

z/OS
3.2

DFSMSStvs Planning and Operating Guide



Note

Before using this information and the product it supports, read the information in [“Notices” on page 113.](#)

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 2003, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | |
|--|-------------|
| Figures..... | vii |
| Tables..... | ix |
| About this document..... | xi |
| Required product knowledge..... | xi |
| z/OS information..... | xii |
| How to provide feedback to IBM..... | xiii |
| Summary of changes..... | xv |
| Summary of changes for z/OS 3.2..... | xv |
| Summary of changes for z/OS 3.1..... | xv |
| Chapter 1. Understanding the DFSMStvs environment..... | 1 |
| Transaction processing and transactional recovery..... | 1 |
| Terminology..... | 1 |
| Transaction processing..... | 3 |
| Transactional recovery..... | 4 |
| VSAM record-level sharing (RLS)..... | 5 |
| Overview of VSAM RLS..... | 5 |
| Read sharing of recoverable data sets..... | 9 |
| VSAM RLS read integrity options..... | 9 |
| Read-sharing integrity across KSDS control-interval and control-area splits..... | 9 |
| Read and write sharing of nonrecoverable data sets..... | 10 |
| Non-RLS access to VSAM data sets..... | 10 |
| Differences between VSAM RLS access and non-RLS access..... | 10 |
| Requirements for VSAM RLS request execution mode..... | 12 |
| VSAM options that RLS and DFSMStvs do not support..... | 12 |
| DFSMStvs overview..... | 13 |
| Transaction processing from a batch job..... | 13 |
| Recovery coordination..... | 13 |
| Access to recoverable VSAM data sets..... | 14 |
| Transaction processing..... | 14 |
| How DFSMStvs works with RRS and other resource managers..... | 17 |
| How DFSMStvs complements CICS..... | 17 |
| Context services and RRMS..... | 17 |
| Native contexts..... | 18 |
| Privately managed contexts..... | 18 |
| Units of recovery..... | 18 |
| Chapter 2. Planning for DFSMStvs..... | 21 |
| Planning tasks..... | 21 |
| Coupling-facility planning..... | 22 |
| Coupling facilities..... | 22 |
| Number of coupling facilities..... | 22 |
| Standalone or internal coupling facility..... | 23 |
| Volatile or nonvolatile coupling facility..... | 23 |
| Contents of a coupling facility..... | 23 |

| | |
|--|-----------|
| Coupling-facility size..... | 24 |
| Coupling facility links..... | 26 |
| Processor-capacity planning..... | 26 |
| Software-configuration planning..... | 27 |
| System-logger planning..... | 27 |
| Logging flow overview..... | 27 |
| Log streams..... | 28 |
| Structures and log streams..... | 30 |
| DASD-only log streams..... | 30 |
| Log stream sizing..... | 31 |
| DASD staging data sets..... | 31 |
| DASD log data sets..... | 32 |
| VSAM operations planning..... | 32 |
| Recovery procedures..... | 32 |
| Forward recovery operation planning..... | 33 |
| Reorganization..... | 33 |
| Automatic Restart Manager planning..... | 33 |
| DFSMStvs and ARM..... | 33 |
| Installation of DFSMStvs..... | 34 |
| Chapter 3. Configuring the DFSMStvs environment and defining resources..... | 35 |
| Defining your Parallel Sysplex environment..... | 35 |
| Setting up the logging environment..... | 35 |
| Using coupling facilities..... | 36 |
| Defining staging data sets..... | 37 |
| Specifying SYS1.PARMLIB parameters for DFSMStvs..... | 38 |
| Defining a PARMLIB member specific to one system..... | 38 |
| Defining a parmlib member that applies to multiple systems..... | 39 |
| Chapter 4. Setting up DFSMStvs logging..... | 41 |
| Determining the amount of logging to do..... | 41 |
| Defining coupling-facility structures for log streams..... | 41 |
| Definition of a coupling-facility structure for a log stream..... | 43 |
| Log structure names..... | 43 |
| Allocating system log streams..... | 44 |
| Examples of system log stream definitions..... | 45 |
| System log stream names..... | 45 |
| Offloading of log data..... | 45 |
| Using backout logging..... | 46 |
| Backout records for in-doubt and long-running units of recovery..... | 46 |
| Backout logging events..... | 47 |
| Defining forward recovery logs..... | 47 |
| Creating a log of logs..... | 49 |
| Authorizing access to log streams..... | 50 |
| Authorization to access log streams..... | 50 |
| RACF RDEFINE coding..... | 51 |
| Chapter 5. Designing and coding applications to use DFSMStvs..... | 53 |
| Determining which applications should use DFSMStvs..... | 53 |
| Modifying an application to use DFSMStvs..... | 53 |
| Coding an application to use DFSMStvs..... | 54 |
| Defining transactions..... | 54 |
| Understanding DFSMStvs restrictions..... | 54 |
| Considering RLS and DFSMStvs restrictions..... | 56 |
| Using VSAM data sets in a transaction..... | 57 |
| Accessing a data set with DFSMStvs..... | 57 |
| Structuring your application for commit and backout..... | 58 |

| | |
|--|------------|
| Understanding the effects of a task ending..... | 59 |
| Understanding record locking that DFSMStvs uses..... | 60 |
| Handling long-running jobs and programs..... | 61 |
| Using restartable applications..... | 62 |
| Establishing positioning after logical errors..... | 63 |
| Using sequential or random access to a data set..... | 63 |
| Deleting and renaming data sets..... | 63 |
| Monitoring and retrying shunted transactions..... | 64 |
| Applying advanced application development techniques..... | 65 |
| Record management requests..... | 66 |
| Multitasking..... | 66 |
| Using the DFSMStvs automatic commit function..... | 68 |
| Chapter 6. Monitoring performance and tuning the DFSMStvs environment..... | 71 |
| Monitoring performance..... | 71 |
| SMF record type 42 (hexadecimal 2A)..... | 71 |
| SMF record type 88 (hexadecimal 58)..... | 72 |
| RMF post-processor reports..... | 72 |
| RMF monitor III..... | 73 |
| CICS monitoring tools..... | 73 |
| System messages..... | 73 |
| Operator commands..... | 73 |
| Shunted units of recovery..... | 73 |
| Effects of DFSMStvs, log stream, and data set states..... | 74 |
| Effects of DFSMStvs states based on events..... | 76 |
| Improving sequential performance..... | 92 |
| Improving logging performance..... | 92 |
| Tuning the DFSMStvs environment..... | 92 |
| Chapter 7. Diagnosing and recovering from DFSMStvs problems..... | 95 |
| Diagnosing system logger and performance problems..... | 95 |
| Categorizing a system logger problem..... | 95 |
| Collecting diagnostic information about logging problems..... | 96 |
| Investigating console messages and dumps..... | 96 |
| Displaying coupling-facility status | 97 |
| Checking global resource serialization (GRS) resource contention..... | 97 |
| Checking SMF and RMF statistics for performance problems..... | 98 |
| Interrupting an operation or resource request..... | 99 |
| Recovering from a log stream problem..... | 99 |
| Resolving waits..... | 100 |
| Restarting DFSMStvs after SMSVSAM address space failure..... | 100 |
| Cold starting DFSMStvs..... | 101 |
| Performing peer recovery..... | 101 |
| Peer recovery initiation..... | 102 |
| SMSVSAM failures while peer recovery is in process..... | 102 |
| System failures while peer recovery is in process | 102 |
| Peer-recovery interference with failed instance restart..... | 103 |
| Appendix A. Quiescing a data set..... | 105 |
| Appendix B. Accessing data sets that have retained locks or lost locks..... | 107 |
| Appendix C. Accessibility..... | 111 |
| Notices..... | 113 |
| Terms and conditions for product documentation..... | 114 |

| | |
|--------------------------------------|------------|
| IBM Online Privacy Statement..... | 115 |
| Policy for unsupported hardware..... | 115 |
| Minimum supported hardware..... | 115 |
| Trademarks..... | 116 |
| Glossary..... | 117 |
| Index..... | 133 |

Figures

| | |
|--|----|
| 1. CICS VSAM non-RLS access..... | 7 |
| 2. CICS VSAM RLS access..... | 7 |
| 3. Batch jobs designed to use transactional recovery..... | 17 |
| 4. Contexts as a series of units of recovery..... | 19 |
| 5. Two-phase commit processing actions..... | 20 |
| 6. Sample definition of the CFRM policy..... | 42 |
| 7. Sample log stream coupling-facility structure definition..... | 43 |
| 8. Sample definitions of the system logs..... | 45 |
| 9. Sample definitions of forward recovery logs..... | 49 |
| 10. Sample definition of the log of logs..... | 50 |
| 11. Example of an RACF PERMIT command..... | 51 |
| 12. Example of RACF RDEFINE commands..... | 52 |
| 13. Example of RACF commands to grant VSAM RLS authority to read and write log streams..... | 52 |
| 14. Example of key hashing..... | 61 |
| 15. Multitasking..... | 67 |
| 16. Example of MVS commands to produce a dump of XCF and system logger address spaces..... | 96 |
| 17. Example of a command to display system logger couple data set status..... | 97 |
| 18. Example of a normal response from a command to display system logger couple data set status..... | 97 |
| 19. Example of a command to reconnect the couple data set..... | 97 |
| 20. Example of a command to display all structures with Failed_Persistent connections..... | 97 |
| 21. Examples of DISPLAY GRS commands..... | 97 |
| 22. Example of a normal response from a DISPLAY GRS command..... | 97 |
| 23. Example of a GRS command showing contention..... | 98 |

| | |
|--|----|
| 24. Example of a GRS command to display log streams with an exclusive enqueue..... | 98 |
| 25. Example of output from a GRS command to display log streams with an exclusive enqueue..... | 98 |

Tables

1. Effect of MAXSYSTEM value on lock-table-entry size..... 25

2. Lock-allocation estimates..... 25

3. Opening recoverable and nonrecoverable VSAM data sets from batch jobs..... 57

4. Effects of DFSMStvs states on DFSMStvs processing..... 74

5. Effects of log states on DFSMStvs processing.....75

6. Effects of data set states on DFSMStvs processing 76

7. Installation exit environment and state..... 107

About this document

This document is intended for system programmers, application programmers, and operators who are responsible for customizing, writing applications for, and operating z/OS DFSMS Transactional VSAM Services (DFSMTsvs).

You can use this document to perform these tasks:

- Plan for using DFSMTsvs to share VSAM data for concurrent batch updates
- Install and operate DFSMTsvs
- Design and code applications to use DFSMTsvs
- Diagnose and recover from DFSMTsvs problems.

For information about accessibility features of z/OS, for users who have a physical disability, see [Appendix C, “Accessibility,”](#) on page 111.

Required product knowledge

Before you read this document, you should understand storage management concepts and be familiar with the virtual storage access method (VSAM) information in *z/OS DFSMS Using Data Sets*.

To use this document effectively, you should also be familiar with the following IBM products, programs, and components:

- CICS® (Customer Information Control System)

DFSMTsvs extends the availability of CICS by enabling multiple batch jobs and CICS to share access to the same data sets to update data as well as to read it. DFSMTsvs and CICS can use the same log of logs, a log stream that provides information for forward recovery programs.

- CICS VSAM Recovery (CICSVR)

CICSVR is a forward recovery program. It can use written to forward recovery logs by CICS or DFSMTsvs to recover VSAM data sets.

- Data Facility Storage Management Subsystem data facility product (DFSMSdfp)

DFSMSdfp provides functions for storage management, data management, program management, device management, and distributed data access. For information about DFSMSdfp, see *z/OS DFSMSdfp Storage Administration*.

- Hierarchical Storage Manager (DFSMSHsm)

DFSMSHsm provides automatic space management and availability functions through a hierarchy of storage devices. For information about DFSMSHsm, see *z/OS DFSMSHsm Storage Administration*.

- Recoverable resource management services (RRMS)

RRMS provides the context and unit-of-recovery management under which DFSMTsvs participates as a recoverable resource manager. Part of the operating system, RRMS comprises registration services, context services, and recoverable resource services (RRS). For information about RRMS, see *z/OS MVS Programming: Resource Recovery*.

- System logger

DFSMTsvs uses the services of the system logger for logging. For information about the system logger, see *z/OS MVS Setting Up a Sysplex*.

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- None.

Changed

The following content is changed.

September 2025 release

- None.

Deleted

The following content is deleted.

September 2025 release

- None.

Summary of changes for z/OS 3.1

The following content is new, changed, or no longer included in z/OS 3.1.

Deleted

The following content was deleted.

September 2023 release

- Removed the section *In order to enable DFSMStvs on your z/OS system* in [Chapter 3, “Configuring the DFSMStvs environment and defining resources,”](#) on page 35.

Chapter 1. Understanding the DFSMStvs environment

Before you plan for z/OS DFSMS Transactional VSAM Services (DFSMStvs), you need to understand it. This topic provides an overview of DFSMStvs in these topics:

- [“Transaction processing and transactional recovery” on page 1](#)
- [“VSAM record-level sharing \(RLS\)” on page 5](#)
- [“DFSMStvs overview” on page 13](#)
- [“Context services and RRMS” on page 17](#)

Transaction processing and transactional recovery

Different applications often need to share VSAM data sets. Sometimes the applications need only to read the data set. Sometimes an application needs to update a data set while other applications are reading it. The most complex case of sharing a VSAM data set is when multiple applications need to update the data set and all require complete data integrity.

Consider these examples:

- Transactions initiated by different applications might need to access the same VSAM data set at the same time.
- A transaction might need to access a VSAM data set at the same time that a batch job is using the data set.

Transaction processing provides functions that coordinate work flow and the processing of individual tasks for the same data sets. VSAM record-level sharing and DFSMStvs provide key functions that enable multiple batch update jobs to run concurrently with CICS access to the same data sets, while maintaining integrity and recoverability.

Terminology

This topic introduces several new terms or uses terms with definitions that are specific to either VSAM record-level sharing (RLS) or DFSMStvs.

Atomic

When an application changes data in multiple resource managers as a single transaction, and all of the changes are accomplished through a single commit request by a syncpoint manager, the transaction is called atomic. If the transaction is successful, all the changes will be committed. If any piece of the transaction is not successful, then all of the changes will be backed out. An atomic instant occurs when the syncpoint manager in a two-phase commit process logs a commit record for the transaction.

Backout

A request to remove all changes to resources since the last commit or backout or for the first unit of recovery, since the beginning of the application. Backout is also called rollback or abort. Any of these events can initiate backout:

- A user request
- An inability of a resource manager to commit resource changes
- An abnormal end of a user task while a transaction is in-flight

Commit

A request to make all changes to recoverable resources permanent since the last commit or backout or, for the first unit of recovery, since the beginning of the application.

Context

Sometimes called a work context, a context is a representation of a work request, or part of a work request, in an application. A context might have a series of units of recovery associated with it. A context represents a work request in an application, and the life of a context consists of a series of units of recovery, with zero or one unit of recovery associated with the context at any point in time.

Forward recoverable data set

A data set that was defined with the LOG(ALL) attribute option.

Forward recovery

A process used to recover a lost data set. The data is recovered from a backup copy and all the changes that were made after the backup copy was taken are applied. The forward recovery process requires a log of the changes made to a data set, together with a date and time stamp. The log of changes is called the *forward recovery log*.

Forward recovery log

A log that contains copies of records after they were changed. The forward recovery log records are used by forward recovery programs and products such as CICS VSAM Recovery (CICSVR) to reconstruct the data set in the event of hardware or software damage to the data set.

Log of logs

A log that DFSMStvs and CICS write to provide information to forward recovery programs such as CICS VSAM Recovery (CICSVR). The log of logs is a form of user journal that contains copies of the tie-up records that DFSMStvs or CICS has written to forward recovery logs. This log provides a summary of which recoverable VSAM data sets that DFSMStvs or CICS has used, when they were used, and to which log stream the forward recovery log records were written.

If you have a forward recovery product that can utilize the log of logs, ensure that all CICS regions that share the recoverable data sets write to the same log-of-logs log stream.

Nonrecoverable data set

A data set for which no changes are logged. Neither backout nor forward recovery is provided.

Primary system log

The *undo log*.

Recoverable data set

A data set that can be recovered using backout or forward recovery processing, defined with the LOG parameter set to UNDO or ALL.

Resource manager

A subsystem or component, such as CICS, IMS, or DB2®, or DFSMStvs, that manages resources that can be involved in transactions. There are three types of resource managers: work managers, data resource managers, and communication resource managers.

Secondary system log

The *shunt log*.

Shunt log

The secondary system log, which contains entries that were shunted to the log when DFSMStvs was unable to finish processing sync points. If a *unit of work* fails, it is removed (shunted) from the primary system log to the secondary system log, pending recovery from the failure.

Sync point

An end point during processing of a transaction. A sync point occurs when an update or modification to one or more of the transaction's protected resources is logically complete. A sync point can be either a commit or a backout.

Syncpoint manager

A syncpoint manager is a function that coordinates the two-phase commit process for protected resources, so that all changes to data are either committed or backed out. In z/OS, RRS can act as the system level syncpoint manager.

Two-phase commit

The process used by syncpoint managers and resource managers to coordinate changes in an *ACID transaction*, which is a transaction that involves multiple resource managers using the two-phase commit process to ensure atomic, consistent, isolated, and durable properties.

In the first phase of the process, resource managers prepare a set of coordinated changes, but the changes are uncommitted pending the agreement of all the resource managers involved in the transaction. In the second phase, those changes are all committed if the resource managers all agreed to them; or, the changes are all backed out if any of the resource managers failed or disagreed.

Using the two-phase commit process, multiple changes across multiple resource managers can be treated as a single ACID transaction.

Undo log

The primary system log, which contains images of changed records as they existed prior to being changed. Backout processing uses the undo log to back out the changes that a transaction made to resources.

Unit of recovery (UR)

A set of changes on one node that is committed or backed out as part of an ACID transaction.

A UR is implicitly started the first time a resource manager touches a protected resource on a node. A UR ends when the two-phase commit process for the ACID transaction changing it completes.

Unit of work

In DFSMStvs, one or more logical units of recovery that are committed or backed out together as a transaction.

Transaction processing

Transaction processing provides data sharing of recoverable resources and ensures that data is synchronized. In z/OS, transaction processing means that each logical unit of work runs as a single transaction. Products such as CICS, IMS, DB2, and DFSMStvs provide a transaction-processing environment. An application that maintains a database must be capable of managing transactions. The transactions can arrive virtually simultaneously from clients.

Transactional recovery

Transactional recovery is the capability to isolate the changes made to recoverable resources by a transaction. This means that when the transaction makes a change, that change is seen only by that transaction. After the transaction reaches the commit point, all changes that the transaction made are visible to other transactions. If an application decides to back out a transaction rather than complete it, then the resource manager backs out all changes that the transaction made. This transactional recovery is a major part of transaction processing.

For example, consider a context in which changes are made to two different records in a recoverable VSAM data set. A field in one record is decremented from 200 to 100. A field in the other record item is incremented from 700 to 800. Transactional recovery ensures that either both changes are made or neither change is made. When the application requests that the changes be committed, both changes are made *atomically*.

If an application makes these changes to nonrecoverable data and the application or the system fails, one or both of the changes could be lost.

Coordination of recovery

DFSMStvs requires three key programs and three key functions to coordinate the recovery of protected resources.

Key programs

These three programs work together to coordinate the recovery of protected resources and the records within a recoverable VSAM data set:

Application program

The application program accesses protected resources and requests changes to the resources.

Resource manager

A resource manager controls and manages access to a resource. A resource manager is an authorized program that provides an application programming interface (API) that enables the application program to read and change a protected resource. The resource manager, in conjunction with the syncpoint manager, controls whether changes to resources are committed or backed out. For DFSMStvs, the resources consist of records in recoverable VSAM data sets.

Syncpoint manager

A syncpoint manager uses a two-phase commit protocol to coordinate changes to protected resources, so that either all changes are made or none of the changes are made. Recoverable resource services (RRS) functions as the syncpoint manager.

Key functions

These three programs work in concert to provide the three basic functions needed to implement transactional recovery:

Resource locking

This function provides serialized access to changed resources.

Resource recovery logging

This function enables a resource manager to keep a record of the changes made to recoverable resources by a transaction.

Two-phase commit processing

This function provides the services that ensure that a transaction appears as an atomic operation. A two-phase commit protocol and backout processing work together to ensure that either all changes that are made by a transaction are committed, or that all of the changes are backed out.

Resource locking

Locking serializes access to the records that are being changed. Locking at a more granular level, such as record level rather than at a control interval (CI) level, is done to minimize the need to wait for locks. Record locks are generally released at syncpoint time, when commit processing occurs and the unit of recovery ends. A transaction might be required to wait if it requests a change to a record that is locked by another transaction. This wait generally lasts until the lock-holding transaction reaches a sync point or the transaction reaches a timeout.

Resource recovery logging

Using the services of the system logger, logging provides a means of backing out the changes to resources. The undo log contains images of changed records as they existed prior to being changed. When forward recoverability is requested, records are also written to a forward recovery log, which contains copies of records after they were changed. In the event that a unit of work fails, a secondary system log is used. The failed unit of work is moved from the primary log to the secondary log, pending resolution of the failure by the user.

Two-phase commit processing

When an application is ready to commit or back out its changes, the application invokes RRS through its commit or backout interface. RRS invokes commit or backout interfaces of the participating resource managers to begin the two-phase commit protocol or to back out the changes. The two-phase commit protocol is a set of actions. The actions ensure that all participating resource managers either make all changes to the resources represented by a unit of recovery or make no changes to the resources. The protocol verifies that the all-or-nothing changes are made even if the application program, the system, RRS, or a resource manager fails.

After an application program completes the set of updates to resources, it can invoke the RRS commit interface to make those changes permanent. The commit occurs in two phases. The first phase is called *prepare*, and the second phase is called *commit*.

During the prepare phase, the commit coordinator, RRS, invokes the *prepare exits* of each of the resource managers. The resource managers determine whether they can make the changes permanent during the commit phase. This means that they must keep their backout information until the commit phase in case they are told to back out the changes. If any of the resource managers are unable to perform the prepare function, RRS directs all of the resource managers to back out the transaction.

During the commit phase, the resource managers commit the changes represented by a unit of recovery and release any resource serialization. When all resource managers have completed the task, the application is notified, the unit of recovery is completed, and the two-phase commit process ends.

After making a number of uncommitted changes, an application program can choose to invoke the RRS backout interface to reverse all the changes. In this case, RRS calls backout exits for each of the resource managers involved in the transaction. In this exit, the resource managers restore the resources to their prior state and release any resource serialization.

VSAM record-level sharing (RLS)

VSAM record-level sharing (RLS) extends the DFSMS storage hierarchy to support a data-sharing environment across multiple systems in a Parallel Sysplex®. This support is primarily for VSAM data sets that online-transaction-processing applications use.

Overview of VSAM RLS

VSAM RLS processing involves support from multiple products:

- CICS Transaction Server
- CICS VSAM Recovery (CICSVR)
- DFSMS

VSAM RLS is a data set access mode that enables multiple address spaces, CICS application-owning regions on multiple systems, and batch jobs to access recoverable VSAM data sets at the same time.

With VSAM RLS, multiple CICS systems can directly access a shared VSAM data set, eliminating the need to ship functions between the application-owning regions and file-owning regions. CICS provides the logging, commit, and backout functions for VSAM recoverable data sets. VSAM RLS provides record-level serialization and cross-system caching. CICSVR provides a forward recovery utility.

The level of sharing that is allowed between applications is determined by whether or not a data set is recoverable. For example:

- Both CICS and non-CICS jobs can have concurrent read or write access to nonrecoverable data sets. However, there is no coordination between CICS and non-CICS, so data integrity can be compromised.
- Non-CICS jobs can have read-only access to recoverable data sets concurrently with CICS jobs, which can have read or write access.

VSAM RLS uses a coupling facility to perform data set-level locking, record locking, and data caching. VSAM RLS uses the conditional write and cross-invalidate functions of the coupling facility cache structure, thereby avoiding the need for control interval (CI) level locking. VSAM RLS uses the coupling facility caches as store-through caches. When a control interval of data is written, it is written to both the coupling facility cache and to direct access storage device (DASD). This ensures that problems occurring with a coupling facility cache do not result in the loss of VSAM data.

Data set types that VSAM RLS supports

VSAM RLS supports access to these types of data sets:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative-record data set (RRDS)
- Variable-length relative-record data set cluster (VRRDS)

VSAM RLS also supports access to a data set through an alternate index but does not support opening an alternate index directly in RLS mode. Also, VSAM RLS does not support access through an alternate index to data stored under z/OS UNIX System Services.

How CICS uses VSAM RLS

The CICS file-control program is a transactional file system built on top of VSAM. Without VSAM RLS, the CICS file-control program must perform its own record-level locking. The VSAM data sets are accessed through a single CICS. Local data sets are accessed by the CICS application-owning region (AOR) that submits requests directly to VSAM. Remote (shared) data sets are accessed by the CICS AOR that submits (function ships) a data set control request to a CICS file-owning region (FOR) and by the FOR that submits a request to VSAM.

[Figure 1 on page 7](#) illustrates the AOR, FOR, and VSAM request flow. CICS AORs function ship VSAM requests to access a specific data set to the CICS FOR that owns that data set. This distributed access form of data sharing has existed in CICS for some time.

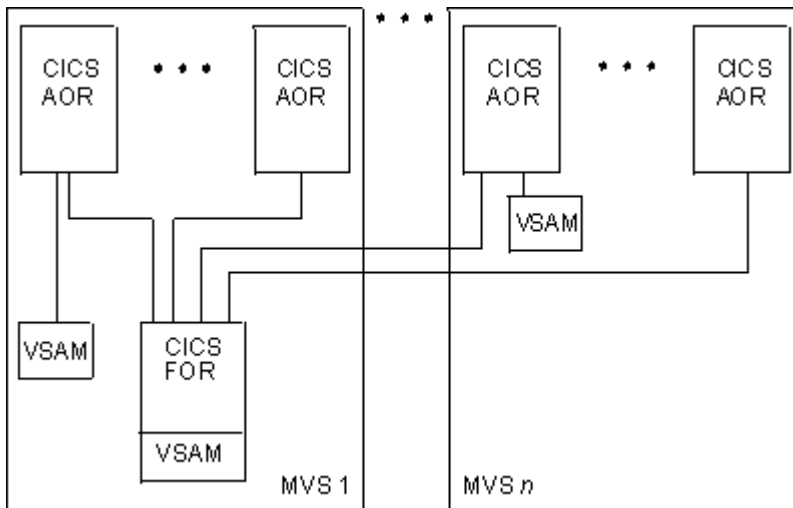


Figure 1. CICS VSAM non-RLS access

Figure 2 on page 7 illustrates how VSAM RLS enables multiple CICS transaction server AORs to access VSAM data sets for update while preserving data integrity. The updating regions can reside in any MVS image within a Parallel Sysplex. Each MVS image has one SMSVSAM server. Each AOR that needs to access the VSAM data in RLS mode connects automatically to the SMSVSAM server address space. VSAM RLS stores locking information in a coupling-facility structure that SMSVSAM owns.

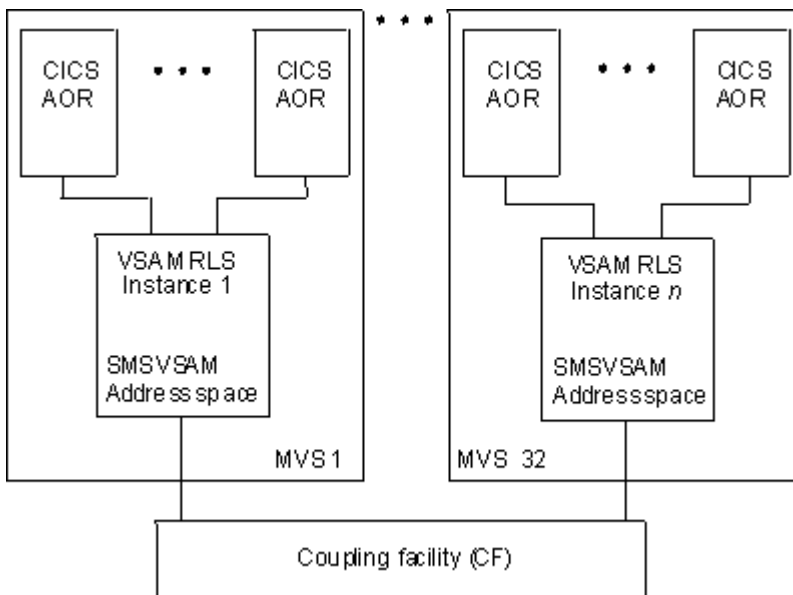


Figure 2. CICS VSAM RLS access

The CICS file-control program provides commit, backout, and forward recovery logging functions for VSAM recoverable data sets. With VSAM RLS, CICS continues to provide the transactional functions. VSAM RLS itself does not provide the transactional functions but does provide record locking based on a coupling facility as well as coupling-facility data caching.

Recoverable and nonrecoverable data sets

VSAM RLS introduced a VSAM data set attribute named LOG. With this attribute, you can declare a data set as recoverable or nonrecoverable. For recoverable data sets, a log of changed records is maintained and used to commit or back out a transaction's changes to a data set. CICS maintains logs of its changes to recoverable data sets.

The CICS file-control program supports recoverable and nonrecoverable data sets. The data set definition includes a recoverability attribute. You can specify these attribute options:

LOG(NONE)

Nonrecoverable.

This option declares the data set to be nonrecoverable. CICS does not perform any logging of changes for a data set that has this attribute. Neither backout nor forward recovery is provided.

LOG(UNDO)

Recoverable.

This option declares the data set to be backward recoverable.

LOG(ALL)

Recoverable.

This option declares the data set to be both backward recoverable and forward recoverable.

In addition to the logging and recovery functions provided for backout, CICS records the image of changes to the data set, after they were made. The forward recovery log records are used by forward recovery programs and products such as CICS VSAM Recovery (CICSVR) to reconstruct the data set in the event of hardware or software damage to the data set.

You can use CICS recoverable file attributes that correspond to VSAM data set attributes. The file definition must match the data set attributes in the catalog. You can use the IDCAMS DEFINE and ALTER (LOG(xxx)) commands to set the data set attributes, either by running an IDCAMS job or by defining them in the SMS DATACLAS parameter. You can also use the BWO and LOGSTREAMID parameters in the IDCAMS commands.

For CICS and DFSMStvs access, you can define VSAM data sets as recoverable with the backup-while-open (BWO) parameter set to TYPECICS. Both CICS and DFSMStvs can open a recoverable data set for output.

You can open a non-RLS data set in nonshared resources (NSR), local shared resources (LSR), or global shared resources (GSR) mode. When you open a data set in NSR, LSR, or GSR access mode, the recoverable attributes of the data set do not apply and are ignored. Thus, existing programs that do not use VSAM RLS access are not impacted by the recoverable data set rules.

Related reading:

- For more information about the IDCAMS DEFINE and ALTER commands, see [z/OS DFSMS Access Method Services Commands](#).
- For more information about NSR, LSR, and GSR access mode, see [z/OS DFSMS Using Data Sets](#).

CICS transactional recovery for VSAM recoverable data sets

In most cases, changes to recoverable data sets made by a transaction remain invisible to other transactions until the modifying transaction reaches the commit. However, if you are using the no-read integrity (NRI) option, you might see uncommitted changes. Exclusive locks that VSAM RLS holds on the modified records cause other transactions that have read-with-integrity requests and write requests for these records to wait. After the modifying transaction is committed or backed out, VSAM RLS releases the locks and the other transactions can access the records.

The CICS backout function removes changes made to the recoverable data sets by a transaction. When a transaction abnormally ends, CICS performs a backout implicitly.

The commit and backout functions protect an individual transaction from changes that other transactions make to a recoverable data set (or other recoverable resource).

How VSAM RLS provides functions

The SMSVSAM server provides VSAM RLS functions. This server resides in a system address space. The address space is created and the server is started at MVS IPL time. VSAM internally performs cross-address space accesses and linkages between requestor address spaces and the SMSVSAM server address space.

The SMSVSAM server owns two data spaces. One data space is called the SMSVSAM data space. It contains some VSAM RLS control blocks and a system-wide buffer pool. The other data space, named MMFSTUFF, is used by VSAM RLS to collect statistical information that is used to produce VSAM RLS system management facilities (SMF) 42 subtype 16-19 records.

VSAM RLS is an access option interpreted at open time. To enable RLS access, specify the MACRF=RLS parameter in the ACB macro, or specify the RLS parameter in JCL. The RLS MACRF option is mutually exclusive with the MACRF NSR (nonshared resources), LSR (local shared resources), and GSR (global shared resources) options.

Read sharing of recoverable data sets

A non-CICS application can open a recoverable data set for input in RLS mode. CICS provides the necessary transactional recovery for the write operations to a recoverable data set. Concurrently, non-CICS applications can have the sphere open for read-RLS access.

VSAM RLS read integrity options

VSAM RLS provides three levels of read integrity:

1. NRI (no read integrity)

This level tells VSAM not to obtain a record lock on the record accessed by a GET or POINT request. This avoids the overhead of record locking. This is sometimes referred to as a “dirty” read because the reader might see an uncommitted change made by another transaction.

Even with this option specified, VSAM RLS still performs buffer validity checking and refreshes the buffer when the buffer is invalid. Thus, a sequential reader of a KSDS does not miss records that are moved to new control intervals by CI and CA splits.

There are situations where VSAM RLS temporarily obtains a shared lock on the record even though NRI is specified. A shared lock is one in which several tasks can hold the lock. This happens when the read encounters an inconsistency within the VSAM sphere while attempting to access the record.

2. CR (consistent read)

This level tells VSAM to obtain a shared lock on the record that is accessed by a GET or POINT request. It ensures that the reader does not see an uncommitted change made by another transaction. Instead, the GET or POINT request waits for the change to be committed or backed out. The request also waits for the exclusive lock on the record to be released.

3. CRE (consistent read explicit)

This level has a similar meaning as CR, except that VSAM RLS holds the shared lock on the record until the end of the unit of recovery, or unit of work. CRE is not available to VSAM RLS (non-DFSMStvs) non-CICS applications. CRE is valid only when used by CICS or in DFSMStvs mode because of the syncpoint nature of the locks. VSAM can only handle an end-of-transaction for CICS. This capability is also called a repeatable read.

The record locks obtained by the VSAM RLS GET requests, with CRE option, inhibit the ability to update or erase the records by other concurrently executing transactions. However, the CRE requests do not inhibit other transactions from inserting other records.

Read-sharing integrity across KSDS control-interval and control-area splits

VSAM does not ensure read integrity across splits for non-RLS access to a data set with cross-region share options 2, 3, and 4. If the application requires read integrity, the application must ensure it. When

KSDS control-interval (CI) and control-area (CA) splits move records from one CI to another CI, the writer cannot invalidate the data and index buffers for the reader. This can result in the reader not seeing some records that were moved.

VSAM RLS can ensure read integrity across splits. It uses the cross-invalidate function of the coupling facility to invalidate copies of data and index control intervals in buffer pools other than the writer's buffer pool. This invalidation ensures that all VSAM RLS readers, CICS and non-CICS, are able to see any records moved by a concurrent CI or CA split. On each GET request, VSAM RLS tests the validity of the buffers, and if invalid, refreshes the buffers from the coupling facility or DASD.

Read and write sharing of nonrecoverable data sets

Nonrecoverable data sets do not participate in transactional recovery. Commit, backout, and forward recovery logging do not apply to these spheres. Because CICS and non-CICS applications are not required to use transactional recovery, VSAM RLS allows concurrent read-and-write sharing of nonrecoverable data sets. Any application can open the sphere for output in RLS mode.

VSAM RLS provides record locking and buffering across the CICS and non-CICS read or write sharers of nonrecoverable data sets. VSAM RLS releases the record lock when the buffer that contains the change is written to the coupling-facility cache and DASD. This record locking differs from the case in which a CICS transaction modifies VSAM RLS recoverable data sets. In that case, the corresponding locks are held until the end of a recoverable unit of work. For a CICS transaction, however, the locks are not held until the end of a transaction in either of these cases:

- The transaction is made up of multiple units of work.
- The locks are released at a sync point (at the end of each logical unit of work).

For sequential and skip-sequential processing, VSAM RLS does not write a modified CI until one of these actions occurs:

- The processing moves to another CI.
- The application issues an ENDREQ.

If an application or the VSAM RLS server abnormally ends, the changes to nonrecoverable data sets that were buffered are lost.

VSAM RLS permits read-sharing and write-sharing of nonrecoverable data sets across CICS and non-CICS programs. However, most programs are not designed to tolerate this type of sharing. The absence of transactional recovery requires very careful design of the data and the programs.

Non-RLS access to VSAM data sets

VSAM RLS access does not change the format of the data in the VSAM data sets. The data sets are compatible for non-RLS access. If you set the cross-region share option to 2, you can issue a non-RLS open-for-input request while the data set is already open for VSAM RLS access. However, a non-RLS open-for-output request fails. If the data set is already open for non-RLS output, an open for VSAM RLS fails. Therefore, at any point in time, a data set (sphere) can be open for non-RLS write access or open for VSAM RLS access.

CICS and VSAM RLS provide a quiesce function to assist in the process of switching a sphere from CICS RLS usage to non-RLS usage.

Recommendation: Be careful about quiesce with a path name; always use a base name.

Related reading: For information about the quiesce function, see [Appendix A, “Quiescing a data set,” on page 105](#).

Differences between VSAM RLS access and non-RLS access

This topic describes the differences between VSAM RLS access and non-RLS access.

Share options

For non-RLS access, VSAM uses the share options settings to determine the type of sharing allowed. VSAM provides full read and write integrity for the VSAM RLS users, but does not provide read integrity for the non-RLS user. You cannot specify a non-RLS open-for-output request when the data set is already opened for VSAM RLS.

VSAM RLS provides read sharing and write sharing for multiple users; it does not use share option settings to determine levels of sharing. When you request an RLS open and the data set is already open for non-RLS input, VSAM does check the cross-region setting. If the share option is 2, the data set can be opened in RLS mode. The open fails if you use any other share option, or if the data set is opened for non-RLS output.

Locking

Non-RLS access provides local locking (within the scope of a single buffer pool) at the VSAM CI level. A locking contention can result in an exclusive control conflict error response to a VSAM record management request.

All SMSVSAM servers (one per MVS image) use the coupling facility for locking. When contention occurs on a VSAM record, the request that encountered the contention waits for the contention to be removed. The lock manager provides deadlock detection. When a lock request is in deadlock, the request is rejected resulting in the VSAM record management request completing with a deadlock error response.

VSAM RLS supports a timeout value that can be specified through the request parameter list (RPL), in the PARMLIB member, or in the JCL. CICS provides two places where you can supply a timeout value:

- By transaction, you can specify the DTIMOUT value in the transaction definition.
- By CICS region, you can use the FTIMOUT system initialization value.

CICS uses DTIMOUT when it is supplied; otherwise, it uses FTIMOUT. This ensures that a transaction does not wait indefinitely for a lock to become available. VSAM RLS uses a timeout function of the lock manager.

Lock Retention

VSAM RLS uses share and exclusive record locks to control access to the shared data. An exclusive lock ensures that a single user is updating a specific record. The exclusive lock causes a read-with-integrity request for the record by another user, such as a CICS transaction or non-CICS application, to wait until the update is complete and the lock is released.

Failure conditions can delay completion of an update to a recoverable data set. This occurs when a CICS transaction enters in-doubt status. This means that CICS waits for a distributed unit of work to complete before CICS can back out or commit the unit of work. Therefore, the recoverable records modified by the transaction must remain locked. Failure of a CICS AOR also causes the current transaction's updates to recoverable data sets to not complete. They cannot complete until CICS restarts the AOR.

When a transaction enters an in-doubt state or a CICS AOR abnormally ends, exclusive locks on records of recoverable data sets held by the transaction must remain held. However, other users waiting for these locks should not continue to wait. The outage is likely to be longer than the user would want to wait. When these conditions occur, VSAM RLS converts these exclusive locks into retained locks.

Exclusive and retained locks are not available to other users. When another user encounters a lock contention with an exclusive lock, the user's lock request waits. When another user encounters a lock contention with a retained lock, the lock request is immediately rejected with a retained lock error response. This causes the VSAM record management request, which produced the lock request, to fail.

Non-RLS access while retained locks exist

Retained locks are created when a failure occurs. The locks need to remain until completion of the corresponding recovery. The retained locks have meaning only for RLS access. Lock requests issued by RLS access requests can encounter the retained locks. Non-RLS access does not perform record locking and does not encounter the retained locks.

To ensure integrity of a recoverable sphere, VSAM does not permit non-RLS update access to the sphere while retained locks exist for that sphere. An installation might need to run a non-CICS application that requires non-RLS update access to the sphere. VSAM RLS provides an IDCAMS command that sets the status of a sphere to enable non-RLS update access to a recoverable sphere while retained locks exist. This command does not release the retained locks. VSAM informs the CICS transaction servers that hold the retained locks when they later open the sphere with RLS.

Related reading: For information about non-RLS access to data sets that have retained locks, see [Appendix B, “Accessing data sets that have retained locks or lost locks,” on page 107.](#)

Requirements for VSAM RLS request execution mode

When a program makes a request while a data set is open in RLS mode (OPEN, CLOSE, or record management request), the program must be running in the following execution mode, with the listed constraints:

- Task mode (not SUB mode)
- Address space control=primary
- Home address space=primary address space=secondary address space
- No functional recovery routine (FRR) is in effect, but an ESTAE routine might be in effect.

The VSAM RLS record management request task must be the same task that opened the ACB, or the task that opened the access method control block (ACB) must be in the task hierarchy.

VSAM options that RLS and DFSMStvs do not support

VSAM RLS does not support these options and capabilities:

- Linear data sets
- Addressed access to a KSDS
- Control interval processing (CNV or ICI) to any VSAM data set type
- User buffering (UBF)
- Clusters that have been defined with the IMBED option
- Key range data sets
- Temporary data sets
- GETIX and PUTIX requests
- DFSMS checkpoint/restart facility
- ACB SDS (system data set) specification
- Hiperbatch
- Catalogs, VVDS, JRNAD exit, any AMP=JCL parameters
- Data that is stored under UNIX System Services

In addition, VSAM RLS does not support these usages:

- If the caller, executing in cross-memory mode, issues a request, RLS does not honor the request.
- When accessing a VSAM data set using the ISAM compatibility interface, you cannot specify RLS access.
- You cannot open individual components of a VSAM cluster for RLS access.
- You cannot specify a direct open of an alternate index for RLS access, but you can specify an RLS open request of a data set through an alternate index path.
- You cannot specify an RLS open request to implicitly establish position to the beginning of the data set. For sequential or skip-sequential processing, you must specify a POINT or GET DIR, NSP request to establish position.

DFSMStvs overview

Without DFSMStvs, a CICS system might have been available only during normal business hours. After business hours, the CICS system or application would be shut down for the supporting batch work to run. As soon as CICS stopped, backups were taken of key data sets as a point of recovery. Batch jobs could then be scheduled to run. If several jobs updated the same data set, they ran in sequence because they could not update the data set at the same time. After the updates were complete, a job would read the updated data and produce reports, and another backup would be taken. Finally, CICS could be restarted and become active again. However, the need to extend the availability of CICS has increased due to growing business volume.

DFSMStvs is an enhancement to VSAM RLS access that enables multiple batch update jobs and CICS to share access to the same data sets. DFSMStvs provides two-phase commit and backout protocols, as well as backout logging and forward recovery logging.

DFSMStvs provides transactional recovery directly within VSAM. DFSMStvs is an extension to VSAM RLS. It enables any job or application that is designed for data sharing to read-share or write-share VSAM recoverable data sets. VSAM RLS provides a server for sharing VSAM data sets in a sysplex. VSAM RLS uses coupling-facility-based locking and data caching to provide sysplex-scope locking and data access integrity, while DFSMStvs adds logging, commit, and backout processing.

DFSMStvs supports data sets that are defined as recoverable. That is, the log attribute for the data set is either UNDO (backout logging only) or ALL (backout and forward recovery logging). A batch job can open a recoverable data set for update in DFSMStvs mode. DFSMStvs provides the necessary transactional recovery for the data set. If a recoverable VSAM data set is opened for output and RLS is specified either in the JCL or the ACB, the data set is opened for DFSMStvs access.

Related reading: For information about programming language environments that support DFSMStvs, see [*z/OS DFSMStvs Administration Guide*](#).

Transaction processing from a batch job

DFSMStvs enables a batch job to open a VSAM recoverable data set for output concurrently with CICS online access. DFSMStvs is an extension of the function provided by VSAM RLS and, therefore, uses the same record-locking protocols as VSAM RLS. These locking protocols provide the data integrity that is required for concurrent access. DFSMStvs uses the record locking protocols when it does forward and backout logging. This provides a transaction-processing environment for batch jobs.

If you use DFSMStvs, you can run some batch jobs without taking your data sets offline. Most likely, you need to make changes to enable batch applications to use DFSMStvs.

DFSMStvs also provides batch program support for the repeatable read option. This option enables a transaction to reread a record and see the same data the second time. This is because a share record lock is obtained for this type of access, which prohibits other transactions from making updates until the transaction is committed or backed out.

Related reading For more information about application changes, see [“Coding an application to use DFSMStvs”](#) on page 54.

Recovery coordination

The records within a recoverable VSAM data set are protected resources. The application program, the resource manager, and the syncpoint manager work together to protect resources.

DFSMStvs builds on these components:

- VSAM record-level sharing (RLS)
- System logger
- Recoverable resource management services (RRMS)

These three components work in concert to provide the three basic functions necessary for implementing transactional recovery:

- Resource locking

This function provides serialized access to changed resources. VSAM RLS and DFSMStvs use a locking function based on a coupling facility, and the locking is done at the VSAM record level.

- Resource recovery logging

This function enables a resource manager to keep a record of the changes that a transaction makes recoverable resources. The system logger is the service that is used to accomplish this task.

- Two-phase commit and backout protocols

This function provides the services that ensure that a transaction appears as an atomic operation. It is the responsibility of two-phase commit and backout protocols to ensure that *all* of the changes made by a transaction are either committed or backed out. DFSMStvs uses the resource recovery services (RRS) provided by the MVS syncpoint manager to accomplish this task.

Access to recoverable VSAM data sets

For a non-CICS application to operate in DFSMStvs mode, begin by requesting RLS processing in either of these ways:

- Specify the MACRF=RLS parameter in the ACB macro.
- Specify the RLS parameter in the JCL.

In addition, do either of these tasks:

- Open a recoverable data set for output from a batch job.

The data set is open for DFSMStvs access if its recoverability attribute, LOG, is set to UNDO or ALL:

LOG=UNDO

Specifies backout logging.

LOG=ALL

Specifies backout logging and forward recovery logging.

- Request the CRE read-integrity option in the JCL or through the ACB.

VSAM RLS support enables batch jobs to access recoverable data sets in RLS mode, but only for reading; batch jobs cannot write to these data sets in RLS mode. With DFSMStvs, batch jobs are able to read and write to recoverable data sets while CICS is concurrently processing those same data sets. A non-CICS application must use RRS to partition the updates done to recoverable VSAM data sets into units of recovery. The application must invoke the RRS commit and backout interfaces to cause points of synchronization. The points of synchronization delineate units of recovery.

If a batch job attempts to communicate with RRS and the system logger goes down, RRS suspends the job until the logger comes back up. If this happens, you cannot cancel the job. To make RRS release the job, you need to bring the system logger back up or take RRS down.

Transaction processing

When an application uses DFSMStvs to process a transaction, DFSMStvs and RLS perform these tasks:

- Resource locking
- Logging the updates to records in recoverable data sets and logging the image of records before they are updated. This allows the updates to be backed out, if necessary. Optionally, DFSMStvs also records the images of records after they are updated so that forward recovery programs can use the records.

Serializing resources with locking

Locking serializes access to the records that are being added, changed, or deleted; this applies to both VSAM RLS as well as DFSMStvs access. In this way, CICS is allowed to access data sets in RLS mode, while a DFSMStvs application is accessing the same data sets.

VSAM RLS uses the coupling facility to provide locking for read and write integrity and caching. DFSMStvs uses the same locking structure and cache structure as other users of VSAM RLS. This allows applications that use DFSMStvs to read and write recoverable data sets concurrently.

When a transaction requests a change to a record, that record is locked. Any new request to change that record must wait. If a DFSMStvs job requests a record lock that is held by another task, the requesting job must wait until the lock is freed. Similarly, if a job holds a lock with data sets that are open in DFSMStvs mode, another task cannot obtain the lock.

VSAM RLS obtains the exclusive lock at GET UPD time, not when you change the record. DFSMStvs does backout logging at this time because GET UPD implies that the you are going to update the record, and a complete before image of the record is available then. Spanned and compressed records are not necessarily available any more when you get to the PUT or ERASE.

Because of the locking and logging overhead, GET UPD is not an efficient way to browse a data set when you use DFSMStvs. If you are browsing and updating only one record in ten or so, use GET NUP instead. If you need read integrity, you can specify CR or CRE. When you find a record that you want to update, do a GET UPD followed by PUT UPD or ERASE.

Logging resource recovery data

DFSMStvs uses the services of the system logger for logging. Logging captures the data needed for backout. An undo log is private to a unit of recovery. The undo log contains images of changed records as they existed *before* they were changed. The system logger writes log entries to the coupling facility, data space, DASD, or some combination of the three.

Log records can also be written to a forward recovery log. The forward recovery log contains copies of records *after* they were changed. The records are written to the system logger log streams. Forward recovery logs are shared between systems; these logs are shared across all CICS and DFSMStvs instances. This provides sysplex-wide merged log streams that can be used as input to forward recovery products such as CICSVR.

Initiating transactions

VSAM record management requests initiate transactions. DFSMStvs calls RRS to register an interest in a unit of recovery, which creates the unit of recovery and associates it with the task control block (TCB) or context.

Sync points are initiated by calling RRS interfaces for commit and backout processing.

Two-phase commit

To RRS, a unit of recovery consists of a set of changes that is to be made as one unit. A unit of recovery represents an application program's changes to resources since the last commit or backout. For the first unit of recovery, a unit of recovery represents changes since the beginning of the application. Each unit of recovery is associated with a context. The context consists of the unit of recovery together with the associated application programs, resource managers, and protected resources. A context represents a work request in an application, and the life of a context consists of a series of units of recovery, with zero or one unit of recovery associated with the context at any point in time.

In processing the first record management request of a transaction, DFSMStvs calls RRS to register an interest in a unit of recovery. When an application is ready to commit or back out its changes, the application invokes RRS. The invocation begins either the two-phase commit protocol or the backout of the changes.

The two-phase commit protocol is a set of actions. These actions are used to ensure that an application program makes either all changes to the resources represented by a unit of recovery or makes no changes to the resources. The protocol verifies that the all-or-nothing changes (sometimes called *atomic* changes) are made even if the application program, the system, RRS, or a resource manager fails. The data sets and the transaction are at a point of consistency based on the data committed with the last sync point, including when DFSMStvs restarts.

The first phase of the commit process is called *prepare*, the second phase is called *commit*. The phases of commit processing are described as follows:

Phase 1, prepare

During the prepare phase, the commit coordinator, RRS, invokes the prepare exits of each of the participating resource managers. In these exits, each of the resource managers determine whether they can make the changes permanent during the commit phase. This means that they must keep their backout information until the commit phase, in case they are told to back out the changes.

The process of making the changes permanent is called *committing*. If all of the resource managers are able to commit their changes, processing continues with the second phase of commit processing. If any of the resource managers is unable to perform the commit function, RRS directs all of the resource managers to back out the unit of recovery.

For data sets that have the forward recovery attribute, backout creates compensating forward recovery records. After a media failure and the restoration of a dated backup copy of the data, forward recovery reapplies all updates since that backup.

Phase 2, commit

In the second phase, the resource managers either commit or back out the changes represented by a unit of recovery, and they release the held locks. When all resource managers have completed the task, the following processing occurs:

- The application is notified.
- The unit of recovery is completed.
- The locks are released.
- The two-phase commit protocol ends.

If a batch application is using recoverable and nonrecoverable data sets, a commit does not affect the nonrecoverable data sets. A close or ENDREQs are required to get buffers written and locks released for nonrecoverable data sets.

Backout

DFSMStvs logs all updates to recoverable data sets that are modified in a unit of recovery. In this way, if a unit of recovery fails, the changes that were made as part of the unit of recovery can be reversed.

Each instance of DFSMStvs has a private undo log. This log contains the status of the units of recovery in a DFSMStvs instance and the backout records required to back out changes that were made to VSAM recoverable data sets by these units of recovery. Any of the following events can trigger a backout request:

- A backout request from the application
- An abnormal end of the unit of recovery or the TCB with which it is associated
- Detection of an error that DFSMStvs cannot fix, such as an I/O error on one of the data sets

If DFSMStvs detects such an error, it votes against committing the unit of recovery when the application calls prepare.

Should an instance of DFSMStvs fail or abnormally end, all in-flight units of recovery that were using that instance at the time of failure are backed out. The backouts for these units of recovery are not performed at the time of the failure by the failed DFSMStvs instance. The backouts are performed either at the time of the failure or later by a peer recovery instance of DFSMStvs.

Records added to the end of an ESDS are not backed out by DFSMStvs because no such function as ERASE exists for an ESDS. You cannot actually delete a record that has been written over in an ESDS. CICS has an exit that can modify the record to imply that it has been deleted.

How DFSMStvs works with RRS and other resource managers

For DFSMStvs, RRS performs syncpoint management for the resource managers, which provide the two-phase commit and backout services that ensure a transaction appears as one unit. Two-phase commit and backout processing must ensure that either all of the changes made by a transaction are committed or all are backed out.

Applications designed to follow the transaction model make it easy to share recoverable resources. Resource managers provide the sharing isolation when a transaction fails or when the execution environment fails. IMSDB and DB2 are resource managers that provide transactional recovery for their databases. The CICS file-control program provides transactional recovery for VSAM recoverable data sets accessed through CICS.

Restriction: When you are running DFSMStvs concurrently CICS and bad input was used for a batch job, you cannot restore the data set, bring CICS up, and then rerun the batch job later with good input as you could with just CICS. The forward recovery logs do not identify job names, which you would need to identify the specific records updated by the batch job with the bad input. If other batch jobs or CICS updated the same records, backing out the changes is not possible.

How DFSMStvs complements CICS

VSAM RLS provides an environment in which multiple CICS transaction servers can directly access a shared VSAM data set. At the same time, batch jobs can share nonrecoverable data sets for reading and updating while CICS is using them. VSAM RLS and CICS can also share recoverable data sets for reading when the data sets have share options defined. Batch jobs are able to share recoverable data sets for reading when they are opened in RLS mode.

Figure 3 on page 17 is an example of how DFSMStvs can be used with CICS and with batch jobs. DFSMStvs works with RRS to commit the VSAM data set changes and release the corresponding VSAM locks.

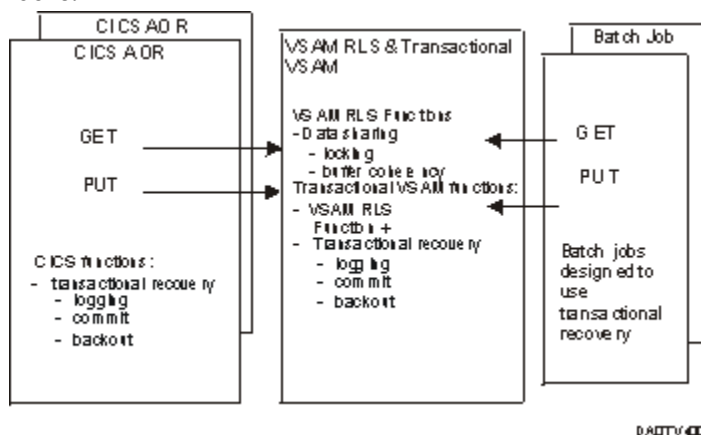


Figure 3. Batch jobs designed to use transactional recovery

While VSAM RLS provides multisystem data sharing for CICS, DFSMStvs is intended for non-CICS applications. From a logical perspective, CICS and DFSMStvs do the same kind of work. DFSMStvs and CICS complement one another by using the same locking protocols and the same type of commit and backout protocol. If both DFSMStvs and CICS want to update the same group of records, one application has to complete its current unit of recovery before the other can begin.

Context services and RRMS

Recoverable resource management services (RRMS) is part of the operating system and comprises registration services, context services, and recoverable resource services (RRS). RRMS provides the context and unit of recovery management under which DFSMStvs participates as a recoverable resource manager. This topic briefly discusses conceptual information about contexts and units of recovery as they relate to DFSMStvs.

A context is a unit of recovery together with the associated application programs, resource managers, and protected resources. Resource managers perform services for contexts and units of recovery. A resource manager allocates resources to a context and assigns units of recovery to it. A resource manager also allocates resources to a unit of recovery and assigns lock ownership to it. A unit of recovery is work that is done by or on behalf of a context between one point of consistency and another. A context represents a work request in an application, and the life of a context consists of a series of units of recovery, with zero or one unit of recovery associated with the context at any point in time.

Related reading: For more information about RRS, units of recovery, and contexts, see [*z/OS MVS Programming: Resource Recovery*](#).

Native contexts

A *native context* is the automatically occurring context of the application program, and the 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 MVS dispatchable unit (task) and always exists.

Privately managed contexts

A *privately managed context* is one that a resource manager creates. The resource manager owns any privately managed contexts it creates, and the resource manager can switch privately managed contexts from one task to another. A privately managed context is usually used by a work manager that is a resource manager. For example, IMS accepts and manages work, such as transactions, from outside the system.

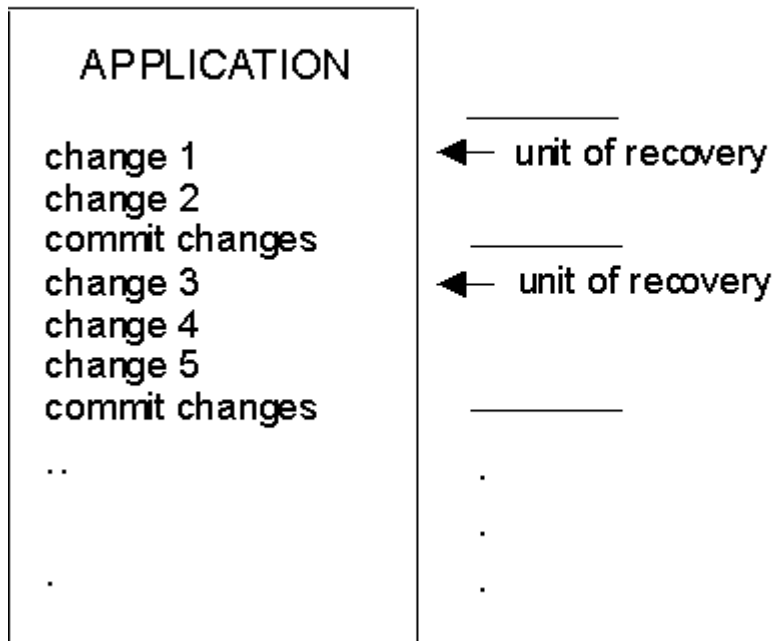
Every task in the system has an associated context, thus, for a given task there is always a context. When a task is created, context services provide the native context for the task. Resource managers can create privately managed contexts and associate them with a specific task. The privately managed 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. The native context would also become current if the privately managed context ends but the task with which it is associated does not. DFSMStvs does not create or switch contexts; it does, however, allow others to do so. DFSMStvs runs under a privately managed context only if another resource manager has created one. That context must be the current context associated with a task when DFSMStvs receives control.

Units of recovery

Using the two-phase commit protocols, applications use the commit service of the syncpoint manager to make changes permanent, and the backout service to back out the changes. When the syncpoint manager receives a commit request, the syncpoint manager synchronizes commitment of resources.

A unit of recovery is an entity in which a resource manager can express interest, letting the syncpoint manager know that the unit of work is using some protected resources that it manages. The protocol verifies the all or nothing changes, even if the application program, the system, RRS, or a resource manager fails. [Figure 4 on page 19](#) shows how contexts become a series of units of recovery.

CONTEXT



DA0TV510

Figure 4. Contexts as a series of units of recovery

A context has only one active unit of recovery at a time. This unit of recovery must complete before another unit of recovery begins. The syncpoint manager knows about the context and tracks resource managers that express interest in a given unit of recovery. When the application invokes the commit service, the syncpoint manager initiates the two-phase commit protocol with the interested resource managers. The syncpoint manager reports the result back to the application.

Unit-of-recovery states

The unit of recovery changes from state to state throughout the course of the two-phase commit protocol. Figure 5 on page 20 shows the flow of the two-phase commit process. In the first *in-reset* state, the application is at the start of its unit of recovery and has not used any protected resources. When protected resources are accessed for add, delete or update, the unit of recovery moves into the *in-flight* state with the resource manager expressing interest in the unit of recovery. Once the application issues its commit, the *in-prepare* state is entered. Phase one begins when the syncpoint manager tells each resource manager to prepare its resources to move forward or backward. When all resource managers reply positively, the unit of recovery transitions to the *in-commit* state and phase 2 begins. The syncpoint manager tells each resource manager to make its changes permanent.

When all resource managers have completed the task, the application is notified, the unit of recovery is completed, and the two-phase commit protocol ends. The application might have altered protected resources using multiple resource managers (for example, DB2, IMS, and DFSMStvs) within a recoverable unit of work.

The protocol also has provisions if a resource manager replies negatively to the prepare notification. In this case, the in-commit state becomes an in-backout state, and resource managers are told to back out any changes.

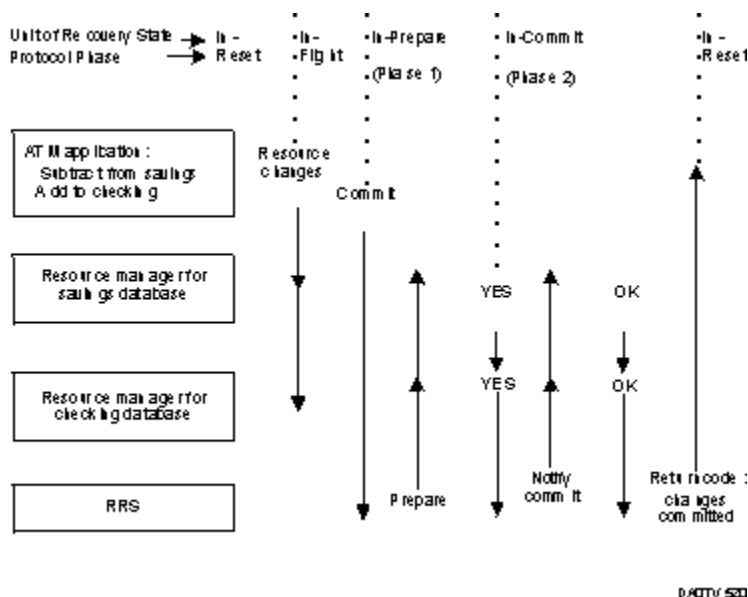


Figure 5. Two-phase commit processing actions

Distributed units of recovery

The resources that a unit of recovery updates can be distributed; that is, they can reside on more than one system. A unit of recovery can be distributed among multiple resource managers. Those resource managers can be on a local system or even on another system in the network. Two programs that participate in distributed resource recovery are DB2 and IMS.

For example, when multiple resource managers are involved, an application might do some DB2 work, and then some DFSMStvs work, and finally some IMS work. All of the work can be interrelated so that it is all under the umbrella of a single unit of recovery that RRS coordinates. The application completes its transactions and then calls RRS to commit or back out those transactions. In this way, the work within this unit of recovery spans several different resource managers. A unit of recovery can be distributed not only across multiple resource managers but across systems and networks.

Chapter 2. Planning for DFSMStvs

This information covers the following topics to help you plan for the use of DFSMStvs:

- [“Planning tasks” on page 21](#)
- [“Coupling-facility planning” on page 22](#)
- [“Processor-capacity planning” on page 26](#)
- [“Software-configuration planning” on page 27](#)
- [“System-logger planning” on page 27](#)
- [“VSAM operations planning” on page 32](#)
- [“Installation of DFSMStvs” on page 34](#)

Planning tasks

Because DFSMStvs builds on VSAM RLS, many of the planning steps for the implementation of VSAM RLS are common to a DFSMStvs implementation. You should perform the following tasks to plan for using DFSMStvs:

- Determine the hardware and availability requirements for coupling facilities. For DFSMStvs, the coupling facilities not only support lock tables and caching of data but should also be used for logging.
- Determine which applications need the ability to update recoverable data sets concurrent with CICS usage of those data sets, and then determine which of them can use the DFSMStvs support.
- Determine the size and number of coupling-facility cache structures and their connectivity.
- Determine the size of the coupling-facility lock structure. With the use of DFSMStvs, batch jobs update recoverable data sets while CICS is also using them, which results in increased locking activity.
- Determine the requirements for the SMS configuration, including storage-class specifications of coupling-facility structures, connectivity, and modifications to the ACS routines. If you have already established storage classes for VSAM RLS and coupling-facility cache structures, you can use those for both VSAM RLS and DFSMStvs. You might need to make the structures larger, however, due to increased usage.
- Add new parameters to the IGDSMSxx member of SYS1.PARMLIB for DFSMStvs initialization, quiesce time out, and activity keypointing.
- Convert batch jobs for use of DFSMStvs to update recoverable data sets:
 - Modify each job to recognize sync points and to take the appropriate action, whether commit or backout, by calling z/OS RRS services.

Recommendation: Do not perform a commit or backout through an RRS panel.

- Modify the programs or JCL to specify that DFSMStvs is to be used.
 - Examine each job to ensure that the use of multiple request-parameter lists within a single program will not cause lock contention within that program.
 - Code each job to handle the loss of positioning that occurs at commit or backout for unpaired requests (an unpaired request consists of a GET UPD not followed by a PUT UPD).
 - Examine each job to ensure that it handles any new return and reason codes from VSAM appropriately.
 - Introduce restart logic to prevent duplicate or missing updates in the event of a rerun.
- Update your recovery procedures to make use of forward recovery.

Coupling-facility planning

VSAM record-level sharing and DFSMStvs use coupling facilities to hold cached copies of individual records and to hold lock and log structures. Planning the use of coupling facilities is important to ensure that you meet your availability and performance objectives. This topic introduces some of the coupling-facility functions and provide information to help you decide the type and number of coupling facilities and how much storage you need to implement DFSMStvs.

Related reading: For more information about coupling facilities, see [z/OS MVS Setting Up a Sysplex](#).

Coupling facilities

A coupling facility provides locking, caching, and list services between coupling-capable z/OS processors. Coupling-facility links connect a coupling facility to the coupling-capable processors. The coupling-facility control code (CFCC) provides the coupling-facility functions. The CFCC can run in a processor resource/systems manager (PR/SM) logical partition (LP) in the following processors

- IBM Enterprise System/9000 (ES/9000) 9021 711-based processor
- An IBM S/390 Parallel Enterprise Server 9672 (IBM 9672)
- An IBM S/390® coupling facility 9674 (IBM 9674)

You can use the Integrated Coupling Migration Facility (ICMF) for migration and for testing software that requires a coupling facility. ICMF provides the CFCC functions without coupling-facility links and is limited to z/OS systems that run in an LP on a single physical system. ICMF is available on the 9021 711-based processors, 9121 511-based processors, and IBM 9672s.

A coupling facility enables software on different systems in the Parallel Sysplex to share data with the assurance that the data will not be corrupted and will be consistent among all sharing users. To share data, systems must have connectivity to the coupling facility through coupling-facility links.

DFSMStvs is one of many exploiters of coupling facilities. You might already have coupling facilities in use for one or more of these functions:

- Enhanced catalog sharing
- Tape drive sharing
- JES2 checkpoint
- Operator log
- Global resource serialization
- VTAM® generic resources
- SmartPipes
- Resource recovery services (RRS)
- CICS logs, temporary storage, and data tables
- DB2 locks and buffer pools
- IMS short message queue, locks and buffer pools
- VSAM record-level sharing (RLS) for caching and locking

Requirement: You must have at least CF level 2 microcode installed before using DFSMStvs.

Number of coupling facilities

A subsystem builds a structure in one coupling facility. If that coupling facility should fail, the subsystem is notified. Most subsystems attempt to rebuild the structure in an another coupling facility. A subsystem running on any z/OS system in the Parallel Sysplex that attempts a rebuild will rebuild the structure from information kept in processor storage.

Recommendation: For maximum availability, set up at least two coupling facilities in your Parallel Sysplex, with global connectivity to the multiple z/OS images in a multiple-processor environment. One is

not enough because a coupling facility could fail, or maintenance could require the temporary removal of a coupling facility from the Parallel Sysplex. Both coupling facilities should have enough processor power and storage resources to run as the only coupling facility, if necessary.

While it is possible to set a coupling facility up as an LPAR, this is not the ideal configuration. If you set the coupling facility up that way and the machine goes down, you lose both the coupling facility and the SMSVSAM server that was running in another LPAR. If there is only one coupling facility, this results in the loss of the lock structure as well as the SMSVSAM server. Any data sets it might have been accessing are now in lost locks throughout the Parallel Sysplex until the SMSVSAM image comes back up and does lost locks recovery.

Related reading: For information about non-RLS access to data sets that have lost locks, see [Appendix B, “Accessing data sets that have retained locks or lost locks,”](#) on page 107.

Standalone or internal coupling facility

You can configure the coupling facilities both in the processor where z/OS images run and in a special coupling-facility processor like the IBM 9674 S/390 coupling facility. The former is called an Internal coupling facility, and the latter is called a standalone coupling facility. A

You can configure an internal coupling facility (ICF) in either of these processors:

- IBM Enterprise System/9000 (ES/9000) 9021 711-based processor
- IBM S/390 Parallel Enterprise Server 9672 (IBM 9672)

The 9672 G3 servers (or later models) support an ICF, in which one of the central processors is used only to run CFCC, the licensed internal code of the coupling facility.

Both z/OS images and a coupling facility could fail at the same time in the event of a processor complex failure. In this situation, some structures cannot be rebuilt in another coupling facility. In general, an internal coupling facility should be used as backup for a production coupling facility if you do not also use coupling-facility duplexing.

A typical standalone coupling facility is an IBM 9674 S/390 coupling facility. If an IBM 9672 in which CFCC runs does not also have a z/OS image from the same Parallel Sysplex, this internal coupling facility has the same availability as a standalone coupling facility, which gives you the highest availability. At least one of the configured coupling facilities should be standalone.

If you use DFSMStvs under one z/OS system or in a one-processor environment, one internal coupling facility is sufficient. But, if you have a multiple-processor environment, two standalone coupling facilities are recommended for maximum availability. At a minimum, you should have at least one internal coupling facility and one standalone coupling facility.

Volatile or nonvolatile coupling facility

A volatile coupling facility is one in which interruption of the power supply causes loss of the memory contents. You can make a standalone coupling facility nonvolatile by adding to it an uninterruptible power supply, which provides power during external power failures. As an alternative, you can use a battery backup, which uses internal batteries to provide power during an outage.

The batteries last for a few minutes if the coupling facility continues to run. You can use the internal batteries to supply power for the power-saver state, in which they keep only the memory active during a power outage.

In general, you should put lock structures and list structures for the system logger in nonvolatile coupling facilities.

Contents of a coupling facility

A coupling facility stores information in structures. The structure types are cache, list, and lock, each providing a specific set of services to the exploiting system component. Coupling-facility users such as

CICS TS or SMSVSAM, or authorized programs, use coupling-facility storage to implement data sharing and high-speed serialization. This topic discusses the structures that VSAM RLS and DFSMStvs use.

Lock structure for VSAM sharing

In a Parallel Sysplex, you need only one lock structure for VSAM RLS because only one VSAM sharing group is permitted.

Recommendation: For high-availability environments, use a nonvolatile coupling facility for the lock structure.

If you maintain the lock structure in a volatile coupling facility, a power outage could cause a failure and loss of information in the coupling-facility lock structure. Should that happen, all outstanding recovery (CICS restart and backout) for any affected data sets must be completed before new sharing work is allowed for those data sets.

The name of the coupling-facility lock structure is IGWLOCK00.

Related reading: For more information about the IGWLOCK00 lock structure, see [z/OS DFSMSdjp Storage Administration](#).

Cache structures for VSAM sharing

The coupling-facility cache structures provide a level of storage between local memory and DASD cache. They are also used as a system buffer pool for VSAM RLS data with cross-invalidation being done when data is modified on other systems. Each coupling-facility cache structure is contained in a single coupling facility. You might have multiple coupling facilities and multiple coupling-facility cache structures.

List structures for the system logger

The system logger writes log data to log streams. The log streams are put in list structures. You can design both a single structure and multiple structures. CICS, SMSVSAM, and RRS use log streams to write logs.

Related reading: For more information about log streams, see [Chapter 4, “Setting up DFSMStvs logging,” on page 41](#).

Coupling-facility size

The size of a coupling facility is defined as the total amount of storage it uses. Total storage includes both the control areas that the coupling-facility control code (CFCC) requires and data areas that applications use. The coupling-facility allocation rules and coupling-facility allocation-increment size, a function of the level of CFCC, affect the size of the coupling facility.

The storage in a coupling facility is divided into distinct objects called structures. The majority of coupling-facility storage is used for structures. The Coupling Facility Resource Management (CFRM) policy defines these structures.

To determine the size of the system logs, look at the individual batch jobs that you plan to use with DFSMStvs and estimate how much space they are likely to require. Determine how many batch jobs there are and how much data they are likely to update per transaction.

An estimate of the amount of storage that you need in a coupling facility for DFSMStvs should include an estimate of the space required for the primary and secondary system logs (undo log and shunt log) and the additional space required in the log of logs and forward recovery logs. The undo log and shunt log need to be large enough for backout logging from all the batch jobs that you plan to use with DFSMStvs, based on how much data that the jobs are likely to update for each transaction. You also need to make the log of logs and forward recovery logs large enough to account for the extra activity from the batch jobs; if these logs already exist, you need to increase their sizes. These logs might need to be twice as large if the batch activity would be roughly equivalent to CICS activity and you do not plan to change your frequency of backups or of clearing out space in the forward recovery logs.

This topic discusses sizing of the structures that DFSMStvs uses:

- The IGWLOCK00 lock structure
- Coupling-facility ache structures
- List structures for log streams

Lock-structure sizing

You define the coupling facility lock structure, IGWLOCK00, in the CFRM policy, using the IXCMIAPU utility. The following formula is for estimating the size of IGWLOCK00 (in megabytes) for VSAM RLS. You can use this formula and then add additional space for the extra locking activity by batch jobs that use DFSMStvs.

$$10 \text{ MB} * \text{number_of_systems} * \text{lock_entry_size}$$

In the formula, the variables have these values:

number_of_systems

Is the number of systems in the Parallel Sysplex.

lock_entry_size

Is the size of each lock entry. This value depends on the MAXSYSTEM value that is specified to the IXCL1DSU couple-data-set format utility.

Use the information in [Table 1 on page 25](#) to determine the actual lock-entry size for different MAXSYSTEM values.

| Table 1. Effect of MAXSYSTEM value on lock-table-entry size | |
|---|-----------------|
| MAXSYSTEM value | Lock-entry size |
| 7 or fewer | 2 bytes |
| > = 8 and < 24 | 4 bytes |
| > = 24 and < 56 | 8 bytes |

[Table 2 on page 25](#) shows some sample lock-allocation estimates.

| Table 2. Lock-allocation estimates | | |
|------------------------------------|-------------------|--------------------------------|
| MAXSYSTEM value | Number of systems | Total lock-structure size (MB) |
| < = 7 | 2 | 40 |
| | 4 | 80 |
| default = 8 | 2 | 80 |
| | 4 | 160 |
| | 8 | 320 |
| 32 | 2 | 160 |
| | 4 | 320 |
| | 8 | 320 |

Use these estimates as rough initial values. Other factors influence the size of the lock structure, such as false contentions and retained locks.

Related reading:

- For more information about the sizing of the lock structure, see [z/OS DFSMSdfp Storage Administration](#).
- For more information about the MAXSYSTEM parameter, see [z/OS MVS Setting Up a Sysplex](#).

Cache-structure sizing

You need to define cache structures large enough for the activity of VSAM RLS as well as the additional activity of batch jobs that use DFSMStvs.

For the best performance with VSAM RLS buffering without DFSMStvs, the total of all of the coupling-facility cache-structure sizes that you define (the coupling-facility cache) should be the total of the VSAM local-shared-resource (LSR) buffer-pool sizes used in non-RLS mode. The VSAM LSR buffer-pool size is the sum of the LSR pool size and if used, the corresponding Hiperspace pool size.

You can run VSAM RLS with less coupling-facility cache storage than this, but the coupling-facility cache must be large enough for the coupling-facility cache directories to contain an entry for each of the VSAM RLS local buffers across all instances of the RLS server. Otherwise, the VSAM RLS local buffers become falsely not valid and must be refreshed. To minimize false invalidation, the size of the coupling-facility cache structure should be at least one-tenth of the sum of the local buffer-pool sizes.

Performance should improve when the coupling-facility cache is larger than the sum of the local VSAM LSR buffer pool sizes. When the coupling-facility cache is smaller, performance depends on the dynamics of the data references among the systems involved. In some cases, you might want to consider increasing the size of a very small (2 MB to 10 MB) coupling-facility cache.

Related reading: For more information about CICS TS structures, see [CICS Transaction Server for z/OS \(www.ibm.com/docs/en/cics-ts\)](http://www.ibm.com/docs/en/cics-ts).

List-structure sizing

The size of list structures is calculated by determining the sum of the log stream sizes. To estimate the total size of the log streams, see [“Log stream sizing” on page 31](#)

Coupling facility links

Three kinds of links are available between coupling facilities and z/OS:

- Single-mode Intersystem Coupling (ISC) link

This is an old kind of link with a data-transfer speed of 100 MB/second. If the distance between the coupling facility and the z/OS processor complex is more than 7 meters or your processor is not a 9672 G5 or later processor, you must use this kind of link.

- Integrated Cluster Bus (ICB)

This kind of link is supported by 9672 G5 and later processors and by z/OS Version 1 Release 3 and later systems. The data transfer speed is 280 MB/second. You can use this kind of link, however, only if the distance between the coupling facility and the z/OS processor complex is less than 7 meters. If your installation meets these requirements, you should use ICB links.

- Internal Coupling Channel (ICC)

This is not a real physical link. If you run both the coupling facility and z/OS in the same processor complex, you can use an ICC to connect the coupling facility and z/OS to get maximum performance. An ICC is supported by 9672 G5 and later processors onwards and by z/OS Version 1 Release 3 and later systems.

Processor-capacity planning

Before you can use DFSMStvs functions, you must have implemented a Parallel Sysplex and VSAM record-level sharing (RLS). In the same way that it is difficult to estimate the resources needed by coupling facilities, it is difficult to estimate the additional processor use for DFSMStvs applications.

These applications will, however, cause a noticeable increase in processor use. If you have spare capacity at the time you expect to run these jobs, this might not be an issue.

Software-configuration planning

If you have multiple z/OS images in a Parallel Sysplex, you can get near-continuous availability of DFSMStvs. If one z/OS system fails, the others can continue to run. When you stop one of the systems for hardware or software maintenance, you do not need to stop all the systems. If you have an environment with multiple z/OS JES2 systems, you should use JES2 Multiple Access Spool (JES/MAS). If you use JES2 MAS, batch jobs can be executed on any of the systems. In the case of a system failure, ARM supports started task and job restart on the other systems.

If your current software configuration has only a single z/OS image, you can still use DFSMStvs if you are not running it in LOCAL mode and the system is set up as a single-system sysplex. You can share VSAM data sets between CICS systems and batch programs, or among batch programs. By using sharing, you can aim for 24-hour availability for your online CICS system, and you can minimize or even eliminate your batch window. For batch jobs alone, DFSMStvs offers you the ability to run jobs in parallel against the same data.

The following configuration rules apply to DFSMStvs:

- The z/OS systems that share VSAM data sets must be in the same Parallel Sysplex and in the same SMSplex.

These z/OS systems must have connectivity to the same coupling facilities, Sysplex CDSs, SMS ACDS, SMS COMMDS, and CFRM CDSs.

- Each z/OS system can have only one SMSVSAM address space.
- You can have only one sharing group in one Parallel Sysplex because the names of the lock structures are fixed.

System-logger planning

The system logger provides a sysplex-wide logging mechanism for VSAM, CICS, and RRS. An integrated MVS function designed to support system and subsystem components in a Parallel Sysplex, the system logger implements a set of services that enables applications to write, browse, and delete log data. The system logger exploits coupling facility technology.

You can use the system logger services to merge data from multiple instances of an application, including from different systems across a Parallel Sysplex. The system logger significantly reduces the complexity of managing multiple logs (which would otherwise be required) and allows a single-system view of the log data in a Parallel Sysplex environment.

The system logger gives subsystems the following functions:

- Real-time merged logs
- Easy management of logs
- Continuous availability of logs

Subsystems that use these functions do not need separate log archival and log merge procedures.

Logging flow overview

Log entries that must be merged are written to the same log stream. The logging flow is as follows:

1. Each subsystem (CICS, SMSVSAM, or RRS) sends log entries to the system logger, IXGLOGR.
2. The system logger writes log entries, not only to log streams in the coupling facility, but also to the local buffer in the IXGLOGR data space.
3. If you set up DASD staging data sets (optional), the system logger also writes log entries to the DASD staging data sets.
4. If the utilization reaches the HIOFFLOAD value set in your LOGR policy, the system logger moves the log entries to DASD log data sets. This process is called offloading.
5. If a DASD log data set becomes full, the system logger allocates a new one.

6. When offloading is complete, the system logger deletes the offloaded log entries from the coupling facility, from the local buffers in the IXGLOGR data space, and from the DASD staging data sets.

Log streams

The log data area that subsystems use to write log entries is called a log stream. DFSMStvs, SMSVSAM, RRS, and CICS use log streams to write logs.

Log streams for DFSMStvs

DFSMStvs uses two log streams for backward recovery: the primary system log and the secondary system log. Each DFSMStvs instance must have both log streams. They cannot be shared between different instances of DFSMStvs.

Primary system log

The primary system log provides a separate backout log stream for each instance of DFSMStvs, called the undo log. The name of the undo log must be as follows:

```
IGTWTVnnn.IGWLOG.SYSLOG
```

In the name, **nnn** completes the DFSMStvs instance name.

The undo log must be allocated before you start DFSMStvs. This log stream is connected during DFSMStvs initialization and disconnected when the DFSMStvs instance ends.

A DFSMStvs instance logs its backout records in its undo log stream. These undo log records are used to back out the uncommitted changes made by a unit of recovery and are therefore written before the data is changed. An application can initiate backout, or it can occur as a result of an error, such as an abnormal end of the application.

Secondary system log

The secondary system log stream for DFSMStvs is called the shunt log. The name of the shunt log must be as follows:

```
IGTWTVnnn.IGWSHUNT.SHUNTLOG
```

In the name, **nnn** completes the DFSMStvs instance name.

The shunt log must be allocated before you start DFSMStvs. This log stream is connected during DFSMStvs initialization and disconnected when the DFSMStvs instance ends.

This log stream is used for the following units of recovery:

- Units of recovery that DFSMStvs is unable to complete (for example, due to an I/O error or unavailability of a resource, such as a volume or a cache)
- Long-running units of recovery. The log entries are moved to the shunt log.

RRS log streams

RRS uses five log streams that are shared by all the systems in a sysplex. Every MVS image with RRS running needs access to the coupling facility and the DASD on which the system logger log streams are defined.

Related reading: For information about the tasks you need to perform related to logging, see [z/OS MVS Setting Up a Sysplex](#).

Resource manager data log

This log includes the information about the resource managers using RRS services.

Restart log

This log includes the information about incomplete units of recovery needed during restart. This information enables a functioning RRS instance to take over incomplete work left over from an RRS instance that failed.

Main UR state log

This log includes the state of active units of recovery. RRS periodically moves this information into the RRS delayed unit-of-recovery state log when unit-of-recovery completion is delayed.

Delayed UR state log

This log includes the state of active units of recovery, when unit-of-recovery completion is delayed.

Archive log

This log includes the information about completed units of recovery. This log is optional but recommended.

Log streams for forward recovery

Forward recovery is optional. If you want to use it, you need two sets of log streams: forward recovery logs and a log of logs.

Forward recovery logs

The forward recovery logs compose a set of user-defined log streams that is connected when the first data set that uses it is opened and disconnected when the last data set that uses it is closed.

These log streams are shared across all instances of DFSMStvs. The forward recovery log stream used for recording the redo records for a data set is specified by the log stream ID attribute of the data set in the catalog entry. You can define the log stream ID at a data set level. The minimum number of log streams is one, and the maximum number is the same as the number of forward recoverable data sets. You can determine the number of forward recovery log streams and assign the log streams to the recoverable data sets.

Because redo log records contain changed images of records, they are written after the changed records are written to the data set. One exception to this is the case where sequential I/O is used and the records are buffered prior to being written out. The redo log stream might also contain compensating records, which are written during backout processing to indicate that records were restored to their original state.

DFSMStvs and CICS write to forward recovery log streams, and forward recovery programs, such as CICSVR, use them.

Log of logs

This log stream is used only if one is specified in the IGDSMSxx member of SYS1.PARMLIB or the CICS definition. Definition of the log of logs is optional. If you define a log of logs, it must be present; otherwise, DFSMStvs jobs would fail. It contains copies of the tie-up records and file-close records written to forward recovery logs and is used by forward recovery utilities, such as CICSVR, to automate the forward recovery process.

CICS log streams

Each CICS region uses a system log, which can be defined as DUMMY if recovery action is not required for the region. A DUMMY system log prevents warm restarts. If CICS uses only nonrecoverable data sets, this system log is optional. However, if CICS uses recoverable data sets, it is mandatory, because the system log is used for dynamic transaction backout.

The CICS system log is a single logical stream made up of two separately defined log streams: the primary system log and the secondary system log. If CICS uses recoverable data sets, one primary system log and one secondary system log are needed for each CICS instance.

CICS and DFSMStvs should share the log of logs so that forward recovery can be done correctly.

Structures and log streams

You can put multiple log streams in one list structure, thereby facilitating the management of structures.

Recommendation: Put the same kinds of log streams in the same structure.

Consider the following guidelines for structures and log streams:

- All data sets used by the same job should use the same log stream to reduce the number of log streams written to at a sync point.
- Consider sharing a forward recovery log stream between data sets that have similar security requirements and a similar backup frequency.
- Avoid mixing data sets that have a high number of updates with data sets with a low number of updates to avoid reading excessive amounts of log data when recovering a low update data set.
- Do not place all your high-update data sets in one log stream as this might exceed the throughput that a single log stream can provide.

For example, in a cloned CICS environment, all the primary system logs of the application owning regions have the same attributes, so they can be put in the same structures. In contrast, the primary system log of an application owning region and the primary system log of a TOR have completely different characteristics, so they should not be put in the same structure.

Recommendation: For DFSMStvs log streams, do not put the primary system log and the secondary system log in the same structure. You can put all primary system logs in the same structures. Similarly, you can put all secondary system logs in the same structures.

For RRS log streams, the main UR state log, the delayed UR state log, and the archive log can be put in one structure. The resource-manager data log and the restart log should be put in another structure.

DASD-only log streams

A DASD-only log stream has a single-system scope; only one system at a time can connect to a DASD-only log stream. Multiple applications from the same system can, however, simultaneously connect to a DASD-only log stream. No coupling facility is needed for a DASD-only log stream.

When a system logger application writes a log block to a DASD-only log stream, the system logger writes it first to the local storage buffers for the system and duplexes it to the DASD staging data set associated with the log stream. When the DASD staging data set's space allocated for the log stream reaches the installation-defined threshold, the system logger offloads the log blocks from its local storage buffers to VSAM linear data sets. From a user's point of view, the actual location of the log data in the log stream is transparent.

Both a DASD-only log stream and a coupling facility log stream can have data in multiple DASD log data sets; as a log stream fills log data sets on DASD, the system logger automatically allocates new ones for the log stream.

If you use DFSMStvs in a single-z/OS environment, you can use DASD-only log streams for all log streams. Even if you run in a multiple z/OS environment, the following log streams can be placed in DASD-only log streams because they are not shared:

- Primary system log for DFSMStvs
- Secondary system log for DFSMStvs
- Primary system log for CICS
- Secondary system log for CICS

If you use DASD-only log streams, performance problems can easily occur. In general, you should use the coupling facility for all production log streams because you must set up the coupling facility for DFSMStvs locking anyway.

Recommendation: Do not use DASD-only log streams for the log of logs or for forward recovery logs in a multisystem sysplex environment because they cannot be shared. The first system that opened a data set would be able to use the logs, but any subsequent opens on other systems would fail because they could not connect to the forward recovery logs.

Log stream sizing

To decide how much storage you need in a coupling facility, you must estimate the size of the log streams that you will store in it. Estimate a size for each of the logs that DFSMStvs uses:

- Primary system log for SMSVSAM
- Secondary system log for SMSVSAM
- Forward recovery logs
- Log of logs
- Log streams for RRS
- Primary system log for CICS
- Secondary system log for CICS

You can use the DFHLSCU utility, which CICS supplies, to help calculate the amount of coupling-facility space that you need and the average buffer size of your log streams.

Related reading:

- For information about how to use the DFHLSCU utility, see [CICS Transaction Server for z/OS \(www.ibm.com/docs/en/cics-ts\)](http://www.ibm.com/docs/en/cics-ts).

DASD staging data sets

In the system logger environment, log entries are written to both the coupling facility and the IXGLOGR data space before offloading. So, if a coupling facility or MVS failure occurs, log data is not lost. However, if both the coupling facility and MVS fail at the same time, log data is lost because the coupling facility and the data space are not permanent media.

Recommendation: Do not set up your sysplex with two LPARs, your MVS image and your coupling facility, on one machine.

If you want to protect against this double failure, you can keep the log data on disk in DASD staging data sets. The performance for log writes can degrade, however, if you choose DASD staging data sets, which IBM does

Recommendation: Do not use DASD staging sets for DFSMStvs logs except to protect against a double failure of MVS and the coupling facility.

If you decide to use DASD staging data sets, you need one per log stream. A DASD staging data set is a VSAM linear data set, and it should be managed by SMS. The following recommendations apply to defining log streams:

- Define STG_DUPLEX(YES) and DUPLEXMODE(COND) for log streams that are associated with the system log. This ensures that the system logger automatically duplexes to DASD staging data sets if it detects that the coupling facility is not failure independent.
- If you are operating with only a single coupling facility, define STG_DUPLEX(YES) and DUPLEXMODE(UNCOND) for log streams that are associated with the system log.
- Define STG_DUPLEX(YES) for log streams that are associated with forward recovery logs. If you do not and a failure causes loss of data from the log stream, you would need to take a new image copy of the associated VSAM data sets. There would be a consequent period of time until this was complete when the data sets would not be fully protected.
- Define each DASD staging data set to be at least the same size as the associated log stream's share of the coupling facility, but round the average block size up to 4 K.

DASD log data sets

The system logger writes log data to log streams in a coupling facility and to its data space. When the utilization of a log stream reaches the value of HIGHOFFLOAD that you have defined in your LOGR policy, the system logger will move the log data to disk data sets and delete the log data from the log stream. These data sets, called DASD log data sets, are allocated by the system logger as VSAM linear data sets and should be managed by SMS.

The recommended value for HIGHOFFLOAD is 85 percent for the primary system log.

Log trimming

DFSMStvs deletes most of the log stream entries in the undo log and shunt log automatically. For example, SMSVSAM periodically requests that the system logger delete old entries in the system log. This activity is called *log trimming*.

Log trimming does not apply to the log of logs or to any forward recovery log. Your installation is responsible for the maintenance of these logs.

DFSMStvs deletes entries that were needed for the backout of a unit of recovery that has now completed. Trimming keeps the size of the log down and makes it more likely that offloading of the log is not necessary.

Excessively large units of recovery might defeat the process of log trimming. Units of recovery might become excessively large if you run a batch job that does not issue commit or backout calls. Such programs would cause performance problems because they could force log offloads that would otherwise be unnecessary. They could also interfere with your online applications by holding a large number of locks.

Recommendation: Modify your batch jobs to issue periodic sync points before you convert them to use DFSMStvs.

Sizing

In general, the primary and secondary system logs should not be offloaded because you need high performance access. So if you have enough storage in the coupling facility for the system logger log streams, you do not need to spend much time estimating the capacity needed for DASD log data sets.

In contrast, you should estimate the capacity for DASD log data sets needed by forward recovery log streams and the RRS archive log stream because you will keep the log information until you think it is no longer needed to perform a forward recovery, probably because you have taken one or more backup copies. When you decide that you no longer need the forward recovery log entries, you need to archive or delete them yourself.

For example, if you take a backup copy of your VSAM data sets once a week, you might choose to delete the forward recovery log stream in DASD log data sets one week after taking the backup. In this case, you would estimate the capacity needed by the DASD log data sets for your redo logs for one week.

VSAM operations planning

DFSMStvs gives you sharing and forward recovery logging for VSAM data sets. To take advantage of these, you should review your backup and recovery procedures and your operational procedures for jobs using DFSMStvs.

Recovery procedures

DFSMStvs gives you the ability to do both forward recovery and backward recovery using a separate program such as CICSVR. You can either select both of them or just backward recovery at a VSAM data set level.

If you select forward recovery, the cost of running applications increases because of the additional logging activity. You also need operational procedures for log management and forward recovery.

Forward recovery operation planning

If you want to use forward recovery, you need to plan the following tasks:

- Regular operational tasks
 - Backup of VSAM data sets
 - Archive of redo logs
 - Eventual deletion of redo logs
- VSAM recovery tasks
 - Reallocate the VSAM data set
 - Restore the backup
 - Use a product such as CICSVR to apply the forward recovery logs.

Basically, backups of VSAM data sets are not used on their own. You perform forward recovery to return the data sets right up to the point of a failure.

Reorganization

If you need to reorganize your VSAM data sets to remove the effects of control interval and control area splits or to increase the space allocated, you will still need to perform these tasks while nobody else is using the data sets.

Automatic Restart Manager planning

z/OS includes a component known as the Automatic Restart Manager (ARM). ARM offers automated restart capabilities that are attractive in a CICS TS environment.

ARM is a recovery function that provides improved availability for batch jobs and started tasks by automatically restarting them after they unexpectedly end. In a Parallel Sysplex, ARM provides additional benefit through its ability to restart registered clients on another MVS system image in the Parallel Sysplex if the MVS system image they were originally using fails. Transaction managers, resource managers, and restartable subsystems can be restarted automatically. Any system that require automatic restart must register with ARM. Systems affected by a failure are usually restarted on the same MVS system image, or on a different one if the MVS system image itself has failed.

The cost of computing, productivity, and availability are improved because shared resources and transactions in progress can be recovered and lost function and services restored. The following points apply to the use of ARM:

- ARM provides job and started task recovery. Transaction or database recovery is the responsibility of the restarted applications.
- ARM does not start the applications initially (that is, at the first IPL or subsequent IPLs after failures). This is the function of automation or production control products. Interface points are provided through exits, the Event Notification Facility (ENF), and macros.
- The MVS system image or Parallel Sysplex must have sufficient spare capacity to guarantee a successful restart.
- Elements that are on an MVS system image that fails are restarted on another MVS system image.

ARM support is part of the overall task of planning for and installing Parallel Sysplex support.

Related reading: For more information about ARM, see [*z/OS MVS Setting Up a Sysplex*](#).

DFSMStvs and ARM

In case of an SMSVSAM failure or a CICS failure, locked records within some VSAM data sets might not be accessible because of retained locks. This lack of accessibility could prevent many transactions and jobs from being restarted. To maintain high availability, you should resolve the retained locks as soon as

possible. They will be resolved by a restart of CICS TS and SMSVSAM. The SMSVSAM address space will restart automatically. It would be helpful to use ARM for CICS automatic restart.

If the MVS system image fails, ARM restarts a peer recovery instance of DFSMStvs on another MVS system image.

If the SMSVSAM address space fails, all in-flight units of recovery that were using the DFSMStvs instance at the time of failure are backed out. The backouts for these units of recovery are not performed at the time of the failure by the failed DFSMStvs instance. The backouts are performed either at the time of the failure or later by a peer recovery instance of DFSMStvs.

However, in the case of a z/OS system failure, restart can take a long time in comparison, and the process is not automatic. So in-flight units of recovery can remain for a long time. To prevent this situation, DFSMStvs provides a function known as *peer recovery*.

When peer recovery occurs, another SMSVSAM instance in the sysplex (a peer) performs the backout for in-flight units of recovery that were being processed by the SMSVSAM address space that failed. Peer recovery will start automatically if there is an ARM policy (created using IXCMIAPU) which includes DFSMStvs and any other resource managers which might be involved in a unit of recovery in a restart group.

If you do not use ARM, you should change your operational procedures to issue the following command in another system as soon as possible in case of a system failure:

```
VARY SMS,TRANSVSAM(tvsname),PEERRECOVERY,ACTIVE
```

Installation of DFSMStvs

For information about enabling DFSMStvs on your z/OS system, see "Installation Information" in the ZOS V1R4 PSP upgrade, subsets ZOSGEN and DFSMS.

Chapter 3. Configuring the DFSMStvs environment and defining resources

Before your applications can use DFSMStvs, you need to set up your environment and define resources for DFSMStvs. You can use these list to help plan the tasks that you need to complete.

- Determine which applications need the ability to update recoverable data sets concurrently with CICS usage of those data sets.

Determine which of them can use DFSMStvs support.

- Determine the sizes and number of coupling-facility cache structures and their connectivity.
- Determine the size of the coupling-facility lock structure.

With the use of DFSMStvs, batch jobs update recoverable data sets while CICS is also using them. This might result in an increase in locking activity.

- Determine the requirements for the SMS configuration, including storage-class specifications, specification of coupling-facility cache structures, connectivity, and modifications to the ACS routines.

If you have already established storage classes for VSAM RLS and coupling-facility cache structures, it might be possible to use those for both VSAM RLS and DFSMStvs.

This information covers these topics:

- [“Defining your Parallel Sysplex environment” on page 35](#)
- [“Setting up the logging environment” on page 35](#)
- [“Using coupling facilities” on page 36](#)
- [“Defining staging data sets” on page 37](#)
- [“Specifying SYS1.PARMLIB parameters for DFSMStvs” on page 38](#)

For information about enabling DFSMStvs on your z/OS system, see "Installation Information" in the ZOSV1R4 PSP upgrade, subsets ZOSGEN and DFSMS. For migration considerations, see the migration chapter of *z/OS DFSMStvs Administration Guide*.

Defining your Parallel Sysplex environment

Two web-based assistants are available to ease your migration to a Parallel Sysplex environment.

- The z/OS Parallel Sysplex Customization Wizard is for system programmers who plan to migrate to a Parallel Sysplex environment for the first time. You can also use it to verify your Parallel Sysplex setup.
- The Coupling Facility Structure Sizer Tool simplifies the task of estimating the amount of storage you need for the coupling-facility structures in your installation.

You can find both assistants on the [IBM z/OS Parallel Sysplex \(www.ibm.com/products/zos/parallel-sysplex\)](http://www.ibm.com/products/zos/parallel-sysplex) website.

Setting up the logging environment

This topic outlines the tasks that you need to complete to set up the DFSMStvs logging environment correctly. These steps are in the order in which you should complete them. You might have completed some of the tasks in your planning phase.

1. Plan your DFSMStvs logging environment by gathering this information:
 - a. The number of DFSMStvs instances in your environment
 - b. The number of forward recovery logs that you require

- c. The expected average buffer sizes and transaction rates in your environment
- d. The number of coupling-facility structures that you require and their sizes
2. Ensure that you are authorized to use the MVS IXCMIAPU utility. This utility enables you to define, update, and delete entries in the LOGR couple data set.
3. Format and define the LOGR couple data set. You need to know how many log streams and structures that you are likely to need. Each DFSMStvs instance needs two system log streams. Optionally, you might want to have a log of logs and a number of forward recovery log streams.

Perform these steps:

- a. Use the MVS IXCMIAPU utility to create the LOGR couple data set.
- b. Define the LOGR couple data set to the sysplex and make it available.
4. Define coupling-facility structures in the CFRM policy couple data set.
5. Activate the LOGR subsystem.
6. Using RACF®, a component of the Security Server for z/OS, define RACF profiles in LOGSTRM classes so that DFSMStvs can access log streams.
7. Use the MVS IXCMIAPU utility to add information about log streams, log stream models, and coupling-facility structures to the LOGR couple data set.

Because DFSMStvs writes to forward recovery log streams, the space required for these log streams might increase. Similarly, because DFSMStvs writes to the primary and secondary system log streams (undo and shunt log streams), the demand increases for space in the coupling-facility structure. DFSMStvs moves long-running transactions and transactions that become shunted (due to backout failure) from the undo log stream to the shunt log stream.

The size of the coupling-facility structure for the primary log stream can affect performance. If the structure is too small, the system logger moves log buffers off to DASD spill data sets. While the system logger is moving log buffers, it is unable to service write requests from DFSMStvs.

Chapter 4, “Setting up DFSMStvs logging,” on page 41 describes how to set up DFSMStvs logging.

Related reading: For more information about the utility, see [z/OS MVS Setting Up a Sysplex](#).

Using coupling facilities

To plan and use a coupling facility configuration requires attention to the storage volatility of the coupling facility where the structures reside. The advantage of a nonvolatile coupling facility is that, if you lose power to a coupling facility that is configured to be nonvolatile, the coupling facility enters power save mode. This saves the data contained in the structures.

The safest environment is one in which there are two or more nonvolatile coupling facilities that are failure-independent from any of the exploiting MVS images, using dedicated processor resources.

One dedicated coupling facility plus a coupling facility LPAR provides the next safest environment for normal logging and lock-structure use. This environment has the same advantages of rebuilding with minimal impact to DFSMStvs instances that are running. Furthermore, MVS detects that the LPAR coupling facility is not in a failure-independent domain and causes the system logger to write log stream data to staging data sets for extra security.

The DFSMS cache structures are store-through caches, and no changed data resides in them. *Store-through* caching means that a user writes changed data to the cache structure and to permanent storage at the same time and under the same serialization. This way, the data in the cache structure always matches the data in permanent storage.

Because there is no chance of data loss, the DFSMS cache structures do not need to be in a nonvolatile coupling facility. The DFSMS lock structure has persistent connections and is a persistent structure. Data loss in the lock table requires a double failure. For example, a double failure occurs at the loss of the coupling-facility structure and the failure of one system in the sysplex at the same time. VSAM RLS also has a sharing-control data set to protect the integrity of the data even if there is a double failure. If this protection is not enough, the DFSMS lock structure can be placed in a nonvolatile coupling facility.

DFSMStvs caches all of its data in DFSMS cache structures that you define. The DFSMS lock structure and the system logger do all the logging for DFSMStvs. The IGWLOCK00 coupling-facility lock structure controls all VSAM RLS and DFSMStvs locks. DFSMStvs defines the names of its undo and shunt log streams, and you define the DFSMStvs log of logs and forward recovery log streams. RRS, which must be active for DFSMStvs to function, needs five log streams to be defined.

If you run with a single coupling facility, its failure would cause the system logger and any other users of the coupling facility to suspend normal operation until access to the coupling facility is restored. DFSMStvs would, effectively, be unusable in such a situation.

If a batch job attempts to communicate with RRS and the system logger goes down, RRS suspends the job until the logger comes back up. You cannot cancel the batch job because RRS has suspended it. If you cancel RRS or bring the system logger back up, however, RRS releases the batch job, which then ends; otherwise, the job remains suspended.

Recommendation: Do not perform a commit or backout through an RRS panel.

Unless you specify that the system logger is to use staging data sets, the recovery of log stream data depends on the MVS images remaining active. This is so that the system logger can use copies of log records that are held in storage to repopulate the coupling facility when it is again available. If you must run with a single coupling facility, specify the DUPLEXMODE(UNCOND) command to force the use of staging data sets.

Defining staging data sets

To ensure maximum protection of log data, the system logger allows duplexing of log write requests to both the coupling facility and to DASD. Duplexing provides failure-isolation of committed log data, even when the coupling facility and a connected MVS are in the same failure domain. For example, a failure domain is when the coupling facility and MVS are in the same central processor complex (CPC), or if the coupling facility storage is volatile. The records are written temporarily to a staging data set associated with the log stream. They are later offloaded asynchronously to permanent storage on the persistent log stream data sets. Offloading occurs when the HIGHOFFLOAD threshold of either the log stream or the staging data set is reached.

You can specify whether duplexing to staging data sets is either conditional or unconditional, depending on whether or not the coupling facility is in an independent failure domain. If conditional, the system logger chooses either the data space option or a staging data set; if unconditional, duplexing to a staging data set is forced.

MVS normally keeps a second copy of the data written to the coupling facility in a data space. The second copy is used to build a coupling-facility log in the event of an error. This is satisfactory as long as the coupling facility is failure-independent (in a separate central processor complex (CPC) and nonvolatile) from MVS.

The coupling facility is in the same CPC or uses volatile storage, the system logger supports staging data sets for copies of log stream data. That log stream data would otherwise be vulnerable to failures that impact both the coupling facility and the MVS images.

Use these guidelines when defining log streams:

- Unless you are operating with only a single coupling facility, define STG_DUPLEX(YES) and DUPLEXMODE(COND) for those log streams associated with the system log. This ensures that the system logger automatically copies log stream data to staging data sets if it detects that the coupling facility is not failure-independent.
- Define STG_DUPLEX(YES) for log streams that are associated with forward recovery logs. If you do not, you must take a new image copy of the associated VSAM data sets after any failure that causes loss of data from the log stream. Until the copying is complete, the data sets are not fully protected.
- Define all log streams with STG_DUPLEX(YES) and DUPLEXMODE(UNCOND) if both of these conditions are true:
 - The coupling facility is a nonvolatile, standalone coupling facility for normal logging

- A PR/SM LPAR, configured as a coupling facility, is acting as backup
- Define each staging data set to be at least the same size as the log stream share of the coupling facility, but round the average block size up to 4 KB.

Use this formula to calculate the staging data set size corresponding to the basic coupling facility space requirement for each DFSMStvs system log stream:

$$4 \times akp \times \text{average buffer size}$$

Round up the average buffer size to a multiple of 4096.

The activity keypoint frequency value (*akp*) specifies the number of physical outputs to the DFSMStvs system log before an activity keypoint is taken.

Specifying SYS1.PARMLIB parameters for DFSMStvs

You can define the following fields in the **IGDSMSxx** member of **SYS1.PARMLIB** for DFSMStvs:

```
LOG_OF_LOGS(logstream)
QTIMEOUT({nnn|300})
RLSTMOUT({nnn|0})
MAXLOCKS({max|0},{incr|0})
SYSNAME(sysname[,sysname[...]])
SYSTEMS({8|32})
TVSNAME(nnn[,nnn][...])
AKP(nnn[,nnn][...])
TV_START_TYPE({WARM|COLD}[,{WARM|COLD}[...])
TVSAMCOM({minval|1},{maxval|0})
```

For more information about the syntax and descriptions of these fields, see [z/OS MVS Initialization and Tuning Reference](#).

Some of the parameters in the **IGDSMSxx** member of **SYS1.PARMLIB** apply only to the system on which they are found; others apply across the sysplex. Values are not remembered across IPLs, so always specify a complete set of the parameters for DFSMStvs.

Recommendations

- Share the **IGDSMSxx** parmlib member across all systems in the sysplex.
- After a cold start, follow these steps for any data set for which recovery was not complete:
 1. Recover the data set manually (without using forward recovery).
 2. Take a backup of the data set and of any other data set that uses the same forward recovery log.
 3. Delete and redefine the data set's forward recovery log.

If recovery was not complete for a data set, it is most likely in a damaged state, and you must recover it manually. If the data set is forward recoverable, then its forward recovery log might also be damaged.

The DFSMStvs instance names must be unique. Because some parameters are positional, ensure that you specify the parameters in the proper order.

Defining a PARMLIB member specific to one system

This example defines a PARMLIB member that is specific to **one system**, and sets up these properties:

- An ACDS of SYS1.ACDS9
- A communications data set of SYS1.COMMDS
- An interval of 10 seconds before synchronizing with any other SMS subsystems in the complex
- One DFSMStvs name in the form of IGWTVnnn, for example, IGWTV001
- A quiesce timeout value of 240
- An activity keypoint (AKP) value of 2000

- No log of logs
- DFSMStvs is to perform a warm start

Code this statement in the IGDSMSxx member of SYS1.PARMLIB:

```
SMS ACDS(SYS1.ACDS9) COMMD(SYS1.COMMD5) INTERVAL(10)
    TVSNAME(1) QTIMEOUT(240) AKP(2000) TV_START_TYPE(WARM)
```

Defining a parmlib member that applies to multiple systems

This example defines a PARMLIB member that applies to **multiple systems** and sets up these properties:

- An ACDS of SYS1.ACDS10
- A communications data set of SYS1.COMMDS10
- An interval of 20 seconds before synchronizing with any other SMS subsystems in the complex
- Three instances of the DFSMStvs name, IGWTV001, IGWTV002, and IGWTV003, to be established on the systems named SYSTEM1, SYSTEM2, and SYSTEM3, respectively
- The default quiesce timeout value (300)
- The default activity keypoint value (1000)
- A log of logs, which has the log stream name: IGWLOG.LOGLOGS
- DFSMStvs performs a cold start on all systems

The IGDSMSxx member of PARMLIB includes this statement:

```
SMS ACDS(SYS1.ACDS10) COMMD(SYS1.COMMDS10) INTERVAL(20)
    LOG_OF_LOGS(IGWLOG.LOGLOGS)
    SYSNAME (SYSTEM1,SYSTEM2,SYSTEM3)
    TVSNAME( 1, 2, 3)
    TV_START_TYPE(COLD,COLD,COLD)
```

Recommendation: You should cold start DFSMStvs only when the log has been damaged and cannot be trusted to use for backouts. In any other situation, you should always warm start DFSMStvs because you would leave your data sets damaged if you did a cold start. Before a cold start, print the log so that you have some information about what might be damaged.

Chapter 4. Setting up DFSMStvs logging

DFSMStvs uses the system logger to record log records in log streams. A *log stream* is a collection of data in log blocks that reside in the coupling facility or on DASD. These log blocks are timestamped for ease of reference. Timestamps are stored in Greenwich Mean Time (GMT) and are independent of local time zones and time changes. DFSMStvs uses the system logger because it provides both sysplex-wide logging services and the integrity needed for protected resources. The system logger merges data from multiple instances of an application, including merging data from different systems in a sysplex.

This information covers these topics:

- “Determining the amount of logging to do” on page 41
- “Defining coupling-facility structures for log streams” on page 41
- “Allocating system log streams” on page 44
- “Using backout logging” on page 46
- “Defining forward recovery logs” on page 47
- “Creating a log of logs” on page 49
- “Authorizing access to log streams” on page 50

Determining the amount of logging to do

By logging updates, you can reconstruct data sets from backup copies, if necessary. When you specify the kind of logging that DFSMStvs should do, you must balance the need for logging with the disadvantages of logging. For example, logging can cause performance delays. Delays include the time that it takes to do the logging, from the standpoint of I/O, and the amount of logging that is done.

The recoverability of the data set determines the amount of logging that DFSMStvs does:

LOG(NONE)

The data set is nonrecoverable, and no logging is done. Nonrecoverable data sets are not normally accessed in DFSMStvs mode, but are accessed in DFSMStvs mode if CRE is specified.

LOG(UNDO)

The data set is recoverable and receives only backout logging. Syncpoint processing is done on its behalf, but the data set is not forward recoverable.

LOG(ALL)

The data set is recoverable and receives both backout and forward recovery logging.

Defining coupling-facility structures for log streams

You have to define log structures in two places: the coupling facility resource management (CFRM) policy and the system logger LOGR policy. Use the DEFINE STRUCTURE keyword of the ICXMIAPU utility to define the coupling facility structures needed for all the log streams that your DFSMStvs instances will use. Define structures in the LOGR policy in the system logger couple data sets.

Related reading: For details on how to code the ICXMIAPU utility, see [z/OS MVS Setting Up a Sysplex](#).

Use the CFRM policy to divide coupling facility space into structures. The CFRM policy enables you to define how MVS should manage coupling facility resources. [Figure 6 on page 42](#) shows the beginning of the definition of a CFRM policy:

```
//RLSPOL01 JOB CLASS=A,MSGCLASS=H,
//      MSGLEVEL=(1,1),REGION=4096K,TIME=1440
//*****
//* This job defines the CFRM policy. Log structures are only part
//* of it. This contains the structures for all the TVS logs and the
//* RRS logs. The structures must have also been defined to the
//* logger policy.
//*****
//*
//* UPDATE THE ADMINISTRATIVE POLICY DATA IN THE COUPLE DATA SET
//* FOR COUPLING FACILITY RESOURCE MANAGER (CFRM)
//*
//*****
//STEP20 EXEC PGM=IXCMIAPU
//*STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD SYSOUT=H
//SYSIN DD *
DATA TYPE(CFRM)
DEFINE POLICY NAME(RLSPOL01) REPLACE(YES)

    STRUCTURE NAME(OPERLOG_STR      )
        SIZE(60512)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)

    STRUCTURE NAME(LOG_DFHLOG_001  )
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHLOG_002  )
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHLOG_003  )
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHLOG_004  )
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHSUNT_001)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHSUNT_002)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHSUNT_003)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_DFHSUNT_004)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_USERJRNL_001)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_USERJRNL_002)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_USERJRNL_003)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_USERJRNL_004)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)
    STRUCTURE NAME(LOG_GENERAL_001)
        SIZE(10240)
        PREFLIST(CFSVTX,CFSVTY,CFSVTZ)

/*
```

Figure 6. Sample definition of the CFRM policy

Multiple log streams can write data to a single coupling-facility structure. This does not mean that the log data is merged; the log data stays segregated according to log stream. You can specify the number of log streams that use the resources of a single coupling facility structure. Use the LOGSNUM parameter on the IXCMIAPU service to define a structure.

Each log stream is allocated a proportion of the structure space, based on the number of currently connected log streams (up to the limit specified in LOGSNUM). For example, a structure can be defined to contain a maximum of 30 log streams. If only 10 log streams are connected, each log stream can use one tenth of the space in the structure. As other log streams are connected and disconnected, the system logger adjusts the proportion of space to each log stream used.

It is important to plan carefully before specifying a value for LOGSNUM. This parameter determines how much storage space in the structure is available to each log stream. A number in the range 10 to 20 is optimum in many environments.

Definition of a coupling-facility structure for a log stream

The example JCL shown in [Figure 7 on page 43](#) defines log stream coupling-facility structures to the system logger. The example is a guide; substitute values that are appropriate for your system.

```
//DEFSTRUC JOB ...

//POLICY EXEC PGM=IXCMIAPU
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
/*****
/* Define log stream CF structures to the MVS logger.
/* AVGBUFSIZE and LOGSNUM values are for illustration,
/* substitute values appropriate for your intended usage.
*****/
//SYSIN DD *
DATA TYPE(LOGR) REPORT(YES)
/* Primary system logs */
DEFINE STRUCTURE NAME(LOG_IGWLOG_001) LOGSNUM(10)
        MAXBUFSIZE(64000) AVGBUFSIZE(4096)
/* Secondary system logs */
DEFINE STRUCTURE NAME(LOG_IGWSHUNT_001) LOGSNUM(10)
        MAXBUFSIZE(64000) AVGBUFSIZE(4096)
/* Forward recovery logs and user journals that are forced */
DEFINE STRUCTURE NAME(LOG_IGWFR1) LOGSNUM(10)
        MAXBUFSIZE(64000) AVGBUFSIZE(8192)
/* Log of logs */
DEFINE STRUCTURE NAME(LOG_IGWLGLGS_001) LOGSNUM(10)
        MAXBUFSIZE(64000) AVGBUFSIZE(4096)
/*
//
```

Figure 7. Sample log stream coupling-facility structure definition

Log structure names

As you define your log structures, consider adopting a naming convention for your log structures that helps to identify the purpose of the structure. For example, you can use a format such as LOG_purpose_nnn:

- *purpose* identifies how you use the structure.
- *nnn* is a sequence number to allow for more than one structure for each purpose.

For example, you might choose the following names:

LOG_IGWLOG_001

For the DFSMStvs system log. The structure should be large to avoid the need to write data to DASD.

LOG_IGWSHUNT_001

For the DFSMStvs secondary system log. The structure should be small. However, it requires a large buffer size. A structure of 100 KB per log stream is usually sufficient.

LOG_FORWARD_001

For forward recovery logs in which block writes are forced periodically.

LOG_IGWLGLGS_001

For the log of logs.

These examples define one structure per log. Multiple log streams can be within a single structure so defining this many log structures is optional.

Related reading: For more information about the LOG_IGWLGLGS_001 log stream, see [“Creating a log of logs” on page 49](#).

Allocating system log streams

Each DFSMStvs instance requires its own pair of system logs: a primary (undo) log and a secondary (shunt) log. The system logs are used for recovery purposes. For example, they are used during backout and DFSMStvs restart. They are not meant to be used for any other purpose or by any other instance of DFSMStvs except in the case of peer recovery. *Peer recovery* is the process of completing the work that was left in an incomplete state by another instance of DFSMStvs when the system on which the instance was running failed. You can find more information about peer recovery in [Chapter 7, “Diagnosing and recovering from DFSMStvs problems,”](#) on page 95. DFSMStvs connects to its system logs automatically during initialization.

You must define the log streams for the DFSMStvs undo log and shunt log to the system logger before accessing a recoverable VSAM data set in DFSMStvs mode.

For each log stream that is in use, DFSMStvs needs two buffers. Each buffer can be a maximum block size of 64 KB for the log stream, and approximately 600 bytes of state data, all of which are above the 16 MB line. The SMSVSAM server address space contains the buffers.

A log stream is a sequence of data blocks, with each log stream identified by its own log stream identifier, which is called the log stream name. The DFSMStvs system logs, log of logs, and forward recovery logs map to specific MVS log streams. Each log stream is a sequence of blocks of user data, which the system logger internally partitions over three different types of storage:

Primary storage

This is a structure within a coupling facility that holds the most recent records written to the log stream. Log data written to the coupling facility is also copied to either a data space or a staging data set.

Secondary storage

When the primary storage structure for a log stream becomes full, the older records automatically spill into secondary storage. The secondary storage structure consists of data sets that are managed by the storage management subsystem. Each log stream, identified by its log stream name, is written to its own log data sets.

Tertiary storage

The tertiary storage is specified in your hierarchical storage manager (HSM) policy. Tertiary storage is used to migrate older records to some form of archive storage. This can be either DASD data sets or tape volumes. HSM manages this log data migration option.

The system logger provides access to these logs. DFSMStvs has access to the log stream data during recovery operations, when it can browse the data, read forward and backward, or read any block directly. [Figure 8 on page 45](#), [Figure 9 on page 49](#), and [Figure 10 on page 50](#) show you how you can define log streams for DFSMStvs to use. In these examples, each of the log streams has its own structure; your log streams can share structures.

Recommendations:

- Specify DIAG(YES) for the DFSMStvs system logs. This parameter indicates that special logger-diagnostic system activity is allowed for this log stream. This is especially useful when certain system logger errors occur, such as an X'804' type condition, which indicates that some data might have been lost.
- Log stream and log stream staging data sets are single extent VSAM linear data sets that require SHAREOPTIONS(3,3). The default is SHAREOPTIONS(1,3).

Recommendation: Explicitly specify the SHAREOPTIONS values.

Examples of system log stream definitions

The sample JCL in [Figure 8 on page 45](#) defines system log streams. These definitions are a guide, Substitute values that are appropriate for your system.

```
//JTVS01 JOB , 'DEF/DEL LOG STRMS',
//      CLASS=S,MSGCLASS=H,NOTIFY=SYSUSER
/*JOBPARM SYSAFF=SY02
//*****
//*This job deletes and redefines the system logs for TVS
//*INSTANCE IGWTV001, WHICH RUNS ON SY01.
//*****
//LGDELDEF EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*,DCB=RECFM=FBA
//SYSIN    DD *
DATA TYPE(LOGR) REPORT(NO)
DELETE LOGSTREAM NAME(IGWTV001.IGWLOG.SYSLOG)
DELETE LOGSTREAM NAME(IGWTV001.IGWSHUNT.SHUNTLOG)
DEFINE LOGSTREAM NAME(IGWTV001.IGWSHUNT.SHUNTLOG)
                STRUCTNAME(LOG_IGWSHUNT_001)
                LS_SIZE(6000)
                LS_DATACLAS(LOGRDC)
                LS_STORCLAS(S1P02S01)
/*                STG_DUPLEX(NO)                */
                STG_DUPLEX(YES)
                DUPLEXMODE(COND)
                STG_SIZE(6000)
                STG_STORCLAS(S1P02S01)
                STG_DATACLAS(LOGRDC)
                HIGHOFFLOAD(95)
                LOWOFFLOAD(15)
                DIAG(YES)

DEFINE LOGSTREAM NAME(IGWTV001.IGWLOG.SYSLOG)
                STRUCTNAME(LOG_IGWLOG_001)
                LS_SIZE(6000)
                LS_DATACLAS(LOGRDC)
                LS_STORCLAS(S1P02S01)
/*                STG_DUPLEX(NO)                */
                DIAG(YES)
                STG_DUPLEX(YES)
                STG_SIZE(6000)
                STG_STORCLAS(S1P02S01)
                STG_DATACLAS(LOGRDC)
                DUPLEXMODE(COND)
                HIGHOFFLOAD(95)
                LOWOFFLOAD(15)

/*
```

Figure 8. Sample definitions of the system logs

System log stream names

System log stream names are qualified names in which the high-level qualifier, IGWTV nnn , is the DFSMStvs instance name, such as IGWTV001. DFSMStvs requires the following log stream names:

- IGWTV nnn .IGWLOG.SYSLOG
- IGWTV nnn .IGWSHUNT.SHUNTLOG

DFSMStvs instance names must be unique throughout the sysplex. Each DFSMStvs instance accesses its system logs for its exclusive use. All other logs are separate from the system logs. The log stream names for forward recovery logs and the log of logs must not be the same as the names of the DFSMStvs system logs.

Offloading of log data

For better system performance, you can store the entire system log within the coupling facility to prevent the log from spilling to disk.

Recommendation: Take frequent sync points to minimize the amount of data being held in the coupling facility.

Generally, the volume of data that DFSMStvs keeps in the primary system log at any one time covers between two and three activity keypoints. The activity keypoint frequency, which is measured in blocks of log data, and defined by the AKP parameter in the IGDSMSxx member of SYS1.PARMLIB, determines this volume.

The way that DFSMStvs manages the system log data makes the frequency of activity keypointing an important factor when planning the size of the primary log stream. Use the AKP parameter to specify keypointing frequency. Review the activity keypoint (AKP) frequency defined for each DFSMStvs instance. The larger the value, the more coupling facility space you need for the system logs. But do not set AKP so low that transactions last longer than an activity keypoint interval. DFSMStvs manages the size of the system log by deleting old, completed units of work (log tail deletion). If you need long-term data retention, then you might want to copy the data from the log stream into alternative archive storage.

A *log tail* is the oldest end of the log. At each activity keypoint, DFSMStvs deletes the tail of the system log by establishing a point on the system log before which all older data blocks can be deleted. If the oldest "live" unit of work is in data block *x*, all data blocks older than *x* (*x*-1 and older) can be deleted. DFSMStvs keeps the two most recent, complete activity keypoints on the primary system log and deletes data from complete units of work older than this.

The system log is designed to ensure the availability of logged data following a system or DFSMStvs failure, thereby maintaining data integrity. Do not use the system log for any other purpose. Forward recovery logs cannot be written to system log streams.

Related reading: For more information about the AKP parameter, see [*z/OS MVS Initialization and Tuning Reference*](#).

Using backout logging

Each instance of DFSMStvs has a private backout log stream in the undo log. This log stream contains both the status of the units of recovery and the backout records that are required to back out changes to a VSAM recoverable data set that are made by units of recovery. Any of the following scenarios can trigger a backout:

- The application requests a backout.
- The unit of recovery abnormally ends.
- The application requests commit, but one of the resource managers detects a problem and responds *no* during the prepare phase. (An application can use multiple resources managers within a single unit of recovery.)

If DFSMStvs fails or abnormally ends, all in-flight units of recovery that were using DFSMStvs at the time of its failure are backed out. A unit of recovery that is in-flight is one that has made a change to a recoverable resource but has not yet committed or backed out that change. The backouts for these units of recovery are not performed at the time of the failure by the failed DFSMStvs instance. The backouts are performed either at the time of the failure or later by a peer recovery instance of DFSMStvs. You can find more information about DFSMStvs restarts and peer recovery in [Chapter 7, "Diagnosing and recovering from DFSMStvs problems,"](#) on page 95.

In the event of a DFSMStvs failure during the backout, the recovery or restart processing repeats the entire backout. Backout processing tolerates duplicate records and attempts to delete nonexistent record conditions, which arise from the attempt to repeat a backout operation that was previously completed.

Backout records for in-doubt and long-running units of recovery

A unit of recovery can become *in-doubt* between the time when the resource managers reply positively to the prepare notification from RRS and the time when RRS begins the commit phase. A unit of recovery can become in-doubt only if one of the interested resource managers has taken on the role of distributed or communication resource manager, and if more than one system is involved. In this case, after a commit request is issued and both systems have responded positively to the prepare request, the following processing occurs:

- The unit of recovery on the system that issued the commit request goes into the in-commit state

- The unit of recovery on the other system goes into the in-doubt state until the syncpoint resource manager receives the prepare response from the system that initiated the commit request.

A unit of recovery is considered *long-running* if it survives two activity keypoints without a sync point. This can cause the unit of recovery to hold a large number of locks until the next sync point, as well as to write a large number of log records.

The backout records for in-doubt and long-running units of recovery present space-management problems within undo log streams. Ideally, the backout records in an undo log stream have a short life cycle. This enables the obsolete portion of the log stream to be deleted. Also, the system logger does not need to off-load the log data from the coupling facility to DASD data sets. Units of recovery that do not reach an end-of-unit of recovery status within a short period of time interfere with this space management algorithm.

If too much old data is left in the log, there are two conditions that can occur when attempting to write records to a log stream. First, the system logger could return a return-and-reason code that indicates that the coupling facility storage limit was reached. Second, the system logger can return a return-and-reason code that indicates that the staging data set storage limit was reached. In either case, the system logger offloads data to DASD. DFSMStvs cannot write any further information to its log streams until the problem has been resolved.

DFSMStvs uses a secondary log, called a *shunt log*. This log tracks units of recovery for which DFSMStvs is unable to complete processing, for example due to an I/O error or unavailability of a resource, such as a volume or a cache.

Backout logging events

Logging begins when a request is made to add, delete, or update a record. An application might issue a pure read request and then later issue a get-for-update request (the first read did not initiate logging). Two VSAM requests, put (either update or add) and erase, can modify a record. The purpose of the undo log is to contain a copy of the data prior to any changes being made, so backout logging is always done when any indication is given of intent to update the data. Backout logging is done at the following times:

GET UPD

Get a record with the intent to update or erase it.

Recommendation: With RLS and DFSMStvs, use GET NUP instead of GET UPD for browsing because the use of GET UPD results in logging and obtaining an exclusive lock. If you need read integrity, you can specify CR or CRE. When you find a record that you want to update, do a GET UPD followed by PUT UPD or ERASE.

PUT ADD

Add a new record.

Backout records are not written to the undo log for a PUT UPD or ERASE request because these operations modify existing records that are first obtained using GET UPD.

Defining forward recovery logs

DFSMStvs supports a set of forward recovery log streams. These log streams are separate from the DFSMStvs undo log streams. Each installation determines the number of forward recovery log streams.

DFSMStvs logs the forward recovery records to system-logger log streams. These log streams are shared across DFSMStvs instances and provide sysplex-wide log streams that you can use as input to forward recovery products, for example, CICSVR.

For data sets with LOG(ALL) specified, DFSMStvs backout processing creates the forward recovery log records. These records, due to their more recent time stamps, supersede the corresponding forward recovery records generated during earlier processing of the unit of recovery. The log records are in the order in which the changes are made. When forward recovery is needed, the forward recovery product

reads through the records and actually makes the change, and then find the compensating record and undoes the change. The records do not supersede one another; both changes are applied.

You can use one forward recovery log stream for multiple data sets, so you do not need to define a log stream for each forward recoverable data set. Consider the following criteria before you decide to share a forward recovery log stream:

- Your installation's ability to balance transaction performance
- Rapid recovery
- The work involved in managing a large number of log streams

The system logger can store all the forward recovery log records from multiple DFSMStvs instances in a shared forward recovery log.

Here are some guidelines for defining forward recovery logs:

- Have all the data sets used by one transaction share the same log stream to reduce the number of log streams that are written to at a sync point.
- Share a forward recovery log stream between data sets if these conditions are true:
 - The data sets have similar security requirements.
 - The data sets have similar backup frequencies.
 - The data sets are likely to need restoring in their entirety at the same time.
- Choose log stream names that relate to the data sets. For example, PAYROLL.data_sets could be mapped to a forward recovery log named PAYROLL.FWDRECOV.PAYLOG.
- Do not mix high-update frequency data sets with low-update frequency data sets. This causes a disproportionate amount of unwanted log data to be read during recovery of low-update frequency data sets.
- Do not put all high-update frequency data sets on a single log stream because you could exceed the throughput capacity of the stream.
- Do not define too many data sets to a single log stream. Doing so could cause frequent structure-full events when the log stream cannot keep up with data flow.
- Delete redundant data from log streams periodically so that the log streams do not become excessively large. Typically, for a forward recovery log, the amount of old data that is deleted is related to the frequency that data is backed up. For example, you might keep the four most recent generations of backup. When you delete a redundant backup generation, you should also delete the corresponding redundant forward recovery log records. These are the records older than the redundant backup because they are no longer needed for forward recovery. The log of logs provides information to forward recovery programs such as CICSVR.

You can have a separate forward recovery log by data set, rather than by job. A separate forward recovery log by data set has these benefits:

- Recovery can be more straightforward with a one-to-one relationship between logs and the data sets to be recovered.
- Shared logs are related to multiple jobs and units of work using the same data set.

The log of logs provides a summary of which recoverable VSAM data sets DFSMStvs has used, when the data sets were used, and to which log stream the forward recovery log records were written. If you have a forward recovery product that can utilize the log of logs, ensure that all DFSMStvs instances sharing the recoverable data sets write to the same log of logs. Do not share the log of logs between test and production DFSMStvs instances. Sharing the log of logs could compromise the contents of production data sets during a restore operation.

[Figure 9 on page 49](#) gives an example of defining forward recovery log streams for DFSMStvs and CICS.


```

//DELFR   JOB , 'DEFINE FR LOGS',
//        CLASS=A,MSGCLASS=H,NOTIFY=SYSUSER
//*****
//*THIS JOB DELETES AND REDEFINES THE LOG STREAMS USED
//*FOR FORWARD RECOVERY BY THE TVS AND CICS WORKLOADS.
//*Note that this defines the structures and logs to the logger
//*policy. To use the structures, you must have defined them
//*to the CFRM policy.
//*****
//LOGDEF   EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=A,DCB=RECFM=FBA
//SYSIN    DD *
DATA TYPE(LOGR) REPORT(YES)

DELETE LOGSTREAM NAME(TVSRVY.GROUP1E)
DELETE LOGSTREAM NAME(TVSRVY.GROUP2E)
DELETE LOGSTREAM NAME(TVSRVY.GROUP1F)
DELETE LOGSTREAM NAME(TVSRVY.GROUP2F)
DELETE LOGSTREAM NAME(TVSRVY.GROUP3F)
DEFINE LOGSTREAM NAME(TVSRVY.GROUP1E)
    STRUCTNAME(LOG_IGWFR) LS_SIZE(4720)
    AUTODELETE(YES) RETPD(2)
    LS_DATACLAS(LOGRDC)
    LS_STORCLAS(S1P02S01)
    DIAG(YES)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    STG_SIZE(6000)
    STG_STORCLAS(S1P02S01)
    STG_DATACLAS(LOGRDC)
DEFINE LOGSTREAM NAME(TVSRVY.GROUP2E)
    STRUCTNAME(LOG_IGWFR) LS_SIZE(4720)
    AUTODELETE(YES) RETPD(2)
    LS_DATACLAS(LOGRDC)
    LS_STORCLAS(S1P02S01)
    DIAG(YES)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    STG_SIZE(6000)
    STG_STORCLAS(S1P02S01)
    STG_DATACLAS(LOGRDC)
DEFINE LOGSTREAM NAME(TVSRVY.GROUP1F)
    STRUCTNAME(LOG_IGWFR1) LS_SIZE(4720)
    AUTODELETE(YES) RETPD(2)
    LS_DATACLAS(LOGRDC)
    LS_STORCLAS(S1P02S01)
    DIAG(YES)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    STG_SIZE(6000)
    STG_STORCLAS(S1P02S01)
    STG_DATACLAS(LOGRDC)
DEFINE LOGSTREAM NAME(TVSRVY.GROUP2F)
    STRUCTNAME(LOG_IGWFR1) LS_SIZE(4720)
    AUTODELETE(YES) RETPD(2)
    LS_DATACLAS(LOGRDC)
    LS_STORCLAS(S1P02S01)
    DIAG(YES)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    STG_SIZE(6000)
    STG_STORCLAS(S1P02S01)
    STG_DATACLAS(LOGRDC)
/*

```

Figure 9. Sample definitions of forward recovery logs

Creating a log of logs

If you want DFSMStvs to write to a log of logs, you must specify so in the IGDSMSxx member of SYS1.PARMLIB. The log of logs contains copies of these records and information:

- Start-of-run records
- Tie-up records
- File-close records for recoverable data sets

- Log-stream exception information

The log of logs provides data set recovery products such as CICSVR with the information required to control forward recovery. If you use both DFSMStvs and CICS, use the same log of logs for both. When an application opens a file to access a data set, a tie-up record records the association between the file and the data set in the forward recovery log.

If you do not want DFSMStvs to write to a log of logs, omit the LOG_OF_LOGS parameter from the IGDSMSxx member of SYS1.PARMLIB.

If you use DFSMStvs in a sysplex and you use a log of logs, all DFSMStvs instances that access the same recoverable data sets should share the log of logs as a single log stream. [Figure 10 on page 50](#) gives an example of defining a log of logs for DFSMStvs and CICS.

```
//DDL0L JOB , 'DEF/DEL LOG STRMS',NOTIFY=SYSUSER,
//      CLASS=4,MSGCLASS=H
//*****
//* This job defines the log of logs that is identified
//* in the IGDSMSnn parmlib member and to CICS.
//* TVS does not allow multiple log of logs as CICS does.
//*****
//LGDELDEF EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*,DCB=RECFM=FBA
//SYSIN DD *
DATA TYPE(LOGR) REPORT(NO)
DELETE LOGSTREAM NAME(IGWTVS.LOG.OF.LOGS)
DEFINE LOGSTREAM NAME(IGWTVS.LOG.OF.LOGS)
                     STRUCTNAME(LOG_IGWLOG_01)
                     AUTODELETE(YES) RETPD(4)
                     STG_DUPLEX(YES)
                     DUPLEXMODE(COND)
                     STG_SIZE(6000)
                     STG_STORCLAS(S1P02S01)
                     STG_DATACLAS(LOGRDC)
                     LS_SIZE(1180)
                     LS_DATACLAS(LOGRDC)
                     LS_STORCLAS(S1P02S01)
/*
```

Figure 10. Sample definition of the log of logs

Authorizing access to log streams

You must define authorization for system logger resources so that DFSMStvs can access, read, and write to its log streams. DFSMStvs uses undo, shunt, log of logs, and forward recovery log streams. This authorization applies to log streams in the coupling facility and DASD-only log streams. You can use RACF or an equivalent security product to implement access to log streams.

Authorization to access log streams

Assign an RACF attribute of PRIVILEGED or TRUSTED to the VSAM RLS server address space (SMSVSAM) to define authorization to access the log streams. Privileged means that most RACROUTE REQUEST=AUTH macro instructions done for SMSVSAM are considered successful, without actually performing any checking. In addition, follow these guidelines:

- RACF does not call any exit routines.
- RACF does not generate any SMF records.
- RACF does not update any statistics.

Bypassing the checking step also applies to the checking done for the CHKAUTH operand on the RACROUTE REQUEST=DEFINE macro instruction. All other RACF processing occurs as usual.

The attribute TRUSTED is similar to PRIVILEGED. It means that RACF assumes most RACROUTE REQUEST=AUTH macro instructions that are done for SMSVSAM are successfully completed, without actually performing any checking. RACF performs the following functions:

- Does not call any exit routines

- Does not update any statistics
- Does generate SMF records that are based on the audit options specified in SETROPTS LOGOPTIONS and the UAUDIT setting in the USER ID profile

Bypassing the checking step also applies to the checking done for the CHKAUTH operand on the RACROUTE REQUEST=DEFINE macro instruction. All other RACF processing occurs as usual.

If the VSAM RLS server address space is neither PRIVILEGED nor TRUSTED, grant the SMSVSAM server the appropriate access authorization for log stream profiles. Specify the authorization in the RACF LOGSTRM general resource class for the user ID of the VSAM RLS address space, SMSVSAM. First you must associate SMSVSAM with an RACF-defined user ID, such as SYSTASK, in the started procedures table. Do not use a user ID of SMSVSAM.

DFSMStvs has no facility for controlling LOGSTRM security checks. These are controlled when the MVS security administrator activates the LOGSTRM general resource class by means of the SETROPTS command. If the LOGSTRM resource class is active, DFSMStvs needs update authority for the log stream profiles for the logs to which it writes. Use the user ID of the VSAM RLS address space, SMSVSAM, when you grant access. Define the log streams to MVS.

Permit read access to users who need to read the DFSMStvs system log streams, but not update access. Only DFSMStvs instances should have update access to the primary system log stream (undo log stream) and secondary system log stream (shunt log stream).

RACF RDEFINE coding

You can define the generic profile to cover all the log streams referenced by a DFSMStvs instance. For example, issue the command: `RDEFINE LOGSTRM tvname.* UACC(NONE)`. The *tvname* is in the form of `IGWTVnnn`.

The examples in [Figure 11 on page 51](#) give access to three categories of users:

```
PERMIT tvname.* CLASS(LOGSTRM) ACCESS(UPDATE)
      ID(smsvsam_userid)
PERMIT tvname.* CLASS(LOGSTRM) ACCESS(READ)
      ID(authorized_browsers)
PERMIT tvname.* CLASS(LOGSTRM) ACCESS(UPDATE)
      ID(archive_userid)
```

Figure 11. Example of an RACF PERMIT command

In these examples, *smsvsam_userid* is the user ID of the VSAM RLS address space in which DFSMStvs makes its calls to the system logger. The number of profiles you define depends on the following criteria:

- The naming convention that you used for the logs
- The extent to which you can use generic profiling

DFSMStvs also writes to forward recovery log streams and a log of logs that is used to optimize forward recovery. To protect these log streams, code the appropriate RDEFINE commands and PERMIT commands for each of them.

For all forward recoverable data sets that DFSMStvs accesses, grant DFSMStvs access to these logs:

- The log of logs
- The forward recovery log streams

Each of these log streams requires update authority for each of the other log streams. For example, issue the commands shown in [Figure 12 on page 52](#):

```
RDEFINE LOGSTRM FORWARD.RECOVERY.** UACC(NONE)
RDEFINE LOGSTRM FR.LOG.** UACC(NONE)
RDEFINE LOGSTRM LOG.OF.LOGS UACC(NONE)
PERMIT FOWARD.RECOVERY.** CLASS(LOGSTRM) ACCESS(UPDATE)
ID(smsvsam_userid)
PERMIT FR.LOG.** CLASS(LOGSTRM) ACCESS(UPDATE)
ID(smsvsam_userid)
PERMIT LOG.OF.LOGS CLASS(LOGSTRM) ACCESS(UPDATE)
ID(smsvsam_userid)
```

Figure 12. Example of RACF RDEFINE commands

For peer recovery to be possible, grant the VSAM RLS server the authority to read and write to the log streams of the DFSMStvs instances on other systems. Issue the commands shown in [Figure 13 on page 52](#) to grant the VSAM RLS server authority:

```
PERMIT IGWTV001.** CLASS(LOGSTRM) ACCESS(UPDATE)
ID(smsvsam_userid)
PERMIT IGWTV002.** CLASS(LOGSTRM) ACCESS(UPDATE)
ID(smsvsam_userid)
```

Figure 13. Example of RACF commands to grant VSAM RLS authority to read and write log streams

Chapter 5. Designing and coding applications to use DFSMStvs

Application programs that use DFSMStvs are written in much the same way as any other program that uses VSAM to read and write to data sets. For an application to participate in transactional recovery, you need to code the application to process data within transactions.

This topic covers this information:

- [“Determining which applications should use DFSMStvs” on page 53](#)
- [“Modifying an application to use DFSMStvs” on page 53](#)
- [“Coding an application to use DFSMStvs” on page 54](#)
- [“Handling long-running jobs and programs” on page 61](#)
- [“Using restartable applications” on page 62](#)
- [“Establishing positioning after logical errors” on page 63](#)
- [“Using sequential or random access to a data set” on page 63](#)
- [“Deleting and renaming data sets” on page 63](#)
- [“Monitoring and retrying shunted transactions” on page 64](#)
- [“Applying advanced application development techniques” on page 65](#)
- [“Using the DFSMStvs automatic commit function” on page 68](#)

Determining which applications should use DFSMStvs

You can determine which of your installation's applications should use DFSMStvs support by sorting them into these categories:

DFSMStvs intolerant applications

These applications use facilities that VSAM RLS does not support, access VSAM internal data structures, or are incompatible with VSAM RLS and DFSMStvs sharing.

DFSMStvs tolerant applications

These applications operate correctly in a multiple-updater environment when you specify either record-level sharing (RLS) in the JCL (job control language) or MACRF=RLS in the ACB (access control block). Without further modifications to an application, however, specifying RLS mode would make the entire application a single unit of recovery. RLS mode would also cause locks and log data to be kept for the length of the job. This could impact system resources and performance.

For these reasons, do not convert this type of application to use DFSMStvs access without additional application modification.

DFSMStvs exploiting applications

These applications recognize data sets that can be accessed using DFSMStvs, and they use DFSMStvs to access those data sets. Such an application also understands the scope of changes and sync points and makes use of resource recovery services (RRS) for commit and backout. These applications can read and write recoverable data sets that are concurrently in use by CICS.

Modifying an application to use DFSMStvs

You can modify an application to use DFSMStvs by specifying RLS in the JCL or the ACB and having the application access a recoverable data set using either open for input with CRE or open for output from a batch job.

If you want an application that currently uses nonshared resources to use DFSMStvs, consider these points:

- Write the application so that it uses RRS services and understands sync points. Write the application so that it issues commit or backout requests at regular intervals. An application that does all of its work under a single unit of recovery can significantly impact other applications and system resources. Holding locks for an excessive period of time could lock out other applications. An extremely long-running unit of recovery could result in your using significant processor storage to maintain control blocks, as well as overusing the undo log stream.
- Write applications that use DFSMStvs so that they can function in a multiple-updater environment. Do not convert an application that cannot function in this environment to use DFSMStvs. For example, do not convert an application that uses DISP=OLD, if you cannot convert it to use DISP=SHR.
- Use the no read integrity (NRI) level of integrity for read processing. Using NRI gives the application similar locking rules with RLS and DFSMStvs as it does with nonshared resources. Using the consistent read (CR) or consistent read explicit (CRE) level of integrity can result in waiting for the availability of a record. Waiting could result in a deadlock, a timeout, or a retained lock condition with other applications.
- Do not use open and record management requests if you do not write the application to handle a return code of 16 from these requests. Return code 16 implies that either RLS or DFSMStvs is not currently available. You do not need to be concerned with the application handling a return code of 16 for nonshared resources.
- Use SHOWCB or TESTCB macros to access control blocks other than the user access method control block (ACB), request parameter list (RPL), and exit list (EXLST). These control blocks are not available to the application other than through the facilities provided on SHOWCB or TESTCB macros.
- Understand that for update processing, unlike nonshared resources, the application waits for an exclusive record lock if another user has the record locked. The application is then subject to deadlock or timeout return codes.
- Understand that if you code a user processing exit routine (UPAD) exit, RLS and DFSMStvs ignore it.

Use care if a batch job uses multiple RPLs. Using different RPLs to access the same record can cause lock contention within a unit of recovery. Because sync points (commits or backouts) cause all locks to be released, the application cannot depend on locking across a sync point.

Coding an application to use DFSMStvs

This topic describes how to code an application to use DFSMStvs.

Defining transactions

You should not simply modify an existing batch job to use DFSMStvs with no further changes. This would cause the entire job to be seen as a single transaction and locks would be held. The log records would need to exist for the entire life of the job. This could cause a tremendous amount of contention for locked resources, and it could also cause performance degradation as the undo log becomes exceedingly large.

Understanding DFSMStvs restrictions

Any reader can share a data set, whatever the access mode. Batch jobs can also open nonrecoverable data sets for either input or output. Batch jobs can open recoverable data sets for update in non-RLS mode under the one of these conditions:

- Only when there is no program currently using them
- Non-RLS readers are the only users of the data sets

Batch jobs cannot open recoverable data sets for output in RLS mode; only CICS can do that. In an application that shares a recoverable data set, a batch job must use DFSMStvs to access the data set for output. The application must also support transactional recovery.

The following restrictions apply to DFSMStvs processing:

- Defer processing

The DFR/NDF specification in the ACB does not apply to either RLS or DFSMStvs.

For sequential and skip-sequential processing, RLS and DFSMStvs defer writing the current data buffer of the string. A subsequent request against the RPL in the string that positions the string to another control interval (CI), causes the modified buffer to be written to DASD.

- Load mode

VSAM hides load mode with either RLS or DFSMStvs access. Loading a data set in either RLS or DFSMStvs mode forfeits load mode optimizations available with nonshared resource access (for example, SPEED). If you want these options, load the data set with nonshared resource access.

- Positioning

An OPEN request for a data set with RLS or DFSMStvs does not establish an implicit position to at beginning of the data set. An explicit POINT request or GET NSP request is required.

- Locking

VSAM RLS and DFSMStvs obtain a record lock on POINT for consistent read and consistent read explicit. The record positioned to by the POINT request is the record that is returned on a subsequent GET SEQ request.

- Sharing

You can use the DISP=OLD parameter for data sets that are accessed with RLS or DFSMStvs, but this is not recommended.

Recommendation: Use the DISP=SHR parameter. The DISP=OLD parameter causes backouts to fail because DFSMStvs tries to allocate the data set with the DISP=SHR parameter.

The access method services DELETE command causes an exclusive enqueue on the data set name for the length of the job. Backouts in the same job fail, even if they are in different steps, because they are unable to allocate the data set dynamically.

- SHAREOPTIONS

For RLS and DFSMStvs access, use SHAREOPTIONS(2,X). This allows concurrent non-RLS readers while the data set is in use in RLS or DFSMStvs mode. RLS and DFSMStvs ignore all other specifications and assume that there are multiple concurrent writers and readers.

- DEFINE parameters

The parameters on the DEFINE command that are ignored or changed for RLS are also ignored or changed for DFSMStvs.

- Alternate indexes

RLS and DFSMStvs support alternate indexes through path access. However, RLS and DFSMStvs do not support a direct open request of an alternate index.

- Exits

Neither RLS nor DFSMStvs access supports the JRNAD exit. RLS and DFSMStvs ignore it. Use the RLSWAIT exit to perform a similar function; RLS and DFSMStvs access supports the RLSWAIT exit.

The RLSWAIT exit is optional. Applications that cannot tolerate VSAM suspending the execution unit that issued the original record management request use it. The exit must do its own wait processing. The wait processing is associated with the record management request that is being asynchronously executed. When the record management request is complete, VSAM posts the event control block (ECB) specified in the request parameter list of an event control block (RPLECB).

- Request environment

A VSAM RLS or DFSMStvs record management request task must be the same as the task that opened the ACB or at the same hierarchical level as the record management request. Issue VSAM RLS and DFSMStvs record management requests in AMODE 24 or AMODE 31. Issue OPEN, CLOSE, IDAQUIES, and record management requests in task mode, non-cross-memory mode, and primary ASC mode. A functional recovery routine (FRR) must not be in effect.

DFSMStvs does not support callers who are running in service request block (SRB) mode.

Unless privately managed contexts are used, the application must issue all record management, commit, and backout requests pertaining to a given unit of recovery under the same task control block (TCB). This is because a unit of recovery is associated with a context and a context is associated with a TCB. Any requests issued under a different TCB would be handled under the context associated with that TCB and would, therefore, belong to a different unit of recovery. The only exception to this is the case in which a resource manager is using privately managed contexts and switching them between TCBs.

A work manager, which takes work requests and parcels them out to tasks, might create privately managed contexts. These contexts and the units of recovery associated with them can be moved around, unlike the default contexts that every TCB in the system has. So, if the work manager knows that a piece of work it is about to submit belongs to a particular unit of recovery, the work manager can take that unit of recovery's privately managed context and attach it to the TCB before giving that TCB the work to do.

- Global resource serialization (GRS)

Like VSAM RLS, DFSMStvs requires GRS or an equivalent product that provides cross-system serialization to serialize:

- VSAM OPEN requests
- CLOSE requests
- EOVS processing
- Access to DFSMS control structures (for example, catalog)

Considering RLS and DFSMStvs restrictions

Like VSAM RLS, DFSMStvs does not support the following options and capabilities:

- Linear data sets
- Control interval access (CNV) for any data set organization.
- Addressed access to a KSDS (relative byte address or relative record address)
- Access to key-range data sets
- Access to clusters with the IMBED attribute
- Access to temporary data sets
- Access using the ISAM compatibility interface
- Opening the individual components of a cluster
- Direct open of an alternate index
- GETIX and PUTIX macros
- Checkpoint/restart facility
- Catalogs accessed in RLS or DFSMStvs mode
- VSAM volume data sets (VVDS) accessed in RLS or DFSMStvs mode
- The ACBDS specification
- Implicit positioning when a data set is opened (explicit POINT or GET NSP request required)
- Hiperbatch
- ADR MACRF option for a KSDS
- CFX MACRF option (ignored; NFX assumed)
- DDN or DSN MACRF options (ignored)

- DFR MACRF option (ignored; NDF assumed for direct requests that do not specify NSP)
- Improved control-interval processing (ICIP)
- Control blocks in common (CBIC)
- User buffering (UBF)
- SHRPOOL parameter of the BLDVRP macro (ignored)

Using VSAM data sets in a transaction

DFSMStvs supports access to these types of data sets:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative record data set (RRDS) (both fixed and variable record lengths)

VSAM data sets are stored on direct access storage devices (DASD). VSAM divides its data set storage into control areas, which are further divided into control intervals. Control intervals are the unit of data transmission between virtual and auxiliary storage. Each one is of fixed size and, in general, contains a number of records. A KSDS or ESDS can have records that extend over more than one control interval. These are called *spanned records*.

Accessing a data set with DFSMStvs

To enable DFSMStvs processing, begin by requesting RLS processing in either of these ways:

- Specify the MACRF=RLS parameter in the ACB macro.
- Specify the RLS parameter in the JCL.

In addition, do either of these tasks:

- Open a recoverable data set for output from a non-CICS batch job.

The data set is recoverable if its LOG parameter is specified as UNDO or ALL.

- Request the CRE read-integrity option in the JCL or through the ACB.

VSAM RLS enables batch jobs to access recoverable data sets in RLS mode, but only for reading; batch jobs cannot write to these data sets in RLS mode. With DFSMStvs, batch jobs that support two-phase commit and backout protocols are able to read and write to recoverable data sets. Concurrently, CICS is processing those same data sets.

For DFSMStvs, a commit protocol application includes programs that can be invoked from a batch job and that use RRS to support two-phase commit and backout. To use DFSMStvs, batch jobs must support and understand sync points (commit and backout) and use RRS services.

Table 3 on page 57 shows the type of open resulting from the parameters specified for a VSAM data set opened by a batch job (non-CICS). The type of data set and type of open appear in the first column. Other column headings indicate the RLS option specified.

| Table 3. Opening recoverable and nonrecoverable VSAM data sets from batch jobs | | | |
|--|----------|----------|----------|
| Data set type and type of open | NRI | CR | CRE |
| Recoverable open for input | RLS | RLS | DFSMStvs |
| Recoverable open for output | DFSMStvs | DFSMStvs | DFSMStvs |
| Nonrecoverable open for input | RLS | RLS | DFSMStvs |

Table 3. Opening recoverable and nonrecoverable VSAM data sets from batch jobs (continued)

| Data set type and type of open | NRI | CR | CRE |
|--------------------------------|-----|-----|----------|
| Nonrecoverable open for output | RLS | RLS | DFSMStvs |

The results of opening different types of VSAM data sets with different read integrity options (NRI, CR, or CRE) depend on whether the batch job opens the data set for input or output, as follows:

1. If you open a recoverable data set for output from a batch job (non-CICS) and you specify RLS with any option (NRI, CR, or CRE), you get a DFSMStvs open.
2. If you attempt to open a recoverable data set for output from a batch job (non-CICS) and you do not specify RLS, the open will fail (unless the open attempt is during your batch window processing and you have disabled access by CICS to the data set).
3. If you open a recoverable data set for input from a batch job (non-CICS) and you specify RLS with any option (NRI, CR, or CRE), you get an RLS open.
4. If you open a recoverable data set for input from a batch job (non-CICS) and you do not specify RLS, you get whatever type of open you requested: NSR, LSR, or GSR (provided share option 2 is in effect).
5. If you open a nonrecoverable data set for output from a batch job (non-CICS) and you specify RLS as either NRI or CR, you get an RLS open.
6. If you open a nonrecoverable data set for output from a batch job (non-CICS) and you specify RLS as CRE, you get a DFSMStvs open.
7. If you open a nonrecoverable data set for input from a batch job (non-CICS) and you specify RLS as either NRI or CR, you get an RLS open.
8. If you open a nonrecoverable data set for input from a batch job (non-CICS) and you specify RLS as CRE, you get a DFSMStvs open.
9. If you open a nonrecoverable data set for output from a batch job (non-CICS) and you do not specify RLS, you get whatever type of open you requested: NSR, LSR, or GSR (provided share option 2 is in effect).
10. If you open a nonrecoverable data set for input from a batch job (non-CICS) and you do not specify RLS, you get whatever type of open you requested: NSR, LSR, or GSR (provided share option 2 is in effect).

Structuring your application for commit and backout

Before invoking commit or backout for a unit of recovery, your application program should complete all I/O for the unit of recovery. To commit data, all changes must be saved. If your application program neglects to save the changes, DFSMStvs ensures that they are saved for you.

Recommendation: For best performance, however, an application should not rely on DFSMStvs to save the changes. Also, it is possible that an attempt by DFSMStvs to save the data could fail. If a save attempt fails, DFSMStvs cannot guarantee that all of the data was put on DASD and will back out the unit of recovery, regardless of whether the user issued a commit or a backout.

Other considerations follow:

- If your application program issues ENDREQs or CHECKs, ensure that the ENDREQs or CHECKs complete before invoking commit or backout processing.
- Never invoke a commit or backout from the RLSWAIT exit or from OPEN or EOVS exits.
- Be aware that a commit or a backout normally releases all of the locks held by a unit of recovery.
- An application can successfully close a data set while it has an in-flight unit of recovery that modified the data set, but this is not recommended. If this is the last close of the data set on this system, RLS converts all of the locks held by the unit of recovery for the data set from active locks to retained locks. This causes any other units of recovery that attempt to get those locks to receive a retained lock

reject instead of waiting for the lock to become available. In addition, the data written to the data set during the unit of recovery is not committed or backed out until the unit of recovery issues a commit or backout. An implicit commit occurs when a job ends normally for in-flight units of recovery; an implicit backout occurs when a job abnormally ends for in-flight units of recovery.

Recommendation: Do an explicit commit before closing a data set.

If you close a data set that has active locks for an in-flight unit of recovery before it issues a commit or backout and your close is the last close on the system for the data set, RLS changes all of your active locks into retained locks. After that happens, another user who opens the data set and attempts to process any of the records that you have locked receives a retained lock rejection instead of waiting for the lock to become available.

- DFSMStvs never invokes commit or backout processing directly for in-flight or in-doubt units of recovery during normal processing. (However, it might do so during restart processing for failed units of recovery.)
- Applications can choose to attempt to commit data after a failure, such as an error that was returned by VSAM. However, if DFSMStvs determines that the error might have affected data integrity, it responds negatively during commit phase 1 (prepare). DFSMStvs then requires the unit of recovery to be backed out.

For example, an application might attempt to commit data after an I/O error on either the data set or the system undo log. Before the commit, DFSMStvs determines that the data is not in a consistent state and requires a backout instead.

- Applications can choose to attempt to commit the data after a failure against a forward recovery log. This failure does not affect the success of the commit process.

Recommendation: After the data is committed, back up the data set before continuing. Without a backup, you might lose the ability to recover the data set because its forward recovery log is damaged.

- Failures that occur during commit phase 1 (prepare) result in the unit of recovery being backed out. Failures that occur during commit phase 2 do not result in the unit of recovery being backed out. The commit is retried at a later time.
- A failure that prevents completion of a sync point causes DFSMStvs to shunt those portions of the unit of recovery that DFSMStvs cannot finish processing. I/O errors or unavailability of a resource are examples of failures that prevent completion of a sync point.

All shunted transactions are automatically retried by DFSMStvs every 15 minutes until the retry is successful. For each transaction that is successfully retried, DFSMStvs issues message IGW892I. You can also manually retry a shunted transaction, using the access method services SHCDS command.

Because DFSMStvs automatically retries transactions that are in the shunt log, you need to make sure that the log does not contain anything that cannot be retried. For example, if you delete a data set for which recovery is owed, VSAM RLS releases all of the locks associated with the data set. Suppose you then allocate a data set with the same name as the deleted data set. If the shunt log still contains outstanding work for the deleted data set, DFSMS might automatically retry this work on the new data set of the same name, without any locks.

Recommendation: Before you delete a data set, use the SHCDS PURGE command to rid the shunt log of any outstanding work for the data set.

Various levels of authorization are required to use the SHCDS parameters.

Understanding the effects of a task ending

When a task ends, its context ends with it. If a unit of recovery is active, the unit of recovery ends as well. Normally, your applications should invoke the commit or backout process prior to the end of a task. If they do not, RRS issues an implicit sync point when the task ends. If the task is ending normally, RRS attempts to commit the data implicitly. If the task is abnormally ending, RRS attempts to back out the data implicitly.

Do not rely on these implicit sync points. Performing syncpoint processing in this manner can cause units of recovery to be much longer than necessary and adversely affect system performance.

Understanding record locking that DFSMStvs uses

VSAM maintains a single central lock structure using the MVS coupling facility. This central lock structure provides sysplex-wide locking at a record-level. The lock is used to control updates, for CR and CRE reads, and to serialize adds to an ESDS data set when used for DFSMStvs processing.

For CICS, the lock owner name is the application identifier (APPLID) of the CICS region, plus the unit of work ID. For DFSMStvs, the lock owner name is based on the DFSMStvs instance name combined with the 16-byte URID supplied by RRS.

There is no need for an application that is using DFSMStvs services to provide the URID when invoking VSAM. When VSAM invokes DFSMStvs services, DFSMStvs determines the URID by using the RRS unit of recovery under which the VSAM request was issued. VSAM builds the lock name using the unit of work identifier (UOW ID) or URID, the record key, and the name of the CICS APPLID or DFSMStvs instance.

If a request is made to update a recoverable data set, the associated exclusive lock must remain held until the next sync point. This ensures that the resource remains protected until a decision is made to commit or back out the request. If DFSMStvs fails, VSAM continues to hold the lock until DFSMStvs restarts.

Using read integrity options in your application program

DFSMStvs supports three options to control read integrity. You can specify these options in the JCL, in the ACB, or at the RPL level. Which one you choose depends on the application programming language in which your application is coded. If you specify different read integrity options, the RPL specification overrides the ACB, which overrides the JCL. If you do not specify an option, no read integrity (NRI), the default option, is in effect. The read integrity options are as follows:

NRI (no read integrity)

There is no read integrity and shared locks are not used for read requests. This is the default.

CR (consistent read)

A request to read a record is queued if the record is being updated by another task. The read completes only when the update is complete, and the updating unit of recovery relinquishes exclusive control by issuing a sync point. Specifying CR ensures that your applications do not see uncommitted data. However, it is possible that another application could modify the record after it has been read, but before your application has finished its operations on the record.

CRE (consistent read explicit)

Processing of the read request is the same as for consistent read requests. However, in this case, the reader holds on to its shared lock until the next sync point. This ensures that a record read in a unit of recovery cannot be modified while the unit of recovery makes further read requests. It is particularly useful when issuing a series of related read requests to ensure that none of the records are modified before the last record is read. This option is also known as a *repeatable read*.

Understanding reasons for retained locks and locking duration

The following conditions delay the release of exclusive record locks for an indefinite period of time:

- A sysplex failure
- An MVS failure
- A failure of an instance of the SMSVSAM server or DFSMStvs
- The last close by all users of the data set on this system image

However, when the application that closed the data set reaches the sync point, this problem is resolved.

- A commit or backout processing failure

When these conditions occur, the affected exclusive record locks are converted to retained locks. Any subsequent attempt to obtain a retained record lock fails immediately rather than waiting for the lock to be released.

A VSAM request obtains the record lock and the lock is held until released by DFSMStvs during commit or backout processing.

If an application uses sequential, skip-sequential, or direct NSP (note string position) requests to modify data sets that were accessed by DFSMStvs, ensure that your application writes the modified buffers to the coupling facility before issuing a commit or backout request.

Avoiding false lock contention

Real contention occurs when two units of recovery are trying to lock the same record if the key length is 16 bytes or less. However, if the key length is more than 16 bytes, VSAM or VSAM RLS locks a hashed value of the key.

While the example shown in [Figure 14 on page 61](#) is not an accurate flow of hashing, it provides an example to help you understand how false contention can happen. The example shows how the hash table would look (17-byte keys) if the key is hashed by just taking the last 16 bytes.

| KEY | HASH |
|--------------------|------------------|
| 188888888888888888 | 8888888888888888 |
| 18888888888888885 | 8888888888888885 |
| 28888888888888885 | 8888888888888885 |

Figure 14. Example of key hashing

The first two keys have different hash values but the last two have the same value. If UR1 is accessing key 18888888888888885 and UR2 tries to access key 28888888888888885, UR2 gets an indication that it has contention with UR1. Since they are trying to lock different records, this is false contention. The program would treat it as a real contention, back out the unit of recovery and try again.

Avoiding deadlocks

When applications access data sets in VSAM RLS or DFSMStvs mode, deadlocks can occur within a single instance. They can also arise between two instances of DFSMStvs running under different MVS images. VSAM RLS performs deadlock detection and resolution across systems, within its own resources. If it detects a deadlock condition, VSAM RLS fails one of the requests to break the deadlock cycle. DFSMStvs writes messages that identify the members of the deadlock chain.

However, VSAM RLS cannot detect a cross-resource deadlock (for example, a deadlock arising from the use of VSAM RLS and DB2 resources) in which another resource manager is involved. VSAM RLS resolves a cross-resource deadlock when the timeout period expires and the waiting request times out. In this situation, VSAM RLS cannot determine whether the cause of the timeout is a cross-resource deadlock or another unit of recovery acquiring an RLS lock and not releasing it. In the event of a timeout, DFSMStvs writes messages to identify the holder of the lock for which a timed-out unit of recovery is waiting.

Because timeouts are the only way VSAM RLS has for resolving cross-resource deadlocks, you need to specify a timeout value for DFSMStvs requests. You can specify the timeout value in the RLSTMOUT parameter in the IGDSMSxx member of SYS1.PARMLIB, in the JCL, in the ACB, or in the RPL request. If you specify different timeout values, the RPL specification overrides the ACB, which overrides the JCL, which overrides the parmlib member. If you do not specify a timeout value (0), it defaults to no timeout, and requests could wait indefinitely with VSAM RLS unable to resolve the problem.

Handling long-running jobs and programs

For long-running programs, ensure that you do not have a large number of changes that meet these criteria:

- Have accumulated over a period of time
- Are exposed to the possibility of a backout in the event of a task or system failure

You can avoid problems by using the commit services provided by RRS to split the program into logically separate sections, thus creating units of recovery. The end of a unit of recovery is initiated by a sync point. If a task fails after a sync point but before the task has been completed, only changes made after the sync point are backed out.

A unit of recovery should be entirely logically independent, not merely with regard to protected resources but also with regard to execution flow.

If an abend occurs during either application or DFSMStvs processing, do not issue further requests for the same unit of recovery. The abend can cause some DFSMStvs control blocks to be invalidated. Similarly, do not try to recover from I/O errors. Because of the risk to data integrity, DFSMStvs does not allow a commit of a unit of recovery after it has received a hard I/O error for one of its VSAM record management requests. Also, it does not allow a commit of a unit of recovery for any type of error occurring during a close if a previous explicit commit was not already done.

Using restartable applications

A restartable application is one that can be rerun after a failure. Batch applications that update VSAM data sets in a nonshared environment can create backup copies of the data sets to establish a restart or recovery point for the data. If such a batch application fails, the data sets can be restored from the backup copies, and the batch jobs can be rerun to reapply all the changes.

You cannot use this restart or recovery procedure in a data-sharing DFSMStvs environment. Restoring to the point in time of backup would erase any changes made by other applications that are sharing the data set.

Instead, the batch application must have a built-in method of tracking its processing position within its string or series of transactions. One possible method is to use a VSAM recoverable data set or a job-processing position file to track the commit position for the job. This is a nonshared VSAM recoverable data set in which the application records its progress. Every time it reaches a sync point, the application makes a note of it in the job processing position file. Because it is a recoverable file, if the unit of recovery is committed, so is the job processing position file. Likewise, if the unit of recovery is backed out, the job processing position file is backed out and correctly indicates that the last successful unit of recovery was the prior one.

When the application fails, any uncommitted changes are backed out. The already committed changes *cannot* be backed out because they are already visible to other jobs and transactions. In fact, the records that were changed by previously committed units of recovery might have since been changed again by other jobs or transactions. Therefore, when the job is rerun, it is important that it determine its restart point. The job should not attempt to redo any changes it had committed before the failure.

For this reason, it is important that jobs and applications using DFSMStvs be written to execute as a string of transactions. They must use a commit point tracking mechanism for restart. When they restart, they should skip any units of recovery that were already completed and begin from the first unit of recovery that was either not attempted or backed out.

One method of tracking the position of a job within the input stream is with a VSAM recoverable data set. As the application executes its units of recovery, the application writes information indicating which units of recovery have been processed to the position file. At commit time, the position file and the application data sets are in sync. If the unit of recovery is backed out, the change to the position file is also backed out. It is not necessary for each input record to correspond to a commit point, but the position file must have a record of each commit or backout request.

Use a private (nonshared) recoverable data set to track the input stream of the job at commit time. Use these steps as a guide in setting up your application to determine the restart point:

1. When the job is rerun, position the input stream to the position shown in the private data set.
2. Read the next record from the input stream.
3. Update the private data set to reflect the new position.
4. Perform the indicated processing.

5. Issue the commit at the end of the transaction.

The position file and the application data sets are in sync.

Recommendation: Do not use an entry-sequenced data set as your position file because it is impossible to erase anything from an entry-sequenced data set.

Establishing positioning after logical errors

VSAM is unable to maintain positioning after every logical error. Whenever VSAM does not maintain positioning after an error request, you must reestablish positioning before processing resumes. After a direct or sequential access request that resulted in a logical error, positioning can be in one of four states:

Yes

VSAM is positioned at the position in effect before the request in error was issued.

No

VSAM is not positioned because no positioning was established at the time the request in error was issued.

New

VSAM is positioned at a new position.

U

VSAM is positioned at an unpredictable position.

Undefined

The reason code is undefined.

Related reading: For a list of the positioning states for reason codes for sequential, direct, and skip-sequential processing, see the [*z/OS DFSMStvs Administration Guide*](#).

Using sequential or random access to a data set

Skip-sequential processing is generally used to speed browsing when reading through a data set. VSAM locates the next record by reading through the lowest level index rather than repositioning from scratch. This method is faster if the records being retrieved 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 to which you are positioning is much higher and there might be a long index scan, use the POINT request to reset your positioning.

Deleting and renaming data sets

It is possible to delete or rename a data set that has retained locks and shunted log records. This enables you to delete and re-create a data set in the 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 by using the access method services SHCDS command prior to deleting the data set. If the locks are unbound prior to the delete request, any retained locks and log records associated with the data set are kept across the delete request. They are again associated with the data set when the locks are rebound. If the unbind is not done prior to deleting the data set, the locks are discarded when the data set is deleted.

It is possible for an application to close a data set that has uncommitted changes (that is, a unit of recovery is in-flight). It is also possible for a data set to be deleted or renamed before the unit of recovery completes. The data sets that have retained locks and shunted log records associated with them could be deleted or renamed.

Recommendation: Do not delete or rename a data set unless you are doing so in preparation for forward recovery (in which case, locks should first be unbound). Deleting or renaming a data set can result in loss of any association between the data set and its log records and locks.

An access method services DELETE command results in an exclusive enqueue on the data set, making it impossible for DFSMStvs to allocate the data set during backout. For example, consider the following scenario:

- Step 1
 1. Execute PGM=IDCAMS
 2. Specify a DD statement for data set MY.TVS.KSDS with DISP=SHR
 3. Delete MY.TVS.KSDS
- Step 2
 1. Execute PGM=IDCAMS
 2. Define MY.TVS.KSDS
- Step 3
 1. Open MY.TVS.KSDS for DFSMStvs access
 2. Update records in MY.TVS.KSDS
 3. Attempt to backout the changes

This results in a failure such as this message describes:

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

The problem occurs because the access method services DELETE command upgrades the enqueue for the data set MY.TVS.KSDS from shared to exclusive. The solution is to delete the data set at the end of the job or in a previous job. The data set cannot be deleted in a different step of the same job because, in this case, the job would still hold the enqueue. In the case where a unit of recovery is in-flight, deleting or renaming the data set causes failures if the unit of recovery needs to be backed out. It also results in any work that cannot be completed being moved to the shunt log. Then you need to manually clean up the shunted log records by using the access method services SHCDS PURGE command.

Various levels of authorization are required to use the SHCDS parameters. For information about this authorization, see [z/OS DFSMS Access Method Services Commands](#).

After 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 access method services SHCDS PURGE command, they could be applied to the new data set in error.

Although users might not be using a data set, the data set could be in use if DFSMStvs restart processing is in progress. If this is the case, it is not possible to delete or rename the data set until DFSMStvs is finished.

Monitoring and retrying shunted transactions

If the VSAM RLS server address space does not have a RACF attribute of PRIVILEGE or TRUSTED assigned, using RACF you must authorize SMSVSAM server to any data sets that can be accessed using Transactional VSAM. This is needed to perform a retry of shunted transactions.

All shunted transactions are automatically retried by DFSMStvs every 15 minutes until the retry is successful. For each transaction that is successfully retried, DFSMStvs issues message IGW892I. You can also manually retry a shunted transaction, using the access method services SHCDS command.

Because DFSMStvs automatically retries transactions that are in the shunt log, you need to make sure that the log does not contain anything that cannot be retried. For example, if you delete a data set for which recovery is owed, VSAM RLS releases all of the locks associated with the data set. Suppose you then allocate a data set with the same name as the deleted data set. If the shunt log still contains outstanding work for the deleted data set, DFSMS might automatically retry this work on the new data set of the same name, without any locks.

Recommendation: Before you delete a data set, use the SHCDS PURGE command to rid the shunt log of any outstanding work for the data set.

Various levels of authorization are required to use the SHCDS parameters. For information about this authorization, see [z/OS DFSMS Access Method Services Commands](#).

Alternatively, if you have units of recovery that were shunted due to backout failures, you can resolve those units of recovery by following these steps:

1. Use the access method services SHCDS LISTSHUNTED(*dsname*) command to list all the shunted entries for a particular data set.
2. Make certain that the problem that caused the failure has been corrected. This might involve ensuring that the volume is available or that the data set has been recovered.
3. Determine whether the shunted entries should be retried.
4. If the shunted entries should be retried, use the access method services RETRY command to request that DFSMStvs take action on them.
5. If the shunted entries should not be retried, use the access method services SHCDS PURGE command to request that DFSMStvs delete the shunted information.
6. It might also be necessary to use interfaces provided by RRS to inform RRS that the unit of recovery has been resolved.

If you used the access method services SHCDS RESETLOCKS command to reset the locks for a data set, the reset causes any attempt to take action on shunted log entries for the data set to fail.

The same is true if you delete the data set without unbinding the locks. If you used the access method services SHCDS FRUNBIND command, followed by the access method services SHCDS RFDELETEUNBOUNDLOCKS command to reset the locks for a data set, it causes any attempt to take action on shunted log entries for the data set to fail.

Both problems cause DFSMStvs to see a unit of recovery that has log records but no locks, which can be referred to as *suspended state*. A unit of recovery that is in a suspended state cannot be retried because it is not safe to do so without the locks. Use the access method services SHCDS command to purge the unit of recovery.

Applying advanced application development techniques

If you use DFSMStvs for more than just running batch jobs that are in a CICS batch window, this topic can help you apply advanced techniques for coding your application.

You need to convert batch jobs that use DFSMStvs to update recoverable data sets. This involves the following tasks:

- Modifying jobs to use sync points and take appropriate action (commit or backout)
- Modifying jobs to use RRS services to invoke commit and backout processing
- Modifying jobs to specify that DFSMStvs is to be used
- Examining jobs to ensure that multiple RPLs in use within a single application does not cause lock contention within a unit of recovery
- Coding jobs to handle loss of positioning. Loss of positioning might occur at commit or backout for unpaired requests (a GET UPD request is not followed by a PUT UPD request, or an IDALKADD request not followed by a PUT NUP request)
- Examining jobs to ensure that any new reason codes are handled appropriately.
- If any applications act as work dispatchers, examining jobs to ensure that all work intended to be part of a single unit of recovery is handled under the same context

Another reason to use more advanced techniques would be if you would like to split a batch process into several jobs. By splitting a batch process, originally written to be a single threaded job, you can use DFSMStvs and run the jobs in parallel.

Another reason to use more advanced techniques would be if you have a batch process that was originally written to be a single threaded job, but using DFSMStvs, you would like to split it into several jobs that run in parallel.

For example: You have a recoverable data set that contains 100 000 records, and a batch job that processes all the records and updates them. Assuming that the updates are independent of each other,

you might split the batch job into four jobs. The first job would process records 1-25 000. The second job would process records 25 001 - 50 000. The third job would process records 50 001 - 75 000, and the fourth job would process records 75 001 - 100 000. Alternatively, your batch job could run as a mother task which attached four subtasks. The mother task would give work to each of the subtasks, and as each subtask completed its job, the mother task would give it more work. Using this approach, each of the subtasks would be dealing with an independent context and unit of recovery.

Record management requests

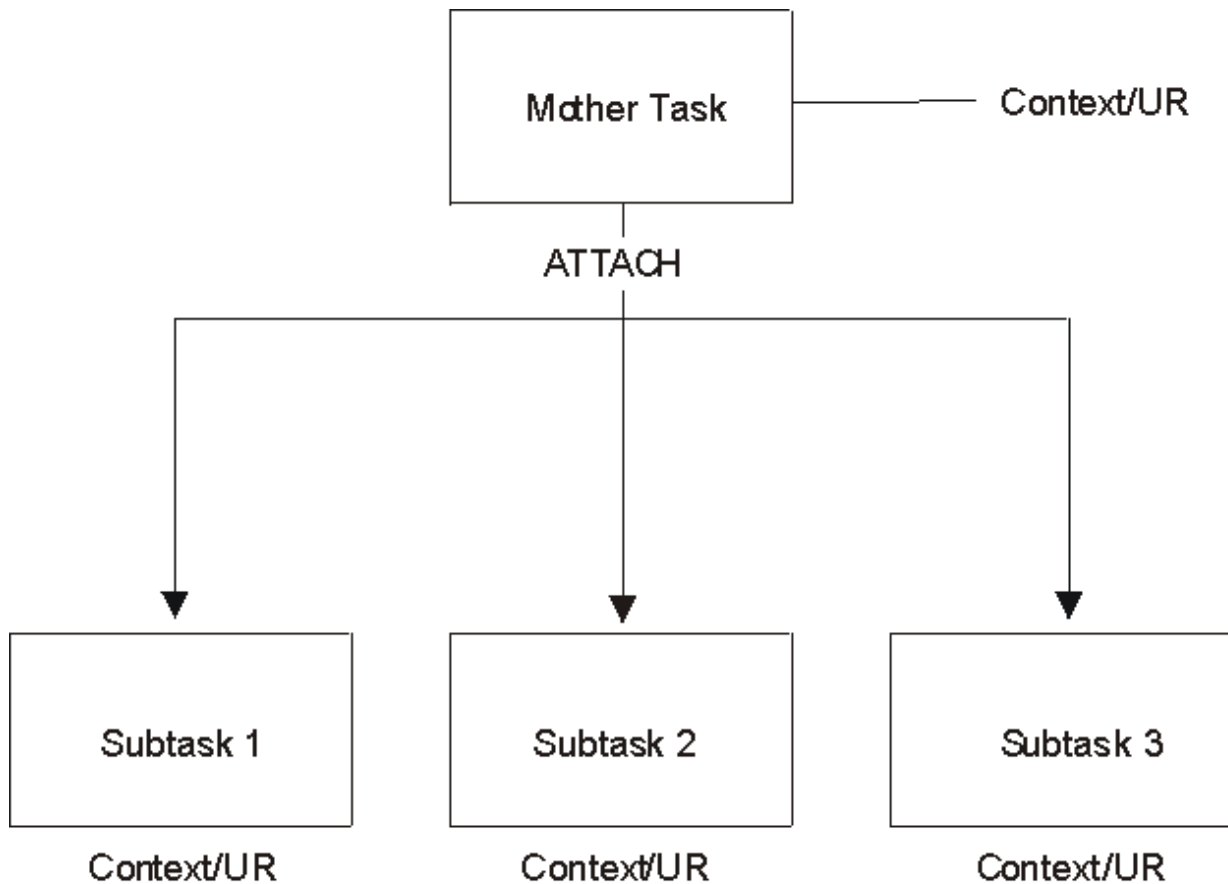
Unless privately managed contexts are used, all record management, commit, and backout requests pertaining to a given unit of recovery must be issued under the same task control block (TCB). This is because a unit of recovery is associated with a context and a context is associated with a TCB. Any requests issued under a different TCB would be handled under the context associated with that TCB and would, therefore, belong to a different unit of recovery. The only exception to this is the case in which a resource manager is using privately managed contexts and switching them between TCBs.

A work manager, which takes work requests and parcels them out to tasks, might create privately managed contexts. These contexts and the units of recovery associated with them can be moved around, unlike the default contexts that every TCB in the system has. So, if the work manager knows that a piece of work it is about to submit belongs to a particular unit of recovery, the work manager can take that unit of recovery's privately managed context and attach it to the TCB before giving that TCB the work to do.

Record management requests get the locks; commit and backout processing release the locks. For example, GET UPDATE, PUT ADD requests get locks. ENDREQ, PUT UPDATE and DELETE requests do not release locks. Commit and backout processing release the locks unless there is a problem, in which case the locks are retained.

Multitasking

DFSMStvs can be used within an application that uses multitasking. That is, the application can consist of a mother task with multiple daughter tasks to which work is allotted. For example, [Figure 15 on page 67](#) shows a mother task and three subtasks.



DA0TV800

Figure 15. Multitasking

One use for multitasking might be for an application that functions as a transaction dispatcher. The following rules apply; if they are not followed, unpredictable results can occur. The application must follow these rules because DFSMStvs has no way to detect violations or to enforce these rules.

1. When DFSMStvs is invoked, it uses the current context. Normally, this is the RRS native context that is associated with a dispatchable unit of work. A context is a representation of a work request, and a unit of recovery is the work done by or on behalf of the context between one point of consistency and another. Therefore, all work within a single unit of recovery (transaction) must come under a single context. When using native contexts, a single unit of recovery cannot cross task boundaries. In this example, subtask 1 and subtask 2 cannot do work on behalf of the same unit of recovery.
2. If an application wishes to share work for a single unit of recovery across task boundaries, it must create one or more privately managed contexts. The application is responsible for managing those contexts and ensuring that the correct context is associated with a dispatchable unit when DFSMStvs is invoked. This ensures that the work being done is associated with the correct unit of recovery. Because privately managed contexts take precedence over native contexts, associating a privately managed context with a dispatchable unit causes it to become the current context.
3. Applications must ensure that the same context is used for operations that are part of the same unit of recovery. If an application issued a GET UPD request then switched contexts before issuing the corresponding PUT UPD request unpredictable results could occur.
4. DFSMStvs allows context switching provided that your application completes all asynchronous I/O. You can use the CHECK macro to wait for I/O associated with an RPL to complete.

Using the DFSMStvs automatic commit function

It is sometimes possible to modify a batch job step to use DFSMStvs without making source code changes. However, that approach means that the entire job step is treated as a single VSAM transaction, which can cause locking, logging, concurrency, and backout issues. To avoid those issues, sync points need to be added to the source code usually. Sometimes, such as jobs that perform only a few updates, adding and testing sync points might require more effort than is necessary. For such programs, the DFSMStvs automatic commit function may be the answer.

The automatic commit function can provide automatic commits of transactions for eligible batch jobs that use DFSMStvs. Eligible jobs are those for which a single update to a single data set constitutes an atomic transaction or logical unit of work. For example, if a batch job modifies two data sets, depending on the parameters that are specified, DFSMStvs might issue a sync point between the two updates, and one data set might become out of sync with the other data set. If such updates need to be atomic, automatic commit is not suitable for the application.

To enable DFSMStvs automatic commit, code the TVSAMCOM parameter on the JCL EXEC statement. The TVSAMCOM parameter specifies two values; a minimum and maximum number for update requests. DFSMStvs uses those values to determine when and if to call RRS services to issue a commit point on behalf of the batch application. DFSMStvs adjusts the commit frequency to a number between the two specified values, based on record lock contention for the current unit of recovery and CICS transactions' timeout value. Also, you can specify a similar system-level commit parameter in the **IGDSMSxx** member of **SYS1.PARMLIB**. Values that are specified in the JCL override any values specified in **IGDSMSxx**.

JCL changes

To enable the automatic commit function by using JCL, specify the TVSAMCOM parameter on the following JCL EXEC statement:

```
//stepname EXEC positional-parm, TVSAMCOM=({minval},{maxval})
```

minval

Specifies a number (0-99999) that represents the minimum number of update requests that need to be performed before DFSMStvs issues an automatic commit on behalf of the batch application.

maxval

Specifies a number (0-99999) that represents the maximum number of update requests that are performed before DFSMStvs issues an automatic commit on behalf of the batch application. This value is used only if DFSMStvs did not dynamically adjust the commit frequency to a number less than the maximum value. In this case, DFSMStvs issues a commit after the number of update requests reaches the maximum.

If you specify the same value (a number greater than zero) for *minval* and *maxval*, DFSMStvs performs automatic commits after the specified number of update requests are performed. In this case, DFSMStvs does not try to adjust the commit frequency based on the number of waiters and timeout values for record locks obtained by the unit of recovery.

If the value specified for *minval* is lesser than *maxval*, with *maxval* greater than zero, DFSMStvs adjusts the commit frequency to a number between *minval* and *maxval*. If no critical record lock contention was found and a commit point was not issued, one is performed when the number of updates reaches the maximum *maxval*.

If the maximum number specified in *maxval* is zero (*minval*>0, *maxval*=0), the default 0 is used. DFSMStvs adjusts the commit frequency if needed and issues commit points after the minimum number of updates is reached. However, a default commit point is not issued if DFSMStvs doesn't encounter eligible lock contention for the unit of recovery.

If a zero is specified for both the minimum and the maximum values, the automatic commit feature is turned off. DFSMStvs does not issue commits on behalf of the application, even if the parameter is specified in member **IGDSMSxx**.

DFSMStvs performs an automatic commit when any of the data sets accessed by the unit of recovery is closed. This is done to avoid leaving record locks held by the unit of recovery until the implicit commit is issued at end of task.

If any of the units of recovery that are eligible for automatic commit causes the lock structure to start becoming full as a result of obtaining record locks, DFSMStvs performs an automatic commit when the lock structure is approximately 80% full.

If the TVSAMCOM is not specified in the JCL EXEC statement or if *minval* and *maxval* are omitted, it defaults to the value in the IGDSMSxx member of SYS1.PARMLIB.

IGDSMSxx changes

In addition to specifying TVSAMCOM on the JCL step level, you can code it as a parameter in an **IGDSMxx** member of **SYS1.PARMLIB** as follows:

```
TVSAMCOM({minval|1}, {maxval|0})
```

If *minval* or *maxval* are not specified, the default value is 1 for *minval* and 0 for *maxval*. If both values are 0 or the keyword is not specified, the automatic commit feature is disabled for the jobs that do not specify TVSAMCOM in the JCL EXEC.

If the parameter is specified in both JCL EXEC statement and member **IGDSMSxx**, DFSMStvs uses the following priority:

1. The values specified in JCL at the step level, if any
2. The values specified in the **IGDSMSxx** member of **SYS1.PARMLIB**, if any.

Chapter 6. Monitoring performance and tuning the DFSMStvs environment

This topic contains information that is NOT Programming Interface information.

The additional work that DFSMStvs performs to obtain locks, to perform two-phase commits, and to log updates consumes additional resources and affects performance. This topic describes how to monitor performance and how to tune the DFSMStvs environment for the best performance.

Use of DFSMStvs can have these effects on performance:

- Increased application run time
- Increased processor usage
- Increased response time for CICS transactions
- Increased coupling facility use
- Increased use of processor storage
- Logs spilling from the coupling facility to disk

This information covers these topics:

- [“Monitoring performance” on page 71](#)
- [“Improving sequential performance” on page 92](#)
- [“Improving logging performance” on page 92](#)
- [“Tuning the DFSMStvs environment” on page 92](#)

Monitoring performance

You can use this information and tools to monitor DFSMStvs performance:

- SMF record type 42 (hexadecimal 2A)
- SMF record type 88 (hexadecimal 58)
- RMF post-processor reports
- RMF monitor III
- CICS monitoring tools
- System messages
- Operator commands

SMF record type 42 (hexadecimal 2A)

In SMF record type 42 (hexadecimal 2A), subtypes 15 to 18 include statistics generated by VSAM RLS:

- Subtype 15: VSAM RLS storage class response time summary

Look for a low read-hit rate in the coupling facility or a high false-invalidation rate, either of which points to a lack of space in the coupling-facility structure.

- Subtype 16: VSAM RLS data set response time summary

Look for a low read-hit rate in the coupling facility or a high false-invalidation rate, either of which points to a lack of space in the coupling-facility structure.

- Subtype 17: VSAM RLS coupling facility lock structure usage

The key measurement in this subtype is the number of false contention events. False contention happens when the hash code calculated for an entry collides with that for another entry. This can

happen purely by accident because of the way the hashing algorithm works; however, false contention events also occur if the lock structure is too small.

- Subtype 18: VSAM RLS coupling facility cache partition usage

Monitor the read-hit counter as a proportion of the number of reads, the number of read misses with directory hits, and the number of cross-invalidations due to directory-entry reclamation or due to local cache-vector replacement. These numbers can be symptoms of insufficient space in the coupling facility.

Related reading: For more information about SMF records, see [z/OS MVS System Management Facilities \(SMF\)](#).

SMF record type 88 (hexadecimal 58)

This record contains information about the system logger. z/OS provides a sample reporting program, IXGRPT1, in SYS1.SAMPLIB. Monitor the following key values, which you can obtain from type 88 records:

- Average buffer size

This size is not reported directly. Divide the number of bytes written by the number of writes completed to calculate this value.

- Structure-full events

If you see these events, the amount of space in the coupling facility is insufficient. Review your sizing assumptions and provide more space.

Related reading: For more information about SMF records, see [z/OS MVS System Management Facilities \(SMF\)](#).

RMF post-processor reports

The coupling facility activity report (use SYSRPTS(CF) in your RMF options to get this) is in three parts:

- Usage summary

This summary has the following information:

- What structures are in use
- Whether they are list, lock, or cache structures
- The space used
- The number of requests

- XES structure activity

This report shows you, by structure, the number of requests (both synchronous and asynchronous), the service time, and the number of requests that were delayed. The following key indicators in the structure-activity report that could alert you to configuration problems:

- More directory entries than data entries
- High number of directory reclaims
- High number of cross-invalidations
- Synchronous requests changed to asynchronous
- Low number of successful reads compared to writes

- Subchannel activity

Key indicators in this report follow:

- High degree of subchannels busy
- Number of subchannels used less than the number defined in the IOCDs

RMF monitor III

The coupling-facility activity summary, CFACT, shows you information by structure name and system. Interesting items are request rates, service times, and lock contention.

The data set by job summary, DSNJ, might help you to identify how individual data sets are being used.

The RMF component of z/OS contains additional Monitor III displays for RLS.

CICS monitoring tools

If you are monitoring the effect that RLS and DFSMStvs have on online CICS transactions, you should look at CICS statistics as well as system logger and SMSVSAM information to gain the complete picture.

The CICS statistics titled *File: requests information* give you information about the types of accesses to files open in CICS. You can produce reports with this information by using the DFHSTUP program supplied with CICS.

You can also use CICS Monitoring Facility data to break down transaction response time into its components, including I/O time.

System messages

Message IGW859I is issued to the system console when an individual unit of recovery is holding more unique locks than the limit specified in the MAXLOCKS parameter in SYS1.PARMLIB(IGDSMSxx). This message can help you identify a job that is not issuing commits frequently enough or that has a logic problem, which is probably causing performance problems for other users of DFSMStvs.

Operator commands

The options for the DISPLAY SMS command show you the status of these items:

- DFSMStvs facilities
- A specific job that is using DFSMStvs
- One or more units of recovery
- Entries in the shunt logs
- Log streams in use by DFSMStvs
- A data set being accessed in DFSMStvs mode

Related reading: For information about the options for the DISPLAY SMS command, see [z/OS MVS System Commands](#).

Shunted units of recovery

The following list gives reasons why a unit of recovery might be shunted.

- A VSAM RLS cache structure, or connection to it, has failed.
- A log stream became or was made unavailable.
- A sync point for a unit of recovery failed for one of the data sets that it was using. In this case, the name of the data set and the reason for the failure are included. Failure can occur for any of these reasons:
 - The commit failed.
 - An error occurred at some point when RLS locks were in the process of being released. This error can normally be resolved by recycling the SMSVSAM server (which should happen automatically).
 - The locks were acquired as a result of recoverable requests having been issued against the data set.
 - A data set is full; no space is available on the direct access device for adding records to it. You need to reallocate the data set with more space. You can then retry the backout using SHCDS RETRY. You can also find information in

Various levels of authorization are required to use the SHCDS parameters.

- Backout failed. This occurs as a result of a severe error being identified during backout, and is possibly an error in either DFSMStvs or VSAM. The problem might go away if the backout is retried.
- Index record is full. A larger alternate index record size needs to be defined for the data set.
- A hard I/O error occurred during backout. To correct this error, restore a full backup copy of the data set and perform forward recovery.
- An attempt to acquire a lock during backout of an update to this data set failed because the RLS lock structure was full. You must allocate a larger lock structure in an available coupling facility and rebuild the existing lock structure into it, then use the SHCDS RETRY command to drive the backout retry.
- An error occurred during an open of the data set for backout. A console message notifies you of the reason for the open error. One likely reason could be that the data set was quiesced.

Related reading:

- For information on moving and reallocating data sets, see [z/OS DFSMSdfp Storage Administration](#).
- CICS Transaction Server for z/OS (www.ibm.com/docs/en/cics-ts)
- For information about the level of authorization required to use the SHCDS parameters, see [z/OS DFSMS Access Method Services Commands](#).

Effects of DFSMStvs, log stream, and data set states

These tables describe the effects of the DFSMStvs states, log stream states, and data set states on DFSMStvs processing.

You can use the VARY SMS command to change the state of DFSMStvs, log streams that DFSMStvs uses, and data sets that DFSMStvs processes.

Recommendation: Be careful about quiesce with a path name; always use a base name.

Related reading: For information about the VARY SMS command, see [z/OS MVS JCL Reference](#).

Table 4 on page 74 describes the effects of the states of a DFSMStvs instance.

| Table 4. Effects of DFSMStvs states on DFSMStvs processing | |
|--|--|
| DFSMStvs state | Effect |
| Initializing | DFSMStvs is in the process of coming up and cannot accept any work until it has completed this process. |
| Active | DFSMStvs is fully initialized and is accepting work. It is possible for DFSMStvs to be in the active state while it is still completing restart processing. |
| Quiescing | DFSMStvs is in the process of completing any units of recovery that were in progress when the quiesce occurred but will not accept any new ones. DFSMStvs transitions from quiescing to quiesced when all in-progress units of recovery are completed and all data sets that are open in DFSMStvs mode are closed. |
| Quiesced | DFSMStvs has completed all units of recovery that were in progress when the quiesce occurred, and there are no active DFSMStvs opens. DFSMStvs does not accept any work until it is enabled. |
| Disabling | DFSMStvs is not accepting any requests. DFSMStvs transitions from disabling to disabled when all data sets that are open in DFSMStvs mode are closed. |

Table 4. Effects of DFSMStvs states on DFSMStvs processing (continued)

| DFSMStvs state | Effect |
|-----------------|--|
| Disabled | DFSMStvs is not accepting any requests, and all data sets that were open in DFSMStvs mode have been closed. DFSMStvs does not accept any new work until it is enabled. |

Table 5 on page 75 describes the effects of undo log states, shunt log states, log-of-log states, and forward recovery log states.

Table 5. Effects of log states on DFSMStvs processing

| Log type | Log state | Effect |
|-------------------------|-----------|---|
| Undo or shunt | Enabled | DFSMStvs services are available for use both by in-progress units of recovery and new units of recovery. |
| Undo or shunt | Quiescing | DFSMStvs allows requests by existing units of recovery to be processed but does not allow new units of recovery to start. Quiescing the undo or shunt log is equivalent to quiescing DFSMStvs. |
| Undo or shunt | Quiesced | There are no units of recovery using DFSMStvs services and new units of recovery cannot start |
| Undo or shunt | Disabling | All DFSMStvs requests are failed, including those submitted by in-progress units of recovery. Units of recovery are neither committed nor backed out and cannot be completed until DFSMStvs restart processing is run. Disabling the undo or shunt log is equivalent to disabling DFSMStvs. |
| Undo or shunt | Disabled | There are no units of recovery using DFSMStvs services and new units of recovery cannot start. All data sets that were open for DFSMStvs processing have been closed, and DFSMStvs has unregistered with RRS and the lock manager and disconnected from its logs. |
| Log of logs | Enabled | DFSMStvs writes tie-up records and file-close records to the log of logs so that they can be used to optimize forward recovery processing. |
| Log of logs | Quiescing | DFSMStvs writes file-close records to the log of logs for any forward recoverable data sets that are currently open but does not write any additional tie up records to the log of logs. |
| Log of logs | Quiesced | All paired tie-up records and file-close records were written to the log of logs, but no additional records are being written to the log of logs. |
| Log of logs | Disabling | DFSMStvs runs without the log of logs. If forward recovery is needed for any forward recoverable data sets, the forward recovery utility will not be able to use the optimization provided by the log of logs. |
| Log of logs | Disabled | DFSMStvs runs without the log of logs. If forward recovery is needed for any forward recoverable data sets, the forward recovery utility will not be able to use the optimization provided by the log of logs. All data sets that had written tie up records to the log of logs have been closed. |
| Forward recovery | Enabled | DFSMStvs allows data sets that use the forward recovery log to be opened and processed. |

| <i>Table 5. Effects of log states on DFSMStvs processing (continued)</i> | | |
|--|-----------|--|
| Log type | Log state | Effect |
| Forward recovery | Quiescing | DFSMStvs allows all jobs that are using the forward recovery log to continue processing but does not allow any new opens of data sets that use the forward recovery log. |
| Forward recovery | Quiesced | There are no jobs that have data sets that use the forward recovery log open, and DFSMStvs does not allow any new opens of data sets that use the forward recovery log. |
| Forward recovery | Disabling | DFSMStvs fails any requests that need to write to the forward recovery log, which causes logging failures in jobs that had data sets open that use it. Because no further records can be written to the forward recovery log, units of recovery that were in progress and using it can be neither committed nor backed out. They are shunted and must be either purged or manually retried when the forward recovery log is later enabled. |
| Forward recovery | Disabled | All data sets that had been using the forward recovery log have been closed. Any open that requires the use of the forward recovery log will be failed. |

Table 6 on page 76 describes the effects of data set states.

| <i>Table 6. Effects of data set states on DFSMStvs processing</i> | |
|---|--|
| Data set state | Effect |
| Not quiesced | The data set is available for normal use. |
| Quiescing | VSAM RLS allows those jobs that were using the data set to continue using it but does not allow additional opens. |
| Quiesced | The data set is not open for either RLS or DFSMStvs access and cannot be opened for RLS or DFSMStvs access until it is unquiesced. |
| Quiescing for copy | DFSMStvs allows any units of recovery that had previously updated the data set (or issued a GET UPD) to continue using the data set. It will also allow both new and in-progress units of recovery to read the data set, but it will not allow any unit of recovery that had not previously updated the data set (or issued a GET UPD) to issue GET UPD, PUT, or ERASE against the data set. |
| Quiesced for copy | There are no units of recovery that have updated the data set (or issued a GET UPD), and DFSMStvs will not allow any unit of recovery to issue GET UPD, PUT, or ERASE against the data set. Units of recovery can read the data set. |
| Quiesced for backup-while-open (BWO) | DFSMStvs ensures the writing of any log records required to enable forward recovery of the sphere from the backup copy that is about to be taken. DFSMStvs also writes a record to indicate that it received a QUIBWO. The sphere remains open and normal processing of the data set is allowed. |

Effects of DFSMStvs states based on events

This topic describes the effects of DFSMStvs states when different events occur.

1. The DFSMStvs state is initializing.

In this state, all opens and record management requests are failed with nonzero return and reason codes. All closes are allowed; the opens with which the closes are associated probably belonged to a previous instance of DFSMStvs.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. DFSMStvs is already initializing. |
| VARY command to quiesce DFSMStvs | The command waits until DFSMStvs finishes initializing to quiesce DFSMStvs processing and change its state to quiescing. |
| VARY command to disable DFSMStvs | The command waits until DFSMStvs finishes initializing to disable DFSMStvs processing and change its state to disabling. |
| RRS becomes unavailable | DFSMStvs makes a note that RRS is unavailable. DFSMStvs begins disabling itself and sets its state to disabling when initialization reaches a point where it can do so. |
| A system log I/O error occurs | DFSMStvs makes a note that the I/O error occurred. DFSMStvs begins quiescing itself and sets its state to quiescing when initialization reaches a point where it can do so. |

2. The DFSMStvs state is initializing and an operator command was issued to quiesce DFSMStvs processing.

In this state, DFSMStvs initialization completes and then the DFSMStvs state changes from initializing to quiescing. It performs DFSMStvs restart processing as part of the process of quiescing, but no work is allowed to start. All opens and record management requests are failed with nonzero return and reason codes. All closes are allowed; the opens with which the closes are associated probably belonged to a previous instance of DFSMStvs.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | Command waits until DFSMStvs finishes initializing and then stacks behind the quiesce command. |
| VARY command to quiesce DFSMStvs | Command waits until DFSMStvs finishes initializing and is then rejected because such a command was already issued. |
| VARY command to disable DFSMStvs | Command waits until DFSMStvs finishes initializing and then stacks behind the quiesce command. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. DFSMStvs begins disabling itself and sets its state to disabling when initialization reaches a point where it can do so. |
| A system log I/O error occurs | DFSMStvs makes a note that this occurred. DFSMStvs will begin quiescing itself and will set its state to quiescing when initialization reaches a point where it can do so. |

3. When the DFSMStvs state is initializing, a system log I/O error occurred, and an operator command has been issued to quiesce the previous instance of DFSMStvs.

In this state, DFSMStvs would not be initializing unless a cold start has been requested to clear the system log I/O error. All opens and record management requests are failed with nonzero return and reason codes. All closes are allowed; the opens with which the closes are associated probably belonged to a previous instance of DFSMStvs.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | Command waits until DFSMStvs finishes initializing and then is rejected because DFSMStvs is already initialized. |
| VARY command to quiesce DFSMStvs | Command waits until DFSMStvs finishes initializing and then DFSMStvs begins quiescing its processing and sets its state to quiescing. |
| VARY command to disable DFSMStvs | Command waits until DFSMStvs finishes initializing and then DFSMStvs begins disabling its processing and sets its state to disabling. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. DFSMStvs begins disabling itself and sets its state to disabling when initialization reaches a point where it can do so. |
| A system log I/O error occurs | This cannot occur. In this state, a cold start must be done, and DFSMStvs would not be writing to the system log during a cold start. |

4. The DFSMStvs state is initializing, a system log I/O error occurred, and an operator command has been issued to quiesce the previous instance of DFSMStvs processing.

In this state, DFSMStvs would not be initializing unless a cold start has been requested to clear the system log I/O error. All opens and record management requests are failed with nonzero return and reason codes. All closes are allowed; the opens with which the closes are associated probably belonged to a previous instance of DFSMStvs.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | Command waits until DFSMStvs finishes initializing and then stacks behind the previous quiesce command. |
| VARY command to quiesce DFSMStvs | Command waits until DFSMStvs finishes initializing and then is rejected because such a command was already issued. |
| VARY command to disable DFSMStvs | Command waits until DFSMStvs finishes initializing and then stacks behind the previous quiesce command. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. DFSMStvs begins the process of disabling itself and sets its state to disabling when initialization reaches a point where it can do so. |
| A system log I/O error occurs | This cannot occur. In this state, a cold start must be done, and DFSMStvs would not be writing to the system log during a cold start. |

5. DFSMStvs has initialized and its state is active.

In this state, opens, closes, and record management requests are all processed normally.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | The command is rejected. DFSMStvs is already available. |
| VARY command to quiesce DFSMStvs | DFSMStvs begins the process of quiescing itself and sets its state to quiescing. |

| Event | Effect |
|----------------------------------|--|
| VARY command to disable DFSMStvs | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| RRS becomes unavailable | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| A system log I/O error occurs | DFSMStvs begins the process of quiescing itself and sets its state to quiescing. |

6. DFSMStvs has initialized and its state is quiescing because a VARY command was issued to quiesce DFSMStvs processing.

In this state, opens and closes are all processed normally. Any record management requests for units of recovery that are already in progress are also processed normally. Record management requests for new units of recovery are rejected.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs sets its state to initializing and reinitializes itself. |
| VARY command to quiesce DFSMStvs | The command is rejected. DFSMStvs is already in the process of quiescing itself. |
| VARY command to disable DFSMStvs | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| RRS becomes unavailable | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| A system log I/O error occurs | The DFSMStvs state does not change, because DFSMStvs is already quiescing. However, DFSMStvs makes a note of the error, because a cold start will be required to correct the problem. |

7. DFSMStvs has initialized and its state is quiescing because a system log I/O error occurred.

In this state, opens and closes are all processed normally. Any record management requests for units of recovery that are already in progress are also processed normally. Record management requests for new units of recovery are rejected.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. A cold start is required to recovery from the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs is already in the process of quiescing itself, but make a note that this occurred so that it will not come back up until another VARY command is issued to tell it to ENABLE itself. |
| VARY command to disable DFSMStvs | DFSMStvs begins the process of disabling itself and sets its state to disabling. |

| Event | Effect |
|-------------------------------|--|
| RRS becomes unavailable | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

8. DFSMStvs has initialized and its state is quiescing because a system log I/O error occurred and because a VARY command was issued to quiesce DFSMStvs processing.

In this state, opens and closes are all processed normally. Any record management requests for units of recovery that are already in progress are also processed normally. Record management requests for new units of recovery are rejected.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that it was told to enable, but it issues a message to indicate that it is still quiescing due to the system log I/O error. |
| VARY command to quiesce DFSMStvs | The command is rejected. DFSMStvs is already quiescing. |
| VARY command to disable DFSMStvs | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| RRS becomes unavailable | DFSMStvs begins the process of disabling itself and sets its state to disabling. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

9. The DFSMStvs state is quiesced because a VARY command was issued to quiesce DFSMStvs processing.

DFSMStvs does not reach this state until the last data set that was open for DFSMStvs processing is closed. In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no closes or record management requests can occur after the last data set open for DFSMStvs processing is closed.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | The command is rejected. DFSMStvs is already quiesced. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already quiesced. |
| RRS becomes unavailable | DFSMStvs unregisters with RRS and RLS, disconnects from its logs, and sets its state to disabled. |
| A system log I/O error occurs | This cannot happen. DFSMStvs processing is quiesced; therefore, DFSMStvs cannot write to its system logs. |

10. The DFSMStvs state is quiesced because a system log I/O error occurred.

DFSMStvs does not reach this state until the last data set that was open for DFSMStvs processing is closed. In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no closes or record management requests can occur after the last data set open for DFSMStvs processing is closed.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | The command is rejected. A cold start is required to recover from the system log I/O error. The SMSVSAM server must be recycled. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that the command was issued so that it will not come up automatically if the SMSVSAM server happens to recycle with TV_START_TYPE set to COLD. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that the command was issued so that it will not come up automatically if the SMSVSAM server happens to recycle with TV_START_TYPE set to COLD. |
| RRS becomes unavailable | DFSMStvs unregisters with RRS and RLS, disconnects from its logs, and sets its state to disabled. |
| A system log I/O error occurs | This cannot happen. DFSMStvs processing is quiesced; therefore, DFSMStvs cannot write to its system logs. |

11. The DFSMStvs state is quiesced because a VARY command was issued to quiesce DFSMStvs processing and a system log I/O error occurred.

DFSMStvs does not reach this state until the last data set that was open for DFSMStvs processing is closed. In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no closes or record management requests can occur after the last data set open for DFSMStvs processing is closed.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs resets the indicator that a command was issued to quiesce but does not reinitialize. A cold start is required to recover from the system log I/O error. The SMSVSAM server must be recycled. |
| VARY command to quiesce DFSMStvs | The command is rejected. DFSMStvs is already quiesced. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already quiesced. |
| RRS becomes unavailable | DFSMStvs unregisters with RRS and RLS, disconnects from its logs, and sets its state to disabled. |
| A system log I/O error occurs | This cannot happen. DFSMStvs processing is quiesced, therefore, DFSMStvs cannot write to its system logs. |

12. The DFSMStvs state is disabling because RRS became unavailable, and then RRS became available again.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with

nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. DFSMStvs must complete the disable process before can reinitialize. It will do so automatically once the disable process completes. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that the command was issued. When RRS comes back up, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that the command was issued so that it will not automatically reinitialize when the disable completes. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred and reinitializes when RRS becomes available again. |
| A system log I/O error occurs | DFSMStvs makes a note that this error occurred so that it does not automatically reinitialize until DFSMStvs is cold started. |

13. The DFSMStvs state is disabling because RRS became unavailable, and then RRS became available again; in addition, an operator command was issued to quiesce DFSMStvs processing.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It will reinitialize automatically once the disable process completes. |
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was previously issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that the command was issued so that it will not automatically reinitialize when the disable completes. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It does <i>not</i> reinitialize automatically when RRS becomes available again because a command was issued to quiesce it. |
| A system log I/O error occurs | DFSMStvs makes a note that this error occurred so that it does not automatically reinitialize until DFSMStvs is cold started. |

14. The DFSMStvs state is disabling because an operator command was issued to disable DFSMStvs processing.

In this state, all opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | The command is rejected. Quiescing is a less restrictive state than disabling. As a result, transitioning from disabling to quiescing is not allowed. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabling. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It does <i>not</i> reinitialize automatically when RRS becomes available again because a command was issued to disable it. |
| A system log I/O error occurs | DFSMStvs makes a note that this error occurred so that it does not automatically reinitialize until DFSMStvs is cold started. |

15. The DFSMStvs state is disabling because RRS became unavailable and then became available again; in addition, a system log I/O error occurred.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. A cold start must be done to recover from the system log I/O error, which requires recycling the server. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. When the SMSVSAM server recycles with TV_START_TYPE set to COLD, DFSMStvs reinitializes in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred so that it does not automatically reinitialize if the SMSVSAM server recycles with TV_START_TYPE set to COLD. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It does <i>not</i> reinitialize automatically when RRS becomes available again because of the system log I/O error. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

16. The DFSMStvs state is disabling because RRS became unavailable.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. DFSMStvs will automatically reinitialize when the disable completes and RRS becomes available. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that the command was issued. When RRS comes back up, DFSMStvs reinitializes in a quiescing state long enough to clean up any incomplete units of recovery. It does not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred so that it does not automatically reinitialize when RRS becomes available. |
| RRS becomes available | DFSMStvs finishes disabling and then reinitializes itself automatically. |
| A system log I/O error occurs | DFSMStvs makes a note that this occurred. A cold start is required to recover from the error. |

17. The DFSMStvs state is disabling because an operator command was issued to disable DFSMStvs processing and a system log I/O error occurred.

In this state, all opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | The command is rejected. Quiescing is a less restrictive state than disabling. As a result, transitioning from disabling to quiescing is not allowed. |
| VARY command to quiesce DFSMStvs | DFSMStvs switches from disabling to quiescing and sets its state to quiescing. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabling. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It does <i>not</i> automatically reinitialize when RRS becomes available because an operator command was issued to disable it. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

18. The DFSMStvs state is disabling because RRS is unavailable and because an operator command was issued to disable DFSMStvs processing

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It will reinitialize automatically when RRS becomes available. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred, but the state remains disabling because of the RRS failure. When RRS comes back up, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabling. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It does <i>not</i> reinitialize itself automatically because an operator command was issued to disable DFSMStvs processing. |
| A system log I/O error occurs | DFSMStvs makes a note that this occurred. A cold start will be required to recover from the error. |

19. The DFSMStvs state is disabling because RRS is unavailable; in addition, an operator command was issued to quiesce DFSMStvs processing.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It will reinitialize automatically when RRS becomes available. |
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was already issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize automatically when RRS becomes available again because an operator command was issued to quiesce DFSMStvs processing. |
| A system log I/O error occurs | DFSMStvs makes a note that this occurred. A cold start will be required to recover. |

20. The DFSMStvs state is disabling because RRS is unavailable; in addition, a system log I/O error occurred.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. A cold start is required to recover from the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. When the SMSVSAM server recycles with TV_START_TYPE set to COLD, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred so that it will not automatically reinitialize if the SMSVSAM server recycles with TV_START_TYPE set to COLD. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize automatically because a cold start is required to recover from the system log I/O error. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

21. The DFSMStvs state is disabling because RRS is unavailable; in addition, a system log I/O error occurred and an operator command was issued to disable DFSMStvs processing.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It will <i>not</i> automatically reinitialize when RRS becomes available because of the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. When the SMSVSAM server recycles with TV_START_TYPE set to COLD, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | The command is rejected. Such a command was already issued. |

| Event | Effect |
|-------------------------------|--|
| RRS becomes available | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize automatically because a cold start is required to recover from the system log I/O error. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

22. The DFSMStvs state is disabling because RRS is unavailable; in addition, a system log I/O error occurred and an operator command was issued to quiesce DFSMStvs processing.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It will <i>not</i> automatically reinitialize when RRS becomes available because of the system log I/O error. |
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was already issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize automatically because a cold start is required to recover from the system log I/O error. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

23. The DFSMStvs state is disabling because RRS became unavailable and then became available again, and an operator command was issued to disable DFSMStvs; in addition, a system log I/O error occurred.

In this state, DFSMStvs must complete the disable process before it can reinitialize because RRS reinitialized, which requires that DFSMStvs reinitialize as a new instance. All opens are failed with nonzero return and reason codes. All closes are processed normally. All record management requests are failed with nonzero return and reason codes.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|---------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred but does not reinitialize. cold start must be done to recover from the system log I/O error, which requires recycling the server. |

| Event | Effect |
|----------------------------------|--|
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was already issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize automatically because of the system log I/O error. |
| A system log I/O error occurs | There is no operational change because such an error has already occurred. |

24. The DFSMStvs state is disabled because RRS became unavailable and then became available again.

When DFSMStvs reaches this state, it begins the process of reinitialize itself and sets its state to initializing, as long as no commands are issued to the contrary. In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. DFSMStvs is about to reinitialize on its own. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that the command was issued. When RRS comes back up, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. The state remains DISABLE and DFSMStvs does not reinitialize. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It begins the process of reinitializing itself and waits for RRS to become available. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

25. The DFSMStvs state is disabled because RRS became unavailable and then became available again, and an operator command was issued to quiesce DFSMStvs processing before DFSMStvs could begin to reinitialize.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was already issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize itself automatically when RRS becomes available again because an operator command was issued to quiesce DFSMStvs processing. |

| Event | Effect |
|-------------------------------|--|
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

26. The DFSMStvs state is disabled because an operator command was issued to disable DFSMStvs processing.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | The command is rejected. DFSMStvs is already disabled. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabled. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize itself automatically when RRS becomes available again because an operator command was issued to disable DFSMStvs processing. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

27. The DFSMStvs state is disabled because an operator command was issued to disable DFSMStvs processing; in addition, a system log I/O error occurred. A subsequent VARY command might also have been issued to change the state of DFSMStvs, as follows:

- Enable DFSMStvs, but DFSMStvs could not reinitialize because of the system log I/O error
- Quiesce DFSMStvs, which made a note of this, but the state did not change because DFSMStvs was already disabled

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|--|
| VARY command to enable DFSMStvs | The command is rejected. A cold start is required to recover from the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabled. |
| RRS becomes unavailable | DFSMStvs makes a note that this occurred. It will <i>not</i> reinitialize itself automatically when RRS becomes available again because of the system log I/O error. |

| Event | Effect |
|-------------------------------|--|
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

28. The DFSMStvs state is disabled because RRS is unavailable.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that the command was issued. When RRS comes back up, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. It will <i>not</i> automatically reinitialize when RRS becomes available. |
| RRS becomes available | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

29. The DFSMStvs state is disabled because RRS is unavailable and an operator command was issued to disable DFSMStvs processing.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. It will <i>not</i> automatically reinitialize when RRS becomes available. |
| VARY command to disable DFSMStvs | The command is rejected. DFSMStvs is already disabled. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It does <i>not</i> automatically reinitialize itself because an operator command was issued to disable DFSMStvs processing. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

30. The DFSMStvs state is disabled because RRS is unavailable and an operator command was issued to quiesce DFSMStvs processing.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs begins the process of reinitializing itself and sets its state to initializing. |
| VARY command to quiesce DFSMStvs | The command is rejected. Such a command was already issued. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It does <i>not</i> automatically reinitialize itself because an operator command was issued to quiesce DFSMStvs processing. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

31. The DFSMStvs state is disabled because RRS is unavailable; in addition, a system log I/O error occurred.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | The command is rejected. A cold start is required to recover from the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. When the SMSVSAM server recycles with TV_START_TYPE set to COLD, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred so that it will not automatically reinitialize if the SMSVSAM server recycles with TV_START_TYPE set to COLD. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It does <i>not</i> automatically reinitialize itself because the system log I/O error requires a cold start. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

32. The DFSMStvs state is disabled because RRS is unavailable; in addition, an operator command was issued to disable or quiesce DFSMStvs processing, and a system log I/O error occurred.

In this state, all opens are failed with nonzero return and reason codes. All closes and record management requests are rejected with X'0F4' abends because no data sets should be open for DFSMStvs processing if DFSMStvs is disabled.

Recommendation: After a cold start, any data sets for which recovery was not completed are most likely left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the forward recovery log, and then delete and redefine the forward recovery log.

| Event | Effect |
|----------------------------------|---|
| VARY command to enable DFSMStvs | DFSMStvs makes a note that this occurred. It does not reinitialize. A cold start is required to recover from the system log I/O error. |
| VARY command to quiesce DFSMStvs | DFSMStvs makes a note that this occurred. When the SMSVSAM server recycles with TV_START_TYPE set to COLD, DFSMStvs will reinitialize in a quiescing state long enough to clean up any incomplete units of recovery. It will not allow new work to begin. |
| VARY command to disable DFSMStvs | DFSMStvs makes a note that this occurred so that it will not automatically reinitialize if the SMSVSAM server recycles with TV_START_TYPE set to COLD. |
| RRS becomes available | DFSMStvs makes a note that this occurred. It does <i>not</i> automatically reinitialize itself because the system log I/O error requires a cold start. |
| A system log I/O error occurs | This cannot occur when DFSMStvs is disabled because there is no DFSMStvs activity. |

Improving sequential performance

You can tune VSAM NSR to provide better sequential performance by requesting additional data buffers using BUFND. The performance difference depends on the degree of tuning in your applications. If you use the default number of buffers, the difference in performance is much less pronounced than if you have optimized buffer use to transfer entire cylinders of data in a single I/O.

For sequential processing, the way that locks are held and released is predictable. Locks are obtained and released in sequence through the data set as the job progresses.

Sequential update applications that use large buffers can have the greatest performance difference when moved to DFSMStvs, which caches data in the coupling facility. DFSMStvs does not attempt to read ahead for sequential access. Unless a record is found in the SMSVSAM buffer pool, each VSAM request results in the transfer of one control interval. This is a lower degree of buffering than you can do for sequential nonshared access to VSAM data. So, the elapsed time of sequential updates increases because the data transfer is less efficient.

Improving logging performance

An application that commits changes regularly incurs some additional cost because DFSMStvs logs changes. DFSMStvs uses the z/OS system logger, which keeps log entries in the coupling facility for as long as the size of the log permits.

However, an application that does not commit often enough could cause the log entries to spill from the coupling facility to disk. DFSMStvs regularly trims its system log entries when they are no longer needed after a commit. If a long time passes between commits, the effectiveness of this process is diminished. Should this occur, the cost of logging could rise significantly.

You should plan to review logging activity using the SMF type 88 records produced by the system logger.

Tuning the DFSMStvs environment

In addition to improving sequential performance and logging, you can tune several things in the DFSMStvs environment to influence the performance. What you tune depends on the information you gained from performance monitoring, as “Monitoring performance” on page 71 describes. The following list describes some things that you might want to tune:

- Applications

If you detect that there is excessive lock contention or that the system logger is forced to spill active log records to disk, you might want to tune the commit frequency that you have implemented in an application program.

- Coupling-facility storage

- SMSVSAM use

If you see structure-full events for the SMSVSAM structures, you might want to change the amount of storage in the coupling facility available for SMSVSAM.

- System logger use

If you see structure-full events for the logger structures, you might want to change the amount of storage within the coupling facility available to the system logger.

When a log becomes full and the system logger has to offload log data, the system logger starts surfacing temporary errors. While this occurs, it is impossible to write anything to the log.

- DFSMStvs SMS settings

If you see many occurrences of a coupling facility log structure filling and spilling to disk, you might want to reduce the activity keypoint frequency. Setting the activity keypoint frequency too low, however, would increase the amount of processor time needed to trim logs.

- Application parallelism

When you run multiple batch jobs against the same shared VSAM data sets, you can obtain benefits by rescheduling existing jobs. You can take advantage of application parallelism still further by taking existing jobs and splitting them into multiple parallel jobs. This reduces the overall run time substantially, depending on how many ways you split a single job. A shorter run time does, however, mean that the total amount of resources consumed by the job is now consumed in a shorter period of time, so creating more parallel jobs can cause a peak in processor use and I/O demand.

Recommendation: Improve performance by setting up your z/OS system optimally in these ways:

- Run DFSMStvs batch and CICS from separate z/OS images rather than combining the two within one z/OS image.
- Place a couple data sets and JES2 checkpoint on different volumes.
- Place the primary sysplex CDS and the coupling facility resource management (CFRM) CDS on different volumes.
- Ideally, spread the primary and alternate couple data sets and the CFRM data sets across four volumes.
- Give SMSVSAM a higher dispatching priority than VTAM and CICSplex® System Manager, which in turn should have a higher dispatching priority than CICS. When you run in goal mode, however, you should allow these system address spaces to default to SYSTEM/SYSSTC.
- As a starting point, set the activity keypoint to 5000.
- Use GRS star mode.
- Define only the number of systems that will actually join the sysplex in a couple data set MAXSYSTEM value.

The RLS lock structure, IGWLOCK00, bases the size of each lock entry on the number of systems permitted to join the sysplex. Each lock entry increases in size as more systems are defined. If you define more systems in the couple data set MAXSYSTEM value than will actually join, each record is larger than necessary and you can fit fewer records in a given amount of coupling facility space.

- Increase the size of your lock structure (IGWLOCK00). DFSMStvs introduces additional locking because batch jobs do not hold locks in today's processing environment.

Chapter 7. Diagnosing and recovering from DFSMStvs problems

This topic contains information that is NOT Programming Interface information.

This information covers these topics:

- [“Diagnosing system logger and performance problems” on page 95](#)
- [“Interrupting an operation or resource request” on page 99](#)
- [“Recovering from a log stream problem” on page 99](#)
- [“Resolving waits” on page 100](#)
- [“Restarting DFSMStvs after SMSVSAM address space failure” on page 100](#)
- [“Cold starting DFSMStvs” on page 101](#)
- [“Performing peer recovery” on page 101](#)

Diagnosing system logger and performance problems

Problems within the system logger or other areas of MVS can cause extended waits by DFSMStvs for logging. If the system logger is not available, DFSMStvs cannot continue processing. This topic provides information to help you resolve problems with the system logger.

Categorizing a system logger problem

DFSMStvs could encounter the following categories of problems, shown in ascending order of impact on the user:

1. Those problems within the system logger that the system logger resolves for itself.
 DFSMStvs has no involvement in this category and the problem might be perceived as merely an increase in response times.
 There is also a dependency on RRS. If RRS encounters a problem, perhaps with one of its log streams, DFSMStvs is impacted. If RRS is not active or fails, DFSMStvs is impacted.
2. The system logger is unable to immediately satisfy a DFSMStvs request. This problem state can be encountered on a structure full condition where the coupling facility has reached its capacity before offloading data to DASD. This state can also be encountered during the rebuilding of a coupling facility structure.
 DFSMStvs is able to recognize this situation and retries the request every three seconds until the request is satisfied. Typically, this can take up to a minute. Message IGW838I is issued to indicate that this has occurred.
3. If the system logger fails, DFSMStvs becomes disabled. If the system log has not been damaged, a subsequent restart of DFSMStvs should succeed.
4. If a return code implies that the system log has been damaged or records have been lost, DFSMStvs is quiesced. This means that transactions run to completion as far as possible with no further records written to the system log. You must then specify TV_START_TYPE(COLD) in your IGDSMSxx member of SYS1.PARMLIB and use the SET SMS command to cold start DFSMStvs.

Recommendation: After a cold start, data sets for which recovery was not completed could be left in a damaged state and must be recovered manually. If the data sets are forward recoverable, their forward recovery logs might also be damaged. Manually recover the data sets (without using forward recovery), take backups of them and of any other data sets that use the same forward recovery log, and then delete and redefine the forward recovery log.

If a return code implies damage to a forward recovery log or one of the system logs, DFSMStvs quiesces the log, and the units of recovery run to completion. The data needed to back out in-flight units of recovery is lost, and some manual recovery might be necessary.

Before you continue to use the log stream, follow these steps:

- a. Delete and redefine the log stream.
- b. Take an image copy of all data sets that reference the log stream.
- c. Unquiesce the affected logs.
- d. In some circumstances, DFSMStvs quiesces a data set. You must unquiesce it. Then you can explicitly open the data set.

DFSMStvs does not provide any mechanism to automatically delete records from forward recovery log streams. You must delete such data in order to manage the size of forward recovery log streams. If you need long-term data retention of data in the log stream, you might want to copy the data from log stream storage to alternative archive storage.

Collecting diagnostic information about logging problems

If you think there is a problem within the system logger, you can begin to collect additional diagnostic information. The dumps generated by DFSMStvs generally do not contain sufficient information about the system logger. DFSMStvs log records include the subsystem ID, like IGWTVS01 and IGWTVS02.

Recommendation: Define the system log streams with DIAG(YES). This forces the system logger to create memory dump in the event of a lost data condition.

A dump of XCF and system logger address spaces from all systems are useful in diagnosing such problems. Issue the following series of MVS commands as shown in [Figure 16 on page 96](#):

```
DUMP COMM=(meaningful dump title)
R ww,JOBNAME=(IXGLOGR,XCFAS,DFSMStvs_jobname),DSPNAME=('XCFAS'.*),CONT
R xx,STRLIST=(STRNAME=structure,(LISTNUM=ALL),ACC=NOLIM),CONT
R yy,REMOTE=(SYSLIST=('XCFAS','IXGLOGR'),DSPNAME,SDATA),CONT
R zz,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,TRT,CSA,GRSQ,XESDATA),END
```

Figure 16. Example of MVS commands to produce a dump of XCF and system logger address spaces

If you suspect a problem with the coupling facility structure, use the `R xx,STRLIST=(STRNAME=structure,(LISTNUM=ALL),ACC=NOLIM),CONT` instruction.

Error records written to the log data set might also be useful.

Related reading: For more information about how to code the commands, see [z/OS MVS Diagnosis: Tools and Service Aids](#).

Investigating console messages and dumps

You can diagnose some problems by investigating MVS console messages and dumps. Look for the following clues:

- Outstanding WTOR messages
- IXGxxx messages
- Allocation, catalog, and HSM error messages
- I/O errors for log stream data sets or LOGR couple data sets

Log stream data sets that are named `IXGLOGR.stream_name.Annnnnnnn`. The high-level qualifier (IXGLOGR) can be different if the HLQ parameter was specified when the log stream was defined.

- IXCxxx messages, which indicate problems with the coupling facility structure or couple data sets.
- 1C5 abends and other abends from the IXGLOGR address space.

Displaying coupling-facility status

To display the system logger couple data set status, issue the MVS command as shown in [Figure 17 on page 97](#)

```
D XCF,CPL,TYPE=LOGR
```

Figure 17. Example of a command to display system logger couple data set status

A normal response looks like the example shown in [Figure 18 on page 97](#):

```
D XCF,CPL,TYPE=LOGR
IXC358I 14.47.51 DISPLAY XCF 391
LOGR COUPLE DATA SETS
PRIMARY DSN: SYS1.SYSPLEX2.SEQ26.PLOGR
VOLSER: P2SS05 DEVN: 230D
FORMAT TOD MAXSYSTEM
12/20/95 09:25:48 8
ALTERNATE DSN: SYS1.SYSPLEX2.SEQ26.ALOGR
VOLSER: P2SS06 DEVN: 2C10
FORMAT TOD MAXSYSTEM
12/20/95 09:27:45 8
LOGR IN USE BY ALL SYSTEMS
```

Figure 18. Example of a normal response from a command to display system logger couple data set status

If the response shows that LOGR is not in use by all systems, this could be a sign of a problem you need to investigate further. Look for IXCxxx messages that might indicate the cause of the problem, and issue the command as shown in [Figure 19 on page 97](#) to attempt reconnection to the couple data set:

```
SETXCF CPL,TYPE=(LOGR),PCOUPLE=(couple_dataset_name)
```

Figure 19. Example of a command to reconnect the couple data set

To display all structures with Failed_Persistent connections, issue the MVS command as shown in [Figure 20 on page 97](#):

```
D XCF,STR,STRNM=*,STATUS=FPCONN
```

Figure 20. Example of a command to display all structures with Failed_Persistent connections

You might also see latch set name SYS.IXGLOGGER_MISC. The system logger should resolve any failed connections.

Checking global resource serialization (GRS) resource contention

To check GRS resource contention by displaying GRS enqueues and latch usage on all machines in the sysplex, issue either of the following MVS commands, as shown in [Figure 21 on page 97](#):

```
D GRS,C
D GRS,RES=(SYSZLOGR,*)
```

Figure 21. Examples of DISPLAY GRS commands

A normal response looks like one shown in [Figure 22 on page 97](#):

```
D GRS,C
ISG020I 12.06.49 GRS STATUS 647
NO ENQ CONTENTION EXISTS
NO LATCH CONTENTION EXISTS
D GRS,RES=(SYSZLOGR,*)
ISG020I 14.04.28 GRS STATUS 952
NO REQUESTORS FOR RESOURCE SYSZLOGR *
```

Figure 22. Example of a normal response from a DISPLAY GRS command

A response showing GRS contention is shown in [Figure 23 on page 98](#):

```
GRS,C
ISG020I 12.06.31 GRS STATUS 619
LATCH SET NAME: SYS.IXGLOGGER_LCBVT
CREATOR JOBNAME: IXGLOGR CREATOR ASID: 0202
LATCH NUMBER: 7
REQUESTOR ASID EXC/SHR OWN/WAIT
IXGLOGR 0202 EXCLUSIVE OWN
IXGLOGR 0202 SHARED WAIT
D GRS,RES=(SYSZLOGR,*)
ISG020I 19.58.33 GRS STATUS 374
S=STEP SYSZLOGR 91
SYSNAME JOBNAME ASID TCBADDR EXC/SHR OWN/WAIT
MV26 MSLDELC1 002F 008F6370 EXCLUSIVE OWN
S=STEP SYSZLOGR 93
SYSNAME JOBNAME ASID TCBADDR EXC/SHR OWN/WAIT
MV26 MSLWRTC1 002E 008DED90 EXCLUSIVE OWN
MV26 MSLWRTC1 002E 008DB990 EXCLUSIVE WAIT
MV26 MSLWRTC1 002E 008DB700 EXCLUSIVE WAIT
MV26 MSLWRTC1 002E 008F60C8 EXCLUSIVE WAIT
S=SYSTEMS SYSZLOGR LPAYROL.TESTLOG.TLOG1
SYSNAME JOBNAME ASID TCBADDR EXC/SHR OWN/WAIT
MV27 IXGLOGR 0011 008F7398 EXCLUSIVE OWN
MV26 IXGLOGR 0011 008F7398 EXCLUSIVE WAIT
```

Figure 23. Example of a GRS command showing contention

This example shows which tasks have exclusive enqueues on the log streams and which tasks are waiting for them. It is quite normal for enqueues and latches to be obtained, occasionally with contention. They are indications of a problem only when they last for more than a minute or so.

Long term enqueueing on the SYSZLOGR resource can be a sign of problems even if there is no contention.

You can choose to display only those log streams exclusively enqueued on, or being waited on, by DFSMStvs jobs in the sysplex. Issue the MVS command shown in [Figure 24 on page 98](#):

```
D GRS,RES=(SYSDSN,*)
```

Figure 24. Example of a GRS command to display log streams with an exclusive enqueue

A typical response to this command looks like the example in [Figure 25 on page 98](#):

| S=SYSTEMS | SYSDSN | JOBNAME | ASID | TCBADDR | EXC/SHR | STATUS |
|-----------|--------|--------------|------|----------|-----------|--------|
| SY02 | ANTHEM | | 002B | 009FFBF8 | EXCLUSIVE | OWN |
| S=SYSTEMS | SYSDSN | ASM.SASMMOD1 | | | | |
| S=SYSTEMS | SYSDSN | JOBNAME | ASID | TCBADDR | EXC/SHR | STATUS |
| SY03 | XCFAS | | 0006 | 008FFBF8 | SHARE | OWN |
| SY03 | LLA | | 0017 | 008FFBF8 | SHARE | OWN |
| SY01 | XCFAS | | 0006 | 008FFBF8 | SHARE | OWN |
| SY01 | LLA | | 0017 | 008FFBF8 | SHARE | OWN |
| SY05 | XCFAS | | 0006 | 008FFBF8 | SHARE | OWN |
| SY05 | LLA | | 0017 | 008FFBF8 | SHARE | OWN |
| SY04 | XCFAS | | 0006 | 009FFBF8 | SHARE | OWN |
| SY04 | LLA | | 0017 | 009FFBF8 | SHARE | OWN |
| SY02 | XCFAS | | 0006 | 009FFBF8 | SHARE | OWN |
| SY02 | LLA | | 0017 | 009FFBF8 | SHARE | OWN |
| S=SYSTEMS | SYSDSN | ASM.SASMMOD2 | | | | |
| S=SYSTEMS | SYSDSN | JOBNAME | ASID | TCBADDR | EXC/SHR | STATUS |
| SY03 | XCFAS | | 0006 | 008FFBF8 | SHARE | OWN |

Figure 25. Example of output from a GRS command to display log streams with an exclusive enqueue

Checking SMF and RMF statistics for performance problems

SMF type 88 log stream statistics records and RMF coupling facility usage reports are useful for analyzing problems that are affecting performance. Increasing the amount of coupling facility storage allocated to a structure can improve both system logger performance and DFSMStvs performance.

The sample program IXGRPT1, supplied in SYS1.SAMPLIB, provides an example of a program to analyze SMF type 88 records.

Interrupting an operation or resource request

When the system logger becomes hung up on an operation or waits for a resource, bottlenecks can occur on that system. Because the system logger manages sysplex resources, a bottleneck on a single system can have an effect on the entire sysplex. Interrupting a particular stream request for a resource might allow much other work to continue in the system and sysplex.

The system logger monitors its allocation and HSM recall service tasks for delays and provides a mechanism (through WTORs) to interrupt these delayed requests. To the system logger, the interruption is an error condition for the current request. Removal of a delayed request enables the processing of other log stream resource requests. Messages IXG271I and IXG272E are issued if the system logger detects that a delay of a service request is inhibiting other log stream resource requests.

You can reply FAIL to message IXG272E to interrupt a delayed logger service request. This reply causes the request to fail and might also allow other work that was waiting to continue.

Use the FAIL option only if you cannot determine why the request is not completing. Replying FAIL might cause undesirable results. This option is meant to keep the rest of the system logger applications running, at the expense of one hung application.

Examine any other error messages from the system logger or from any exploiter of the affected log stream. If you reply FAIL, the system logger might cause other components such as allocation to enter their recovery, create memory dumps, or issue various messages. If you reply FAIL to message IXG272E, you might see a dump in media manager, dynamic allocation, or catalog, depending on what type of I/O delay that the system logger was experiencing at the time IXG272E was issued. There is an unexpected dump that results in abend 0E0 during SVCDMP processing.

Recovering from a log stream problem

In the event of a problem with a specific log stream, you might need to keep DFSMStvs from using the log stream while you correct the problem. To stop DFSMStvs from using the log stream, follow these steps:

1. Use the VARY SMS,LOG operator command to quiesce or disable DFSMStvs from using the log stream. If the log stream is usable but currently experiencing problems, quiesce access to it. This enables DFSMStvs to complete processing of any in-progress units of recovery. If the log stream has become unusable, disable it. This causes DFSMStvs to stop using the log stream immediately.

Recommendation: Be careful about quiesce with a path name; always use a base name.

If you disable DFSMStvs, it goes into a disabling state in which it rejects any new record management requests (get, point, put, or erase). For DFSMStvs to go into a disabled state, all current transactions must be brought to a sync point (committed or backed out) and all ACBs must be closed.

If you quiesce DFSMStvs, it goes into a quiescing state in which it processes record management requests for existing transactions but fails any record management request that would initiate a new transaction. For DFSMStvs to go to a quiesced state, all current transactions must be brought to a sync point and all ACBs must be closed.

Quiescing or disabling a DFSMStvs primary system (undo) log or secondary system (shunt) log is equivalent to quiescing or disabling DFSMStvs. DFSMStvs processing is unavailable until the log is re-enabled.

2. For a system log problem, identify and recover any data sets that might have been involved in in-flight units of recovery.
 - a. Do not try to bring DFSMStvs back up until you have finished this procedure.
 - b. Obtain a dump of XCF and system logger address spaces from all systems by issuing the following series of MVS commands:

```
DUMP COMM=(meaningful dump title)
R ww,JOBNAME=(IXGLOGR,XCFAS,cics_job),DSPNAME=('IXGLOGR'.*,'XCFAS'.*),CONT
R xx,STRLIST=(STRNAME=structure,(LISTNUM=ALL),ACC=NOLIM),CONT
R yy,REMOTE=(SYSLIST=*( 'XCFAS', 'IXGLOGR' ),DSPNAME,SDATA),CONT
R zz,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,TRT,CSA,GRSQ,XESDATA),END
```

- Use the R **xx**,STRLIST=(STRNAME=**structure**, (LISTNUM=ALL),ACC=NOLIM),CONT instruction only where you suspect a problem with the coupling-facility structure.
- c. Use the log data to identify any data sets that might have been involved in in-flight units of recovery.
 - d. For forward recoverable data sets, revert to the last good backup of the data set and apply forward recovery to it. Make sure not to apply anything that might belong to the in-flight unit of recovery because backout would not have been able to write the compensating records.
 - e. For data sets that are not forward recoverable, you probably need to look at the individual log records and fix each of the record manually by looking at what your batch job was doing and putting the records back the way they were. DFSMStvs log records include the subsystem ID, like IGWTVS01 and IGWTVS02.
3. Correct the problem. If the log is severely damaged, this might involve deleting and redefining it. If you delete and redefine the primary system log, ensure that any incomplete units of recovery are discarded. One way to do this is to cold start DFSMStvs, following these steps:
 - a. Use SET SMS=**xx** to assign an IGDSMSxx parmlib member in which a start type of cold is specified (TV_START_TYPE(COLD)).
 - b. Use the V SMS,TRANVSAM(**nnn**),E command to restart DFSMStvs.
 - c. Use the SET SMS command (TV_START_TYPE) to specify a PARMLIB member that has a start type of WARM.
 4. Re-enable the log stream to give DFSMStvs access using the VARY SMS,LOG operator command.

Resolving waits

For a batch job wait that you think might be related to DFSMStvs, diagnosis starts with a dump of the batch job. First examine the linkage stack of the batch job to determine with which component the batch job might have a cross-memory problem. Then perform a dump of that component. Many components can cause hangs, including DFSMStvs, VSAM RLS, a catalog, the system logger, and RRS.

A possible cause of apparent waits is RRS. If there is an in-flight UR and RRS loses access to its logs (for example, because the system logger goes down), RRS hold onto the batch jobs until the logs become available again. While the batch jobs are in this state, you cannot cancel them. The only way to free up the batch jobs is to cancel RRS or to get the system logger working again.

If you encounter a wait that you think is related to logging, check the MVS console for messages that have the prefix IXG. This prefix is for system logger messages. These messages might provide more information about the cause of the wait. The MVS console might also reveal evidence of resource contention within MVS.

Restarting DFSMStvs after SMSVSAM address space failure

If a DFSMStvs instance or the SMSVSAM server fails, normally, it restarts automatically unless it has been specifically requested not to restart automatically. An operator command, for example, is one way to request a DFSMStvs instance to not start automatically. When DFSMStvs restarts, it collects information about any units of recovery that were in progress at the time of the failure. This information comes from three sources:

- The system logs (undo and shunt logs)
- The lock manager (locks that are held on behalf of a unit of recovery)
- RRS, which indicates the state of the unit of recovery (for example, in-commit, in-doubt, or in-backout)

Based on this information, DFSMStvs determines what action to take on behalf of the unit of recovery:

- If the unit of recovery has locks or log records but is not known to RRS:
 - If a commit record is present in the log, DFSMStvs simply releases the locks; processing on behalf of this unit of recovery is complete.

- If a commit record is not present in the log, the unit of recovery was in-flight at the time of the failure, and restart processing backs out the unit of recovery.
- If RRS indicates that the unit of recovery was in-backout, then restart processing backs out the unit of recovery.
- If RRS indicates that the unit of recovery was in-commit, DFSMStvs completes commit processing for the unit of recovery.
- If RRS indicates that the unit of recovery was in-doubt, DFSMStvs waits for the in-doubt conditions to be resolved and takes the action indicated by RRS.
- If a unit of recovery has log records that do not include a commit record and for which no locks are held, DFSMStvs assumes there is an error. The locks are used to protect the modified records. This error can occur under either of these circumstances:
 - A data set that has retained locks and shunted log records is deleted.
 - The access method services SHCDS RESETLOCKS command is used.

Use the access method services SHCDS command to purge the unit of recovery.

Various levels of authorization are required to use the SHCDS parameters. For information about this authorization, see [z/OS DFSMS Access Method Services Commands](#).

Cold starting DFSMStvs

A cold start is a DFSMStvs restart in which restart processing is *not* performed. Instead, DFSMStvs deletes any information remaining in its system logs, releases any locks, and starts as if the log had been empty.

Recommendation: Do not cold start DFSMStvs unless the DFSMStvs system logs have been damaged. Any recovery processing that was to be done for recoverable data sets is not done after a cold start of DFSMStvs. Any data sets for which recovery was not complete are most likely left in a damaged state. If the data sets are forward recoverable, then their forward recovery logs might also be damaged. You should manually recover the data sets (without using forward recovery), make a backup of those data sets and any other data sets that use the same forward recovery log, and then delete and redefine the forward recovery log.

Performing peer recovery

Peer recovery is the process of completing the work that was left in an incomplete state due to the failure of an instance of DFSMStvs by another instance of DFSMStvs. The only function of peer recovery is to complete that work and then end; a peer recovery instance of DFSMStvs does not accept any new work.

Peer recovery occurs only in cases of *system* failure, not merely when DFSMStvs fails. These rules apply to peer recovery:

- Peer recovery occurs only when both DFSMStvs and the system on which it was running fail. Peer recovery does not occur when the DFSMStvs instance failed, but the system continued to run. If this were the case, either DFSMStvs would automatically restart, or the DFSMStvs instance was stopped and was not meant to be restarted.
- All resource managers that had shared interest in units of recovery restart on the same system. For this reason, DFSMStvs uses the automatic restart manager (ARM) to manage the grouping. If the installation is not using ARM, or if ARM is unavailable, you can manually initiate peer recovery by issuing the VARY SMS command, as follows:

```
VARY SMS,TRANVSAM(001),PEERRECOVERY,ACTIVE
```

Issue the VARY SMS command on another system. That system then runs peer recovery for the instance specified on the command, as well as any instances for which that instance was performing peer recovery.

Normally, if a DFSMStvs was disabling or disabled due to an operator command, peer recovery does not run. This is because that instance of DFSMStvs was told to come down and not restart. If you want peer recovery to occur, issue this command:

```
VARY SMS,TRANSAM(001),PEERRECOVERY,ACTIVEFORCE
```

- A peer recovery DFSMStvs instance is only started if there is a primary DFSMStvs instance of DFSMStvs running on the system. If DFSMStvs is not started on the system, peer recovery does not run.
- Peer recovery starts asynchronous tasks to process outstanding units of recovery in parallel. As the tasks complete, additional tasks are started, until all the outstanding units of recovery are processed or an operator command is issued to request the end of peer recovery processing. If such a command is issued, tasks that are already running are allowed to complete, then peer recovery processing ends.
- Peer recovery is allowed to run if the state of the failed DFSMStvs instance was quiescing, enabling, or enabled, since peer recovery would only complete the quiesce process. It is not allowed if the state was quiesced since, in this case, there should be no work to do. It also is not allowed if the state was disabled; this implies that the installation did not want the DFSMStvs instance to do any work. If you want peer recovery to be performed on behalf of a DFSMStvs instance that was disabling or disabled, you must use the VARY SMS command with the ACTIVEFORCE keyword.

If the DFSMStvs instance had been disabling due to an RRS failure, peer recovery is allowed to run. In this case DFSMStvs is reinitialized when RRS became available again.

Peer recovery initiation

You can use the ARM to initiate automatic peer recovery processing or the VARY SMS command to initiate peer recovery manually.

Recommendation: Use ARM as the method of initiating peer recovery processing.

To use ARM, you must create an ARM policy that groups instances of DFSMStvs with other resource managers that might have a shared instance in a unit of recovery. ARM supports this method with the administrative policy RESTART_GROUP utility control statement, which identifies related elements that are to be restarted as a group if the system on which they are running fails.

DFSMStvs registers with ARM as an abstract resource (one that is not associated with a job or started task) when it initializes. DFSMStvs requests that it be restarted only when the system on which it is running fails unexpectedly by specifying an element ending type of SYSTERM and an element bind of CURSYS. DFSMStvs provides its restart method by supplying the text of the required VARY SMS command as its start text when it registers with ARM. If DFSMStvs ends, it deregisters with ARM.

If the installation is not using ARM or if ARM is unavailable, you can initiate peer recovery manually by using the VARY SMS command. For example, issue this command:

```
VARY SMS,TRANSAM(001),PEERRECOVERY,ACTIVE
```

SMSVSAM failures while peer recovery is in process

DFSMStvs registers with ARM, requesting that it be restarted only in event of a system failure. If the SMSVSAM server in which peer recovery was running fails, ARM cannot automatically restart peer recovery. Instead, DFSMStvs remembers that it was performing peer recovery by writing that information to the SHCDS. During initialization, it reads this information and restarts any failed peer recovery work.

System failures while peer recovery is in process

A system failure has the effect of ending the primary instance of DFSMStvs running on the system, as well as any peer recovery instances. All instances register with ARM while they are active. ARM recognizes a failure and initiates peer recovery for all involved instances on another system or systems. If peer recovery is being initiated manually, then a VARY SMS command must be issued on another system. That system then runs peer recovery for the instance specified on the command, as well as any instances for which that instance was performing peer recovery.

For example, DFSMStvs instance IGWTV001 was running peer recovery for instances IGWTV002 and IGWTV003. If the system fails, and peer recovery is initiated for IGWTV001 on another system, that system performs peer recovery for IGWTV001, IGWTV002, and IGWTV003. However, starting peer recovery for IGWTV002 or IGWTV003 on another system would start peer recovery for the specified instance, and *only* for the specified instance.

Peer-recovery interference with failed instance restart

Depending on the amount of work left incomplete by a failure, peer recovery can take a significant amount of time. It is possible that the DFSMStvs instance might attempt to restart while peer recovery is in progress. Since another system is registered with its instance name, its initialization would fail. To prevent this, DFSMStvs has serialization in place to detect this type of error. The DFSMStvs instance finishes coming up when the peer recovery instance finishes.

Peer recovery processing obtains an enqueue, releasing it when peer recovery ends, which allows the DFSMStvs instance to proceed with initialization. If peer recovery is running and you need it to stop to allow the failed instance of DFSMStvs to reinitialize, you can stop it at any time by using the VARY SMS command to vary it INACTIVE. Peer recovery does not stop immediately but instead completes any tasks that it has already started and then ends without starting any additional tasks. The failed instance of DFSMStvs completes the remaining work when the instance reinitializes as part of its restart processing.

Appendix A. Quiescing a data set

DFSMS and application programs can initiate a quiesce of DFSMStvs activity against a sphere across the sysplex. You can quiesce a sphere using the CICS SET DSNAME command, the equivalent CEMT command, or the MVS VARY SMS command.

Recommendation: Be careful about quiesce with a path name; always use a base name.

Unlike CICS, DFSMStvs does not control the open requests and close requests of data sets that it accesses. DFSMStvs supports the ability to quiesce and close ACBs in a limited manner. If a data set is not currently open for DFSMStvs access, the data set is quiesced; otherwise, the quiesce of the data set is rejected. To quiesce a data set completely, ensure that it is not currently open in DFSMStvs mode; take these steps

Take these steps to quiesce a data set:

1. Display the jobs that use the data with the DISPLAY SMS,DSNAME operator command or the access method services SHCDS LISTDS(*dsname*) JOBS command.

Various levels of authorization are required to use the SHCDS parameters. For information about this authorization, see [z/OS DFSMS Access Method Services Commands](#).

2. Either allow those jobs to complete normally or cancel them. An in-flight unit of recovery is backed out if it is cancelled.
3. Use the VARY SMS,SMSVSAM,SPHERE command, or an equivalent CICS command, to quiesce the data set.

Recommendation: Be careful about quiesce with a path name; always use a base name.

4. Perform the operation or operations that required the data set to be quiesced.
5. Use the VARY SMS,SMSVSAM,SPHERE command, or an equivalent CICS command, to enable the data set for DFSMStvs and CICS.

When you specify data sets for a VARY SMS,SMSVSAM,SPHERE command, the data sets are not necessarily quiesced in the order in which you specified them or in any other order. When you use an asterisk (*) to specify data sets, they are not necessarily quiesced alphabetically or in any other order. So, when the last data set that you specified is quiesced, the other data sets that you specified might not have been quiesced yet. Do not issue a quiesce command for a particular data set until the previous operation on that data set completes.

For example, if an operator at a console enters VARY A QUIESCE followed by VARY A ENABLE, either command could take effect first. Also, the operator could enter the VARY QUIESCE or VARY ENABLE command with a path name while someone else enters one of these commands with a base name, and either command could take effect first.

Recommendation: Be careful about quiesce with a path name; always use a base name.

DFSMStvs supports the following VSAM IDAQUIES macro quiesce types:

QUICLOSE

Quiesce and close ACBs

QUIOPEN

Unquiesce

QUICOPY

Quiesce for COPY operation

QUICEND

End of quiesce for COPY operation

QUIBWO

Prepare for BWO copy operation

Quiescing a data set

QUIBEND

End of prepare for BWO copy operation

QUIFRC

Forward recovery complete

QUICMP

Completion of quiesce processing

QUICA

Cache available

Appendix B. Accessing data sets that have retained locks or lost locks

If a data set has retained locks or is in a lost-locks state, an open request from a batch job fails if it is neither for DFSMStvs nor VSAM RLS access. To enable the batch job to access this data set without DFSMStvs or VSAM RLS, the operator can issue the access method services SHCDS PERMITNONRLSUPDATE command. After the operator issues the command, the batch job's open request is successful, even if there are retained or lost locks, provided that there are no concurrent DFSMStvs or VSAM RLS open requests for the data set.

Various levels of authorization are required to use the SHCDS parameters. For information about this authorization, see [z/OS DFSMS Access Method Services Commands](#).

The SHCDS PERMITNONRLSUPDATE command enables the batch job to update or delete records for which there are retained locks. Some time after the batch job runs, DFSMStvs might be required to back out the units of recovery for which it holds retained locks. If the backout is done, DFSMStvs might invalidate the update requests or the delete requests that were performed by the batch job. The backout can be requested in any of the following ways:

- An in-progress unit of recovery requires a backout.
- DFSMStvs restarts.
- An operator issues the access method services SHCDS RETRY command.
- An automatic retry of a shunted transaction occurs.
- A resource manager issues a backout request for a unit of recovery that had been in an in-doubt state.

To prevent damage to the data set, DFSMStvs defers the decision to back out a specific record to an installation-provided exit that is a batch override exit. This is an optional exit that DFSMStvs calls. It uses the exit to back out a unit of recovery involving a data set that a PERMITNONRLSUPDATE command impacts. DFSMStvs calls the exit once for each affected undo log record in the data set. The purpose of the exit is to return to DFSMStvs an indication of whether or not the undo log record should be applied. The input is an undo log record and a data set name. The output is a Boolean response of whether or not to do the backout.

Recommendation: Although the exit can perform other processing, it should *not* attempt to update any recoverable resources.

If you do not provide the installation exit and DFSMStvs encounters backouts that a PERMITNONRLSUPDATE command impacts, DFSMStvs issues a message. It does not apply the backout records. Use the access method services SHCDS RETRY or PURGE command to clean up the backout records.

Requirements: In a DFSMStvs environment, you must quiesce the data set, as Appendix A, “[Quiescing a data set](#),” on page 105 describes. In addition, you must issue the PERMITNONRLSUPDATE command because a failure could cause DFSMStvs to restart.

Recommendation: Be careful about quiesce with a path name; always use a base name.

The exit receives control in the following environment:

| Table 7. Installation exit environment and state | |
|--|------------------------------|
| Environment | State |
| Interrupts | Enabled |
| State and key | Problem program state, key 8 |

| <i>Table 7. Installation exit environment and state (continued)</i> | |
|---|---|
| Environment | State |
| ASC mode | Home address space=Primary address space=Secondary address space , RLS address space |
| AMODE, RMODE | No restrictions |
| Locks | None held |
| Reentrancy | The exit must be reentrant |
| Registers at entry | <p>DFSMSStvs saves the registers before calling the exit, and DFSMSStvs restores registers upon return.</p> <p>Reg 0 Not applicable</p> <p>Reg 1 Points to IGWUNLR (in key 8 storage)</p> <p>Reg 2 Points to an area to be used as an autodata area (in key 8 storage)</p> <p>Reg 3 Length of the autodata area</p> <p>Reg 4 - 13 Not applicable</p> <p>Reg 14 Return address</p> <p>Reg 15 Entry point for exit</p> |
| Registers on return | <p>Reg 0 - 14 Not applicable</p> <p>Reg 15 Return code</p> <p>0 Do not back out this record</p> <p>4 Back out this record</p> |
| Serialization requirements | None |

Restrictions:

1. The exit does not receive a save area and must not attempt to use the value in register 13 or the exit abends. The storage pointed to by register 13 is in a different key than the key in which the exit receives control. As a result, the exit should not attempt to use register 13 to save or restore the registers. It is not necessary for the exit to restore the registers before returning. This is because it receives control through the SYNCHX (SVC 12) exit, and the SYNCHX exit restores the registers.
2. It is difficult for the exit to obtain an autodata area because it runs in key 8 problem state, which is not authorized. This limits the subpools it can use to those subpools that obtain storage in the TCB key. The TCB key is normally key 5, which prevents the exit from referring to the storage. Therefore, the exit receives passed a pointer to an area that can be used as an autodata area in register 2. The length of the autodata area is in register 3; and is normally 8192. Use the value that is in register 3 rather than the hard-coded length.

The name of this exit *must* be IGW8PNRU. DFSMStvs loads the module, which must reside in LINKLIB or LPALIB. If the load fails (for example, no exit is found), DFSMStvs issues a message. If the following conditions exist, DFSMStvs shunts the unit of recovery:

- No exit is found.
- One of the data sets was accessed through PERMITNONRLSUPDATE.

To fix a code error or enhance the function of the exit, restart DFSMStvs to enable the new exit. The exit must be loadable from any system that might perform peer recovery for another system. The exit can issue SVC instructions.

DFSMStvs establishes an ESTAE recovery environment before calling the exit to protect the VSAM RLS address space from failures in the exit. If the exit fails or an attempt to invoke it fails, DFSMStvs shunts the unit of recovery. DFSMStvs provides a dump and disables the exit until the next DFSMStvs restart but does not recycle the server. If the exit abnormally ended, the abend might result in a dump with a title, as in this example:

```
DUMP  TITLE=COMPID=?????,CSECT=????????+FFFF,DATE=????????,MAINT
      ID=????????,ABND=0C4,RC=00000000,RSN=00000004
```

If this problem occurs, examine the dump to investigate why the exit abnormally ended.

Appendix C. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation
Site Counsel
2455 South Road*

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Glossary

This glossary defines technical terms and abbreviations. If you do not find the term you are looking for, refer to the index of the appropriate DFSMS manual.

This glossary includes terms and definitions from the following sources:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). You can purchase copies from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. The symbol (A) after a definition identifies it as a definition from this source.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). The symbol (I) after a definition identifies it as a definition from the published part of this vocabulary. The symbol (T) after a definition identifies it as a definition from a draft international standard, committee draft, or working paper that ISO/IEC JTC1/SC1 is developing, indicating that the participating National Bodies of SC1 have not yet reached final agreement on the definition.
- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

The following cross-reference is used in this glossary:

See

Refers to a preferred term, a synonym, or a term that is the expanded form of an abbreviation or acronym.

See also

Refers to a related term.

Contrast with

Refers to a contrasting term.

A

access method control block (ACB)

A control block that links an application program to VSAM or VTAM programs.

access method services

A multifunction service program that manages VSAM and non-VSAM data sets. Access method services provides the following services:

- Define and allocate space for data sets and catalogs
- Convert indexed-sequential data sets to key-sequenced data sets
- Modify data set attributes in the catalog
- Reorganize data sets
- Facilitate data portability among operating systems
- Create backup copies of data sets
- Assist in making inaccessible data sets accessible
- List the records of data sets and catalogs
- Define and build alternate indexes

ACID transaction

A transaction involving multiple resource managers using the two-phase commit process to ensure ACID (atomic, consistent, isolated, and durable) properties.

- **Atomic:** When an application changes data in multiple resource managers as a single transaction, and all of the changes are accomplished through a single commit request by a syncpoint manager, the transaction is called atomic. If the transaction is successful, all the changes will be committed. If any piece of the transaction is not successful, then all of the changes will be backed out. An

atomic instant occurs when the syncpoint manager in a two-phase commit process logs a commit record for the transaction.

- **Consistent:** Applications involved in an ACID transaction must be written to maintain a consistent view of data. The transaction either makes valid changes to data or returns all the data to its state before the transaction was started.
- **Isolated:** Databases involved in an ACID transaction isolate the updates to their data so that only the application changing the data knows about the individual update requests until the transaction is complete.
- **Durable:** Databases involved in an ACID transaction ensure that the data is persistent, both before and after the transaction, regardless of success or failure.

ACS routine

See *automatic class selection (ACS) routine*.

activity keypoint (AKP)

In DFSMStvs, a record of task-entry status in the system log made on a periodic basis to facilitate the identification of transaction backout information during restart. In the event of an uncontrolled shutdown and subsequent restart, activity keypoints can shorten the process of backward scanning through the system log.

activity keypoint interval

The number of logging operations that the system logger performs between keypoints.

addressed-direct access

In VSAM, the retrieval or storage of a data record identified by its relative byte address (RBA), independent of the record's location relative to the previously retrieved or stored record.

addressed-sequential access

In VSAM, the retrieval or storage of a data record in its entry sequence relative to the previously retrieved or stored record.

addressing mode (AMODE)

An attribute of an entry point in a program that identifies the addressing range in virtual storage that the module is capable of addressing. In 24-bit addressing mode, only 24-bit addresses can be used.

AKP

See *activity keypoint*.

alias

An alternative name for a catalog, a non-VSAM data set, or a member of a partitioned data set (PDS) or partitioned data set extended (PDSE).

alias entry

An entry that relates an alias to the real entry name of a user catalog or non-VSAM data set.

allocation

Generically, the entire process of obtaining a volume and unit of external storage, and setting aside space on that storage for a data set.

The process of connecting a program to a data set or devices.

alternate index

A key-sequenced data set that contains index entries organized by the alternate keys of its associated base data records. It provides an alternate means of locating records in the data component of a cluster on which the alternate index is based.

alternate key

One or more characters within a data record used to identify the data record or to control its use. Unlike the primary key, the alternate key can identify more than one data record. An alternate key is used to build an alternate index or to locate one or more base data records through an alternate index. See also *generic key*, *key*, and *key field*.

application owning region

A CICS address space whose primary purpose is to manage application programs.

application

The use to which an access method is put or the end result that it serves, contrasted to the internal operation of the access method.

ARM

See *automatic restart manager*.

atomic

Pertaining to a transaction's changes to the state of resources: either all changes happen or none happen. It would not be possible for some updates to be made but for others to fail and still maintain data integrity. The process of making final changes to the data is *committing*.

automatic class selection (ACS) routine

A procedural set of ACS language statements. Based on a set of input variables, the ACS language statements generate the name of a predefined SMS class, or a list of names of predefined storage groups, for an MVS data set.

automatic restart manager (ARM)

A z/OS recovery function that can automatically restart a batch job, started task, or abstract resource after it ends unexpectedly or after the system on which it is running goes down unexpectedly.

B**backout**

A request to remove all changes to resources since the last commit or backout or for the first unit of recovery, since the beginning of the application. Backout is also called rollback or abort.

backout log

See *undo log*.

binder

The DFSMS program that processes the output of language translators and compilers into an executable program (load module or program object). It replaces the linkage editor and batch loader in z/OS.

blocking

The process of combining two or more records into one block.

block size

The number of records, words, or characters in a block; usually specified in bytes.

C**CA**

See *control area*.

catalog

A data set that contains extensive information required to locate other data sets, to allocate and deallocate storage space, to verify the access authority of a program or operator, and to accumulate data set usage statistics. (A) (I)

central processor complex (CPC)

A physical collection of hardware that consists of main storage, one or more central processors, timers, and channels.

CI

See *control interval*.

CICS

Customer Information Control System.

CICSVR

Customer Information Control System VSAM Recovery, a forward recovery utility, which can perform forward recovery for DFSMSdfs and others as well as for CICS.

class

See *SMS class*.

cluster

A data component and an index component in a VSAM key-sequenced data set; or a data component alone in a VSAM entry-sequenced data set.

commit

A request to make all changes to recoverable resources permanent since the last commit or backout or, for the first unit of recovery, since the beginning of the application.

component

A named, cataloged collection of stored records. A component, the lowest member of the hierarchy of data structures that can be cataloged, contains no named subsets.

compress

(1) To reduce the amount of storage required for a given data set by having the system replace identical words or phrases with a shorter token associated with the word or phrase. (2) To reclaim the unused and unavailable space in a partitioned data set that results from deleting or modifying members by moving all unused space to the end of the data set.

compressed format data set

A type of extended format data set created in a data format which supports record level compression.

configuration

The arrangement of a computer system as defined by the characteristics of its functional units.

See *SMS configuration*.

consistent read

A level of read integrity that VSAM RLS obtains for a share lock on the record that is accessed by a GET or POINT request. Consistent read ensures that the reader does not see an uncommitted change made by another transaction.

consistent read explicit

A level of read integrity that is the same as *consistent read*, except that VSAM RLS keeps the share lock on the record until the end of the transaction. This option is available only to CICS transactions and to DFSMStvs. VSAM does not recognize the end of the transaction for usage other than by CICS or DFSMStvs. This capability is often referred to as repeatable read.

context

Sometimes called a work context, a context is a representation of a work request, or part of a work request, in an application. A context might have a series of units of recovery associated with it. See also *native context* and *privately managed context*.

control area (CA)

(1) A group of control intervals used as a unit for formatting a data set before adding records to it.

(2) In a key-sequenced data set, the set of control intervals, pointed to by a sequence-set index record, that is used for distributing free space and for placing a sequence-set index record adjacent to its data.

control blocks in common (CBIC)

A facility that allows a user to open a VSAM data set so the VSAM control blocks are placed in the common service area (CSA) of the MVS operating system. This provides the capability for multiple memory accesses to a single VSAM control structure for the same VSAM data set.

control interval (CI)

A fixed-length area of auxiliary storage space in which VSAM stores records. It is the unit of information (an integer multiple of block size) transmitted to or from auxiliary storage by VSAM.

control interval definition field (CIDF)

In VSAM, the 4 bytes at the end of a control interval that contain the displacement from the beginning of the control interval to the start of the free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control program

A routine, usually part of an operating system, that aids in controlling the operations and managing the resources of a computer system.

control unit

A hardware device that controls the reading, writing, or displaying of data at one or more input/output devices. See also *storage control*.

cross memory

A synchronous method of communication between address spaces.

coupling facility (CF)

The hardware that provides high-speed caching, list processing, and locking functions in a Parallel Sysplex.

coupling facility (CF) cache structure

The CF hardware that provides a data cache.

coupling facility (CF) lock structure

The CF hardware that supports Parallel Sysplex-wide locking.

CPC

See central processor complex.

D**data class**

A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

data extent block (DEB)

A control block that describes the physical attributes of the data set.

Data Facility Storage Management Subsystem (DFSMS)

An operating environment that helps automate and centralize the management of storage. To manage storage, SMS provides the storage administrator with control over data class, storage class, management class, storage group, and automatic class selection routine definitions.

Data Facility Storage Management Subsystem data facility product (DFSMSdfp)

A DFSMS functional component or base element of z/OS that provides functions for storage management, data management, program management, device management, and distributed data access.

Data Facility Storage Management Subsystem data set services (DFSMSdss)

A DFSMS functional component or base element of z/OS that is used to copy, move, dump, and restore data sets and volumes.

Data Facility Storage Management Subsystem Transactional VSAM Services (DFSMSStvs)

An IBM licensed program for running batch VSAM processing concurrently with CICS online transactions. DFSMSStvs users can run multiple batch jobs and online transactions against VSAM data, in data sets defined as recoverable, with concurrent updates. DFSMSStvs is a licensed component of DFSMS.

data record

A collection of items of information from the standpoint of its use in an application, as a user supplies it to the system storage. Contrast with *index record*.

data security

Prevention of access to or use of data or programs without authorization. As used in this publication, the safety of data from unauthorized use, theft, or purposeful destruction.

data set control block (DSCB)

A control block in the VTOC that describes data set characteristics.

data synchronization

The process by which the system ensures that data previously given to the system via WRITE, CHECK, PUT, and PUTX macros is written to some form of nonvolatile storage.

device number

The reference number assigned to any external device.

DFSMS

See *Data Facility Storage Management Subsystem*.

DFSMSdftp

See *Data Facility Storage Management Subsystem data facility product*.

DFSMSdss

See *Data Facility Storage Management Subsystem data set services*.

DFSMSStvs

See *Data Facility Storage Management Subsystem Transactional VSAM Services*.

dictionary

A table that associates words, phrases, or data patterns to shorter tokens. The tokens are used to replace the associated words, phrases, or data patterns when a data set is compressed.

direct access

The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data. See also *addressed-direct access*.

direct access device space management (DADSM)

A DFP component used to control space allocation and deallocation on DASD.

direct data set

A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address according to the beginning of the data set. Normally accessed via BDAM.

directly-allocated printer

A printer that is allocated to the application program.

dynamic buffering

A user-specified option that requests that the system handle acquisition, assignment, and release of buffers.

E**entry-sequenced data set**

A data set whose records are loaded without respect to their contents, and whose relative byte addresses (RBAs) cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

ESA

See *Enterprise Systems Architecture*.

exclusive control

A way of preventing multiple write-add BDAM requests from updating the same dummy record or writing over the same available space on a track. When specified by the user, the exclusive control lock requests that the system prevent the data block that is about to be read from being modified by other requests; it is specified in a read macro and released in a write or relex macro. When a write-add request is about to be processed, the system automatically gets exclusive control of either the data set or the track.

extended format

The format of a data set that has a data set name type (DSNTYPE) of EXTENDED, for example, extended format and extended key-sequenced data sets. Data sets in extended format can be striped or compressed. Data in an extended format VSAM KSDS can be compressed.

extended format data set

A sequential data set that is structured logically the same as a physical sequential data set but that is stored in a different physical format. Extended format data sets consist of one or more stripes and can take advantage of the sequential data striping access technique. See also *striping* and *stripe*.

extent

A continuous space on a DASD volume occupied by a data set or portion of a data set.

F**field**

In a record or control block, a specified area used for a particular category of data or control information.

file-owning region (FOR)

A data-owning region, a CICS address space whose primary purpose is to manage files and databases.

file system

In the z/OS UNIX hierarchical file system environment, the collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk. See also *hierarchical file system data set*.

FOR

See *file-owning region*.

forgotten

The state of a unit of recovery that occurs when the unit of recovery has completed and RRS has deleted its log records.

format-D

ASCII variable-length records.

format-DB

ASCII variable-length, blocked records.

format-DBS

ASCII variable-length, blocked spanned records.

format-DS

ASCII variable-length, spanned records.

format-F

Fixed-length records.

format-FB

Fixed-length, blocked records.

format-FBS

Fixed-length, blocked, standard records.

format-FS

Fixed-length, standard records.

format-U

Undefined-length records.

format-V

Variable-length records.

format-VB

Variable-length, blocked records.

format-VBS

Variable-length, blocked, spanned records.

format-VS

Variable-length, spanned records.

Forward recoverable data set

A data set that was defined with the LOG(ALL) attribute option.

forward recovery

A process used to recover a lost data set. The data is recovered from a backup copy and all the changes that were made after the backup copy was taken are applied. The forward recovery process requires a log of the changes made to a data set, together with a date and time stamp. The log of changes is called the forward recovery log.

forward recovery log

A log that contains copies of records after they were changed. The forward recovery log records are used by forward recovery programs and products such as CICS VSAM Recovery (CICSVR) to reconstruct the data set in the event of hardware or software damage to the data set.

free space

Space reserved within the control intervals of a key-sequenced data set for inserting new records into the data set in key sequence or for lengthening records already there; also, whole control intervals reserved in a control area for the same purpose.

G**gigabyte**

1 073 741 824 bytes.

global shared resources (GSR)

. Indicates use of a global resource pool.

GSR

See *global shared resources*.

H**header label**

An internal label, immediately preceding the first record of a file, that identifies the file and contains data used in file control.

The label or data set label that precedes the data records on a unit of recording medium.

hierarchical file system

A POSIX-compliant file system, which is a collection of files and directories organized in a hierarchical structure, that can be accessed using z/OS UNIX System Services. HFS enables an application written in a high-level language to create, store, retrieve, and manipulate data on a storage device. See also *file system*.

index record

A collection of data-record pointers retrieved and stored together. Contrast with *index record*

instance

(1) The code and control blocks that represent access to VSAM data sets through DFSMSHVS.

An instance of DFSMSHVS starts when DFSMSHVS is initialized as part of SMSVSAM address space initialization or enabled by operator command. The instance ends when DFSMSHVS enters a quiesced or disabled state or when the SMSVSAM address space ends.

(2) A peer recovery instance of DFSMSHVS serves to recover from the failure of some other DFSMSHVS instance that ran on another system within a sysplex when the system on which the other DFSMSHVS instance was running failed. The peer recovery instance shares the SMSVSAM address space, and certain control blocks, with a "native" DFSMSHVS instance. Automatic restart manager (ARM) can start a peer recovery instance automatically when a system in a sysplex fails. A peer recovery instance can also be started manually by operator command. The instance ends when it completes its peer recovery process, when it is stopped by operator command and enters a quiesced state, or when the SMSVSAM address space ends.

in-backout

The state of a unit of recovery when one or more resource managers reply negatively to a commit request. The syncpoint manager tells each resource manager to back out the changes. The resources are returned to the values they had before the unit of recovery was processed. When all the resource managers have backed out the changes, the syncpoint manager notifies the application.

in-commit

The state of a unit of recovery when all resource managers reply positively to a commit request. The syncpoint manager tells each resource manager to make its changes permanent. When all resource managers have made the changes, the syncpoint manager notifies the application.

in-completion

The state of a unit of recovery when any enabled completion exit routines run. After this phase completes, RRS passes a return code to the application indicating that the changes have been committed or backed out.

in-doubt

For a distributed request, the state of a unit of recovery on the originating system from the end of the prepare phase of the two-phase commit until the distributed syncpoint resource manager (DSRM) returns a commit or backout request.

in-end

The state of a unit of recovery when the resource managers have responded to the syncpoint manager that commit or backout is complete. The unit of recovery is logically complete.

in-flight

The state of a unit of recovery when an application accesses protected resources. The resource managers express interest in the unit of recovery.

in-forget

The state of a unit of recovery for a distributed request. The unit of recovery has completed, but RRS is waiting for the server distributed syncpoint resource manager (SDSRM) to indicate how to process the log records for the unit of recovery.

in-only-agent

The state of a unit of recovery when only one resource manager has expressed an interest in the unit of recovery. RRS invokes the ONLY_AGENT exit routine to tell the resource manager to process the commit immediately.

in-prepare

The state of a unit of recovery when the application has issued a commit request and the syncpoint manager tells each resource manager to prepare its resources for commit or backout.

in-reset

The state of a unit of recovery before an application program has used any protected resources.

in-state-check

The state of a unit of recovery when the application has issued a commit request and the resource managers check if their resources are in the correct state.

K**key-sequenced data set (KSDS)**

A VSAM data set whose records are loaded in ascending key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in key sequence because of free space allocated in the data set. Relative byte addresses of records can change because of control interval or control area splits.

keyed-sequential access

In VSAM, the retrieval or storage of a data record in its key or relative-record sequence, relative to the previously retrieved or stored record as defined by the sequence set of an index.

kilobyte

1024 bytes.

L**library**

Synonym for partitioned data set. See *partitioned data set*.

linear data set (LDS)

A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

load module

The output of the linkage editor; a program in a format ready to load into virtual storage for execution. Contrast with program object.

local shared resources (LSR)

Resources in the local resource pool.

locate mode

A transmittal mode in which a pointer to a record is provided instead of moving the record. Contrast with *move mode*.

log of logs

A log that DFSMStvs and CICS write to provide information to forward recovery programs such as CICS VSAM Recovery (CICSVR). The log of logs is a form of user journal that contains copies of the tie-up records that DFSMStvs or CICS has written to forward recovery logs. This log provides a summary of

which recoverable VSAM data sets that DFSMStvs or CICS has used, when they were used, and to which log stream the forward recovery log records were written.

If you have a forward recovery product that can utilize the log of logs, ensure that all CICS regions that share the recoverable data sets write to the same log-of-logs log stream.

log stream

A log stream is a collection of data in log blocks that reside in the coupling facility or on DASD.

log tail

In DFSMStvs, the oldest log record of interest. Log tail deletion is the process of deleting unneeded records that are older than the oldest record of interest to DFSMStvs.

log trimming

Removal of records that are no longer required from the DFSMStvs primary system log or secondary system log.

LSR

See *local shared resources*.

M

management class

A collection of management attributes, defined by the storage administrator, used to control the release of allocated but unused space; to control the retention, migration, and back up of data sets; to control the retention and back up of aggregate groups, and to control the retention, back up, and class transition of objects.

member

A partition of a partitioned data set or PDSE.

move mode

A transmittal mode in which the record to be processed is moved into a user work area.

MVS

Multiple Virtual Storage.

N

native context

The automatically occurring context of a work request. A native context is associated with a single task. This context always exists.

non-VSAM data set

A data set allocated and accessed using one of the following methods: BDAM, BPAM, BISAM, BSAM, QSAM, QISAM.

nonrecoverable data set

A data set for which no changes are logged because its LOG parameter is either undefined or set to NONE. Neither backout nor forward recovery is provided for a nonrecoverable data set.

nonshared resources

A data set that does not use shared resources.

NSR

See *nonshared resources*.

O

object

A named byte stream having no specific format or record orientation.

z/OS UNIX System Services (z/OS UNIX)

The set of functions provided by the SHELL and UTILITIES, kernel, debugger, file system, C/C++ Run-Time Library, Language Environment, and other elements of the z/OS operating system that allow users to write and run application programs that conform to UNIX standards.

operand

Information entered with a command name to define the data on which a command operates and to control the execution of the command.

optimum block size

For non-VSAM data sets, optimum block size represents the block size that would result in the smallest amount of space utilization on a device, taking into consideration record length and device characteristics.

P**Parallel Sysplex**

A sysplex that uses one or more coupling facilities.

partitioned data set (PDS)

A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE)

An system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

path

A named, logical entity composed of one or more clusters (an alternate index and its base cluster, for example).

PDS directory

A set of records in a partitioned data set (PDS) used to relate member names to their locations on a DASD volume.

peer recovery

A recovery process that occurs when an application fails. Peer users can perform recovery and clean up resources.

peer recovery instance

See *instance*.

pointer

An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs.

primary key

One or more characters within a data record used to identify the data record or control its use. A primary key must be unique.

primary space allocation

Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

primary system log

See *undo log*.

privately managed context

A context created and owned by a resource manager. The resource manager can switch a privately managed context from one task to another. Privately managed contexts are usually used by a resource manager that is also a work manager, like IMS. This sort of work manager can accept and manage transactions, or other kinds of work, from outside the system.

program library

A type of PDSE which contains program objects only. A PDSE from which programs are loaded into memory for execution by the operating system.

program object

All or part of a computer program in a form suitable for loading into virtual storage for execution. Program objects are stored in PDSE program libraries and have fewer restrictions than load modules. Program objects are produced by the binder.

protected resource

A local or distributed resource that can be changed in a synchronized manner during processing coordinated by a syncpoint manager, such as RRS. Databases, conversations between two communications managers, or product-specific resources can all be protected resources. A protected resource is also often called a *recoverable resource*.

R

random access

See *direct access*.

record definition field (RDF)

A field stored as part of a stored record segment; it contains the control information required to manage stored record segments within a control interval.

record-level sharing

See *VSAM record-level sharing (VSAM RLS)*.

recoverable data set

A data set that can be recovered using backout or forward recovery processing, defined with the LOG parameter set to UNDO or ALL. See also *protected resource*.

recoverable resource

A data set that can be recovered using commit, backout, or forward recovery processing because its LOG parameter is set to UNDO or ALL.

register

An internal computer component capable of storing a specified amount of data and accepting or transferring this data rapidly.

relative byte address (RBA)

The displacement of a data record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

relative record data set (RRDS)

A type of VSAM data set whose records have fixed or variable lengths, and are accessed by relative record number.

residence mode (RMODE)

The attribute of a load module that identifies where in virtual storage the program will reside (above or below 16 megabytes).

resource

A database, a conversation between two systems, or a product-specific item. A resource can be local (residing on the current system) or distributed (residing on another system). A resource is protected when it can be changed in a synchronized manner.

resource manager (RM)

A subsystem or component, such as CICS, IMS, or DB2, or DFSMSStvs, that manages resources that can be involved in transactions. There are three types of resource managers: work managers, data resource managers, and communication resource managers.

reusable data set

A VSAM data set that can be reused as a work data set, regardless of its old contents. It must not be a base cluster of an alternate index.

RLS

See *VSAM record-level sharing (VSAM RLS)*.

S

scheduling

The ability to request that a task set should be started at a particular interval or on occurrence of a specified program interrupt.

secondary space allocation

Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

secondary system log

See *shunt log*.

security

See *data security*.

sequence checking

The process of verifying the order of a set of records relative to some field's collating sequence.

sequential access

The retrieval or storage of a data record in: its entry sequence, its key sequence, or its relative record number sequence, relative to the previously retrieved or stored record. See also *addressed-sequential access* and *keyed-sequential access*.

sequential data set

A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Contrast with *direct data set*.

service request block (SRB)

A system control block used for dispatching tasks.

shared lock

A lock that several tasks can hold.

shared resources

A set of functions that permit the sharing of a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

shunt

The process of moving failed work or long-running work from the primary system log to the secondary system log. If a unit of work fails, it is removed (shunted) from the primary system log to the secondary system log, pending recovery from the failure.

shunt log

The secondary system log, which contains entries that were shunted to the log when DFSMStvs was unable to finish processing sync points. If a *unit of work* fails, it is removed (shunted) from the primary system log to the secondary system log, pending recovery from the failure.

shunted

The state of a unit of recovery when it has been moved from the primary system log to the secondary system log because of a failed or long-running unit of work.

slot

For a fixed-length relative record data set, the data area addressed by a relative record number, which might contain a record or be empty.

single point of failure

An environment in which one failure can result in simultaneous loss of both the coupling-facility list structure for a log stream and the local storage-buffer copy.

skip-sequential access

Keyed-sequential retrieval or storage of records here and there throughout a data set, skipping automatically to the desired record or collating position for insertion: VSAM scans the sequence set to find a record or a collating position. Valid for processing in ascending sequences only.

SMF

See *System Management Facilities*.

SMS class

A list of attributes that SMS applies to data sets having similar allocation (data class), performance (storage class), or backup and retention (management class) needs.

SMS configuration

A configuration base, Storage Management Subsystem class, group, library, and drive definitions, and ACS routines that the Storage Management Subsystem uses to manage storage.

SMS-managed data set

A data set that has been assigned a storage class.

spanned record

For VSAM, a logical record whose length exceeds control interval length, and as a result, crosses, or spans one or more control interval boundaries within a single control area. For non-VSAM, a spanned record that occupies part or all of more than one block.

staging data set

Staging data sets are allocated by the system logger to safeguard log data when there is an error that leaves the only copy of log data in a volatile configuration.

storage class

A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage control

The component in a storage subsystem that handles interaction between processor channel and storage devices, runs channel commands, and controls storage devices.

storage group

A collection of storage volumes and attributes, defined by the storage administrator. The collections can be a group of DASD volumes or tape volumes, or a group of DASD, optical, or tape volumes treated as a single object storage hierarchy.

Storage Management Subsystem (SMS)

A DFSMS facility used to automate and to centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

store-through caching

A process used by the store-through user in which changed data is written to the cache structure and to permanent storage at the same time and under the same serialization so that at any time, the data in the cache structure matches the data in permanent storage.

stripe

The portion of a striped data set (for example, an extended format data set) that resides on one volume. The records in that portion are not necessarily logically consecutive. The system distributes records among the stripes such that the volumes can be read or written simultaneously to gain better performance.

striping

A software implementation of a disk array that distributes data sets across multiple volumes to improve performance.

system-managed storage

Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications. See also *DFSMS environment*.

system management facilities (SMF)

A component of z/OS that collects input/output (I/O) statistics, provided at the data set and storage class levels, which help you monitor the performance of the direct access storage subsystem.

sync point

An end point during processing of a transaction. A sync point occurs when an update or modification to one or more of the transaction's protected resources is logically complete. A sync point can be either a commit or a backout.

syncpoint manager

A syncpoint manager is a function that coordinates the two-phase commit process for protected resources, so that all changes to data are either committed or backed out. In z/OS, RRS can act as the system level syncpoint manager.

T**tie-up record**

A record that associates each data set after-image record in the log with a file name. You can associate a data set with more than one file with the same data set. When a file is opened, DFSMStvs records the association between the file and the data set as a tie-up record in the forward recovery log. This information is also written to the log of logs. For non-BWO backups, the forward recovery utility uses this tie-up record to apply the log records to the correct data sets.

transaction

A unit of application data processing initiated by a single request. A transaction might involve multiple application programs and might require the initiation of one or more jobs for its execution. In DFSMStvs, a transaction is a unit of work, which consists of one or more logical units of recovery.

transaction ID (TRANSID)

A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

TRANSID

See *transaction ID*.

trimming

See *log trimming*.

two-phase commit

The process used by syncpoint managers and resource managers to coordinate changes in an ACID transaction.

In the first phase of the process, resource managers prepare a set of coordinated changes, but the changes are uncommitted pending the agreement of all the resource managers involved in the transaction. In the second phase, those changes are all committed if the resource managers all agreed to them; or, the changes are all backed out if any of the resource managers failed or disagreed.

Using the two-phase commit process, multiple changes across multiple resource managers can be treated as a single ACID transaction.

U**undo log**

The primary system log, which contains images of changed records as they existed prior to being changed. Backout processing uses the undo log to back out the changes that a transaction made to resources.

unit address

The last two hexadecimal digits of a device address. This identifies the storage control and DAS string, controller, and device to the channel subsystem. Often used interchangeably with control unit address and device address in System/370 mode.

unit of recovery (UR)

A set of changes on one node that is committed or backed out as part of an ACID transaction.

A UR is implicitly started the first time a resource manager touches a protected resource on a node. A UR ends when the two-phase commit process for the ACID transaction changing it completes.

unit of recovery identifier (URID)

Persistent tokens used by RRS to identify a transaction.

unit of work

In DFSMStvs, one or more logical units of recovery that are committed or backed out together as a transaction.

universal character set (UCS)

A printer feature that permits the use of a variety of character arrays. Character sets used for these printers are called UCS images.

update number

For a VSAM spanned record, a binary number in the second RDF of a record segment that indicates how many times the segments of a spanned record should be equal. An inequality indicates a possible error.

user buffering

The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control interval between the work area and direct access storage without intermediary buffering.

V

virtual storage access method (VSAM)

An access method for direct or sequential processing of fixed and variable-length records on direct access storage devices. You can organize the records in a VSAM data set in logical sequence by a key field (key sequence), in the physical sequence in which they are written to the data set (entry sequence), or by relative record numbers.

VSAM

See *virtual storage access method*.

VSAM record-level sharing (VSAM RLS)

An extension to VSAM that provides direct record-level sharing of VSAM data sets from multiple address spaces across multiple systems. Record-level sharing uses the z/OS coupling facility to provide cross-system locking, local buffer invalidation, and cross-system data caching.

VSAM volume data set (VVDS)

A data set that describes the characteristics of VSAM and system-managed data sets residing on a given DASD volume; part of a catalog.

z/OS

A network computing-ready, integrated operating system consisting of more than 50 base elements and integrated optional features delivered as a configured, tested system.

Index

A

- access mode [7](#)
- accessibility
 - contact IBM [111](#)
- accessing data sets [14](#)
- activity keypoint frequency (AKP) [45](#)
- advanced application development [65](#)
- application program
 - coordinating recovery [4](#)
- applications
 - coding [54](#)
- archive log [29](#)
- assistive technologies [111](#)
- atomic
 - change [4](#)
 - definition [1](#)
- automatic restart manager (ARM) [101](#)
- Automatic Restart Manager (ARM) [33](#)

B

- backout
 - log stream [46](#)
 - logging
 - GET UPD [47](#)
 - PUT ADD [47](#)
 - PUT UPD [47](#)
 - logs [15](#)
 - protocols [14](#)
 - records [46](#)
- batch job
 - transaction processing [13](#)

C

- cache structures
 - size [26](#)
- CICS
 - transactional recovery [8](#)
 - with VSAM RLS [6](#)
- CICS log streams [29](#)
- coding applications [54](#)
- cold start [101](#)
- commit [1](#)
- commit processing
 - phases of
 - commit [16](#)
 - prepare [16](#)
 - two-phase [15](#)
- commit, two-phase [14](#)
- console messages and dumps [96](#)
- contact
 - z/OS [111](#)
- context
 - native [18](#)
 - privately managed [18](#)

- context (*continued*)
 - services [17](#)
- coordination of recovery [4](#)
- coupling facilities
 - contents [23](#)
 - internal [23](#)
 - links [26](#)
 - number [22](#)
 - overview [22](#)
 - size [24](#)
 - standalone [23](#)
 - volatile or nonvolatile [23](#)
- coupling facility
 - defining structures for log streams [41](#)
 - log stream example [43](#)
 - status [97](#)
 - using [36](#)
- coupling-facility planning [22](#)

D

- DASD log data sets [32](#)
- DASD staging data sets [31](#)
- DASD-only log streams [30](#)
- data sets
 - deleting [63](#)
 - nonrecoverable [10](#)
 - recoverable [13](#)
 - renaming [63](#)
- deadlocks [61](#)
- defining transactions [54](#)
- delayed UR state log [29](#)
- DFSMSHvs
 - accessing data sets [57](#)
 - applications
 - exploiting [53](#)
 - intolerant [53](#)
 - redesigning [53](#)
 - tolerant [53](#)
 - backout logging [46](#)
 - coding your application program to use [53](#)
 - commit and backout [58](#)
 - defining resources [35](#)
 - end of task, effects of [59](#)
 - logging [35](#), [41](#)
 - PARMLIB
 - examples [38](#)
 - recommendations and requirements [38](#)
 - record locking [60](#)
 - restarts [100](#)
- DFSMSHvs environment [1](#)
- direct access storage devices (DASD) [57](#)
- distributed resource recovery [20](#)
- distributed units of recovery [20](#)

E

enabling VSAM RLS [9](#)
execution mode requirements [12](#)

F

false lock contention [61](#)
forward recoverable data set [2](#)
forward recovery [2](#)
forward recovery log [2](#), [15](#)
forward recovery log streams [29](#), [47](#)
forward recovery logging [47](#)
forward recovery logs [29](#)
forward recovery operation planning [33](#)
functional recovery routine (FRR) [55](#)

G

global resource serialization (GRS) [97](#)

I

Integrated Cluster Bus (ICB) [26](#)
Internal Coupling Channel (ICC) [26](#)

K

keyboard
 navigation [111](#)
 PF keys [111](#)
 shortcut keys [111](#)

L

list structures
 size [26](#)
lock structure
 size [25](#)
locking
 non-RLS [11](#)
 retained locks [11](#)
 serializing resources [14](#)
locking, resource [14](#)
log
 backout [15](#), [16](#)
 forward recovery [15](#)
 system
 primary [2](#)
 secondary [2](#)
log forward recovery [2](#)
log log of logs [2](#)
log of logs [2](#), [29](#), [49](#)
log shunt [3](#)
log stream
 forward recovery [47](#)
log streams
 access to [50](#)
 CICS [29](#)
 DASD log data sets [32](#)
 DASD staging data sets [31](#)
 DASD-only [30](#)
 definition examples [45](#)

log streams (*continued*)
 DFSMSHVS [28](#)
 forward recovery logs [29](#)
 log of logs [29](#)
 overview [28](#)
 primary system log [2](#), [28](#)
 RRS [28](#)
 secondary system log [2](#), [28](#)
 shunt [3](#)
 shunt log [46](#)
 size [31](#), [32](#)
 structures [30](#)
 system log names [45](#)
 undo log [3](#), [46](#)
log structures [43](#)
log tail deletion [45](#)
log trimming [32](#)
LOG(ALL) [8](#), [41](#)
LOG(NONE) [8](#), [41](#)
LOG(UNDO) [8](#), [41](#)
logging
 problems
 categories [95](#)
 collecting diagnosis information [96](#)
 recovery from [99](#)
 resource recovery [14](#)
logging flow [27](#)
logging updates [41](#)

M

main UR state log [29](#)
multitasking [66](#)

N

native contexts [18](#)
navigation
 keyboard [111](#)
non-CICS applications [17](#)
non-RLS
 access to VSAM data sets [10](#)
 access with retained locks [11](#)
 locking [11](#)
nonrecoverable data set [2](#)
nonrecoverable data sets
 read and write sharing [10](#)

O

offloading [45](#)
Overview of DFSMSHVS [13](#)

P

parallel sysplex environment [35](#)
peer recovery
 interference [103](#)
 SMSVSAM failures during [102](#)
 starting [102](#)
 system failures during [102](#)
performance delays [41](#)
planning for DFSMSHVS

- planning for DFSMStvs (*continued*)
 - coupling-facility planning [22](#)
 - overview [21](#)
 - planning tasks [21](#)
 - processor capacity [26](#)
 - software configuration [27](#)
 - system logger [27](#)
 - VSAM operations planning [32](#)
- planning tasks [21](#)
- positioning states
 - new [63](#)
 - no [63](#)
 - u [63](#)
 - undefined [63](#)
 - yes [63](#)
- primary system log [2, 28](#)
- privately managed contexts [18](#)
- processing restrictions
 - alternate indexes [55](#)
 - defer processing [55](#)
 - DEFINE parameters [55](#)
 - exits [55](#)
 - global resource serialization (GRS) [56](#)
 - load mode [55](#)
 - locking [55](#)
 - positioning [55](#)
 - request environment [55](#)
 - SHAREOPTIONS [55](#)
 - sharing [55](#)
- processor capacity planning [26](#)

Q

- quiescing [96](#)

R

- random access [63](#)
- read and write sharing
 - nonrecoverable data sets [10](#)
- read integrity options
 - CR (consistent read) [60](#)
 - CRE (consistent read explicit) [60](#)
 - NRI (no read integrity) [60](#)
 - repeatable read [60](#)
- Read integrity options [9](#)
- read sharing integrity [9](#)
- Read-sharing integrity [9](#)
- record locking protocols [13](#)
- record management request [15](#)
- record management requests [66](#)
- record-level sharing [5, 7, 9, 11, 12](#)
- recoverable data set [2, 5, 6, 13](#)
- recoverable data sets
 - CICS [17](#)
 - read sharing [9](#)
 - VSAM RLS [17](#)
- recoverable VSAM data sets
 - access to [14](#)
- recovery
 - application program [4](#)
 - backout [1](#)
 - coordination [4](#)

- recovery (*continued*)
 - forward recoverable data set [2](#)
 - forward recovery [2](#)
 - forward recovery log [2](#)
 - log of logs [2](#)
 - primary system log [2](#)
 - resource locking [4, 5](#)
 - resource manager [4](#)
 - resource recovery logging [4, 5](#)
 - syncpoint manager [4](#)
 - transactional [4, 13](#)
 - two-phase commit processing [3–5](#)
 - undo log [3](#)
- recovery coordination [13](#)
- recovery procedures [32](#)
- recovery tracking [46](#)
- reorganization [33](#)
- repeatable read option [13](#)
- requirements
 - CF level 2 microcode [22](#)
 - lost locks [107](#)
 - retained locks [107](#)
- resource locking [4, 5, 14](#)
- resource manager
 - coordinating recovery [4](#)
 - DB2 [17](#)
 - definition [2](#)
 - IMSDB [17](#)
- resource manager data log [28](#)
- resource recovery logging [4, 5, 14](#)
- restart log [29](#)
- restarting applications [62](#)
- restrictions
 - CICS and DFSMStvs [17](#)
 - DFSMStvs [56](#)
 - DFSMStvs processing [55](#)
 - processing [55](#)
 - RLS [56](#)
 - understanding [54](#)
- retained locks [11, 60](#)
- RRMS [17](#)
- RRS [17](#)
- RRS log streams
 - archive log [29](#)
 - delayed UR state log [29](#)
 - main UR state log [29](#)
 - overview [28](#)
 - resource manager data log [28](#)
 - restart log [29](#)

S

- secondary system log [2, 28](#)
- sequential access [63](#)
- service request block (SRB) [56](#)
- share options [11](#)
- shortcut keys [111](#)
- shunt log [3, 35, 46](#)
- shunted transactions
 - monitoring [64](#)
 - retrying [64](#)
- single-mode Intersystem Coupling (ISC) link [26](#)
- SMF and RMF statistics [98](#)
- SMSVSAM

- SMSVSAM (*continued*)
 - address space failure [100](#)
- software configuration [27](#)
- spanned records [57](#)
- staging data sets [37](#)
- storage types
 - primary [44](#)
 - secondary [44](#)
 - tertiary [44](#)
- summary of changes [xv](#)
- sync point [3](#)
- syncpoint manager
 - coordinating recovery [4](#)
 - definition [3](#)
- system log
 - storage [45](#)
- system log streams
 - names [45](#)
- system logger
 - availability [95](#)
 - list structures [24](#)
 - logging flow [27](#)
 - planning [27](#)

T

- task control block (TCB) [56](#), [66](#)
- trademarks [116](#)
- transaction processing
 - batch job [13](#)
 - description [3](#)
 - DFSMSvts and RLS tasks [14](#)
 - overview [1](#)
 - transactional recovery [4](#)
- transactional recovery
 - CICS [8](#)
 - CICS file-control program [17](#)
 - coordinating recovery [4](#)
 - description [4](#)
 - nonrecoverable data sets [10](#)
 - overview [1](#)
 - programs
 - application program [4](#)
 - resource manager [4](#)
 - syncpoint manager [4](#)
 - resource locking [4](#), [5](#)
 - resource recovery logging [5](#)
 - two-phase commit processing [5](#)
- transactions
 - defining [54](#)
- two-phase commit
 - processing [3](#), [5](#)
- two-phase commit processing [4](#), [5](#)

U

- undo log
 - backout logging [46](#)
- unit of recovery [3](#), [17](#)
- unit of work [3](#)
- unit-of-recovery states
 - in-backout state [19](#)
 - in-commit [19](#)

- unit-of-recovery states (*continued*)
 - in-flight [19](#)
 - in-prepare [19](#)
 - in-reset [19](#)
- units of recovery
 - in-doubt [46](#)
 - long-running [46](#)
- user interface
 - ISPF [111](#)
 - TSO/E [111](#)

V

- VSAM data sets
 - non-RLS access [10](#)
 - RLS access [10](#)
 - share options [11](#)
- VSAM operations
 - ERASE [47](#)
 - PUT [47](#)
- VSAM operations planning
 - Automatic Restart Manager (ARM) [33](#)
 - forward recovery operation [33](#)
 - overview [32](#)
 - recovery procedures [32](#)
 - reorganization [33](#)
- VSAM record management requests [15](#)
- VSAM RLS
 - Read integrity options [9](#)
- VSAM RLS (record-level sharing)
 - access mode
 - GSR [7](#)
 - LSR [7](#)
 - NSR [7](#)
 - and CICS [5](#)
 - data set types
 - ESDS [6](#), [57](#)
 - KSDS [6](#), [57](#)
 - RRDS [6](#), [57](#)
 - VRRDS [6](#)
 - enabling
 - MACRF=RLS [9](#)
 - execution mode requirements [12](#)
 - options not supported [12](#)
 - processing [5](#)
 - read integrity options [9](#)
 - read sharing integrity [9](#)
 - recoverable data set [6](#)
 - recoverable data sets [7](#)
 - request [12](#)
 - retained locks [11](#)
 - share options [11](#)
 - SMSVSAM
 - server [9](#)
 - using the coupling facility [5](#)
- VSAM sharing
 - cache structures [24](#)
 - lock structure [24](#)

W

- waits [100](#)



Product Number: 5655-ZOS

SC23-6877-70

