Caring for Your Rational ClearCase VOBs

Mark Zukowsky July 15, 2002

from The Rational Edge:The material in this article is directed at more elementary Rational ClearCase users; it's intended to help you perform "autopsies" when something goes wrong with a VOB (versioned object base). After some introductory comments about VOBs, it looks at what can go wrong with them, how to recognize when something's gone wrong, and how to minimize VOB problems through preventive maintenance.

This article is based on one of the presentations I gave at the Rational User Conference (RUC) in 2001, to help Rational ClearCase users. The material presented here is directed at more elementary Rational ClearCase users; it's intended to help you perform "autopsies" when something goes wrong with a VOB (versioned object base). After some introductory comments about VOBs, I'll look at what can go wrong with them, how to recognize when something's gone wrong, and how to minimize VOB problems through preventive maintenance.

Another presentation I gave during a session about advanced change management has been turned into the Rational Developer Network article, ***Avoiding Common Problems in Rational ClearCase****. The second article is designed for more advanced users.*

Introduction to VOBs

The VOB storage directory has numerous subdirectories in it. The four main areas of VOB storage are source pools, cleartext pools, derived object pools, and the database. Things can go wrong in any of these areas.

The VOB database (the db subdirectory) is one of the more important parts of the system (along with the source pools, which are in the s subdirectory). Taking a closer look at the database, we see that it uses the Raima proprietary format (where Raima is a company that's gone out of business twice since we started using them). There's one database per VOB; on a VOB server, no matter how many VOBs you have, there will be just one database in each VOB storage directory. The database consists of eight major files:

- three data files (.d01 through .d03), which contain the actual data
- four key files (.k01 through .k04), which contain indexes for random access into the data files
- a string file (.str_file), which contains configuration records, among other things

There's also a schema, which defines how the database is organized internally.

Possible Problems and Their Causes

Three main types of things can go wrong:

- Corruption in source containers
- VOB accesses not working
- Data loss or corruption in the VOB database

Source containers get corrupted for a variety of reasons, usually network-related. These reasons include: NFS problems that create zeroes in the middle of the files; network problems such as fragmentation or reassembly errors for large NFS packets; or faulty NICs (network interface cards).

VOB accesses can stop working because you ran out of disk space or hit an OS file size limitation (2 GB on some systems) or internal database limitation (particularly in ClearCase 3.0 or schema version 53; more on schema versions later). If you encounter one of these problems, you're in big trouble, because you won't be allowed to write anything else into the VOB.

The reasons for data loss or corruption in the VOB database include:

- Hardware-induced failures (such as disk failures or RAID failures, throwing zeroes in the middle of the files or swapping things around a bit). These are sometimes just single-bit errors. We've seen memory checksum problems where a single bit has been flipped in the middle of the data file.
- "Pilot error." For example, if someone attempts to clone a VOB by copying it into the same region, that can cause problems. We've also had people try to remove individual database files to save space. Having a remote database storage location can also be an issue, but it's less so now that we support filers (discussed later).
- Software-induced failures, from either the operating system or ClearCase (in Raima). The last Raima problem was reported four years ago; however, there was an issue last year with multiple derived object pools on HP systems, where we could actually go in and corrupt a database - not corrupt the contents, but mess up the records a bit so that certain information couldn't be referenced. We were able to provide a fix for that.

Detecting Problems

Now let's take a look at how you can recognize when one of the above types of problems has occurred. Table 1 lists the sections that follow and the possible problem areas that might be uncovered as described within them.

Table 1: Sections on detection mapped to problem areas

Section on detection	Possible problem area(s)
Detecting Data Loss or Corruption	VOB accesses not workingData loss/corruption in database
Detecting When a VOB Is Near a Limit	VOB accesses not working
Detecting Source Container Integrity Problems	Corruption in source containers Data loss/corruption in database
Detecting Problems with checkvob	Corruption in source containers

Detecting Data Loss or Corruption

Symptoms of having a corrupt VOB database include the following:

- You're unable to access the VOB or VOB objects. (cleartool commands fail, reporting an error like a database ID not found.)
- Either scrubber or vob_scrubber fails.
- You can't lock the VOB (which in general means you can't do any write transactions to the database).
- You can't replicate (or import replica packets). This indicates possible database corruption, but it might also just indicate divergence, depending on the error logs.
- Reformatting the VOB fails. In this case, the database is almost certainly corrupt.

The only two processes that talk to the VOB database are db_server and vobrpc_server. In their ClearCase log files, you'll see messages like these in the event of data loss or corruption:

- Unable to open VOB database - This isn't critical. It usually means you've tried to move or copy the VOB without maintaining the proper permissions on a subset of files in the VOB storage directory tree).
- db_VISTA error *nnn*- You'll see error numbers like -922, -909, and -6. Note that db_VISTA is essentially synonymous with Raima; anything that's a db_VISTA error is a Raima problem, although not necessarily a corruption. Raima could be having problems with the system, as in not being able to see it, read it, or write to it. This message can also mean that the API being used in ClearCase to interact with Raima is returning an error code.
- Internal error in VOB database, Unexpected error in VOB database, or Something not found in VOB database (basically, "I don't know what you're looking for, but it's not there") - If you see one of these not-so-useful messages, please call Rational Technical Support right away; don't wait a few months or ignore it altogether.

Detecting When a VOB Is Near a Limit

A VOB can stop working if it's near a limit. (I talk about this in more detail in the article *Avoiding Common Problems in Rational ClearCase*. To help you detect how full a VOB database is, two utilities are available from Technical Support: countdb analyzes the data (.d01-.d03) files, and string_report analyzes the string file.

If you see messages in ClearCase log files indicating db_VISTA error -909 or (during a clearmake) 2, that means you're approaching one of the VOB file's limits.

Detecting Source Container Integrity Problems

Missing or unreferenced source containers will be detected by checkvob. If you use VOB snapshots to back up your VOB, you'll need to run checkvob after restoring the backup, to make sure the database and source pools are in sync.

The problem of corrupted source containers is usually detected from user-level errors; specifically, cleartool checkin or checkout will fail. If you find you have a bad container, you can copy the equivalent container from another replica in the VOB family (if you have one), and then run checkvob to make sure everything's healthy.

In addition, we ship a utility called ck_all_tfd_for_nulls.pl in the etc/utils subdirectory that enables you to examine text file delta containers in a cursory way. This script will look for a zero byte in the middle of every text file delta container, which would indicate the presence of binary data in the middle of the file (meaning the file is bad). This utility won't fix anything for you - we don't have any good way of fixing containers - but it will at least give you an idea of whether a container is healthy. If it's not, you'll have to fix it manually, by either finding the container and backing it up or copying it from another replica.

Detecting Problems with checkvob

I've already mentioned that checkvob will detect missing or unreferenced source containers. It will also detect VOB problems like inconsistencies between the VOB database and storage pools; if the VOB database says there has to be five versions of the file, the source container better have five versions as well.

Fortunately, checkvob will fix problems whenever possible. For example, it will fix hyperlinks pointing to nonexistent objects, so if you remove an admin VOB, it will fix that (assuming the VOB is unlocked). It will also detect and fix problems with global types stored in admin VOBs.

Since checkvob isn't run by default in ClearCase, you should run it if you're restoring a VOB from backup, and you must run it if you're using VOB snapshots to back up your VOB or if you copy the database separately from the source containers during your backup. You can also run checkvob regularly to look for inconsistencies; once a week seems to be a reasonable time frame.

Minimizing the Impact of VOB Problems

Now we'll explore how you can minimize the impact of any problems you're noticing (although not necessarily fix them) and how you can prevent them by running various utilities to help keep things working right. Table 2 shows how the sections that follow are organized and the related problem areas for each.

Table 2: Sections on minimizing impact mapped to problem areas

Section on minimizing impact	Related problem area(s)
Minimizing Exposure to Data Loss • Making Backups • Checking for Corruption • Scrubbing	By subsection: • Corruption in source containers; data loss/corruption in database • Data loss/corruption in database • VOB accesses not working
Minimizing Problems with VOBs Approaching a Limit • Upgrading to Large VOB Support" • Working Around a Full Database • Accelerating Reformatting • Testing Reformatting	VOB accesses not working

Minimizing Exposure to Data Loss

The two main ways to minimize your exposure to data loss are to make backups and to maintain VOBs by periodically running various checks, such as the following:

- Checking for corruption, primarily by running dbcheck regularly. (This is the most important maintenance you can do.)

- Scrubbing, to free up disk space in the case of containers (and also to remove minor events for deleted objects that are no longer present in the VOB database).
- Monitoring the log files and any problems that users report.
- Running countdb and string_report to see if you're approaching a limit (in either the OS file size or the number of records in a file).
- Monitoring disk space usage.

These techniques won't necessarily prevent things from going bad, but they'll help you find problems as quickly as possible.

I'll first discuss backups and then elaborate on some of the more useful maintenance techniques.

Making Backups

The most important thing about backups is to actually do them. There are customers who don't think they need backups, but they do. Next in importance is to do the backups correctly. At an absolute minimum, you should back up the following: any files in the main directory (*vob-storage*)

- all of the source containers (*vob-storage*/s)
- the entire database subdirectory (*vob-storage*/db)

There are a lot of different methods for backing up VOBs. Here are some of them along with their advantages and disadvantages:

- **Rational ClearCase MultiSite** - This is the most cost-effective method (you knew I'd say that!). You can use MultiSite one-way from your main VOB to a backup replica, which has the advantage that you never have to lock the VOB. In addition, everything is handled appropriately by the syncing; your backups are as up to date as your last sync. On the other hand, restoring from backup involves recreating the replica, which has the disadvantage that views referencing the old VOB can't access those references in the new VOB. In a moment we'll look at a utility that will help with this, actually switching everything around for you. One other slight disadvantage to using MultiSite is that not everything gets replicated, so things like nonversioned derived objects and triggers wouldn't be backed up.
- **Mirroring** - The advantage of this method is that VOB lock time is minimized. However, during the time the mirror is broken and you're backing up from it, you've got reduced redundancy on the database. There's also the disadvantage of disk cost, but that's minimal these days.
- **VOB snapshots** - On the plus side, the VOB is locked for a minimal amount of time if you use vob_snapshot for backup. During a snapshot, we lock the VOB, copy the database files from the db subdirectory, unlock the VOB, and copy the source pools from the s subdirectory; we can then back up the database files and source pools that were copied. Since the VOB is unlocked, the source pools can change on you, which is why you need to run checkvob if you do restore from that backup; checkvob will make sure the database and source pools are in sync.
- **Filers** - Support for this method is new this year. A filer is a network-attached storage device, and the VOB storage directory itself would be on it. I don't know much about the procedure, but apparently you can create snapshots of any file on the filer and back up from that; the

database should be locked when that snapshot is made. The possibility of inconsistent backup is small but, on the down side, there's the cost of purchasing the filers.
- **Your choice of copy programs** - This can be, for example, cpio on UNIX or Backup Exec on Windows NT. Note that the VOB has to be locked when you're copying the database file. Also, the utility you use must be able to back up open file handles and preserve file permissions. The former is more of a problem on Windows NT; I believe that by default Backup Exec won't back up open file handles but rather will require you to go into Control Panel and set it to do that. Even though you're not writing to the VOB, the VOB files are still open.

As mentioned above, if you use ClearCase MultiSite and you then need to recreate the replica from your backup, any view that points to the old replica won't be able to access the new replica. In this case, cleartool recoverview won't help you. However, we started shipping a utility in ClearCase 4.0, called view_sr (where sr stands for "switch replica"), that will go through the entire view database and switch every reference from the old replica to point to the new replica. Any checkout that happens to make it to the backup replica before you restore from the backup will be preserved; it will still exist in the view. Any checkout that hasn't made it over (that is, you did a checkout but you never had a chance to sync) won't be known in the view after this utility is run, but it will become view-private and will eclipse checked-in versions. All other view-private files will be preserved. You won't lose anything, either; anything view_sr can't figure out what to do with will go into the lost-and-found areas.

Some additional notes on view_sr:

- Under no circumstances should you use the old replica after running view_sr. That's equivalent to cloning a VOB by copying it into the same region.
- You can use view_sr in two cases. Let's say replica A is your production VOB and replica B is your backup VOB. When replica A is lost, you can use view_sr in one of two ways to get up and running again. The first method is to move replica B to the original server, run view_sr, and start using replica B as your production VOB; you can then use mkreplica to create a new backup replica C. The second, simpler way is to use mkreplica at replica B to create a new replica, A2, to replace the original replica; you would then run view_sr against replica A2 to use A2 as the production VOB.
- You need to use documented replica recover procedures, which would involve running restorereplica.
- The alternative to view_sr is to replace all the views on the system.

There's a new RPC that the view server needs to respond to that only exists in ClearCase 4.0 and later, so view_sr won't work if your view is stored on a pre-4.0 machine. If you have an earlier version of ClearCase running on the view server, view_sr can't do anything for you, so you'll need to replace the view.

Checking for Corruption

If you think something's wrong in the VOB database itself, not necessarily in the containers, you can use dbcheck, or perhaps reformatvob. Although it's the single most important maintenance you can do, running dbcheck will detect only about 80% of the corruptions that can occur. The

reformatvob utility is much slower but will detect almost all of the remaining possible corruptions (an additional 17% of the total possible). During normal ClearCase use, complaints from users about various cleartool commands will detect any other corruptions in the database (that is, the remaining 3%).

We've shipped dbcheck in every version of ClearCase, in etc/utils. It does a structural integrity check on the VOB database files, but only on the three data files (.d01-.d03) and four key files (.k01-.k04); it doesn't do anything with the string file. dbcheck is concerned only with an individual record and a database. For example, if you have five versions of an element, there's one element record and five version records in the database; dbcheck is concerned only that each of the version records is findable, and not whether there are five of them or whether they all belong to the same element. On the other hand, reformatvob cares more about the interactions between the records (which accounts for the difference in what types of corruption these two tools will detect).

It can take several hours to run dbcheck on large databases, and the VOB must be locked while you're running it (otherwise you'll get false error conditions - pages and pages of output, probably none of it valid). If any true errors surface, please contact Technical Support; we treat these problems seriously, and we'll let you know if we can fix things for you.

To avoid downtime spent running dbcheck on a production database, you can run it on a backup copy of the VOB database. You can either run it on a recent backup of the VOB or lock the VOB, copy the database files to a temporary area, unlock the VOB, and run dbcheck on the copy you made in the temporary area. You don't need the VOB to be registered or tagged; dbcheck is just running on the flat files in the database. You do, however, need read-write access to the database files. (This is a Raima quirk, where everything Raima does has to have write access, even though dbcheck won't actually modify the database.) You also need to be in the directory where the database files reside; if you've copied them to a temporary location, cd to that location and then run dbcheck. To enable the process to run faster, you should run dbcheck local to the machine where the database resides.

The syntax for running dbcheck is illustrated in this example:

```
dbcheck -a -k -p8192 vob_db
```

- The -a and -k options combine to tell dbcheck the maximum amount of error checking it can possibly do. It will check all the data files and key files as well as the internal delete chains, which store unused space in the data files. It will also make sure all the index files are sorted properly (and the like).
- The -p option tells dbcheck how many pages (4096 bytes) of memory to allocate to the process. Note that there's no space between the p and the number following it; dbcheck is picky about this. The maximum number is 32766, and specifying larger numbers will generally speed up the process considerably, assuming there's enough memory on the machine. For the maximum of 32766, you'd need about 134 MB of RAM to run the process. If you have a relatively small database - say, under 100 MB total - increasing the number of pages won't help you; in fact, it will slow things down slightly; on a medium-sized database, the slowdown will be about 16 to 19 minutes. We've found 8192 pages to be an optimal size for small to

medium-sized databases (but we haven't played around much with testing this). On a large database, increasing the number of pages can speed up the process by about 35%-40%.

Finally, you should make sure that when you run dbcheck it's actually doing something. Occasionally the parameters get screwed up - for example, if you run with space between the p and the number following it - so you should examine the output of dbcheck to confirm that it's running correctly. The output should say Processing data file (or key file) for each of the seven files. A clean dbcheck that's gone through all the files successfully will end with the following line:

0 errors were encountered in 0 records/nodes

A problem will cause output like the following:

Problems at node 18:
* key field CONTAINER_DBID(71) error:
slot 15's record-dba=[0:1731] has invalid record-id and/or inconsistent dba

If dbcheck does reveal problems with your database, you can restore it from backup or recreate the replica. Again, we encourage you to report the problem to Technical Support. We'll ask you to send the dbcheck output and possibly also a copy of the database, depending on how bad the problem is. If we request a copy of the database, we want only the db subdirectory and not any containers or source code; that will give us access to host names, user names, and comments, which is all we're interested in. If you see only one error in dbcheck and most things seem to be working OK, we suggest you lock the VOB and send us a copy; you can then continue working in the VOB in any place that doesn't appear to be problematic. Rational Engineering will examine the database and, if possible, we'll send you a tool specific to your database that will fix it. However, note that in general, unless it's a known defect that we've seen before or an easily fixed corruption such as a single-bit problem, recovering from a backup or recreating the replica will usually be faster than going through Engineering.

Scrubbing

Scrubbing on a regular basis will save on disk space and prevent encounters with internal database limitations. It removes unnecessary data from the database (old configuration records, events, and oplogs), from the derived object pools, and from the cleartext pools (cleaning things up that haven't been used for a while).

The two processes you can use to accomplish this are scrubber, which runs daily by default and cleans up the configuration records and the pools, and vob_scrubber, which runs weekly by default and scrubs the database events and oplogs.

Specifically, scrubber does the following:

- It removes cleartext containers that haven't been accessed for some time. You can modify the various parameters; for example, the age parameters enable you to save cleartext for a longer amount of time, if you have enough disk space. But in general, we'll generate cleartext containers as we need them.

- It removes from the database files any configuration records that aren't being referenced by derived objects. Internally that space will be marked as available for use. Note, however, that no matter what you remove, the database files will never shrink. The only way to shrink a database is to run reformatvob.

vob_scrubber cleans up the events in the database, removing minor events based on how old they are. (There's a list of all minor events in the events_ccase reference page.) It also removes old oplogs based on their age. Oplogs can take up a lot of database space, and the default is to keep them around forever. (This is discussed in more detail in *Avoiding Common Problems in Rational ClearCase*.) If you sync successfully on a regular basis, consider reducing the number of stored oplogs to only those six months old or less. You don't need oplogs in order to create new replicas with mkreplica; mkreplica will essentially take a snapshot of the current database.

Minimizing Problems with VOBs Approaching a Limit

As mentioned earlier, it's a good idea to run countdb and string_report to see if you're running into a limit (in either the OS file size or the number of records in a file). Here we'll look at how to reduce the likelihood of that happening by installing the support necessary for large VOBs. I'll also discuss how to work around the problem if you do end up having a full database, and how to speed up VOB reformatting, which can take a prohibitively long time with large databases.

Upgrading to Large VOB Support

Support for large VOBs is currently an installation option in ClearCase version 4.0 and later on Solaris, HP, and Windows NT systems. (It's under investigation for other platforms.) This option allows database files to grow past 2 GB, which is a limitation of the operating system calls used in earlier versions. It also allows more records to be stored in each database file: 2^{56} instead of just 2^{24} (about 16 million) records.

A VOB database uses schema 54 for large VOB support and schema 53 otherwise. During installation, you can (in response to a prompt) specify that you want to install schema 54 for large VOB support. The command

cleartool describe vob:*vob-tag*

will tell you which schema you're using.

On any one VOB server machine, every VOB has to be at the same schema level in order for you to use it. So if you upgrade to schema 54, you have to reformat each of the VOBs individually.

In addition, all replicas in a family should make the transition to schema 54 at the same time. If one replica goes over a limit - either the 2 GB file size or the 16 million record limit - then every other replica that needs to import that packet must also be able to exceed these limits. Things like derived objects take up a lot of room in the string file, and many people need to upgrade to schema 54 because of all the configuration records. Since derived objects don't replicate, the string file at one site may be 2 GB while the string file at another site that doesn't do any builds

may be only 100K. If you want to upgrade the 2 GB one to schema 54, you can leave the other one at schema 53, and syncing will continue to work.

Note that schema level does not equal feature level; you can mix and match. You can have either schema 53 or schema 54 at feature level 1 or 2.

When deciding whether to upgrade to large VOB support, keep in mind that it will result in quite a bit of downtime. As already mentioned, you'll need to reformat all the VOBs on the server. Each one is accessible to your clients as soon as you reformat it, but if you have, say, 50 VOBs, it could take a lot of time before they're all accessible. Some people create a secondary server that uses schema 54 and then move the VOBs over to it one at a time and do the reformatting there; this enables all VOBs except one to be accessed at any given time.

The same database will be about 35% larger under schema 54 than schema 53, and it will take more time to seek across the disk. Even though schema 54 allows the files to grow as large as you want, enabling you to work past the usual limits, you should still be monitoring the string file (using string_report) and the countdb output to make sure things are being cleaned up properly.

Note that the 35% size increase is an estimate for the average VOB. If you haven't reformatted yet for schema 54 and you'd like a closer estimate of what the size increase will be for one of your VOBs, contact Technical Support. We have a tool called vob_size that will run on a schema 53 database and estimate how big it will be when reformatted it to schema 54.

Working Around a Full Database

If you have a full database file and haven't upgraded to schema 54, doing that upgrade for large VOB support (if it's available on the platform you're using) is clearly one way to work around the problem. Another way is to split the VOB using cleartool relocate, although this won't work if you're using UCM. The reason for this restriction is immutable baselines: if an element has ever been in a baseline, it can never be moved to another place; the baseline needs to know where to look for it. So a UCM element really can't be relocated.

If you have a full database file with a lot of oplogs in it, you can run vob_scrubber with more aggressive oplog parameters. If the full database has a lot of labels in it, you need to remove some of them; if it has a lot of other stuff in it, you can contact Technical Support for recommendations.

In the case of a full string file, you should remove derived objects and run scrubber to clean up unreferenced configuration records. This is covered in more depth in *Avoiding Common Problems in Rational ClearCase*.

Accelerating Reformatting

As mentioned earlier, dbcheck catches the majority of corruptions but won't detect all of them; reformatvob will catch more, but it's slow. For large databases, reformatvob can take nine or more hours, and the VOB is inaccessible while you're doing it. Under those conditions it wouldn't be practical for you run reformatvob as a periodic check to determine if the database healthy. There

is, however, an environment variable (named CCASE_LOADER_NEW_CACHE_SIZE) that you can set to speed up part of the reformatvob operation. Its default value is 4096, and (similar to dbcheck) you can increase it to 32766. This will speed up only the load (not the dump phase) by a factor of about 2, which will cut down the reformat phase from about nine hours to roughly five and a half.

Note that on UNIX, this environment variable has to be set in the atria_start script (the script that kicks off the actual load and dump processes), so that any child processes the ALBD server initiates will have it set. On Windows NT, you need to set this variable in the system environment variable area and then restart ClearCase.

Testing Reformatting

To close, I'll describe a procedure that will allow you to carry out the steps reformatvob would perform, but without actually running reformatvob. You can do this as a test to make sure the database is healthy before doing a real upgrade. This will also enable you to determine approximately how much time reformatvob is going to take.

You'll need to run this procedure as root. Also, you'll need enough disk space to dump and load the database - about three times the current size of the VOB database (for the copy of the current database, the dump files, and the copy of the new database).

1. Get a copy of the VOB database in either of these ways:
   - Lock the VOB, copy the contents of the db subdirectory to a temporary location (let's assume /tmp) and unlock the VOB.
   - Alternatively, if you have a backup copy of the VOB, simply copy that backup into the temporary location.

   Note that you only need a copy of the database files; the VOB doesn't need to be tagged or registered.
2. Using cd, change directory to /tmp.
3. Within /tmp, create a directory - named dumpdir, for example - in which to perform the database dump.
4. Change directory to /tmp/dumpdir and run the command /usr/atria/etc/dumpers/db_dumper.53 .. where:
   - .53 denotes the schema version and so should be replaced with the schema version of the existing database if it's not 53. If you've installed support for large VOBs, you'll use db_dumper.54. If you happen to be running V2, my condolences, but you can use db_dumper.38 as well.
   - On Windows NT, these utilities will instead be in *atria-home*/bin/dumpers. ***

   This command will print information as it dumps the reformatted database. The output you're concerned with is the phrase Dumper done, reported at the end of the process to indicate that the dump phase was successful. If you don't see the Dumper done output, please contact Technical Support, letting them know of any database errors that were reported during the

dump. In general, if the dump phase fails, you'll need a copy of the database and you'll run the dumper in the debugger to determine what problems exist.
Assuming the dump phase worked, it will create three ASCII files in /tmp/dumpdir. Next comes the load phase.

5. Optionally, set the CCASE_LOADER_NEW_CACHE_SIZE environment variable, as described in the previous section. You can do this from the command line now, because you're going to start the load phase directly rather than use the ALBD server to kick off the process. If you'd like to see how much time setting the environment variable will save you, you can run the loader once without the variable set and then again after setting it.
6. To run the loader, create a new directory - say, newdb - within /tmp/dumpdir as the location for the new VOB database.
7. From within the /tmp/dumpdir directory, run the command
   /usr/atria/etc/db_loader newdb
   This will take those three ASCII files, read from them, run the loader, and create a new set of VOB database files. The output of interest is Loader done; if it doesn't appear (or if you encounter any other problems during this procedure), please contact Technical Support.

If this procedure succeeds, reformatvob will work on the live VOB; if something fails here, reformatvob will fail in the same manner.

Once you're done, you can delete the copy of the database you started with, the temporary ASCII files, and the new copy of the database that was created.

Summary

Things can and will go wrong with Rational ClearCase VOBs. For a variety of reasons, you could have corruption in source containers, VOB accesses not working, or data loss or corruption in a VOB database. I've discussed how to recognize when one of these types of problems has occurred and how to minimize the impact. Please keep in mind that Rational Technical Support will try to help you as much as possible.

References and Other Resources

- Avoiding Common Problems in Rational ClearCase by Mark Zukowsky
- ClearCase VOB Database Troubleshooting by Carem Bennett

***NOTE:** This article was originally published on Rational Developer Network, the learning and support channel for the Rational customer community. If you are a Rational customer and have not already registered for your free membership, please go to www.rational.net.