# Transactional VSAM: An Application Programmer's Perspective

ZSeries Technical Conference
Session TSS05
Ruth Ferziger
ruthf@us.ibm.com

## What is Transactional VSAM?

The objective of Transactional VSAM is to provide transactional recovery directly within VSAM. It is an extension to VSAM RLS. It allows any job or application that is designed for data sharing to read/write share VSAM recoverable files.

Transactional VSAM is a follow-on project/capability based on VSAM RLS (record level sharing). RLS provides a sysplex-wide server for sharing VSAM files. It provides CF (coupling facility) based locking and data caching with local buffer cross-invalidate. RLS supports CICS as a transaction manager. This provides sysplex data sharing of VSAM recoverable files when accessed through CICS. CICS provides the necessary unit-of-work management, undo/redo logging, and commit/back out functions. VSAM provides the underlying sysplex-scope locking and data access integrity.

Transactional VSAM adds logging and commit/back out support to VSAM RLS. Transactional VSAM requires/supports the OS/390 RRMS (Recoverable Resource Management Services) component as the commit or sync point manager.

Transactional VSAM provides a level of data sharing with built-in transactional recovery for VSAM recoverable files that is comparable to the data sharing and transactional recovery support for data bases provided by DB2 and IMSDB.

## Transactional VSAM Overview

The transaction program execution model provides data sharing of recoverable resources. During the life of a transaction, its changes to recoverable resources are NOT seen by other transactions. The transaction may request that its changes be rolled back (backed out). If the transaction fails, its changes are backed out. This capability is called transactional recovery. It is provided by the resource managers.

Applications that are designed to the transaction model are able to easily share the recoverable resources. The resource managers provide the sharing isolation and recovery when a transaction fails or when the execution environment fails.

IMSDB and DB2 are resource managers that provide transactional recovery for their databases. CICS File Control provides transactional recovery for VSAM recoverable files. Now, Transactional VSAM provides transactional recovery for VSAM recoverable files as well.

## Transactional VSAM Logstreams

Each Transactional VSAM instance has a primary system log (undo log) and a secondary system log (shunt log). Both are implemented as MVS system logger log streams. These logs are intended for use only for recovery purposes--for example, during back out or restart. They are not meant to be used for any other purpose. The primary system log stream holds data for most normal in-flight units or recovery (URs). The secondary system log stream holds information for URs that cannot be completed, normally due to back out failures, or which have been determined to be long running.

System log stream names are generally qualified names, of which the high level qualifier is the Transactional VSAM instance name. Transactional VSAM requires the log stream names to be IGWTVnnn.IGWLOG.SYSLOG and IGWTVnnn.IGWSHUNT.SHUNLOG respectively.

Transactional VSAM instance names must be unique throughout the sysplex. Each Transactional VSAM instance supports the system log for its exclusive use. All other logs are kept separate from the system log. Their stream names are checked to ensure that they are different from that of the system log.

You must define the log streams for the Transactional VSAM primary and secondary system logs to the MVS system logger before starting Transactional VSAM.

Forward recovery logs are kept separate from system logs. Transactional VSAM obtains the log stream name of a VSAM forward recovery log from the ICF catalog entry for the data set. Transactional VSAM only writes to forward recovery logs. It is the responsibility of products which provide forward recovery capability to read them.

If you associate forward recovery log stream ids with VSAM data sets, you must ensure that you define those log streams to the MVS system logger before you open the data sets. If the log stream name is not defined to the MVS system logger, the connect request fails. Note that the log stream definition determines the MVS log stream structure in the coupling facility to which the log stream is written.

If the use of a log-of-logs is requested via the IGDSMSxx member of SYS1.PARMLIB, Transactional VSAM writes a log-of-logs. It contains copies of the start of run records, and the tie-up records and file close records for recoverable data sets, and log stream exception

information. This provides data set recovery products such as CICSVR with the information required for to control forward recovery. If you use both Transactional VSAM and CICS, it is recommended that you use the same log-of-logs for both.

If you do not want Transactional VSAM to write a log-of-logs, omit the LOG_OF_LOGS parameter from the IGDSMSxx member of SYS1.PARMLIB. If you use Transactional VSAM in a sysplex environment, and you run with a log-of-logs, then the log-of-logs should be a single log stream shared by all Transactional VSAM instances that are used to access the same set of recoverable data sets.

## Accessing a Data Set With Transactional VSAM

For the most part, Transactional VSAM only supports those data sets that are defined as recoverable. That is, the log attribute for the data set is either UNDO (back out logging only) or ALL (back out and forward recovery logging). When a batch job opens a recoverable data set for update, the open is done in Transactional VSAM mode. This allows Transactional VSAM to provide the necessary transactional recovery for the data set.

Note that data sets opened for input with the CRE option specified are also open in Transactional VSAM mode. This is because the CRE (consistent read explicit, also known as repeatable read) locks are sync point duration locks. Without the Transactional VSAM support, VSAM RLS would know nothing about sync points, and the locks would never get released.

In either case, read or update access, the application is responsible for defining the sync points by invoking the RRS commit or back out function. It is not possible for Transactional VSAM to define the sync points because it knows nothing about what the application is actually doing. If it tried to imply a sync point every so many operations any of the following could happen:

1. Transactional VSAM could insert the sync point between paired operations (that is, a GET UPD and its paired PUT UPD or ERASE).
2. Transactional VSAM could insert the sync point in the middle of two pieces of work which were meant to be atomic (for example, between subtracting 100 dollars from a checking account and adding it to a savings account).
3. Transactional VSAM could decide to commit a piece of work that the application would have realized should have been backed out (or vice versa).

## Using Transactional VSAM

Transactional VSAM (sometimes abbreviated TranVSAM) permits a batch job to OPEN a recoverable file for OUTPUT. However, most existing batch jobs that modify VSAM files are NOT designed to permit data sharing. Each job assumes the file is NOT being changed by another concurrently executing program.

Transactional VSAM provides the necessary transactional recovery to enable data sharing. Batch jobs that are designed to use the transactional programming model may use Transactional VSAM to read/write shared VSAM recoverable files.

## What Transactional VSAM Does

This is an example of the usage of interfaces to Transactional VSAM. Notice the only change to the already existing VSAM application interface is the new repeatable read option on GET.

Call SRRCMIT invokes the RRS component of RRMS to commit the changes made by the application. RRS interfaces with Transactional VSAM to commit the VSAM file changes and release the corresponding VSAM locks.

If the application also changed other recoverable resources managed by resource managers (like DB2) that support RRMS, the commit applies atomically across Transactional VSAM and the other resource managers.

## What is a Transaction?

This is an example of the usage of interfaces to Transactional VSAM. Notice the only change to the already existing VSAM application interface is the new repeatable read option on GET.

Call SRRCMIT invokes the RRS component of RRMS to commit the changes made by the application. RRS interfaces with Transactional VSAM to commit the VSAM file changes and release the corresponding VSAM locks.

If the application also changed other recoverable resources managed by resource managers (like DB2) that support RRMS, the commit applies atomically across Transactional VSAM and the other resource managers.

## Unit of Recovery

The set of changes processed by a single commit or backout (between commits or back outs, or between the start of the program and the first commit or backout) is called a unit of recovery. A unit of recovery is an indivisible (atomic) entity which permits resource consistency to be maintained.

Change for a single unit of recovery occur atomically. Atomic operations are those that execute in the same manner as machine instructions: either all the changes are made or none are made. The point at which the applications makes the changes permanent is called the atomic instant of commit, or more simply commit or a sync point. If for some reason the applications desires that the changes not be made or an error occurs which makes it impossible for the changes to be made atomically, then the changes are backed out.

We strongly recommend that your not allow your programs to end without committing and that you not convert your applications to use Transactional VSAM without inserting sync points. If you use only the implicit sync points (commit for normal end of step or end of job, and back out for abnormal end of step or end of job) you may end up creating retained locks which can cause problems for other jobs. When these jobs request the retained locks, instead of waiting for the locks, they will receive retained lock rejects, which can only be cleared by the application which caused them committing or backing out.

In addition, not adding commits or back outs to your application can cause the entire program to be viewed as a single unit of recovery. This can cause it to hold a large number of locks and to have a large amount of active data in the undo log. This can have two very negative impacts on your sysplex operations. The first is that by holding a large number of locks, your program or application is locking out any other batch applications or on line systems that need to access those records. This can negatively impact your on line response time. Having a large amount of active data on the undo log can cause it to spill from the coupling facility onto DASD, negatively impacting system performance.

Note: This example is extremely simplified since it leaves out any GET operations, as well as other operations such as POINT and ENDREQ.

## Application Interfaces: VSAM

Transactional VSAM introduces very few changes to the VSAM interfaces. RLS introduced the recoverable/non-recoverable file attributes (LOG and LOGSTREAMID). These attributes are supported by Transactional VSAM.

RLS introduced two read integrity options for batch jobs: NRI - do NOT obtain a share lock on the record (dirty read), and CR - obtain a share lock (wait for any exclusive lock held by another sharer to be released). Transactional VSAM provides a third option: CRE - repeatable read (obtain a share lock and keep lock until end-of-transaction). RLS provided CRE, but only for CICS application access.

Specifying the CRE option on the JCL RLS parameter or on the ACB RLSREAD parameter results in Transactional VSAM access.

When a non-CICS application requests RLS access and OPEN for OUTPUT for a recoverable file, the file is opened for Transactional VSAM access. Prior to Transactional VSAM, the OPEN would have failed.

In general, a data set can be shared by any readers, whatever the access mode. Batch jobs can also open non-recoverable files for either input or output.

Batch jobs can open recoverable files for update in non-RLS mode only when there is no one else using them, or the only users of the files are non-RLS readers. Batch jobs cannot open recoverable files for output in RLS mode; only CICS may do that.

In order for a batch job to access a recoverable file for output when the file is being shared by applications which support transactional recovery, it must use Transactional VSAM access.

A data set is opened for Transactional VSAM access using the same JCL or ACB parameters that are used to open it for RLS access. It is accessed in Transactional VSAM mode when it is opened by a batch job (non-CICS) for RLS access and:

1. Read integrity option CRE is requested on the JCL or via the ACB, or
2. It is opened for output and the LOG parameter is UNDO or ALL.

Note that this means that a non-recoverable data set may be accessed in Transactional VSAM mode when CRE is requested. This is because Transactional VSAM must release all locks at either sync point or at the end of the job (implied sync point).

A data set whose LOG parameter is NONE or undefined which is opened without CRE being specified will be opened for RLS access. If CRE is later specified via an RPL, it will be treated as an error. This is because a single open cannot be treated as both RLS and Transactional VSAM as it would cause confusion with respect to the lock owner and could cause the user to run into conflicts with his/her own locks.

A data set whose LOG parameter is UNDO or ALL which is opened for input without CRE being specified will be opened for RLS access. If CRE is later specified via an RPL, it will be treated as an error.

### New VSAM Error Codes

40 (28) - Transactional VSAM was unable to expand the pool for its context/UR related control blocks. In this case, Transactional VSAM will most likely restart the SMSVSAM server, since this error indicates that there is a storage shortage within the address space.

92 (5C) - This is the same reason code that VSAM RLS uses to indicate that a PUT or ERASE was issued without the preceding GET UPD. For Transactional VSAM it indicates that a PUT UPD or ERASE was issued without previous GET UPD in the same unit of recovery. One potential cause of this is a sync point between the GET and the PUT.

205 (CD) - Transactional VSAM  restarted while the UR was in-flight. RRS considers the existing UR to be 'stale.' To continue processing, the application must issue a commit or a back out and then begin a new unit of recovery.

206 (CE) - The data set is quiesced or quiescing for copy. Wait for the data set to be unquiesced and then retry the request.

207 (CF) - Transactional VSAM is quiescing or disabling. In order for the quiesce or disable process to complete, all data sets that are open for Transactional VSAM access must be closed.

209 (D1) - Transactional VSAM was unable to complete the request because the forward recovery log is disabling, most likely because the operator issued a VARY SMS command to disable it. It will transition for disabling to disabled when all users have disconnected.

210 (D2) - The record length for the updated record is greater than the installation maximum supported by the forward recovery log. The maximum record length the System Logger supports is 64K, but the installation may define log streams such that their maximum is smaller.

211 (D3) - A permanent I/O error was detected on the forward recovery log. See accompanying Transactional VSAM logger messages for appropriate action.

213 (D5) - Transactional VSAM was unable to complete the request because the undo log is unavailable, most likely because a VARY SMS command was issued to quiesce or disable it. There could also have been an I/O error on the log, in which case a cold start will be required.

214 (D6) - A permanent I/O error was detected on the undo log. In this case, Transactional VSAM will quiesce itself, and a cold start will be required. See accompanying Transactional VSAM logger messages for appropriate action.

217 (D9) - RRS restarted while the UR was in-flight. RRS considers any outstanding URs to be 'stale.' To continue processing, the application must issue a commit or a back out and then begin a new unit of recovery. In cases where the commit or back out is issued before Transactional VSAM has an opportunity to reinitialize, it may be necessary to issue the sync point twice. The reason for this is that if they sync point was issued to early, then Transactional VSAM would not have been registered with RRS and would not have seen it.

Note: When an RRS failure occurs, it requires Transactional VSAM to reinitialize as a new instance. To do this, Transactional VSAM must transition from the disabling state to the disabled state and then reinitialize. In order for this transition to occur, it is necessary for all data sets which are open for Transactional VSAM access to be closed.

220 (DC) - Transactional VSAM was unable to complete the request because RRS is currently unavailable. In order to continue processing, the application should wait for RRS to recycle, at which point Transactional VSAM will reinitialize itself. Although Transactional VSAM restart processing will clean up any outstanding work, the application will need to issue a commit or a back out to clean up the 'stale' UR.

235 (EB) - VSAM RLS or Transactional VSAM internal error. VSAM RLS or Transactional VSAM most likely took a dump. This type of error should be reported to IBM service personnel.

249 (F9) - The record length is greater than the installation maximum supported by the undo log. The maximum record length the System Logger supports is 64K, but the installation may define log streams such that their maximum is smaller.

12 (C) -Transactional VSAM processing is currently unavailable because Transactional VSAM is initializing.

## Transactional VSAM & z/OS RRS

Transactional VSAM is a recoverable resource manager. It is NOT a commit or sync point manager. Transactional VSAM interfaces with the OS/390 Sync point Manager. When an application issues a commit request directly to OS/390 or indirectly through a sync point manager that interfaces with the OS/390 sync point manager, Transactional VSAM is invoked to participate in the 2-phase commit process. Other resource managers (like DB2) whose recoverable resources were modified by the transaction are also invoked by the OS/390 sync point manager thus providing a commit scope across the multiple resource managers.

## Application Interfaces: RRS

The UR identity is obtained from OS/390. Transactional VSAM does NOT provide commit/rollback interfaces. They are provided by the RRS component of OS/390 RRMS, which coordinates across any number of interested resource managers. Transactional VSAM registers with RRS as a resource manager and expresses interest in work requests done in Transactional VSAM mode. RRS then invokes exits provided by Transactional VSAM as commits and back outs are required. Note that these are not installation exits or users exits. Exit is just the terminology that RRS happens to use.

If an application neglects to issue sync points, an implicit sync point will be taken when the application terminates. If it terminates normally, a commit is done. If it terminates abnormally, a back out is done. It is NOT recommended that applications rely on these implicit sync points as doing so could result in URs which are extremely long running. This could result in tying up resources for much longer than is actually necessary and potentially impacting other applications and online transaction processing. It can also cause the log to become very long which can potentially impact performance.

## Application Redesign

In order for an application to participate in transactional recovery, it must first understand the concept of a transaction. It is NOT a good idea simply to modify an existing batch job to use Transactional VSAM with no further changes. This would cause the entire job to be seen as a single transaction. As a result locks would be held and log records would need to exist for the entire life of the job. This could cause a tremendous amount of contention for the locked

resources. It could also cause performance degradation, as the undo log becomes exceedingly large.

Once a batch job understands the concept of a transaction, it can be modified to use Transactional VSAM simply by changing the RLS JCL or modifying the ACB parameter specified in the program. If the batch job uses multiple RPLs, care must be taken in how they are used. Using different RPLs to access the same record can cause lock contention within a UR.

Since sync points (commits or back outs) cause all locks to be released, the application can not depend on locking across a sync point. For example, the application cannot issue a GET UPD before a commit and a PUT UPD afterward. Likewise, a POINT CRE should not be issued before a sync point, followed by a GET afterward. In order to enforce this Transactional VSAM forces a loss of positioning during sync point processing so that the application cannot get into trouble by attempting to do something that requires a lock it no longer holds.

## Application Considerations

Transactional VSAM provides isolation of each UR. The application may use repeatable read to inhibit changes to records that it reads. Changed records of recoverable files are locked with exclusive locks. All of the URs locks are released at commit or back out. The URs changes are then visible to other URs.

Notice that Transactional VSAM is providing transactional recovery, and hence is supporting a transactional environment. In order for a batch job to participate in this data sharing, it must be designed as a series of transactions. At end-of-transaction, the job's Transactional VSAM record locks are released. Issuing frequent sync points (commits or back outs) avoids timeout of other jobs/ transactions that may be waiting for the lock(s). It also avoids writing large amounts of data to the log which then have to be offloaded from the coupling facility. Overuse of the log results in a more frequent need to offload and can create a performance bottle neck.

The application must rely on RLS and Transactional VSAM locking of file records. The locks are used so that the application sees consistent information. When reading, three levels of integrity are available: NRI (no read integrity, or dirty read), CR (consistent read, so that the application does not read a record that is in the process of being updated), and CRE (consistent read explicit, so the record remains unchanged for the duration of the UR). Although the application may use repeatable read to inhibit changes to records that it reads, it is recommended that this not be used more than absolutely necessary, since it locks out updaters.

Notice that an application which uses CRE reads MUST issue commits or back outs in order to release the locks, even if it has made no updates. If it does not, it will continue to hold the locks, potentially locking out other applications.

A GET UPD implies that the application reading the record intends to update the record. It is when the GET UPD is done that RLS obtains an exclusive lock on the record on behalf of the UR, and it is also at this time that Transactional VSAM writes a record to the undo log. If the

application is only reading the record without intent to update it, it should use GET NUP in order to avoid the overhead of obtaining the exclusive lock and writing the log record. If read integrity is required, the application should specify CR or CRE on the GET.

## Applications Considerations -- Commit Frequency

This example was created by the ITSO in the process of writing the redbooks. The input transaction file contained 10,000 records and the program was run using a range of different commit frequencies.

As expected, the greater the commit frequency, the longer the elapsed run time, because invoking commit incurs a cost. As the commit frequency decreased, so did the elapsed time. The results using a COBOL program are shown on this chart.

The elapsed time is measured in seconds. Each run was repeated several times to verify that the measurements were repeatable and consistent.

Committing every three updates produced the longest elapsed time. Lengthening the commit frequency reduced the elapsed time, until a level was reached where the cost of committing was a vanishingly small component of the elapsed time. For this particular program, committing every 100 updates seemed to produce reasonable results. However, note that results can vary by program so this may not necessarily be the case for your programs.

It is also important to consider the impact of your program's actions on on line applications and other users. A commit frequency of 100 updates means that at least 100 locks are held. As there were 10,000 records in the data set, this means committing 100 times. Working back from the elapsed time of 60 seconds, this means that a transaction (or unit of recovery) averaged 0.6 seconds between commits. This would be the worst case for the time a CICS transaction might have to wait for a lock; on average, it would wait 0.3 second, if it had to wait at all. It would only need to wait if the record it needed was one of the 100 involved in the current transaction, about a 1% chance, in a data set containing 10,000 records.

As a rough rule of thumb, a commit frequency between 10 and 100 updates seems to be reasonable. A number closer to 10 updates is recommended when the record size is relatively large. This is also the case when the amount of processing for each record is relatively long and would therefore hold records locked for an extended period of time, which could lead to contention. A number closer to 100 is good when the record size is relatively small and the amount of processing is relatively low, so that resource are locked only very briefly.

A well-tuned size for the unit of recovery balances the cost of commits against the impact on other users and the time that would be required in event of a back out.

## Exclusive Control of Resources

A UR updating a record in a recoverable file has exclusive control of that resource until it terminates or indicates that it wants to commit its changes. Other URs requiring the resource must wait until the first UR finishes with it. To reduce delays due to contention, the length of time between claiming the resource and releasing it should be minimized.

Skip sequential processing is generally used to speed browsing when reading forward through a data set. VSAM locates the next record by reading through the index rather than repositioning from scratch. This method is faster if the records you are retrieving are relatively close together but not necessarily adjacent; it can have the opposite effect if they are far apart in a large data set. If you know the key you are repositioning to is much higher and that you may incur a long index scan, you may wish to consider using POINT to reset your positioning.

To update a record you must first retrieve it using GET UPD. After modification by the application, the record can be written out using PUT UPD. Because GET UPD gets an exclusive lock that is not released until sync point, the application should issue commits at regular intervals. For a KSDS, the base key in the record must not be altered when the record is modified. Similarly, if the update is made by way of a path, the alternate key used to identify the record must not be altered either, although other alternate keys may be altered.

After using GET UPD to retrieve a record, you can use ERASE to delete it. You cannot delete records from an ESDS. If you wish to do so, you are responsible for marking the record in way that your application considers deleted.

New records are added to a data set using PUT NUP. When adding a record to a KSDS, the base key identifies the position in the data set where the record is to be inserted. A record added to an RRS is added using the relative record number (RRN). A record added to an ESDS is always added at the end. Note that Transactional VSAM does NOT support backing out records that were added to an ESDS.

## Browsing

When browsing a data set use GET with NRI, CR or CRE rather than GET UPD. GET UPD causes an exclusive lock to be obtained and is not recommended for browsing when using Transactional VSAM access. Users have historically used GET UPD to obtain read integrity because prior to VSAM RLS there was no other way to provide the necessary serialization and provide read integrity. With VSAM RLS and Transactional VSAM, this level of serialization is available using the CR and CRE options. The CRE option should only be used when that level of integrity is really required as it locks updaters out of the record.

## Avoiding Deadlocks

Applications need to be designed to avoid deadlocks. A deadlock occurs if each of two URs needs exclusive control of some resource (a record) that the other already holds. For example, UR A holds record 1 and wants record 2, while UR B holds record 2 and wants record 1. In order

to avoid this type of deadlock, it is important that applications which are intended to run in a shared environment access resources in a predictable order. If, for example, URs A and B in the example above always access resource in ascending order, the deadlock would not occur. Here are some rules for avoiding deadlocks:

1. All applications that update multiple resources should do so in the same order.

2. If a data set has an alternate index, beware of mixing URs that perform updates via the base key with URs that perform updates via an alternate key. URs that perform updates via the base key can deadlock with URs that perform updates via an alternate key because the key that is locked is always the base key.

3. An application that issues a GET UPD should follow it with a PUT UPD or ERASE and complete work on behalf of the UR as quickly as possible. It should then invoke RRS to commit the UR, allowing the locks held on behalf of the UR to be released.

4. Commits should be issued by applications that use CRE to browse data sets. Although the locks for GET CRE requests are shared locks, they are held until sync point processing. This also applies to POINT CRE. In addition POINT CR causes a shared lock to be obtained that is not released until positioning on the RPL is changed or the UR reaches a sync point.

### Non-Shared vs. Shared: Job Rerun

Today's batch applications that update VSAM files in a non-shared environment may create backup copies of the files to establish a restart/recovery point for the data. If the batch application fails, the files may be restored from the backup copies, and the batch jobs can be re-executed. This restart/recovery procedure CANNOT be used in a data sharing Transactional VSAM environment, because restoring to the point in time backup would erase any changes made by other applications sharing the data set.

Instead, the batch application must have a built in method of tracking its processing position within it string or series of transactions. One potential method of doing this is to use a VSAM recoverable file to track the job's commit position.

When the application fails, any uncommitted changes are backed out. The already committed changed CANNOT be backed out, because they are already visible to other jobs/transactions. In fact, the records that were changes by previously committed URs may have since been changed again by other jobs/transactions. Therefore, when the job is rerun, it is important that it determine its "restart point" and NOT attempt to redo any changes it had committed before the failure.

For this reason, it is important that jobs and applications using Transactional VSAM be written to execute as a string of transactions and use a commit point tracking mechanism for restart.

### Job Restart: Change Isolation

This is another way of illustrating the fact that recovery from failure of a sharing update job can NOT be via backup/restore of the file. This example shows job1 updating a record and committing the update. This allows Job2 to update the same record. Job2 commits the change and the record is now updated by Job3. The value of a field in the record has changed from 100 to 400. If JOB1 failed and recovery restored the file to the state is was before JOB1 started, if would reset the value of the updated field to 100 losing the changes to the field made by JOB2 and JOB3.

Since JOB1 committed its change to the field (else, the other jobs could not have changed the field), recovery from failure of JOB1 must NOT change the field. The change was committed and thus should NOT be backed out.

## Job Processing Position File

This illustrates how a VSAM recoverable file may be used to track a job's position within its input stream. As the application executes its units of recovery, it writes information indicating which units of recovery have been processed to the position file. At commit time, the position file and the application files are in sync. If the unit of recovery is backed out, then the change to the position file is also backed out. It is not necessary for each input record to correspond to a commit point, only that the position file have a record of each commit or back out.

Use a private, non-shared recoverable file to track the job's input stream at commit time. On rerun:

1. Position input stream to position shown in private file
2. Read next record from input stream
3. Update private file to reflect new position
4. Perform indicated processing
5. Continue, issue RRS commit at end of transaction
6. Commit is atomic across all recoverable files. Private file and application stay in sync.

## Application Redesign

BEFORE shows an example program used for batch processing. A backup is taken of the master file prior to running the program, to be used in case of restart. The transaction file serves as input to make updates, deletions or new records in the master file. The spooled report could be checks, or some other processing report.

AFTER -
1) Batch program will open files in Transactional VSAM mode, with the appropriate settings and options.
2) Batch program reads the position file, and uses this to determine where to begin processing the transaction file.

3) Just before each commit, batch program writes update to position file. If commit is successful, position file matches. If commit fails, all updates are backed out.
4) Rather than spooled output, Batch program writes to a transactional print file, can be RRDS, so that if transactions are backed out, the print file updates are backed out also.
5) You can use IDCAMS REPRO to send the print file to spool.

## Context & Unit of Recovery (UR)

In order to have transactional recovery, we must have a means of uniquely identifying transactions (units of recovery). For Transactional VSAM, OS/390 provides the interfaces through which the unit of recovery (UR) identity is communicated.

OS/390 provides a Context and UR structure associated with a TCB. The structure includes a UR identifier and it tracks the resource managers that will participate in the commit or rollback of the UR.

The UR consists of the set of changes that are to be made or not made as an atomic unit. Therefore, a UR represents an application program's changes to resources since the last commit or back out, or, for the first UR, since the beginning of the application. Each UR is associated with a context. The life of a context is typically a series of application programs URs.

## Context Management

z/OS provides the context and UR management under which Transactional VSAM participates as a recoverable resource manager. When a TCB is dispatched, it has a context. For the standard case where the work is not managed by a transaction monitor or work manager, z/OS provides a default context, the native context. A native context is the automatically occurring context of the application program and protected resources associated with a work request. A native context is always associated with a single application task. This context is associated with a specific task and always exists.

A transaction/work manager may use z/OS Context Services to create privately-managed contexts. The resource/work manager owns any privately-managed contexts it creates, and can switch privately-managed contexts from one task to another. A privately-managed context is usually used be a work manager which is a resource manager (such as IMS) that accepts and manages work, such as transactions, from outside the system.

Every task in the system has an associated context, thus there is always a context for a given task. When a task is created, context services provides the original (native) context for the task. Resource managers can create privately-managed contexts and associate them with a specific task. The privately-managed context then becomes the current context. The native context still exists, but it is not current. If the resource manager later disassociates the privately-managed context from the task, the native context would again become current.

Transactional VSAM does not create or switch context. It does, however, allow others to do so. When Transactional VSAM receives control, it works under whatever context is current. Since a UR always remains paired with its context, it is important that the correct context be in control when any Transactional VSAM work is done.

## Multitasking Considerations

In this picture, a mother task, perhaps representing a work dispatcher, has attached three daughter tasks. All four tasks have native context associated with them. Since each context has its own UR (every context has zero or one URs at any point in time) work done under each TCB is in fact a different UR.

If the mother task is in fact a work manager, and it wanted the ability to randomly hand work off to which ever daughter task happened to be free, it would need to create privately managed contexts. Before passing work to a daughter TCB, it would need to attach the correct privately managed context to the TCB, so that the work that is being done is associated with the correct UR.

## Performance -- Parallelizing the Workload

Even though the additional overhead associated with locking, logging, and two phase commit and back out results in increased elapsed time, it should be noted that this can be offset by the parallelism that Transactional VSAM allows. Of course, this assumes that the batch jobs are independent of one another, and that one batch job is not waiting for the output of another batch job to provide its input.

In this example created by the ITSO, the master file contained 1,000,000 records. The job which updated this master file was run in three different ways: as a single non-shared resources (NSR) job that updated all 1,000,000 records, as a single Transactional VSAM job that update all 1,000, 000 records, and as four separate Transactional VSAM jobs that each updated 250,000 of the records.

This chart shows a comparison of the elapsed time. Note, however, that these results were obtained in an uncontrolled test environments an should by no means used as benchmarks.

When splitting the Transactional VSAM work load into four parallel batch jobs, each jobs ends after about the same elapsed time because each job is processing the same number of records. They complete in about 41% of the time it took to run the updates as a single Transactional VSAM job. Running them as four parallel jobs even caused them to run slightly faster than the traditional NSR environment when comparing wall clock time. The number of EXCPs and the CPU consumption vary slightly between jobs in the DFSMStvs environment. In addition, CPU time, elapsed time, and the number of EXCPs are higher when using Transactional VSAM. This is due to the extra overhead noted earlier: locking, logging, and two phase commit and back out.

## Peer Recovery

Peer recovery is very similar to restart processing, in that it completes the processing of any units of recovery which were in progress at the time of the failure. The major difference is that during peer recovery, there are multiple instances of Transactional VSAM running within the SMSVSAM address space: the primary instance, which is working with VSAM RLS to process requests on the system, and an instance which is registered with VSAM RLS and RRS as the failed instance. During peer recovery, Transactional VSAM does the following:

1. It registers with VSAM RLS as the failed instance of Transactional VSAM.
2. It registers with RRS as the failed instance of Transactional VSAM.
3. It invokes RRS to indicate that it is beginning restart processing as the failed instance.
4. It reads the failed instance's undo log to retrieve information about in-progress units of recovery.
5. It invokes RRS to retrieve information about unit of recovery status.
6. It processes the log data and the information returned by RRS to determine whether to commit or back out units of recovery.
7. Indoubt units of recovery are left until RRS determines whether they should be committed or backed out.
8. When all outstanding units of recovery have been processed, it unregisters with RRS and VSAM RLS.

It is necessary for the instance of Transactional VSAM which is performing peer recovery to register as the failed instance in order to gain access to the failed instance's resources. Since only one instance can be registered with a given name at a time, the first system which successfully registers performs the peer recovery.

Note that the registration persists until peer recovery is complete. As a result, should the failed instance restart, it will be unable to initialize because its attempt to register will fail. It may be necessary to restart Transactional VSAM manually once peer recovery is complete.

## Permit Non-RLS Update

If a data set has retained locks or is in lost locks state, a batch job's non-Transactional VSAM open of the data set will fail. To allow a non-Transactional VSAM batch job to access this data set, the operator can issue the SHCDS PERMITNONRLSUPDATE command. Once the command has been issued, a non-Transactional VSAM batch open will be successful (even if there are retained or lost locks) provided that there are no concurrent RLS or Transactional VSAM opens of the data set.

The SHCDS PERMITNONRLSUPDATE command allows the batch job update or delete records which may be the records for which there are retained locks. Some time after the batch job has run, Transactional VSAM may be asked to back out the URs for which it holds retained locks. If the back outs are done, Transactional VSAM may invalidate the updates or deletes performed by the batch job. This back out request may come from an in progress UR,

Transactional VSAM restarting, the IDCAMS SHCDS RETRY command, or from a communications resource manager issuing a back out request for a UR which had been in indoubt state.

In a Transactional VSAM environment, it will be necessary to quiesce the data set in addition to issuing the PERMITNONRLSUPDATE for it. This is because, unlike CICS, a failure could cause Transactional VSAM to restart. For example, suppose there were five jobs that needed to be run while the PERMITNONRLSUPDATE was active and the data set was not quiesced:

1. The data set has retained locks or is in a lost locks state, but non-RLS and non-Transactional VSAM work needs to be done, so a PERMITNONRLSUPDATE is issued.

2. Job 1 is run and completes successfully.

3. Job 2 is run and completes successfully.

4. A failure occurs which causes Transactional VSAM to restart. As part of restart, Transactional VSAM reads the undo log and encounters the records for the data set which is in PERMITNONRLSUPDATE status. Not knowing that the work is still in progress, Transactional VSAM attempts to perform the back out, invoking the exit as necessary. When it is finished, it clears the PERMITNONRLSUPDATE status. Note that this step should not run until all of the work being done under the PERMITNONRLSUPDATE is complete.

5. Job 3 is run and completes successfully. It is able to open the data set because it no longer has retained locks and it is no longer open for Transactional VSAM access. However, note that the Transactional VSAM restart may have backed out records that this job thought it was supposed to handle.

6. Job 4 is run and completes successfully. Again, Transactional VSAM may have backed out records that this job thought it was supposed to handle.

7. Job 5 is run and completes successfully. Same notes as above.

To prevent this from happening, the data set also needs to be quiesced. This will cause the open that Transactional VSAM would do as part of restart to fail and prevent Transactional VSAM from attempting to perform the back out until the data set is unquiesced.

### What Happens to the Back Outs?

In order to prevent damage to the data set, Transactional VSAM will defer the decision whether to back out a specific record to an installation exit, the batch override exit. This is an optional exit which Transactional VSAM calls when it backs out a UR which involves a data set which may have been impacted by a PERMITNONRLSUPDATE. The exit will be called once for each affected undo log record for the data set in question. The purpose of the exit is to return to

Transactional VSAM with an indication of whether or not the back out should be applied. The input is an undo log record (mapped by IGWUNLR) and a data set name. The output is a Boolean response of whether or not to do the back out. The interface to the exit is the same as the interface for the CICS XCFBOVER exit. It is possible for the exit to perform other processing, but it is strongly recommended that the exit not attempt to update any recoverable resources.

If the installation exit is not provided and Transactional VSAM encounters back outs which may have been impacted by a PERMITNONRLSUPDATE, it will issue a message. It will not apply the back outs. They will need to be cleaned up using the IDCAMS SHCDS RETRY or PURGE command.

**Closing Data Set with In-Flight UR**

CLOSE of a VSAM recoverable data set that is OPEN for RLS or Transactional VSAM access in output mode results in any update record locks held by the resource manager that is managing the data set being converted into retained locks. If there is more than one ACB OPEN for a sphere, this action is only taken when the last ACB for the sphere is closed. This action is also taken when the TCB or address space that issued the OPEN terminates with the data set OPEN.

For data sets which are forward recoverable, the forward recovery log stream is disconnected at CLOSE time.

If an application attempts to close a data set which has uncommitted changes (that is, a unit of recovery is in-flight), the close will be allowed. However, this is not recommended, because once a data set is closed, it can be deleted or renamed. In the case where it is necessary to back out the changes that were made, Transactional VSAM will reopen the data set and perform back out. If the data set has been deleted or renamed, it will cause failures during back out. The portions of back out which could not be completed will be shunted.

**Data Set Delete/Rename Concerns**

It is possible to delete or rename a data set which has retained locks and shunted log records. This is necessary to allow for delete and recreate of a data set in event that the data set is damaged and must be forward recovered. In this case, the retained locks associated with the data set are normally unbound using the IDCAMS SHCDS command prior to deleting the data set.

If the locks are unbound prior to the delete, then any retained locks and log records associated with the data set are kept across the delete. They are re-associated with the data set when the locks are rebound. If the unbind is not done prior to deleting the data set, then the locks are discarded when the data set is deleted.

Since it is possible for an application to close a data set which has uncommitted changes (that is, a unit of recovery is in-flight), it is also possible for the data set to be deleted or renamed before

the unit of recovery completes. It is also possible to delete or rename data sets which have retained locks and shunted log records associated with them. Unless this is being done in preparation for forward recovery (in which case, locks should first be unbound), this is not recommended. Doing so can result in loss of any association between the data set and it log records and locks.

IDCAMS DELETE results in an exclusive ENQ on the data set, making it impossible for Transactional VSAM to allocate the data set during back out. For example:

STEP1:
Execute PGM=IDCAMS
Specify a DD statement for data set MY.TVS.KSDS with DISP=SHR
Delete MY.TVS.KSDS

STEP2:
Execute PGM=IDCAMS
Define MY.TVS.KSDS

STEP3:
Open MY.TVS.KSDS for Transactional VSAM access
Update records in MY.TVS.KSDS
Attempt to back out the changes

One possible failure this can result in is:

IGW10117I DYNAMIC ALLOCATION OF DATA SET dsn FAILED.
RETURN CODE (00000004) REASON CODE (02100000)
ATR306I RESOURCE MANAGER rmname CAUSED A HEURISTIC-MIXED CONDITION FOR URID = urid

The problem occurs because IDCAMS upgrades the ENQ for MY.TVS.KSDS from shared to exclusive. The solution is to do the delete at the end of the job or in a previous job. It cannot be done in a different step of the same job, since in this case, IDCAMS would still hold the ENQ.

In the case where a unit of recovery is in flight, deleting or renaming the data set will cause failures, if the unit of recovery needs to be backed out. It will also result in any work which cannot be completed being moved to the shunt log, and manual action using the IDCAMS SHCDS PURGE command will be needed to clean up the shunted log records.

Once a data set has been deleted or renamed, it is possible for another data set to be created with the name. If any shunted log records related to the original data set are not purged using the IDCAMS SHCDS PURGE command, they could be applied to the new data set in error.

Note that although users may not be using a data set, the data set could be in use if Transactional VSAM restart processing is in progress. If this is the case, then it will not be possible to delete or rename the data set until Transactional VSAM is finished with it.

## Transactional VSAM Summary

Transactional VSAM provides general VSAM recoverable file sharing. This addresses a long standing requirement for sharing of VSAM files across CICS and batch jobs.

## Appendix B. IDCAMS SHCDS Commands

### RLS Commands

Base-cluster is a fully or partially qualified VSAM data set name. The high-level qualifier must be specified. You can use an asterisk (*) for a subsequent qualifier, but then no lower-level qualifiers are allowed. For example, A.* is allowed, but A.*.B is not.

Subsystem is the name of an online system, such as CICS, as registered to the SMSVSAM server.

Notes:

1. Various levels of authority are required to use the SHCDS parameters.

2. A program that calls the SHCDS command must be APF-authorized.

3. To use the SHCDS command in the TSO foreground, SHCDS must be added to the authorized command list (AUTHCMD) in the SYS1.PARMLIB member IKJTSOxx or added to the CSECT IKJEGSCU. Please see OS/390 TSO/E Customization for more information.

The SHCDS parameters provide for:

- Listing information kept by the SMSVSAM server and the catalog as related to VSAM RLS.

- Controlling forward recovery, as well as preserving retained locks when a data set is moved or copied; and, in rare cases when forward recovery fails, deleting the locks.

- Allowing non-RLS updates when forward recovery is required.

- Removing the SMSVSAM server's knowledge of an inactive subsystem, thus forcing a cold start of the online application. Use REMOVESUBSYS only when procedures provided by the application have failed or you have no intention of ever using the subsystem again.

- Resetting VSAM RLS indicators in the catalog, allowing reconstruction of RLS information or fallback from VSAM RLS.

LISTDS(base-cluster) lists:

- The assigned coupling facility structure name
- The subsystem type and status:
  - Active for batch
  - Active or failed for online
- Whether the VSAM sphere accessed in RLS mode requires recovery and a summary of the recovery indicators:
  - Forward recovery required
  - Retained locks
  - Lost locks
  - Locks unbound
  - Non-RLS update permitted
  - Permit-first-time switch

LISTSUBSYS(subsystem|ALL) lists information about a specific subsystem or all subsystems known to the SMSVSAM server:

- Subsystem status
  - Active for batch
  - Active or failed for online
- A summary showing whether the subsystem's shared data sets have:
  - Lost locks
  - Retained locks
  - Non-RLS update permitted
- For an active subsystem, LISTSUBSYS gives the number of held locks, waiting lock requests, and retained locks. For a failed subsystem, LISTSUBSYS shows the number of retained locks.

LISTSUBSYSDS(subsystem|ALL) lists information about a specific subsystem or all subsystems known to the SMSVSAM server, including data sets that it is sharing. For each subsystem, it lists:

- Sharing protocol (online or batch)
- The status (active or failed)
- Recovery information for each shared data set:
  - Whether it has retained locks owned by this subsystem
  - Whether it has lost locks owned by this subsystem
  - Whether there are locks not bound to the data set
  - If forward recovery is required
  - If non-RLS update is permitted
  - The permit-first-time switch setting

LISTRECOVERY(base-cluster) lists data sets requiring recovery and the subsystems that share those data sets. Recovery indicators listed are:

- Lost locks
- Retained locks
- Non-RLS update permitted
- Forward recovery required

LISTALL Lists all information related to recovery for subsystems and VSAM spheres accessed in RLS mode. The output from this parameter can be quite large.

FRSETRR(base-cluster) This parameter sets the forward-recovery-required indicator. Until reset with the FRRESETRR parameter, access is prevented until forward recovery is complete.

If you use a forward recovery utility that supports RLS, such as CICSVR, do not use this parameter.

FRUNBIND(base-cluster) This parameter unbinds the retained locks prior to restoring or moving the data set. These locks protect uncommitted changes and are needed for eventual back out. They must be rebound by using the FRBIND parameter.

If you use a forward recovery utility that supports RLS, such as CICSVR, do not use this parameter.

FRBIND(base-cluster) Use this parameter after BLDINDEX to rebind the associated locks to the restored data set.

Attention: Between the unbind and the bind, do not delete any clusters in the sphere or change their names.

If you use a forward recovery utility that supports RLS, such as CICSVR, do not use this parameter.

FRRESETRR(base-cluster) Use this parameter after forward recovery is complete and after locks have been bound to the new location of the data set using FRBIND. This allows access to the newly recovered data set by applications other than the forward recovery application.

If you use a forward recovery utility that supports RLS, such as CICSVR, do not use this parameter.

FRDELETEUNBOUNDLOCKS(base-cluster) This parameter lets you delete locks in the rare case when a successful forward recovery is not possible. Every attempt should be made to complete forward recovery, whether using a product such as CICSVR that supports VSAM RLS or using another forward recovery procedure.

If forward recovery does not successfully complete, locks cannot be re-associated (bound) to the new version of the data set, because these locks do not provide the protection that online back out requires.

Before using this parameter, check the documentation for your online application. For CICS, the procedure is documented in the CICS Recovery and Restart Guide.

PERMITNONRLSUPDATE(base-cluster) Allows a data set with pending RLS recovery to be opened for output in non-RLS mode. This command is used when it is necessary to run critical batch updates and RLS recovery cannot first be completed. This is reset the next time the data set is accessed for RLS. If after using PERMITNONRLSUPDATE, you do not run a non-RLS batch job, you must use DENYNONRLSUPDATE to prevent non-RLS updates.

DENYNONRLSUPDATE(base-cluster) If you inadvertently issue PERMITNONRLSUPDATE, use this parameter to reset the effect of PERMITNONRLSUPDATE.

If recovery was pending, but you did not run a non-RLS batch job, you must use this parameter. If not reset, CICS takes action assuming the data set has been opened for update in non-RLS mode.

Do not use DENYNONRLSUPDATE if you do indeed run non-RLS work after specifying PERMITNONRLSUPDATE. The permit status is reset the next time the data set is opened in RLS mode.

REMOVESUBSYS(subsystem) Use this parameter to remove any knowledge of recovery owed to SMSVSAM by the named subsystem, including locks protecting uncommitted updates.

Normally, a failed online application would be restarted so that it can do the required back outs and release locks protecting uncommitted updates. However, sometimes it might be necessary to cold start the online application. For more information about cold starts, see CICS Recovery and Restart Guide.

Use of this parameter is equivalent to cold starting the named subsystem with respect to the SMSVSAM server. Use REMOVESUBSYS for the rare cases where either there is no intention of ever running the subsystem again or the application's cold start procedures cannot be used. An example of an appropriate use of REMOVESUBSYS would be removing a test system that is no longer needed. If the removed subsystem is ever run again, every effort should be made to cold start the subsystem.

Attention: Use of REMOVESUBSYS can result in loss of data integrity.

CFREPAIR({INFILE(ddname)|INDATASET(dsname)} Use this command to reconstruct the RLS indicators for all applicable data sets in a restored catalog. CFREPAIR uses information known to the VSAM RLS server at the time the SHCDS command is used. The catalog must be

import-connected on all systems to the master catalog before the CFREPAIR parameter can be used.

CFRESET({INFILE(ddname)|INDATASET(dsname)} Use this parameter if you've decided to fall back from using VSAM RLS. It clears VSAM RLS indicators in the catalog for all applicable data sets. A detailed fallback procedure is included in the DFSMS/MVS DFSMSdfp Storage Administration Reference. Also, please see the CICS Recovery and Restart Guide for information specific to CICS.

If the catalog is later restored, use CFREPAIR to reconstruct critical information required by the SMSVSAM server.

CFRESETDS(base-cluster) Use this parameter if you've decided to fall back from using VSAM RLS. It clears VSAM RLS indicators in the catalog for all applicable data sets. CFRESETDS This parameter differs from CFRESET in that it lets you select one or more data sets for fallback. CFRESETDS lists all data sets processed, not just those with errors.

A detailed fallback procedure is included in the DFSMS/MVS DFSMSdfp Storage Administration Reference. Also, please see the CICS Recovery and Restart Guide for information specific to CICS.

**Transactional VSAM Commands**

To use most of the SHCDS LIST subcommands, read authority to the facility class STGADMIN.IGWSHCDS.REPAIR is required. The only exception is the LISTSHUNTED subcommand for which update authority is required to the data set specified. The reason for this is that it is expected that the list subcommand will normally be followed by either a RETRY or PURGE subcommand. The RETRY and PURGE subcommands also require update authority to the data set(s) involved.

LISTDS(base_cluster_name) Lists the following information for the specified data sets:

- CF cache structure assigned
- Whether the sphere is recoverable or non-recoverable
- Names of the subsystems sharing the data set and subsystem status
  - Subsystem status
  - Subsystem status relative to the data set
- A summary indication for the data set state (retained locks, lost locks, non-RLS update permitted, forward recovery required, locks unbound).

This command is modified to accept a new, optional JOBS keyword. When this keyword is specified, this command also returns a list of the jobs currently accessing the data set in Transactional VSAM mode.

LISTSHUNTED SPHERE(base-cluster) |URID(urid|ALL) Lists information about work which was shunted due to an inability to complete a sync point (commit or back out) for a given data set or UR, or for all shunted URs when the ALL keyword is specified. The output includes:

- The unit of recovery identifier(s)
- The data set name(s)
- The job with which the unit of recovery was associated
- The step within the job with which the unit of recovery was associated
- The disposition of the unit of recovery (whether it will be committed or backed out if it is retried)

RETRY and PURGE Commands

These subcommands are used to take action on work which Transactional VSAM shunted. Units of recovery are shunted when Transactional VSAM is unable to finish processing them, for example, due to an I/O error. As long as a shunted log entry exists, the locks associated with that entry are retained. You can either take action on a particular unit of recovery, or on all records in the shunt log which apply to a particular data set.

Before taking action, the installation needs to correct the problem which caused the log entries to be shunted. The possible actions are:

RETRY SPHERE(base-cluster) | URID(urid)

Retry the sync point. This command should be used when the data set can be restored to a state where it is consistent with the log entries. "Consistent" here means that the data set reflects the state that existed prior to the particular unit of recovery whose processing Transactional VSAM was unable to complete. This is possible either for data sets which are forward recoverable or for failures which do not damage the data set (such as a dropped path). Attempting to apply a log entry to a data set which is in an inconsistent state can cause unpredictable results. If this command completes successfully, locks associated with the log entries will be released.

PURGE SPHERE(base-cluster) | URID(urid)

Discard the log entries and release the associated locks. This command should be used when the data set is damaged and cannot be restored to a state where it is consistent with the log entries. For example, it may have been necessary to restore the data set to a backup copy which predates updates which were made to it prior to the failure.