRev Server process is running and development activities are ongoing. The **backup mark** command does not actually copy any files to a backup medium; it just marks a consistent state of the repository files. After executing this command, you can back up the repository files, in any order, and on any schedule, while users continue to use AccuRev. For more information, see *Backing Up the Repository* on page 3.

At any time after you've executed an **accurev backup mark** command and copied the repository files to a backup medium, you can restore the repository to its state at the time the **backup** command was executed. This is an offline procedure — the AccuRev Server must be stopped when you run it. The procedure is documented in section *Restoring the Repository* on page 4. The **maintain** commands involved are:

restore

```
maintain restore <depot-name>
```

Rolls back all the database files (**.ndb**) in the specified depot to their state at the time of the **accurev backup mark** command. This is a “logical truncate” operation — a file-system truncate is *not* performed on the database files.

reindex

```
maintain reindex [ <depot-name> ]
```

Reads various database files (**.ndb**) and recreates the corresponding index files (**.ndx**). This operation is performed on the database files of the specified depot; if you omit the *<depot-name>* argument, the operation is performed on the repository-wide database files in the **site_slice** directory

Removing a Depot from the AccuRev Repository

This section describes a procedure for removing a depot completely from the AccuRev repository. Removing a depot:

- Deletes every version of every file and directory in the depot.
- Deletes the entire history of the depot — all transactions involving the depot and its elements.

Removing a depot does not affect any of the workspaces or reference trees that contain copies of the depot’s elements.

Before You Begin

We strongly recommend that you preserve a backup copy of the AccuRev data repository before deleting any depots. See *Backing Up the Repository* on page 3. Much of a depot’s data is stored in its slice of the repository. Use the command **accurev show slices** to determine the pathname of a depot’s slice; you’ll need it in Step 3 below.

Depot Removal Procedure

The following procedure must be performed on the machine where the AccuRev repository resides.

1. Stop the AccuRev Server process:
 - Unix: use the **acsverctl** utility, located in the AccuRev **bin** directory:

```
acsverctl stop
```
 - Windows: use the **Services** applet, or enter the command **net stop accurev** in a Command Prompt window.
2. Execute the **maintain** program’s remove-depot command. This utility program is located in the AccuRev **bin** directory.

```
maintain rmdepot <depot-name>
```

For safety, the **rmdepot** command goes through two confirmation steps, including having you retype the depot name. The command displays some messages, ending with:

```
Site reindex complete.
```

The **rmdepot** command completely removes the depot's records from the repository database in the **site_slice** directory. (There's one exception: the depot's name remains in the database. See Step 5.)

3. Remove the depot directory subtree (slice) from the AccuRev **storage** directory tree. Be careful not to remove any other depot's directory subtree! If you're not sure where the depot's slice is located, use the command **accurev show -fi slices** to determine the pathname. (This command requires the AccuRev Server to be running.)

For example, if the depot is named **widget** and AccuRev is installed on a Unix machine at **/opt/accurev**, use this command:

```
rm -r /opt/accurev/storage/depots/widget
```

4. Restart the AccuRev Server process:
 - Unix: use the **acservctl** utility, located in the AccuRev **bin** directory:

```
acservctl start
```
 - Windows: use the **Services** applet, or enter the command **net start accurev** in a Command Prompt window.
5. At this point, the depot's name remains in the AccuRev repository database. It won't appear in an **accurev show depots** command listing; but you *can* make it appear in a GUI listing of the repository's depots. If you wish to reuse the removed depot's name, perform a **rename-depot** command. For example:

```
> accurev chdepot -p widget deleted_widget
Unknown stream or ver spec: widget
Unknown stream or ver spec: widget
Changed name of depot widget to deleted_widget .
```

At this point, you can create a new depot named **widget**, which will be completely unrelated to the depot you removed.

