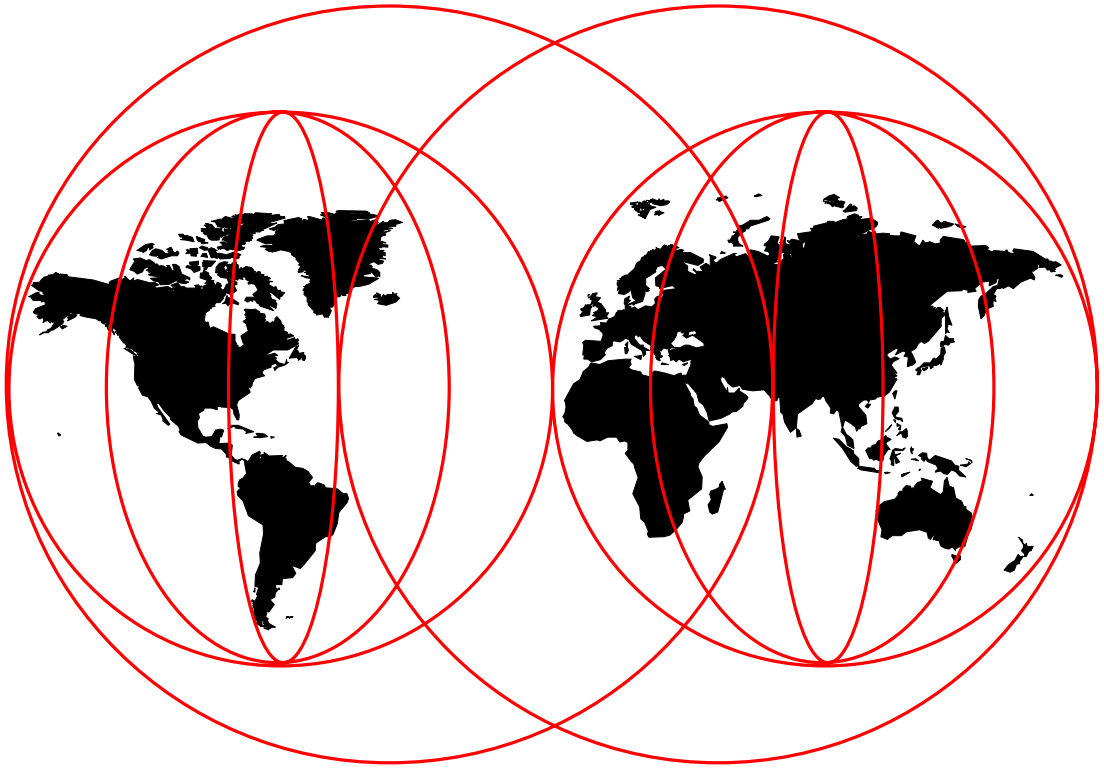
IMS/ESA Version 6 Shared Queues

Geoff Nicholls, Glen Battershell, Jim Boyle, Ulrich Müller
Jennifer Pearcey, Gerhard Schenk, Bill Stillwell



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5088-00

International Technical Support Organization

SG24-5088-00

IMS/ESA Version 6 Shared Queues

July 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 175.

First Edition (July 1998)

This edition applies to the IBM Information Management System (IMS), Transaction and Database Server for System/390, Program Number 5655-158, for use with MVS/ESA SP Version 4 Release 3, OS/390 Version 1 or later Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This redbook describes the IMS Version 6 shared queues feature. Shared queues enable you to spread your transaction processing workload across up to 32 systems in a Parallel Sysplex, thereby increasing your processing capacity and enhancing system availability.

This redbook introduces the architectures available with shared queues in the IMS environment and provides information to help you install, tailor, and configure shared queues in your environment.

Chapter 1 gives an overview of the facilities available with IMS/ESA Version 6 shared queues. Chapter 2 introduces the common queue server, which is the interface between IMS and the shared queues stored on the coupling facility. Several configuration scenarios are presented in Chapter 3. The balance of the redbook details the implementation and operation of shared queues, as well as special considerations for conversational programs, multiple systems coupling, serial transactions, and transactions initiated from APPC and OTMA.

This redbook is especially useful for users of IMS Transaction Manager who plan to distribute their workload across multiple systems in a Parallel Sysplex.

Some knowledge of IMS and transaction processing is assumed.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Geoff Nicholls is an Advisory Systems Engineer at the International Technical Support Organization, San Jose Center. Geoff has a Science degree from the University of Melbourne, Australia, where he majored in Computer Science. He worked as an application programmer and database administrator for several insurance companies before specializing in database and transaction management systems with Unisys and IBM. Since joining IBM in 1989, Geoff has worked extensively with IMS customers in Australia and throughout Asia.

Glen Battershell is an IMS Systems Programmer for IBM Global Services, Australia. Glen has 11 years of experience in the information technology industry, the last 3 focusing on IMS. He holds a Certificate in Computer Programming from Box Hill College, Melbourne, Australia. Glen's areas of expertise include IMS Transaction Manager and automated operations.

Jim Boyle is an Advisory Systems Engineer in IBM Australia. He holds a degree in Engineering from the University of Melbourne and has 30 years of IBM experience, including 23 years working as an IMS specialist in Australia, New Zealand, and Asia. Jim installed the first IMS Fast Path accounts in Australasia and has assisted many customers in the design, installation, and performance enhancement of IMS systems. He has been a joint author of technical studies on IMS capacities and several IBM Redbooks on IMS and MQSeries. Jim is a joint inventor of a software technique to enhance the performance of input/output (I/O) subsystems in MVS/ESA.

Ulrich Müller is an Advisory Systems Engineer for the High End Systems Sales product division in IBM Germany. Ulrich has 18 years of experience with DB/DC software in technical support for large systems accounts. He is assigned to the finance branch in Frankfurt to help customers in the finance industry plan and implement their Parallel Sysplex DB/DC projects.

Jennifer Pearcey is an Advisory Technical Specialist in South Africa. Jennifer has six years of experience as an IMS systems programmer, as well as several years with the systems performance team of a large Bank in Johannesburg. Jennifer holds a Bachelor of Science degree with honors in Computer Science from the University of Witwatersrand, Johannesburg. Her areas of expertise include IMS installation, IMS system definition, and IMS Transaction Manager in large banking environments.

Gerhard Schenk is the manager of the database and transactions systems programming department at Allianz Versicherungs-AG (Insurance Group) in Munich, Germany. Gerhard has 15 years of experience in IMS database management and application development fields. He holds a degree in Mathematics from the University of Regensburg in Bavaria. His areas of expertise include IMS systems design and implementation in large corporate environments.

Bill Stillwell is a Consulting Systems Specialist with IBM USA. He has been providing technical support and consulting services to IMS customers for 15 years as a member of the Dallas Systems Center. During that time Bill has developed expertise in application and database design, IMS performance, Fast Path, data sharing, planning for IMS Parallel Sysplex exploitation and migration, DBRC, and database control (DBCTL). He also develops and teaches IBM Education and Training courses, including IMS/ESA Version 6 Product Enhancements and IMS Fast Path Implementation.

Thanks to the following people for their invaluable contributions to this project:

Maggie Cutler
Matthew Nicholls
Elizabeth Patton
Alan Tippett
International Technical Support Organization, San Jose Center

Cedric Chen
George Denny
Dick Hannan
Michael Morrison
Tom Morrison
Betty Patterson
Sandy Stoob
Judy Tse
Jack Wiedlin
Mark Ziebarth
IBM Santa Teresa Laboratory

Dougie Lawson
Pete Sadler
Andrew Wilkinson
IBM United Kingdom

Curt Jews
IBM Washington Systems Center

Sharon Wang
IBM Education and Training, San Francisco, USA

Heiner Rauschenbusch
IBM Germany

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 189 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com/`
  For IBM Intranet users `http://w3.itso.ibm.com/`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. IMS/ESA Shared Queues in a Parallel Sysplex Environment

Many IMS users are looking to manage their increasing application workloads efficiently now and in the future. In addition they want to develop strategies to improve their current service levels and move toward continuous operations while maintaining or improving acceptable response times.

IMS users recognize the requirements to efficiently distribute and balance the total workload across multiple systems. The introduction of shared queues, a new function in IMS/ESA Version 6, assists users in meeting these requirements by:

- Providing dynamic load balancing among multiple IMS/ESA systems in a Parallel Sysplex environment a Parallel Sysplex environment
- Optimizing overall throughput across the sysplex—no single IMS remains underutilized while others are overloaded
- Achieving improvements in continuous availability for applications
- Improving reliability and providing better failure isolation, as any remaining IMS can continue processing the shared workload
- Adding on new IMS systems as workload increases

With previous releases of IMS, each IMS system has its own queues for both input and output messages and its own expedited message handler for Fast Path messages. Messages are processed in the local IMS system, unless that IMS sends the message to another IMS system.

With IMS Version 6, all IMS systems in a Parallel Sysplex can share a common set of message queues stored in the coupling facility. Full function and Fast Path input and output messages can be processed by any IMS system that has access to the shared queues and is capable of processing the message. Any IMS system running in a Parallel Sysplex can participate in processing application workload as defined in its IMS system definition according to available CPU capacity and resources.

With a shared-queues implementation, new IMS subsystems can be added easily to the sysplex as workload increases or extra load must be processed. The content of shared queues is not affected by hardware or software failures. Remaining IMS systems continue to schedule their applications and process messages stored in the coupling facility.

In this chapter, we introduce shared queues; describe their design, components, and message flow; and provide some information to consider when planning and implementing a shared queues environment.

We assume a general knowledge of IMS message processing.

1.1 Who Should Use Shared Queues

Not clear. The shared queues function increases the number of options available for customers to increase their system throughput and availability. Its approach of processing transactions, wherever in a single-image CPU environment has its resource, is an advantage to any IMS implementation. Additionally to keep applications available, even when hardware or software failures occur.

Not clear. Except for the restriction of SERIAL, advanced program-to-program communication (APPC) and OTMA type transactions theoretically and practically each transaction can be processed by any IMS system in the sysplex.

Not clear. IMS users with high transaction volumes per day, with many large sets of data volumes accessed in data sharing by multiple IMS systems, can now use the shared queues function in IMS/ESA. The current design reduces overhead to maintain the shared queues. In respect to this performance issue the concept of shared queues provides the required workload distribution to any IMS DB/DC system participating in a shared-queues environment.

In general, the best candidates for migration are IMS installations that have already implemented “cloned IMS systems” (identical system definitions in each IMS) and have activated data sharing in a Parallel Sysplex environment. Nevertheless, evaluation of the benefits of shared queues must always done.

IMS installations with Multiple Systems Coupling (MSC) routing are also good candidates for shared queues. They now have the option of eliminating the message traffic overhead by replacing MSC links with shared queues.

Benefits can also accrue to customers who have a single IMS system. Failure isolation and higher application availability are important for such installations too. Setting up your shared queues environment with two or more cloned IMS systems allows you to process messages across multiple IMS systems, thereby increasing application availability.

The shared queues function reduces planned and unplanned outages, because improved workload balancing optimizes the available CPU resources. For more information about the benefits of shared queues to any IMS/ESA production system, see Chapter 4, “Configuration Scenarios with Shared Queues” on page 35.

1.2 Overview of Shared-Queues Message Processing

Before IMS Version 6, workload distribution was defined statically through such methods as network balancing, the MSC facility, Intersystem Communication (ISC) links, Advanced Program-to-Program Communication (APPC) and the Workload Router (WLR).

With IMS/ESA Version 6, messages can be placed on a shared queue that can be processed by any IMS subsystem in the shared-queues group. A shared-queues environment has a has two sets of queue types; one each for full function and Fast Path. The full function shared queues contain all input and output messages for full function shared queue processing. The EMH shared queues are used for Fast Path messages. The queues are maintained as list structures on a coupling facility and can therefore potentially be processed by any IMS in the Parallel Sysplex. Any

IMS Transaction Manager (TM) system has the potential to process any message on its shared queues. The list structures are persistent and recoverable. A shared-queues environment can be also seen as a single-image view of a defined set of multiple IMS system members in a sysplex.

1.2.1 An IMS Shared-Queues Environment

The shared queues function manages and distributes on demand all workload effectively and efficiently to any IMS system in a shared-queues group. Transactions that are entered on one IMS can potentially be processed by any other IMS that has access to the shared queues. In this redbook we call this single-image view of multiple IMS systems connected through their coupling links to a common set of list structures in the coupling facility an *IMS sysplex*. Examples of IMS sysplex groupings are all IMS systems in your systems test environment, an IMS sysplex for inquiry applications only, or a production IMS sysplex where a set of single IMS DB/DC production systems are set up as a production sysplex environment. Each customer's approach to sysplex configuration will vary according to individual needs.

1.2.2 The Common Queue Server

The transactions that are entered on an IMS in an IMS shared-queues environment are put on shared queues by a new interface between IMS and the shared queues: the Common Queue Server (CQS). CQS is a generalized queuing facility that manages data objects on a shared queue on behalf of its clients. CQS runs in its own address space and is started by an IMS system. All IMS systems communicate with their own CQS to manipulate IMS messages sent to and received from shared queues. There is a one-to-one relationship between each IMS and its associated CQS. Each CQS is connected to the common set of list structures in the coupling facility; up to 32 CQSs can be connected to these structures.

1.2.3 Shared Queues Group

An IMS system that shares a defined set of list structures with other IMS systems is a member of a *shared-queues group*. A shared-queues group defines a set of IMS systems, that can put their messages to the list structures, or get their messages from the list structures. Members can be added to or removed from the shared-queues group dynamically. You can remove membership by shutting down the IMS system or by changing the IMS `SHAREDQ` startup parameter to reference to the name of the proclib member containing the shared queue parameters.

1.2.4 Registering Interest in Queues

All IMS systems in their shared-queues group register interest in particular transactions or LTERMs corresponding to their system definition in shared queues. Each CQS notifies its associated client (IMS) that messages exist for it to process. If that IMS has an available dependent region in which to execute a transaction, IMS asks the CQS to get the message from the shared queues. Multiple IMS systems can register interest through their CQS in the same transaction. The first CQS that requests a message from the shared queue receives it and passes the message to its IMS system.

IMS also registers interest in output messages for its defined LTERMs to its CQS. If a terminal is logged on to an IMS and messages are held for this terminal, CQS

delivers the message to its IMS for terminal output transmission. Messages with unknown LTERM destinations stay on shared queues.

Figure 1 shows a sample shared queues configuration.

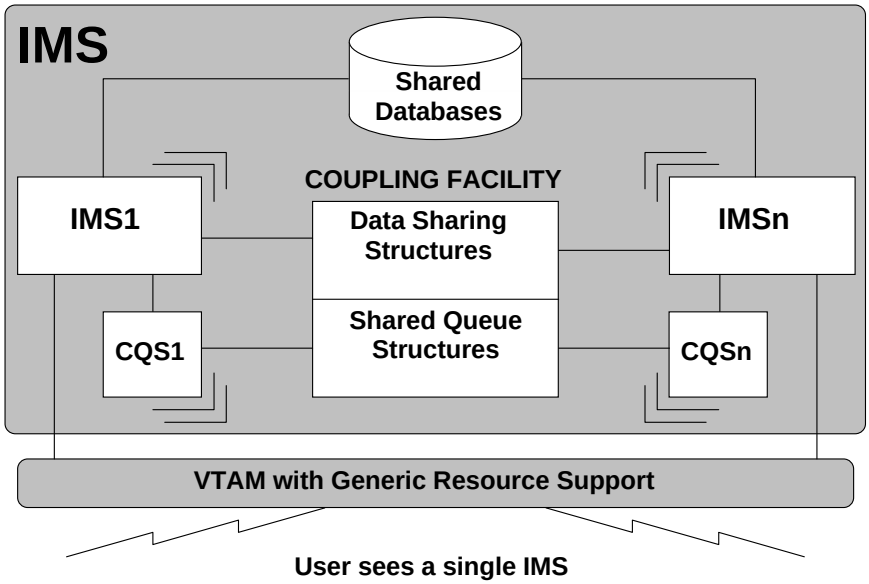


Figure 1. A Shared-Queues Configuration

Any IMS installation can take advantage of the enhanced functionality of shared queues among multiple IMS systems. Chapter 4, “Configuration Scenarios with Shared Queues” on page 35 provides you with information to evaluate and set up an efficient IMS workload management in your production environment. The shared queues function is optional in IMS/ESA Version 6; you can still use conventional message queuing with local queues.

When you implement shared queues, the IMS sysplex manages your incremental capacity needs with enhanced availability for your application systems. The shared queue function also reduces the impact of system failures by continuing the message processing on remaining or additional IMS systems. It can also eliminate message traffic overhead and bottlenecks for multiple MSC links. When possible, cloning your IMS systems may be an attractive alternative. It additionally reduces the system maintenance effort and allows any IMS in the shared-queues group to process all transactions.

1.3 Message Flow within A Shared-Queues Environment

Message flow in a shared-queues environment differs from than in a local queuing environment. Below we briefly review the general message flow for full function and Fast Path messages according to the major components used in a shared-queues environment and explain the steps required to process a transaction with its associated output message.

1.3.1 Full Function Message Queuing

Whenever a full function transaction arrives in an IMS system and that IMS has all the resources available to immediately process the message, it puts it on the shared-queue list structure in the coupling facility. The IMS processes the received message. If the IMS did not have the resources available to process the message, it makes the message available for processing to all other clients in the shared queues group. For further details see 3.1, “Processing a Full Function Transaction” on page 25. Figure 2 illustrates how ff transactions are processed when the message is put on the shared queue.

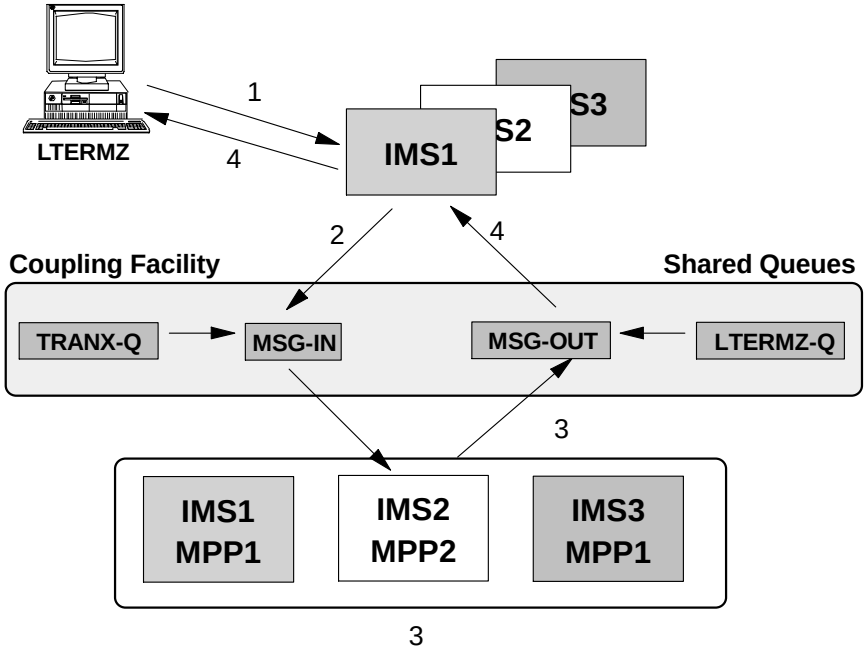


Figure 2. Full Function Message Processing in a Shared-Queues Environment

1. LTERMZ establishes a session with IMS1.
2. LTERMZ enters a message (MSG-IN), which is placed on the coupling facility and queued on a MSGQ structure for TRANX.
3. MPP2 started on IMS2 requests and processes MSG-IN. The output message response (MSG-OUT) is placed in the coupling facility and queued on a MSGQ structure for LTERMZ.
4. MSG-OUT is delivered to LTERMZ by IMS1, as IMS1 has the session with LTERMZ.

1.3.2 EMH Message Queuing

In general, Fast Path messages are processed in their local system, to avoid access to the shared EMH queues. This mode of processing takes place when the **local first** option (the default) is specified in the Fast Path input/edit routine and an IMS Fast Path (IFP) program processing the program specification block (PSB) is waiting for work in the IMS Fast Path system (see 3.2, “Processing a Fast Path Transaction” on page 28). Otherwise the message is placed on the shared queue for workload distribution. For additional information about processing a Fast Path transaction, see 3.2.1, “Local Only” on page 31.

Figure 3 on page 6 illustrates EMH processing in a shared-queues environment.

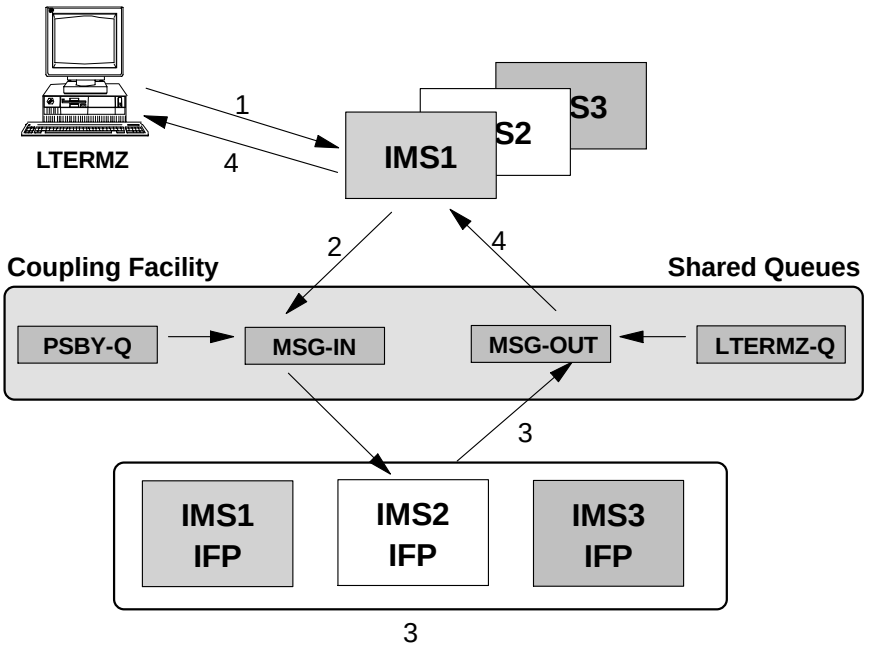


Figure 3. EMH Processing in a Shared-Queues Environment

1. LTERMZ establishes a session with IMS1.
2. LTERMZ enters an EMH transaction (MSG-IN), which is placed in the coupling facility and queued on an EMHQ structure for PSBY.
3. IFP belonging to IMS2 requests and processes MSG-IN and creates an output message (MSG-OUT), which is placed in the coupling facility and queued on an EMHQ structure for LTERMZ.
4. MSG-OUT is delivered to LTERMZ by IMS1, as IMS1 has the session with LTERMZ.

In a shared queues environment:

- Messages are queued on coupling facility structures accessible by all IMS systems in the shared-queues group.
- Any IMS that has registered interest in a PSB can request and subsequently process messages queued for that PSB.
- The transaction output is ultimately delivered by the IMS that is in session with the originating terminal.

Chapter 2. Common Queue Server

The CQS is a generalized queuing facility that manages data objects from a shared queue on behalf of its clients.

CQS runs on either an MVS/ESA or an OS/390 operating system. The CQS client must also run under the same MVS/ESA or OS/390 operating system. CQS uses the coupling facility as a repository for data objects. Clients communicate with CQSs, using CQS requests that are supported by CQS macro statements. Using these macros, CQS clients can direct their CQS to manipulate client data residing on shared coupling facility structures. The CQS runs in a separate address space that can be started by the client system.

IMS DB/DC and DC Control (DCCTL) subsystems use the CQS to access the shared queues. The CQS address space is started during IMS initialization through parameters specified in the CQSIPxxx proclib member. Each IMS in a shared-queues environment accesses the shared queues through its own CQS address space.

CQS performs the following services on behalf of its clients:

- Notifies registered clients when work exists for them on the shared queues
- Provides clients with an architected interface for accessing the shared queues and the CQS. Potentially, subsystems other than IMS DB/DC and DCCTL could use this interface.
- Writes CQS checkpoint information to a CQS checkpoint data set
- Writes structure checkpoint information to structure recovery data sets (SRDSs) for recovery of shared queue list structures.
- Provides structure recovery for the shared queue list structures
- Performs overflow processing for the shared queue list structures
- Drives numerous CQS and user exits. CQS exit routines inform the client when events such as the following occur:
  - CQS abnormal termination
  - CQS restart completion
  - Structure copies
  - Structure recoveries
  - Structure overflow
  - Work available on a shared queue

2.1 Base Primitive Environment

The Base Primitive Environment (BPE) is a software layer on which new components of IMS are built. It is the first step in the rearchitecting of IMS. BPE provides common services such as subdispatching, storage management, tracing, serialization, and message formatting. BPE services are accessed through well-defined architected interfaces. CQS is one of the first components built on the BPE software layer.

For the most part, BPE is not visible. IMS components are built on top of BPE, but BPE itself has very few externals. The externals are:

- BPE modules in RESLIB
- Messages issued to the system console
- CQS JCL
- Proclib members
- BPE dump formatter

For information about setting up the BPE proclib members, see 5.10, “Specifying the Base Primitive Environment Parameters” on page 81.

2.2 Components of a List Structure

Coupling facility list structures are managed by the Coupling Facility Control Code (CFCC). This section is based on CFCC Version 4. CFCC details change frequently; you must obtain up-to-date information from the *OS/390 PR/SM Planning Guide*, GA22-7236.

IMS shared queues are stored on coupling facility list structures. There are two types of list structures that IMS uses; the ff (MSGQ) structure and the Fast Path (EMHQ) structure. The MSGQ structure is always required, and the EMHQ structure is required only if the system has been genned as a Fast Path system. (IMS is a Fast Path system if the IMS system definition includes the FPCTRL macro.)

If the IMS system definition includes the FPCTRL macro, the EMHQ structure is always required. There are cases where the IMS system definition has the FPCTRL macro and EMH is not being used to process transactions, such as with Fast Path databases being used. The EMHQ structure is still required because Fast Path transactions could be implemented through an online change if the system has been defined as Fast Path capable. If the system is defined as Fast Path capable, and you do not process EMH transactions, your EMHQ structure can be defined as the minimum size of 256 K.

Shared-queue structures are persistent. They remain allocated even if all CQSs have disconnected, or all IMSs and their associated CQSs cold start.

List structures can contain four significant components:

- List headers
- List entries
- Lock table
- Event monitor controls (EMCs)

The lock table and the EMCs are optional components of an MVS list structure. CQS uses all four list structure components in the shared-queues environment.

Figure 4 on page 9 shows a ff message queue list structure with two messages. Message 1 is stored as one data object, comprising a list entry control and one data element. Message 2 is stored as two data objects (list entry 2 and list entry 3), each comprising one list entry control and two data elements.

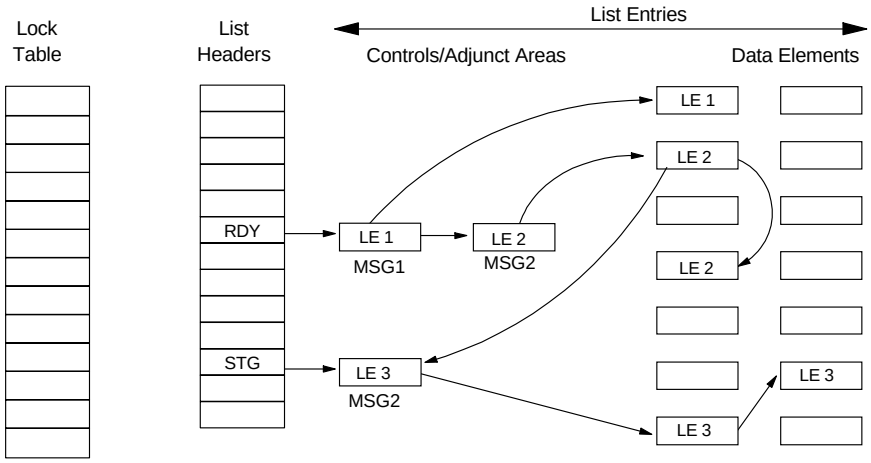


Figure 4. Components of a List Structure

2.2.1 List Headers

List headers are anchor points for CQS or IMS queues. A list header points to the first list entry in the queue. When a structure is allocated, 192 list headers are created; 71 for CQS private queues, and 121 for the IMS queues. See 2.3, “CQS Queue Types” on page 11 for details on the different types of queues. For a MSGQ structure, 99 of the IMS list headers are used, and the remaining 22 are reserved for future use. For an EMHQ structure, 22 of the IMS list headers are used and the remaining 99 are reserved for future use.

2.2.2 List Entries

List entries are chained off list headers and are used to store messages. Each list entry is divided into two sections, a control area and one or more data elements. The control area contains the queue name, which is the object in which IMS registers interest if it is capable of processing that type of message. The control area is also the anchor point for one or more 512-byte data elements in which the message is stored. Each message is divided into 512-byte sections, with each section stored in a data element. The data elements are chained together in order to retain the sequence of the message.

Figure 4 demonstrates how the chaining occurs. In this figure, assume the size of the IMS queue buffers are 1 K (requiring two data elements per list entry). The first message has a single list entry with one 512-byte data element containing the message. The second message is 2 K long and four 512-byte data elements are required to store the message. The IMS queue buffers are 1 K in length; to store the 2 K message requires two list entries. When a message requires more than one IMS queue buffer, the second and subsequent IMS queue buffers are chained off the staging queue. The first queue buffer is chained off the ready queue and points to the list entry in the staging queue. 2.5, “IMS Queue Manager” on page 13 shows how the queue manager divides the message across multiple data elements.

All list entries with the same queue name form a sublist. IMS requests the CQS to *put* a message on the end of the sublist if a new message arrives, or to *get* a message from the front of the sublist case when it wants to process a message. In Figure 5 on page 10, seven list entries are anchored off two list headers. There

are three TRANX transactions. The grouping of them together is referred to as a *sublist*. (The EMC area is described in 2.2.4, “Event Monitor Controls” on page 10.)

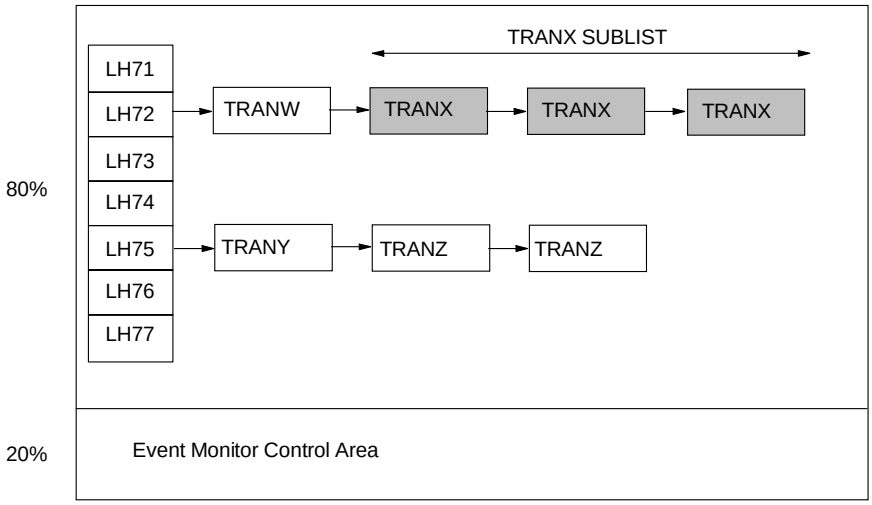


Figure 5. Transaction Sublists in the Shared Queue

2.2.3 Lock Table

The CQS uses the lock table to serialize access to the CQS control queue. The control queue is currently the only queue that is locked.

The control queue is a CQS private queue that has list entries for functions such as overflow processing, structure checkpoint, and rebuild processing. When a CQS wants to perform one of these functions, it must first obtain the lock for the control queue. When it obtains the lock, it proceeds to look at the list entry for the function that it needs to perform. If the list entry already contains the name of a CQS, the function is already being performed. Otherwise, this CQS adds its name, releases the lock on the control queue, and proceeds to perform the function. When a CQS has its name in the list entry for a particular function, it is referred to as the *master CQS*.

2.2.4 Event Monitor Controls

An EMC entry is a control block in the coupling facility list structure that is used to represent a CQS that has a registered interest in a queue. Each EMC entry occupies 64 bytes of storage.

An EMC control block is created for each CQS that has an interest in a particular queue. If you have three CQSs that register an interest in queue TRANA, three EMCs are created. Care should be taken when sizing the shared queue structures to ensure that there is enough space to store the number of EMCs that are required.

CQS reserves 20% of the total structure size for EMCs. If CFCC Level 4 or greater is being used, the area can be dynamically increased in 5% increments up to a maximum of 50% if required. Use the `SETXCF ALTER` command to make this change.

Available data elements are used to create additional space for EMCs. If the EMC area is increased, the amount of space available for list entries is reduced.

If the EMC area runs out of space, any attempt by an IMS system to register an interest in a queue fails.

2.3 CQS Queue Types

Various types of private and client queues are managed in an IMS shared-queues environment. Each queue type is used for a different type of work. In an MSGQ structure, there are nine different client queue types, and in an EMHQ structure, there are two different client queue types. Eleven list headers are assigned to each client queue type, to reduce contention when accessing the client queues. Each queue type has a corresponding queue type number. The queue type number is passed by IMS to CQS with each request to put a message on a queue. The queue type number ensures that the messages are placed on the correct queue.

2.3.1 Private Queue Types

In the following definitions of the queue types, the queue type number is shown in parentheses after the queue name.

There are six types of private queues:

Control queue (01)
: Contains information to manage the shared-queues environment. Global information about the structure, local information about each CQS, structure checkpoint information, structure rebuild information, and information about overflow processing are some of the details that are stored in the control queue.

Cold queue (02)
: Contains messages that a CQS found on the lock queue that were being processed by the IMS that this CQS serves. Messages are left on the lock queue if an IMS system abends while transactions are being processed, and the IMS system is restarted cold. (To emergency restart an IMS system cold, specify either COLDCOMM or COLDSYS on the restart command.) The messages that IMS was processing at the time of the abend would have been on the lock queue. When the IMS system is restarted cold, all messages that were being processed are moved from the lock queue to the cold queue. They remain on the cold queue until they are manually dequeued.

  Messages are also moved to the cold queue if CQS was coldstarted (regardless of what IMS did).

Rebuild queue (03)
: An intermediate queue used during recovery of an EMHQ structure with WAITRBLD=NO specified

Lock queue (08)	Contains messages that are locked. When IMS reads a message, the message is locked by requeuing it to the lock queue. The lock queue is used to prevent more than one IMS system from processing the same message. The message is later deleted when IMS completes processing it, or the message is unlocked if IMS cannot process it.
Move queue (09)	An intermediate queue used to save messages that are being moved from one queue to another
Delete queue (10)	An intermediate queue used to save messages which are being deleted

2.3.2 Full Function Queue Types

There are nine types of full function message queues:

Transaction ready queue (01)	Contains the first IMS queue buffer of messages destined for local or remote transactions
Transaction staging queue (02)	An internal queue that the IMS queue manager uses for holding those parts of input messages that exceed a single IMS queue buffer
Transaction suspend queue (03)	Contains the first IMS queue buffer of transactions that have been suspended
Transaction serial queue (04)	Contains the first IMS queue buffer of serial type transactions (defined with SERIAL=YES on the TRANSACT macro)
LTERM ready queue (05)	Contains the first IMS queue buffer of messages destined for LTERMs
LTERM staging queue (06)	An internal queue that the IMS queue manager uses for holding those parts of output messages that exceed a single IMS queue buffer
APPC ready queue (07)	Contains the first queue buffer of output messages destined for APPC
Remote ready queue (08)	Contains the first IMS queue buffer of messages destined for MSNAMES or LTERMS outside the shared queues group (MSC responses)
OTMA Ready Queue (09)	Contains first IMS queue buffer of output messages destined for Open Transaction Manager Access (OTMA) devices

2.3.3 Fast Path Queue Types

There are two Fast Path (EMHQ) queue types:

Program ready queue (01)	Messages destined for Fast Path programs
LTERM ready queue (05)	Messages destined for LTERMs that were generated by a Fast Path program

2.4 Queue Names

IMS registers interest in queues by the queue name. The queue name is 16 bytes long and is made up of a 1-byte queue type, an 8-byte resource name, and in some cases, a 4-byte IMS identifier with three blanks. The types of resources are transaction names, LTERM names, and MSNAMEs. For APPC and OTMA transactions, a timestamp is used in place of a resource name to ensure that only the IMS that placed the transaction onto the queue can process the transaction.

If a message must be processed by a particular IMS, the identifier of the IMS that is to process the message is placed in the queue name. No other IMS has an interest in this unique queue name. For example, if TRANA has a requirement to be processed by IMSA, it would be added with a queue name of 'TRANAbbbIMSA'.[1] IMSA is the only system that has registered interest in queue 'TRANAbbbIMSA', so it is the only IMS system notified when one of these transactions is queued.

If it did not matter which IMS system processed TRANA, TRANA would be added with a queue name of 'TRANA'. Any IMS system that had registered interest in the queue name of 'TRANA' would be notified.

Full Function Queue Names

Transaction ready queue	X'01' + Transaction Code
Transaction staging queue	X'02' + Transaction Code
Transaction suspend queue	X'03' + Transaction Code
Transaction serial queue	X'04' + Transaction Code + IMSID
LTERM ready queue	X'05' + LTERM Name, for local LTERMs
LTERM staging queue	X'06' + LTERM Name
APPC ready queue	X'07' + Timestamp + IMSID
Remote ready queue	X'08' + MSNAME
OTMA ready queue	X'09' + Timestamp + IMSID

Fast Path Queue Names

Program ready queue	X'01' + PSB Name
LTERM ready queue	X'05' + LTERM Name

2.5 IMS Queue Manager

All requests to process messages on the MSGQ structure go through the IMS queue manager. IMS messages are built in one or more queue manager buffers (Qbuffers). Large messages go in large Qbuffers, and short messages go in short Qbuffers. If the message is longer than one Qbuffer, the second through nth Qbuffers are placed on the staging queue. Part-messages in the staging queue are identified by a unique queue name that is generated by the queue manager. IMS does not register interest in the unique queue name assigned to the staging queue

[1] b is the space character.

entries and therefore is not notified of part-messages placed on the staging queue. The Qbuffer containing the initial segment of the message is placed on the ready queue after the second through nth Qbuffers (if any) have been placed on the staging queue. Before placing the first qbuffer on the ready queue, the unique queue name used to place the second through nth Qbuffers on the staging queue is saved in the message prefix of the first segment of the message. This is required in order to be able to retrieve the segments of the message that were placed in the staging queue.

IMS registers interest in the ready queues and is notified when the queues go from empty to nonempty. Putting the second through nth Qbuffers on the staging queue before putting the first Qbuffer on the ready queue prevents an IMS from retrieving a partial message.

2.6 Registering Interest in Queues

An IMS system registers and deregisters interest in queues according to the type of work it can process. Consequently, the CQS for the associated IMS system notifies the IMS system when work is present on the queue (queue goes from empty to nonempty). The types of work include:

- Transactions (local and remote)
- LTERMs (local)
- MSC resources (remote LTERMs and MSNAMEs)

Registration occurs in the following situations:

- IMS initialization for full function and Fast Path statically defined transactions
- When a logical unit (LU) for a static LTERM logs on, registration to the MSGQ structure occurs
- When an LU for a dynamic extended terminal option (ETO) LTERM signs on, registration to the MSGQ structure occurs
- When the first Fast Path transaction is put on the EMHQ structure, registration in the LTERM occurs on the EMHQ structure.
- MSC logical link started (MSNAME)
- A stopped transaction has been started
- A transaction is added by online change
- A PSB is scheduled in an IFP region (a balancing group becomes

Deregistration occurs in the following situations:

- At IMS termination (CQS performs deregistration for that IMS)
- When IMS is no longer capable of processing the queue; that is,
  - An LTERM is stopped, pstopped or locked
  - A static LTERM logs off
  - A dynamic LTERM (ETO) signs off
  - A transaction is stopped, pstopped or locked
  - An MSC logical link is stopped

– A transaction is deleted by online change

– The last IFP region with a PSB scheduled is stopped

2.7 Accessing Shared Queues

Each IMS that has registered an interest in a queue is notified when the queue goes from empty to nonempty.

IMS can request CQS to perform one of six actions:

- Put
- Read
- Browse
- Move
- Delete
- Unlock

A read function gets the first list entry of a message from a queue, which causes it to be moved to the lock queue. If the message prefix contains a staging queue name, a browse would then be issued to get the remaining parts of the message from the staging queue. Part-messages on a staging queue do not have to be locked, because the only way to access them is from the first segment of the message on the lock queue.

The difference between a read and a browse is that a read gets the contents of the list entry and moves the list entry to the lock queue, whereas a browse only gets the contents of the list entry.

2.8 Message Queue Overflow Structure

When you define your message queue structure, you can define an optional overflow structure. The overflow structure is used to store the largest queues on the primary structure when the primary structure reaches a certain percentage of use. The percentage value is set in the OVFLWMAX parameter in the global CQS proclib member. For more information about defining the OVFLWMAX parameter, see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79. The largest queues are moved from the primary structure to the overflow structure to assist in preventing the primary structure from becoming full. The primary structure and the overflow structure together are known as a *structure pair*. This applies for both MSGQ and EMHQ structures.

If a printer was stopped and output was still being queued to it, rather than this one printer LTERM causing the MSGQ structure to fill up, the worst case scenario would be that it would fill up the overflow structure. All queues that are still in the primary structure are affected by the overflow structure filling up.

2.8.1 Overflow Processing

When the overflow threshold is exceeded and the overflow structure is defined, structure activity is quiesced while the queue names that are using the largest number of data elements are identified. The identified queues are moved to the overflow structure until the data element usage drops by a minimum of 20%. The moving of one large queue could result in the drop being far greater than 20%. A queue cannot exist in both the primary structure and the overflow structure. If a queue is selected to be moved, the whole queue is moved from the primary structure to the overflow structure. A maximum of 512 queues can be moved, to reduce the data element usage by 20%. You can make use of a user exit if you would like to exclude specific queues from overflow processing. For more details on how to exclude specific queues from overflow processing, see 5.10, “Specifying the Base Primitive Environment Parameters” on page 81. You can determine which queues are in the overflow structure by issuing the `/DISPLAY OVERFLOWQ STRUCTURE ALL` command. For more details on how to determine which queues are in the overflow structure, see 7.3.1, “IMS Commands” on page 106. If the defined threshold is exceeded again, the same processing occurs, to reduce the data element usage by 20%.

The following tasks are performed during overflow processing:

1. CQS quiesces the structure.
2. CQS identifies the queue using the largest number of data elements.
3. CQS moves the identified queue to the overflow structure.
4. CQS checks to see what percentage of the structure is being used. If the number of data elements in use has not decreased by 20%, another queue is identified and moved.

This process continues until the number of data elements that are being used has decreased by at least 20%, or 512 queues have been moved.

On completion of overflow processing, a structure checkpoint is taken.

The queue remains in the overflow structure until the number of entries on the queue gets to zero. Periodically, a scan is run to look at the queue names that are in the overflow structure. If these queues have zero entries queued, the queue name is moved back to the primary structure. Having a large queue in the overflow structure isolates and reduces the impact that it can have on the overall system. In the case where we have one stopped printer, the worst case scenario is that the overflow structure eventually fills up, and further messages cannot be be put onto the queues in the overflow structure. The primary structure continues to function normally.

Overflow processing should be performed only during exceptional circumstances. If it is being performed regularly, consider making the primary structure larger and look to see whether there are any problems with message delivery (such as output queuing to a stopped printer).

2.8.2 Processing without an Overflow Structure

If a primary structure is defined without an overflow structure, CQS still goes through the process of determining which queues would have been moved to the overflow structure. All puts to the identified queues are rejected, and the queues remain in the primary structure. Overflow threshold checking is not performed again.

Note: If an overflow structure is not defined, you can add one only by coldstarting the primary structure.

2.8.3 Structure Filled

If either the primary or the overflow structure becomes full, puts to that structure are rejected. A structure can become full if either all the list entries or all the data elements are used. A structure-full condition is usually a temporary one, and IMS does not stop attempting to put messages on the queue. An application program receives an `A7` or `QF` status code if it attempts to do an ISRT call to a queue that is on a full structure. If IMS has committed output messages that have not yet been placed on the shared queue, they are saved in IMS queue buffers and moved to the shared queue when space becomes available. Input messages are rejected with the following messages:

- `DFS070 UNABLE TO ROUTE MESSAGE`, for non-EMH terminals
- `DFS2194 SHARED EMHQ NOT AVAILABLE`, for EMH terminals.

2.9 CQS Checkpoint Data Sets

Each CQS maintains local VSAM entry sequenced data sets (ESDSs) for use as checkpoint data sets. There is always one for the MSGQ structure and, if Fast Path is defined, there is also one for the EMHQ structure. They are dynamically allocated to CQS when it initializes. When a system checkpoint is taken, CQS writes its control blocks and tables to the log stream, in order to support CQS restart. A CQS checkpoint is similar in concept to an IMS simple checkpoint. The location of the system checkpoint information within the log stream is recorded in the CQS checkpoint data set and in the structure on the control queue.

CQS performs a system checkpoint in the following CQS performs a system checkpoint in the following situations:

- Automatically, based on CQS log volume. The SYSCHKPT parameter in the CQSSLxxx proclib member defines how many CQS log records are to be written before a checkpoint is issued.
- Manually, when the `/CQCHKPT SYSTEM` command is entered
- At the completion of a successful structure checkpoint
- At CQS shutdown time
- At the completion of a CQS restart
- After a client resynchronizes with CQS

Figure 6 on page 18 shows two CQS address spaces, each with its own local pair of checkpoint data sets.

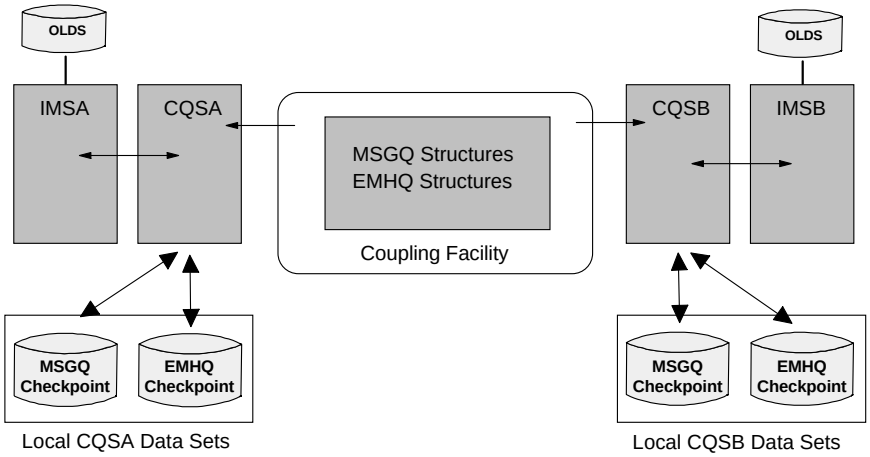


Figure 6. Checkpoint Data Sets

2.10 CQS Structure Recovery Data Sets

CQS uses a pair of SRDSs per coupling facility structure pair for its structure checkpoint processing. These data sets are shared across all CQS address spaces in the shared-queues group. When a structure checkpoint is requested, CQS dynamically allocates an SRDS. Structure checkpoint requests alternate between the two SRDSs. After a successful structure checkpoint has been taken, and the data objects have been copied to the SRDS, that SRDS is used if a structure recovery is required. The alternate SRDS (the oldest) is the one to which the next structure checkpoint is written. The oldest SRDS is also used if a structure recovery is required and the most recent SRDS cannot be read. During structure checkpoint processing, all activity to the structure is quiesced while recoverable data objects on the structure are written to a data space, before they are written to the SRDS. The size of each SRDS should be at least the size of the primary structure plus the overflow structure to ensure that all objects can fit.

To minimize the length of time the structure is quiesced during structure checkpoint processing, CQS first writes the data objects to a data space. The structure is quiesced while the data objects are being written to a logger data space. Once the data objects have been written to the logger data space, activities against the structure are resumed. CQS activities against the structure are resumed. CQS then completes the structure checkpoint process in the background by copying the data objects from the CQS-owned data space to an SRDS. Because no other work for a structure can be processed while CQS is checkpointing the structure, we recommend that structure checkpoints be performed during nonpeak hours.

In addition to recoverable objects being copied from the structure to the SRDS, log records prior to the oldest structure checkpoint are deleted because they are not required for a structure recovery. If a CQS that is part of the shared-queues group is not running while two structure checkpoints are taken, the log records for its last CQS system checkpoint will have been deleted. When the CQS is restarted, it must be brought up cold. To avoid this situation, bring the CQS straight back up after the IMS system that it supports is shut down.

To prevent a log-full condition, which is where the number of log data sets has reached the limit or there is no more DASD space for the allocation of log data

sets, regularly initiate a structure checkpoint. A structure checkpoint reduces the time it takes to perform a structure recovery if required.

CQS performs structure checkpoints in the following situations:

- When the log stream approaches a full status
- After a successful structure rebuild (unless it was a structure copy initiated by an operator)
- After a successful overflow threshold process
- After the `/CQCHKPT SHAREDQ STRUCTURE structurename` command is entered to any CQS in the shared-queues group
- At CQS shutdown if the `/CQSET SHUTDOWN SHAREDQ` command was previously issued.

Figure 7 shows the CQS SRDSs that are shared by CQSs in the shared-queues group.

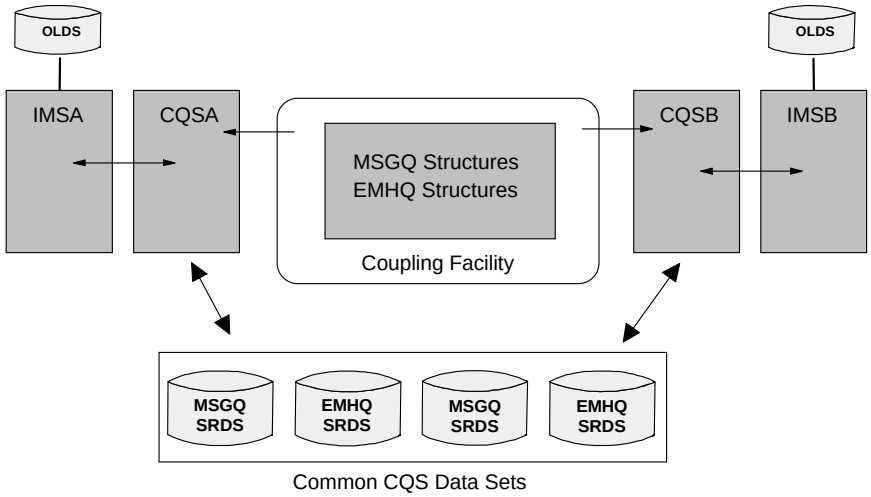


Figure 7. Structure Recovery Data Sets

2.11 Structure Integrity

CQS maintains the integrity of the shared queue structures. CQS maintains the integrity of the shared queue structures. It uses the MVS system logger to record actions that occur against the shared queue structures. CQS also takes checkpoints, which are used to recover the shared queue structures. If a shared queue structure is lost, CQS can recover it from the last structure checkpoint and the appropriate log stream (ff and Fast Path have separate log streams).

The MVS system logger is described in 2.13, “MVS System Logger” on page 20.

2.12 CQS Security

If RACF or another security product is installed at an installation, the security administrator can define profiles that control the ability of clients to connect to and access CQS shared queue structures. When an IMS client issues the CQSCONN request to connect to a queue structure, CQS issues a `RACROUTE REQUEST=AUTH` call to determine whether the client is authorized to access the structure. If the client's userid has at least update authority, the client can connect to the structure. The profile names must be of the form `CQSSTR.structure_name`, where structure_name is the name of the primary structure to be protected. The structure names are specified in the CQSSGxxx and CQSSLxxx proclib members. CQS does not perform a separate check on the overflow structure name because the primary and overflow structures are considered to be one unit.

If a profile is not defined for a CQS structure, the structure is not protected, and any client can issue a CQSCONN request to access the structure. See *OS/390 MVS Programming: Sysplex Services Guide*, GC28-1771, for more information about protecting access to coupling facility structures.

2.13 MVS System Logger

The MVS system logger is an MVS component that allows The MVS system logger is an MVS component that allows applications to log from different systems within a sysplex to a central point. The MVS system logger merges the log data from several systems in the sysplex into a single log stream. A log stream is essentially a collection of data in log blocks residing in a coupling facility structure, on DASD, or a combination of the two. The logging structures and the log streams are defined in the CFRM sysplex policy. See 5.8.1, “Defining a CFRM Policy” on page 67 for details on the setting up of the CFRM policy. For a full description of the MVS system logger, please see *OS/390 Setting Up a Sysplex*, GC28-1779.

CQS uses the MVS system logger to record all information necessary for it to recover shared queue structures. CQS writes log records for each structure pair to a separate log stream. The same log stream is written to by all of the CQS address spaces in the one shared-queues group. The MVS system logger manages the log stream and provides a merged log for all CQS address spaces that are connected to a message queue structure on a coupling facility. Figure 8 on page 21 shows two CQS address spaces writing to a shared log stream.

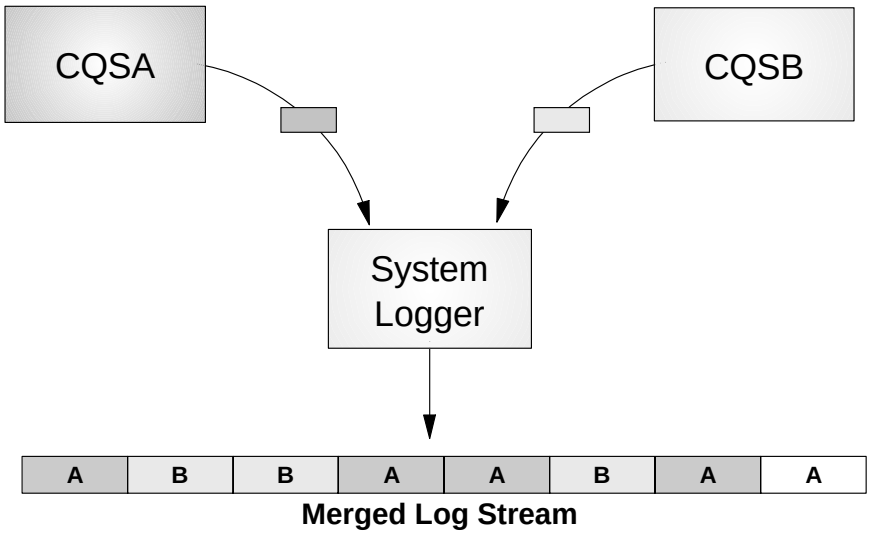


Figure 8. A Merged Log Stream

The MVS system logger can use four components:

- Logging structure
- Logger data space
- Staging log data sets
- Log data sets

2.13.1 MVS Logging Structure

The logging structure stores the logging data written by each of the CQSs that are writing to the same log stream on a coupling facility.

There is one logging structure for each shared queue structure pair. For details on how to define the logging structure, see 5.8.2.1, “Defining the Logging Structure” on page 71.

2.13.2 MVS Logger Data Space

There is one MVS system logger per MVS image. Each MVS system logger maintains a copy of the log data that it writes to the logging structure in its own logger data space unless the structure has a staging log data set defined. If a logging structure on a coupling facility needs to be recovered and a staging log data set was not being used, all of the MVS system loggers are required to participate because each logger data space has part of the data that is required to rebuild the log stream in the logging structure.

Figure 9 on page 22 shows both the logger data space (1a) and the staging log data set (1b). Only one of the two methods of duplexing the log stream can be used. You cannot use both methods at the same time.

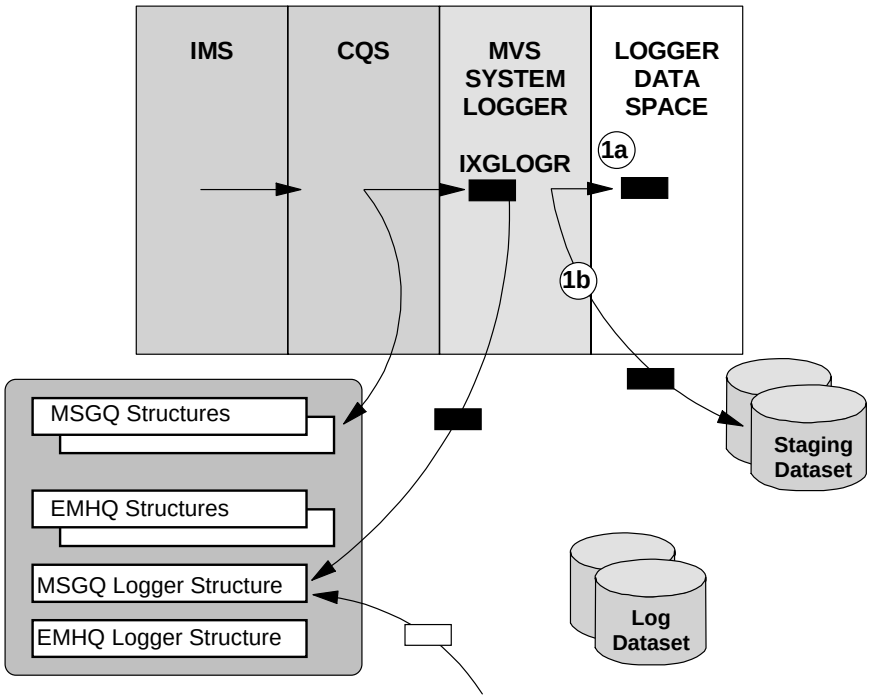


Figure 9. Logging Components

2.13.3 MVS Staging Log Data Set

Staging log data sets can be used to store a copy of what is in the logging structure on the coupling facility. If your configuration contains a single point of failure, then you should use staging log data sets in preference to the logger data space. In the case of the log stream, a single point of failure can be when the log stream is unable to be rebuilt after the failure of a single item. If your coupling facility is volatile (no backup power supply) or is on the same processor box as a system that is connecting to it, you have a single point of failure. In this case, losing power to the site or the processor box failing results in a loss of your logging structure. If you have a single point of failure, we recommended that you use staging log data sets. In a production environment, it is usually unacceptable to allow any single point of failure to interrupt your business processing.

A staging log data set contains a copy of the log data that is written to a logging structure by the local MVS system logger. The staging log data sets from all of the MVS system loggers that are writing to a common log stream are used to recover a logging structure. The staging log data set should be on DASD that is accessible by all of the MVS system loggers in the shared-queues group. Staging log data sets are required only if the logging structure needs to be recovered. The MVS system logger performing the recovery reads each of the staging log data sets in order to reconstruct the logging structure.

2.13.4 Log Data Sets

MVS system logger log data sets are used when the space used in either the logging structure or any of the staging log data sets for a log stream exceeds the value of the HIGHOFFLOAD parameter. This parameter is defined in the LOGR policy. For more details on the defining the LOGR policy see 5.8.2.1, “Defining the Logging Structure” on page 71. When the value of the HIGHOFFLOAD parameter

is exceeded, the MVS system logger offloads data from the coupling facility to a log data set. The amount of data that is offloaded is determined by the LOWOFFLOAD parameter in the LOGR policy. The logging structure is not quiesced while the offload is occurring. The HIGHOFFLOAD parameter should not be set too high, because log data continues to be added while the offload is occurring, and, if the logging structure fills up, puts to the structure are rejected. We recommend that the LOWOFFLOAD parameter be set to 0, so that all of the data in the structure when the offload commences is written to the log data set through the logger data space. All data that is written to the structure after the commencement of the offload process is not offloaded.

The number of log data sets that you can allocate is 168. If you are using OS/390 R3 or later, you can increase the number of data sets by using the DSEXTENT parameter in the LOGR policy. For more information about the DSEXTENT parameter, see *OS/390 Setting Up a Sysplex*, GC28-1779.

Chapter 3. Transaction Flow in a Shared-Queues Environment

In this chapter we discuss the path taken by an IMS transaction from the time it is entered from a terminal, through to the response back to the terminal. We discuss both ff and Fast Path transactions.

3.1 Processing a Full Function Transaction

A ff transaction is always placed on the transaction ready queue in the MSGQ structure so that it can be recovered. When a transaction is placed on the transaction ready queue, a CQS log record is created. The CQS log record is used to recover the structure if required. A ff transaction can be processed by the local IMS system when it is first received, or any of the IMSs in the shared-queues environment that have a registered interest in the queue can bid for the right to process the transaction. The exception to this rule is when the transaction is a serial type or APPC or OTMA sourced. In these cases the transaction has to be processed by the IMS system that put it on the structure.

When an IMS system is notified that work is available on the transaction ready queue, IMS updates a flag (new with shared queues) in the scheduler message block (SMB) control block to indicate that work is available for this transaction type. This flag is updated to indicate that there are no more transactions on a queue when a message processing program (MPP) on the IMS system performs a get unique (GU) call and receives a QC status code.

IMS must have the following resources available to process a transaction:

- A message processing region (MPR) that is waiting for work. This MPR must be be able to process the class for which that transaction has been assigned.
- The PSB that runs the transaction must be started.

3.1.1 Local Processing

When the local IMS system receives a ff transaction, IMS determines whether the transaction can be processed immediately in the local IMS system before it places the transaction on the transaction ready queue. The transaction can be processed immediately only if the resources are available. If the transaction can be processed immediately in the local IMS system, the message is queued to the lock queue rather than the transaction ready queue on the MSGQ structure. The local IMS system then processes the transaction. Because the transaction is not placed on the transaction ready queue, other IMS systems that have a registered interest in this queue are not notified. This is far more efficient in that only one IMS system performs scheduling, and the overhead for notifying all of the registered CQSs is removed. Where possible, you should try to process the majority of ff transactions within the local IMS system. The number of transactions processed by the local IMS system is affected by the MPR occupancy level. Low MPR occupancy levels increase the chance that the full function transaction can be processed by the local IMS system.

3.1.2 Global Processing

If the local IMS sys tem did not have the resources available to process the transaction immediately, CQS passes the transaction to the transaction ready queue. If there are no other entries on the queue for this transaction, all IMSs that have a registered interest in the queue are notified. Each notified IMS having the resources available performs all scheduling functions except passing control to the dependent region controller. IMS requests the message from CQS, and the message is passed to the MPP when the MPP issues the GU call. The first CQS to perform the read function retrieves the first list entry on the queue and passes the transaction to IMS in order to execute it. If there was only one of this type of transaction on the queue, all other CQSs attempting to retrieve the transaction are unsuccessful. All of the MPPs that failed to get the transaction receive a QC status code. Once the QC status code is received, the IMS SMB control block is updated to indicate that there is no more work for the particular transaction.

3.1.3 Full Function Transaction Flow

In this section we describe how a full function transaction is processed. The transaction flow relates to Figure 10, Figure 11 on page 27, Figure 12 on page 27, and Figure 13 on page 28.

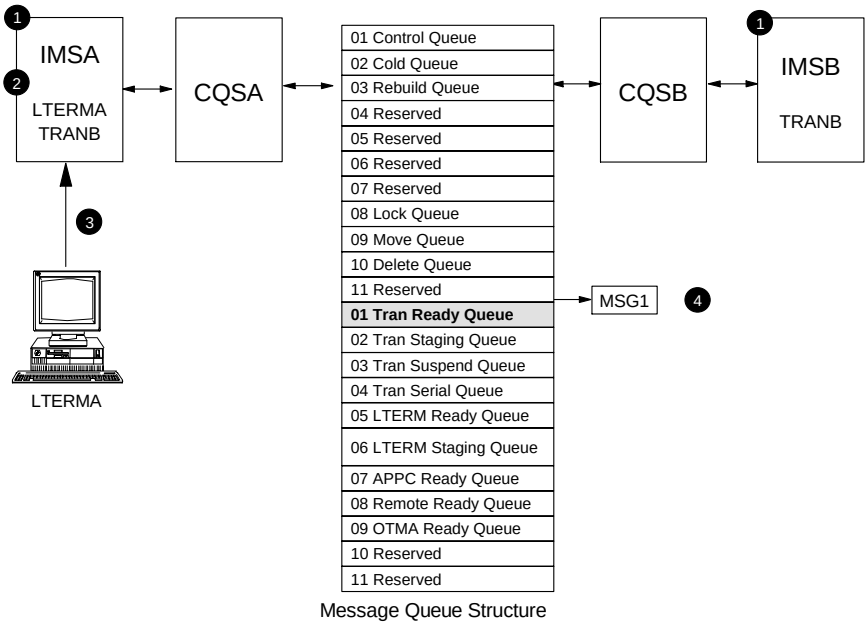


Figure 10. Full Function Transaction Flow (Part 1)

1. IMSA and IMSB register interest in the statically defined transaction, TRANB, when IMS is initialized or when the transaction is started.
2. IMSA registers an interest in statically defined LTERMA when the terminal logs on.
3. IMSA receives MSG1 from LTERMA.
4. IMSA checks to see whether it has resources available in the local IMS system in order to process the message immediately. If the resources are available, IMSA requests that CQSA put MSG1 on the lock queue and proceeds to process the message. In this example, the resources are not available, so

IMSA requests that CQSA put MSG1 on the transaction ready queue for TRANB.

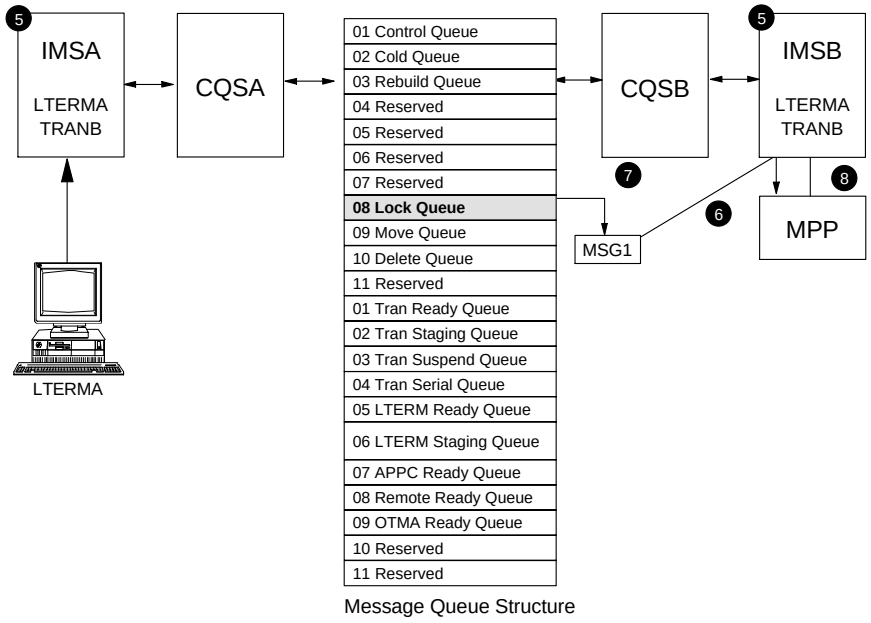


Figure 11. Full Function Transaction Flow (Part 2)

5. IMSA and IMSB are notified by the CQSs that the TRANB queue has gone from empty to nonempty.

   IMSB performs scheduling.

6. IMSB reads MSG1 for TRANB.
7. TRANB is moved from the transaction ready queue to the lock queue.
8. IMSB loads the program for TRANB.

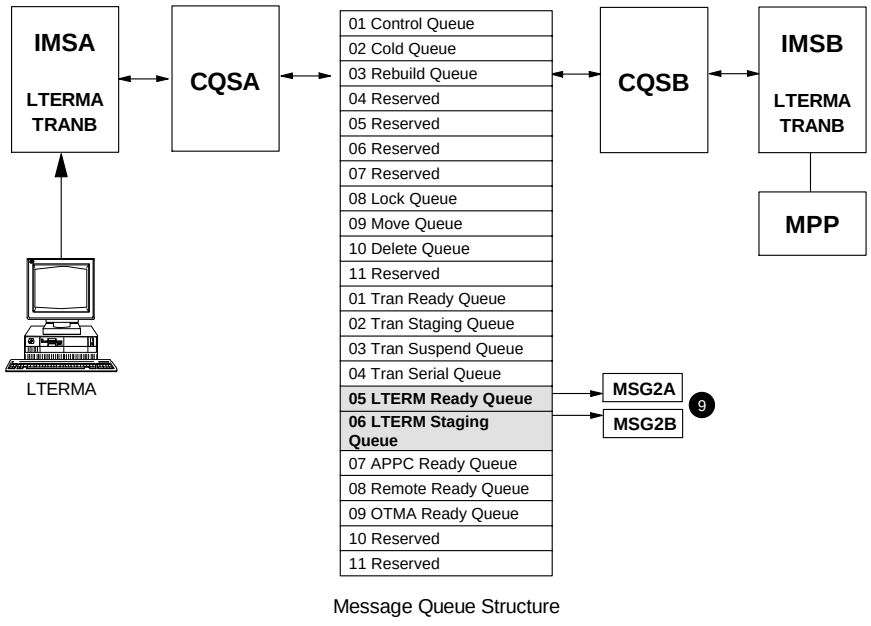


Figure 12. Full Function Transaction Flow (Part 3)

9. The MPP processes TRANB and inserts the reply (MSG2) in two queue buffers for LTERMA. The MPP then commits the processing. MSG2B is placed on the LTERM staging queue, and MSG2A is placed on the LTERM ready queue. CQSB puts MSG2B on the LTERM staging queue before MSG2A is put on the LTERM ready queue. When the MSG2B segment has been put on the LTERM staging queue, the MSG2A segment is put on the LTERM ready queue. Putting the message on the LTERM ready queue causes IMSA to be notified that the LTERMA queue has gone from empty to nonempty. The full message is then available for retrieval.

   IMSB then requests CQSB to delete MSG1 from the lock queue.

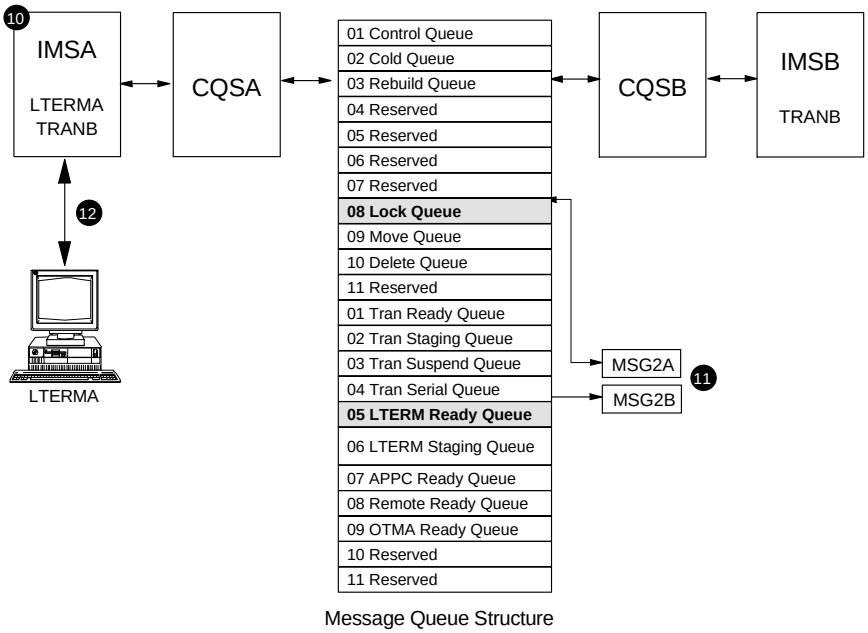


Figure 13. Full Function Transaction Flow (Part 4)

10. CQSA notifies IMSA of work for LTERMA.
11. IMSA passes a read request to CQSA. CQSA then locks MSG2A and reads the segments of the message. The message segments are then passed to IMSA.
12. IMSA returns reply messages to LTERMA.

    When the terminal has acknowledged receipt of the message, IMSA requests CQSA to delete MSG2A and MSG2B from the structure.

3.2 Processing a Fast Path Transaction

Fast Path transactions can also take advantage of shared queues. Fast Path exit DBFHAGU0 can decide whether the transaction is to be processed locally or placed on the EMHQ structure. The default setting is local first. IMS attempts to process the transaction locally, and if that is not possible, the transaction is placed on the EMHQ structure.

Each Fast-Path-defined IMS system maintains a number of tables which are used to indicate the PSBs that are active within the shared-queues environment. When an IMS system receives a Fast Path transaction, it needs to determine whether the

transaction can be processed by an active IFP region. This decision is made by doing a lookup in the tables. If there is no active IFP region in the shared-queues group that can process the transaction, a message is sent to the user indicating that the transaction cannot be processed.

A local PSB name table is created on the system where there is an active IFP region servicing that PSB. It is created only when the first IFP region servicing that PSB is started, and it is deleted when the last IFP region servicing that PSB is stopped. Each local PSB name table maintains a counter recording the number of IFP regions servicing the particular PSB on the local IMS system.

When the local PSB name table is created, a message indicating that a particular PSB is now being serviced by an IFP region is sent to all other IMSs in the shared-queues group. The message indicates the IMS system name that is servicing the particular PSB, and each IMS system in the shared-queues group either to creates or updates its global PSB name table. The global PSB name tables contain a list of the IMS system names where the PSB is being serviced by an active IFP. For each PSB that has an active IFP region servicing it, there is a local PSB name table on the local IMS system, and all other IMSs in the shared-queues group have a global PSB name table. It is possible and probable for an IMS system to have both a local PSB name table and a global PSB name table for a particular PSB. This is the case if the same PSB is being serviced by an IFP region in more than one IMS system.

Figure 14 on page 30 shows three IMS Fast Path systems in a shared-queues group. Each IMS system has a list of global and local PSB name tables. Each global PSB name table lists which IMS systems in the shared-queues group other than the local IMS system have IFP regions active that are servicing a PSB. Each local PSB name table shows which PSBs are being serviced by IFP regions in the local IMS system. The numbers in parentheses, in the local tables, represents the number of IFP regions in the local IMS system servicing the PSB. When a local PSB name table count goes from 1 to 0 the local IMS system notifies other IMS systems in the shared-queues group that it is no longer servicing the PSB. All other IMS systems update their global PSB name table.

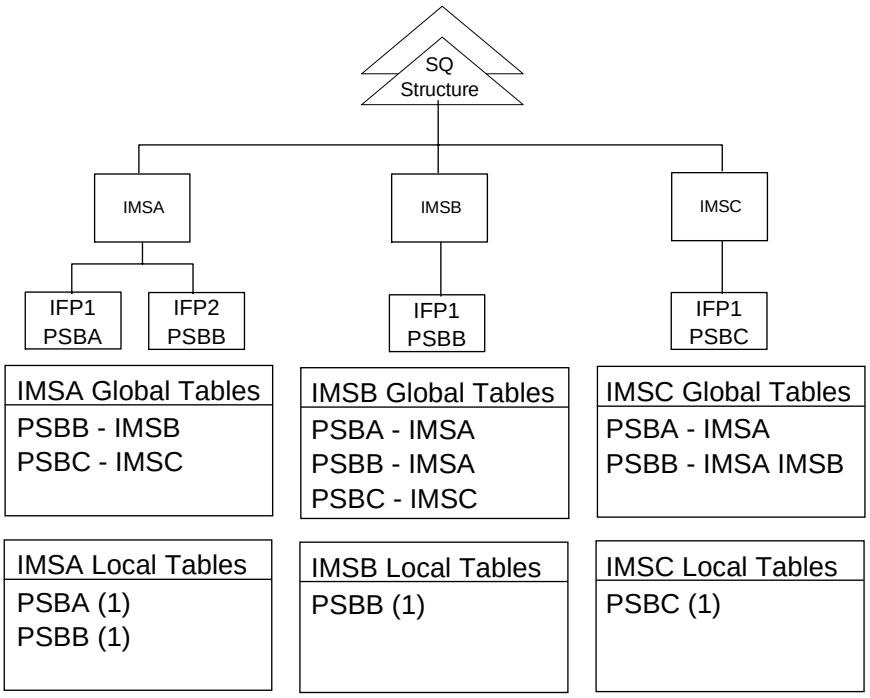


Figure 14. Global and Local PSB Name Tables

If an EMHQ structure contains transactions that are waiting to run, and the last IFP region in the shared-queues group that can process them is stopped, the transactions are deleted from their queue name and the following message is sent to the inputting terminal to unlock it:

```
DFS2529I NO FAST PATH REGION IS ACTIVE
```

When an IMS system shuts down or abends, all other IMS systems in the shared-queues environment update their global PSB name tables for all PSBs that had an active IFP region on the IMS system that has ended. The table is deleted if it was the only IMS system that was servicing the PSB (other than the local IMS system).

In the Fast Path input edit/routing exit routine, DBFHAGU0, you can specify the how the message is to be processed. This is known as the sysplex processing code (SPC), and there are three options:

- Local Only
- Local First
- Global Only

A design objective should be to process the message on the local IMS system wherever possible. If the message is processed on the local IMS system, no interaction is required with the EMHQ structure. This avoids the overhead for:

- Performing a CQSPUT to the EMHQ structure
- Each IMS system registered as interested in the PSB and LTERM being notified
- The IMS system that is to process the message having to read the message from the EMHQ structure

- Extra logging

3.2.1 Local Only

The local only option requires the transaction to be processed only on the local IMS system. The message is never placed on the EMHQ structure. If the routing code is stopped or there are no IFP regions servicing the PSB in the local IMS system, the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE.
```

Otherwise, the transaction is queued to the balancing group.

3.2.2 Local First

The Local First option, which is the default, causes IMS to attempt to process the transaction on the local IMS system first, to avoid access to the EMHQ structure. The transaction is always processed locally if there is at least one IFP region in the local system servicing the PSB that this transaction requires and the number of transactions queued to this balancing group is not greater than 5. If more than five transactions are queued to the balancing group that this transaction requires, it is still processed locally if the number of IFP regions servicing this balancing group divided by 4 is greater than the number of transactions currently queued to this balancing group. If the transaction cannot be processed locally because of the above criteria, it is passed to CQS, which puts the message on the program ready queue.

In order for the transaction to be accepted, the local routing code must not be stopped, and there must be at least one program servicing this transaction within the shared-queues group, otherwise the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE
```

The check as to whether a routing code is active is performed by a lookup on the local PSB name table and then, if required, the global PSB name table.

3.2.3 Global Only

If the Global Only option is specified and the input transaction is accepted, the message is always placed on the EMHQ structure for processing. In order for the transaction to be accepted, the local routing code must not be stopped and there must be at least one program servicing this transaction within the shared-queues group, otherwise the input is rejected with this message:

```
DFS2533 ROUTING CODE NOT ACTIVE
```

The check as to whether a routing code is active is performed by a lookup on the local PSB name table and then, if required, the global PSB name table.

The Global Only option should be selected only for transactions that are not going to run on the front-end IMS system.

3.2.4 Fast Path Transaction Flow

In this section we describe how a Fast Path transaction In this section we describe how a Fast Path transaction is processed. PSBY is defined with an SPC option of Local First. The transaction flow relates to Figure 15, Figure 16 on page 33, Figure 17 on page 33, and Figure 18 on page 34.

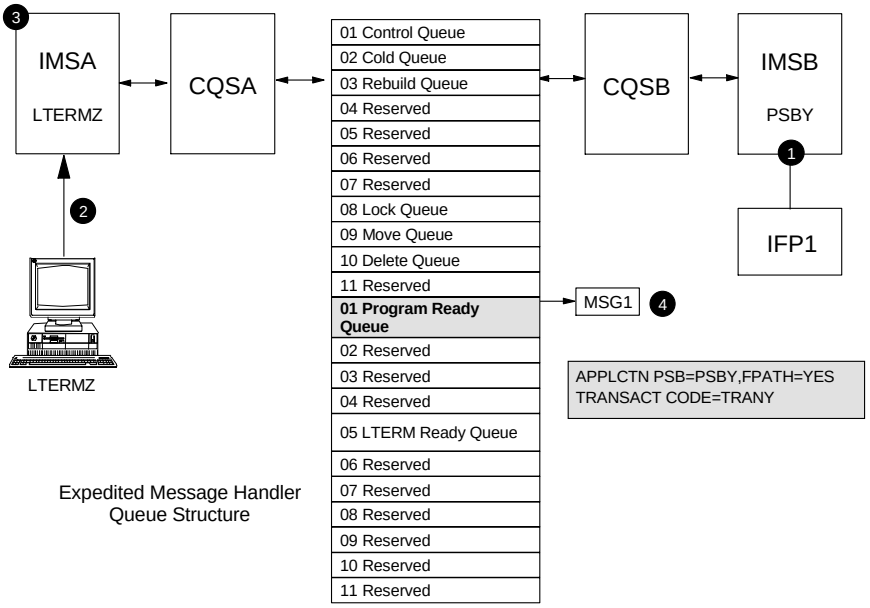


Figure 15. Fast Path Transaction Flow (Part 1)

1. IMSB registers interest in PSBY when IFP1 is started. IFP1 is the only region servicing PSBY in the shared-queues group. IMSB creates a local PSB name table for PSBY and notifies IMSA, which causes IMSA to create a global PSB name table for PSBY.
2. IMSA receives MSG1 (TRANY) from LTERMZ.
3. IMSA registers an interest in statically defined LTERMZ when the first Fast Path transaction is put on the EMHQ structure. Registration has already occurred to the MSGQ structure when LTERMZ logged on. Registration to the EMHQ structure occurs only when the first Fast Path transaction is put on the EMHQ structure.
4. IMSA performs a lookup in the local PSB name table to determine whether it can be processed locally. In this case the lookup would not be successful. A lookup is then performed on the global PSB name table. The lookup finds an entry, and IMSA accepts the transaction from the user. IMSA then requests that CQSA put MSG1 on the program ready queue for PSBY.

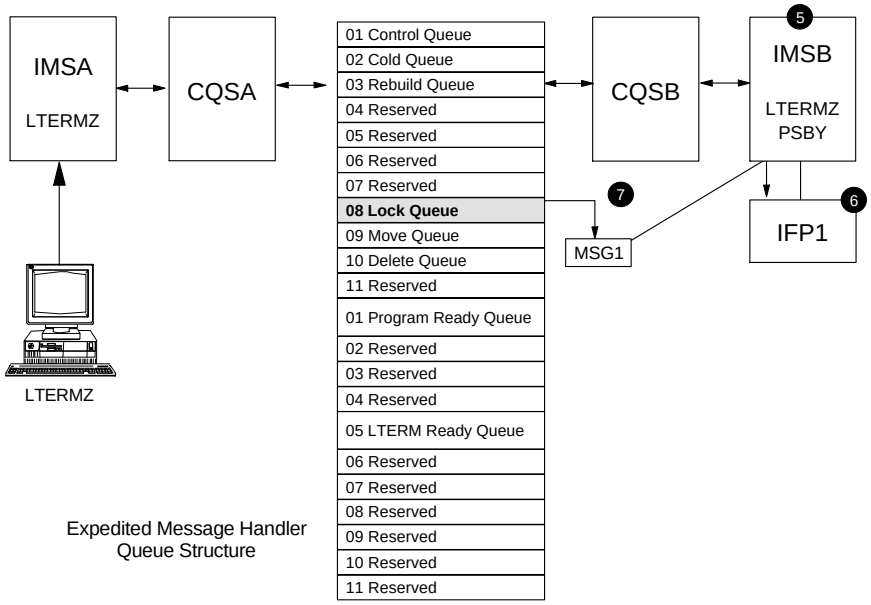


Figure 16. Fast Path Transaction Flow (Part 2)

5. IMSB is notified by CQSB that the PSBY queue has gone from empty to nonempty.
6. IMSB reads TRANY from the PSBY queue and passes it to IFP1.
7. TRANY is moved from the program ready queue to the lock queue.

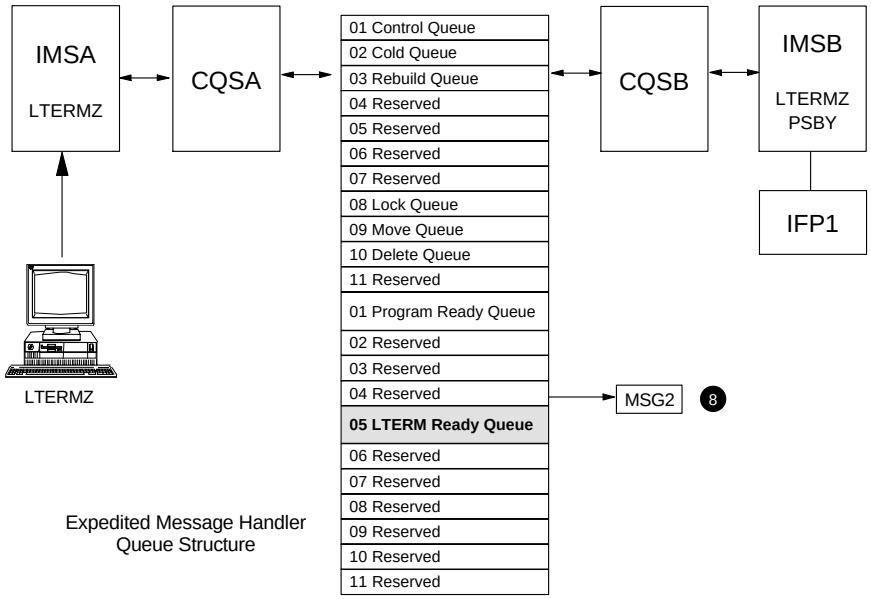


Figure 17. Fast Path Transaction Flow (Part 3)

8. IFP1 processes TRANY, inserts a reply (MSG2) for LTERMZ, and commits the work. IMSB then requests that CQSB place the inserted reply message (MSG2) for LTERMZ on the LTERM ready queue and delete MSG1 from the lock queue.

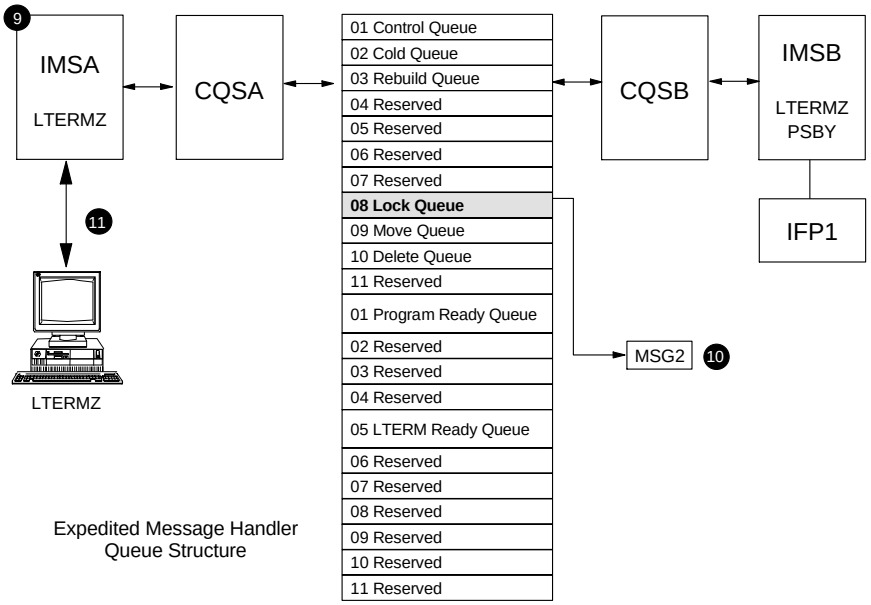


Figure 18. Fast Path Transaction Flow (Part 4)

9. The queue for LTERMZ has gone from empty to nonempty. CQSA then notifies IMSA of work for LTERMZ.
10. IMSA passes a read request to CQSA. The message is retrieved. The reading of the message causes the message to be moved from the LTERM ready queue to the lock queue.
11. IMSA returns the reply message to LTERMZ.

    When the terminal has acknowledged receipt of the message, IMSA requests CQSA to delete MSG2 from the structure.

Chapter 4. Configuration Scenarios with Shared Queues

Many different system configuration scenarios are possible in an IMS sysplex with shared queues. The configuration you implement depends largely on your objectives and the benefits you expect by moving to shared queues.

In this chapter we deal with the most common configurations and explore how to take advantage of the shared queues function in IMS/ESA Version 6.

We define some important terms, identify some customer configurations, and present and discuss the new configurations based upon shared queues corresponding to their advantages and limitations. Some migration and fallback aspects may also be considered.

Chapter 5, “Implementing Shared Queues” on page 55 describes the steps required to implement shared queues.

4.1 Background Information

In this section we define the following concepts, which are important for you to understand as you consider the configuration alternatives for shared queues:

- Registering interest in transactions and terminals
- Significant status
- Front-end and back-end IMS systems

4.1.1 Registering Interest in Transactions and Terminals

Understanding the concept of interest registration for transactions and terminals serves as a good foundation for considering the different shared-queues configuration approaches.

4.1.1.1 Registering Interest in Transactions

Each IMS system in the shared-queues group has an associated CQS address space. CQS (the server) acts on behalf of its IMS system (the client) by placing messages on and removing messages from the shared queues. CQS does not actually know the content or meaning of the IMS messages. Instead, IMS tells CQS to place the messages on a particular queue in the shared queues structure. There are several queue types that are managed by CQS for transactions and LTERMs.

When an IMS system in the shared-queues group starts or restarts, it informs CQS of the transactions defined in the Stage 1 system definition. CQS places these transaction names in a special table, the *interest area table*, in the coupling facility structure. When input messages are placed on the shared queues for a particular transaction, CQS checks whether the transaction is listed in its interest area table. If it is in the table and the queue goes from empty to nonempty, CQS notifies IMS that there are messages to be processed.

If the IMS system has an available dependent region with the proper class and priority to execute the transaction, IMS tells CQS to get the message from the shared queues. Because there could be many IMS systems in the shared-queues

group, the *first CQS* that requests the message receives the message from the shared queues. In shared-queues environments, the same transaction names can exist in different CQS interest area tables if the transaction is defined in more than one IMS system in the shared-queues group.

4.1.1.2 Registering Interest in PSBs for Fast Path

When an IMS system in the shared-queues group starts or restarts, it informs CQS of the programs defined in the Stage 1 system definition. CQS places these program names into a special table in the coupling facility structure, called the *interest area table*. When input messages are placed on the shared queues for particular programs, CQS checks if the program is listed in its interest area table. If it is in the table and the queue goes from empty to non-empty, CQS notifies IMS that messages exist to be processed.

If the IMS system has an available IFP region to execute the transaction, IMS tells CQS to get the message from the shared queues. Since there could be many IMS systems in the shared-queues group, the *first CQS* to request the message receives the message from the shared queues. In shared queues environments, the same program names can exist in different CQS interest tables if the program is defined in more than one IMS system in the shared-queues group.

4.1.1.3 Registering Interest in Terminals

LTERMs are also registered in the CQS interest area table. When an ETO user signs on or a user on a static terminal logs on, IMS notifies CQS to add an entry for the LTERM in the CQS interest area table. When messages are placed on shared queues for a specific LTERM, the CQS with that LTERM in the CQS interest table notifies IMS of the existence of a message, and the message is delivered to the IMS to which the terminal is connected. If the LTERM does not exist in any CQS interest area table, the message stays on the shared queues.

4.1.2 Significant Status

An affinity is created whenever an LU connects to an IMS system. These affinities are usually deleted at session termination. Session termination can consist of user logoff, VTAM LOSTERM exit, IMS normal shutdown, or IMS failure. Affinities can remain after session termination, however, if the terminal has a significant status.

A significant status indicates that an affinity should remain after session termination, to maintain the consistency between a terminal and the IMS system. This is a list of significant status:

- Terminal in response mode
- Terminal in conversational mode
- Exclusive mode (/EXCLUSIVE)
- Preset destination (/SET xxxxxxxxxx)
- MFS test (/TEST MFS)
- 3600 Financial/SLUP (LU0) devices
- MVS failure

An affinity remains in the case of MVS failure, because the IMS ESTAE exit is not executed in this situation. In the event of an IMS failure, affinities are deleted by the abend ESTAE routines.

For dynamic terminals, while a user stays logged on, the significant status is associated with that LU through the user. When the user signs off (which is done before session termination or logoff), the significant status is moved to the user structure, and the user structure is "disconnected" from the virtual terminal control block (VTCB). When the session terminates, there is no significant status associated with the session, and the affinity is always deleted.

For static terminal logoff, the entry in the table is also removed even if there is significant status for the terminal.

Existing affinities may cause problems in later session establishments. If there is an existing affinity for a terminal and a user attempts to connect to a specific APPLID, the session is created for that APPLID. Affinities are not globally known; they are held by the local IMS. When the user later reestablishes a session with the original IMS, significant status is restored.

Affinity deletion can be forced. Two IMS exits have been enhanced to allow deletion of affinities; the Signoff exit (DFSSGFX0) and the Logoff exit (DFSLGFX0). If the DC portion of IMS is cold started, existing affinities are deleted.

Keep in mind that it is your responsibility in a shared-queues environment to manage and control the deletion of significant status to keep terminal sessions consistent across IMS systems. If you have VTAM generic resources installed for static LTERMs, affinities are automatically maintained in VTAM's affinity table.

4.1.3 Front-End and Back-End IMS Systems

With local message queues, the IMS system that receives a message processes it, unless that IMS is set up to send the message to another remote IMS subsystem. This is still the same in shared queues, but transactions are processed locally only if the required definitions and resources are available.

It can be also efficient or necessary to split message handling and message processing across different IMS systems. Those IMS systems in a shared-queues group that contain only the network definitions for the terminals attached to them are called *IMS front-end systems*. They are responsible for the correct routing of input messages and responses and, if necessary, the routing of messages outside the shared-queues group.

For availability reasons you can define multiple front-end IMS systems to split your network. You can also clone your static terminal definitions in another front-end IMS system and define the front-ends to one VTAM generic resource group. The end users log on to IMS without any concern about the particular IMS subsystem to which they are connecting. The duplication of terminals can also be done by using ETO. This provides a single image to the end user and automatic session balancing. If an IMS front-end system fails, the alternate front-end IMS system is still available to the users.

A back-end IMS system is a workload processing engine. It has only application processing resources in the IMS system definition.

Any MPP and batch message processing (BMP) regions process and execute the application workloads in parallel. The shared queues design offers cloning of all application system definitions in the stage 1 deck for each IMS.

If you cannot clone your IMS system definitions because of existing affinities or for business reasons, then:

- Define the IMS system application workload profiles corresponding to your requirements or affinities

  or

- Clone your Stage 1 deck, but restrict registered interest to CQS for processing messages for that application by
  - Starting and/or stopping MPP regions serving the appropriate job class

    or

  - Starting and/or stopping the requisite transactions in each IMS system explicitly.

An IMS system that holds terminal sessions and process applications is called a front-end and back-end IMS system. It combines both routing and processing of messages. The terminal network is either replicated or partitioned in multiple IMS systems. In general, the user must know which IMS subsystems they must log on to to process their appropriate applications, unless a session manager such as VTAM generic resources is used to support direct logon to the appropriate IMS subsystem automatically.

4.2 IMS System Configuration Designs

In this section we cover the following IMS configuration alternatives:

- Vertical partitioning
- Horizontal partitioning
- Horizontal and vertical partitioning - a hybrid environment
- Horizontal and vertical partitioning with Multiple Systems Coupling (MSC)

4.2.1 Vertical Partitioning

With vertical partitioning, IMS applications are split and distributed among multiple IMS systems (see Figure 19 on page 39). These IMS systems may belong to one shared-queues group or run as separate IMS online systems remote or locally. Each specific application type is assigned for processing in just one IMS system. There is probably little data sharing among the systems.

Vertical partitioning can be used to isolate applications to a single IMS system. In the case of CPU capacity constraints in an existing environment, complete and consistent application subsets can be moved to another MVS system.

Vertical partitioning requires a thorough review of the application designs. Often, the database and application designs must be revised.

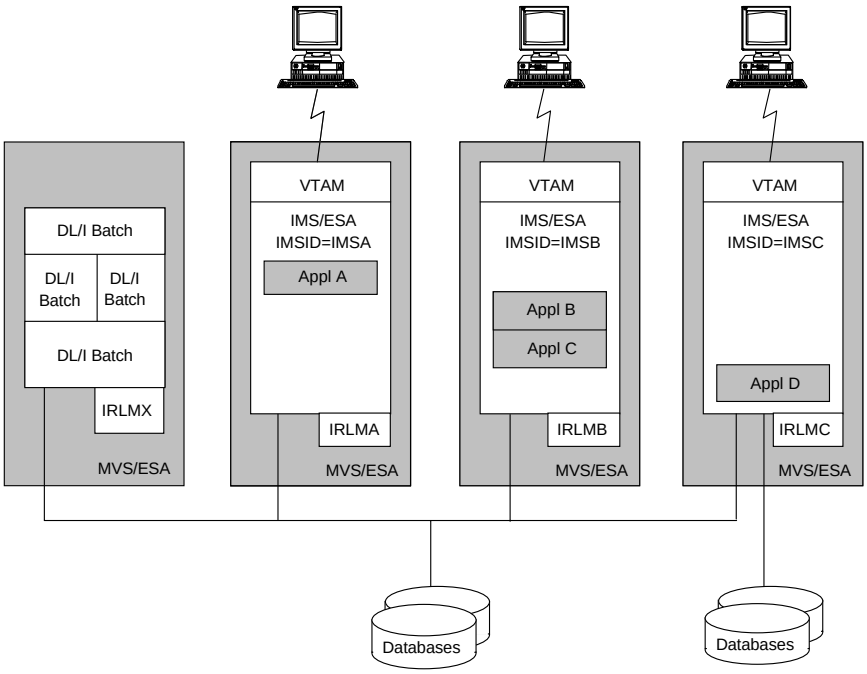


Figure 19. Vertical Partitioning of IMS Applications in a Sysplex

4.2.2 Horizontal Partitioning

With horizontal partitioning, identical applications are defined in each IMS system (see Figure 20 on page 40). Customers who use horizontal partitioning have already implemented data sharing to provide parallel access to their databases by multiple IMS systems.

In most cases, the IMS application system resources have been replicated or cloned. If one of the sharing participants fails, access to the application can still be gained through another IMS system. The terminal networks often are divided into geographical areas, and the users know to which IMS system their terminals belong to or for which IMS system their userids are authorized. In case of failures, one area may not be able to continue processing applications.

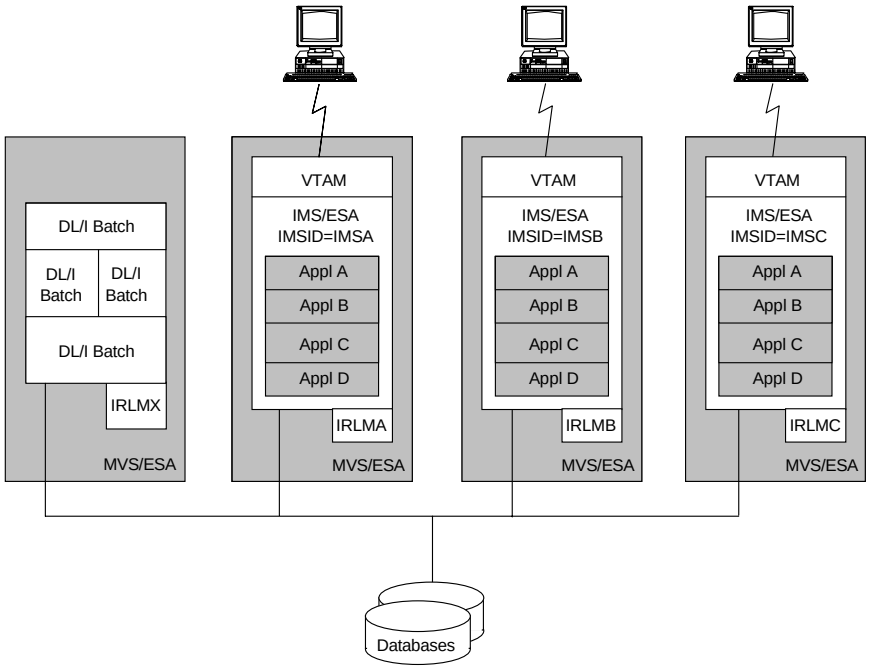


Figure 20. Horizontal Partitioning of IMS Applications across a Sysplex

4.2.3 Vertical or Horizontal Partitioning with Multiple Systems Coupling

Another IMS system configuration design uses MSC to connect multiple vertically or horizontally partitioned IMS subsystems.

MSC facilities allow full function message routing across multiple IMS systems. MSC operations are not supported for the Fast Path EMH.. Fast Path transactions must be processed locally (unless the IMS front-end switch is used with ISC).

In vertically partitioned IMS systems, applications execute in one system with their own unshared databases, or just a few common databases are shared. In horizontally partitioned IMS systems, all databases are shared, and identical application programs execute in more than one system. In each design, MSC can be used to:

- Transfer processing responsibilities among IMS systems in a vertically partitioned environment
- Statically balance the load among IMS systems in a horizontally partitioned environment.

MSC offers many advantages. The first design alternative uses MSC on a front-end IMS system to automate the transaction and message routing, either locally or remotely. The routing path (how a specific transaction or message flows between IMS systems) is determined by having the same SYSID defined for all APPLCTN and TRANSACT macros in each IMS system. No other application interaction is required.

Note that for MSC, geographically remote does not always imply that the IMS system is in a remote location. Remote means that IMS routes the transaction to another IMS subsystem across its MSC links.

A second design alternative with MSC is to split the network and applications into two systems, each acting as front-end and back-end systems. Each may also be coupled to other back-end IMS systems. Each IMS system processes its own application subset and, when required, the MSC links are used to route messages to other IMS systems in this complex.

When a transaction arrives at the front-end/back-end system, the transaction is checked to see whether it is local or remote. If it is defined as local (message is received and processed in same IMS system), the originating front-end/back-end system processes the transaction. If it is defined as remote, the message is sent through its assigned path to another IMS system. This other IMS system can be another front-end/back-end IMS system or any other back-end IMS system in the complex.

Front-end/back-end or back-end only IMS systems can be designed in either a horizontal or vertical partitioning approach.

Figure 21 is a sample configuration of vertical partitioning with MSC.

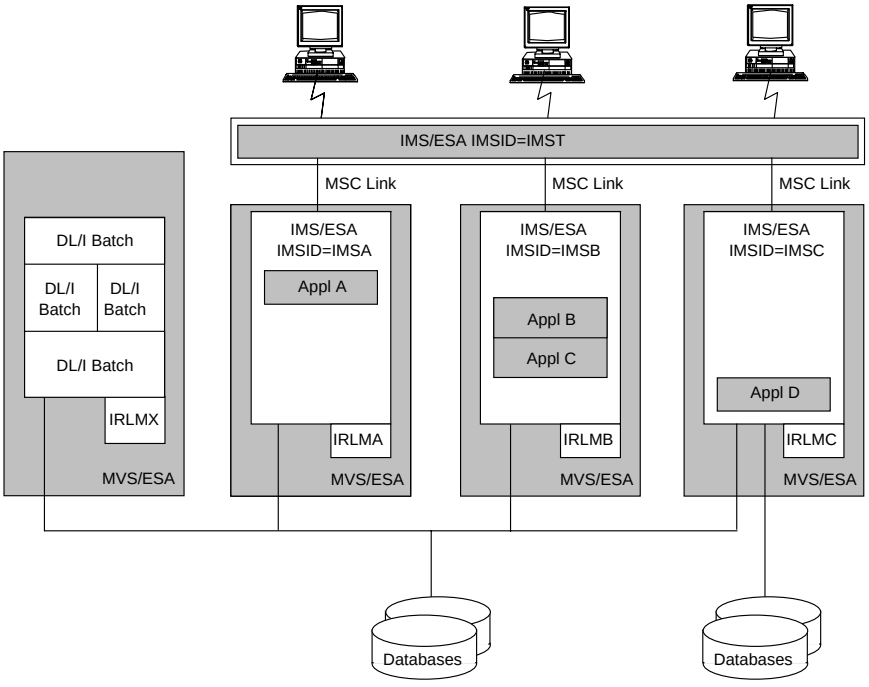


Figure 21. Vertical Partitioning of Applications with MSC

Note: If any of the systems share message queues, you cannot use MSC to route transactions among them.

Figure 22 on page 42 is an overview of the use of an horizontally partitioned IMS systems environment, where each IMS interfaces to the network and routes the workload across MSC links to the attached IMS systems.

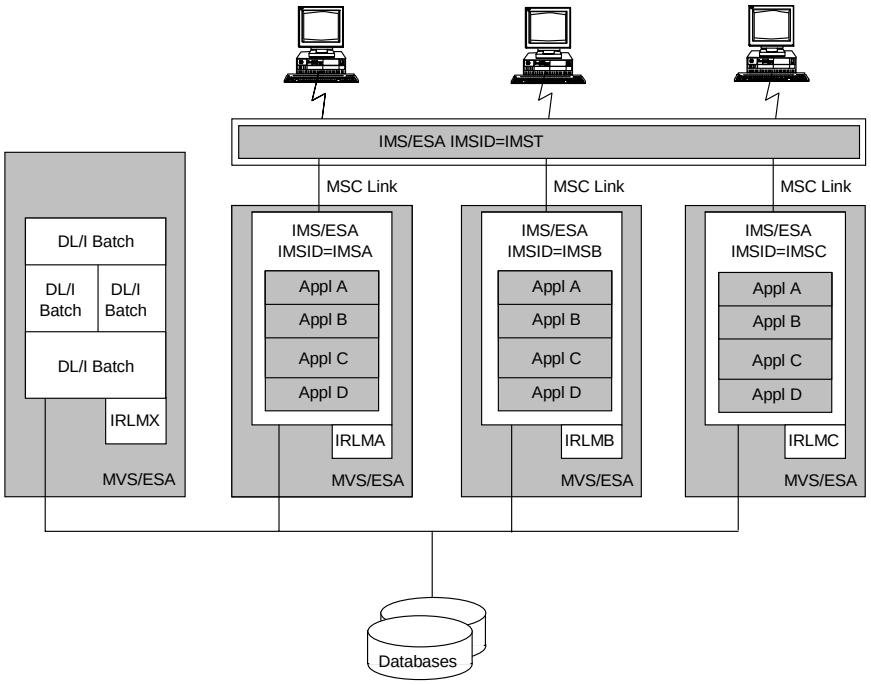


Figure 22. Horizontal Partitioning of Applications with MSC

4.2.4 Hybrid Environment

In reality it may be necessary to configure a combination of vertical and horizontal partitioning with or without MSC. Customers are running multiple IMS systems, where applications can be processed in any IMS system. Also, there are other applications that have to be processed in a dedicated single IMS system. These restrictions are typically based on existing affinities of applications to the system on which they run, such as

- Databases are distinct to geographical areas
- Data sharing has not yet been implemented
- Serialization of transaction processing is required
- A specific attachment to another subsystem is required.

A hybrid environment represents a mixture of cloned IMS system definitions and individual application definitions assigned to only one IMS system.

4.3 Sample Configurations

This section describes several potential configurations, and how they can be moved to a shared-queues environment.

4.3.1 Sample 1: Single IMS System

Most IMS customers, even in large organizations, have one or more noncoupled single IMS systems installed, either for historical reasons, such as having split their applications and databases into geographical areas, or because the total IMS workload can still be managed within a single CPU.

Within the single IMS system (Figure 23 on page 43), often a DB2 connection is established to provide access to DB2 databases, or a CICS/ESA attachment through DBCTL or ISC is set up to access the same IMS databases too.

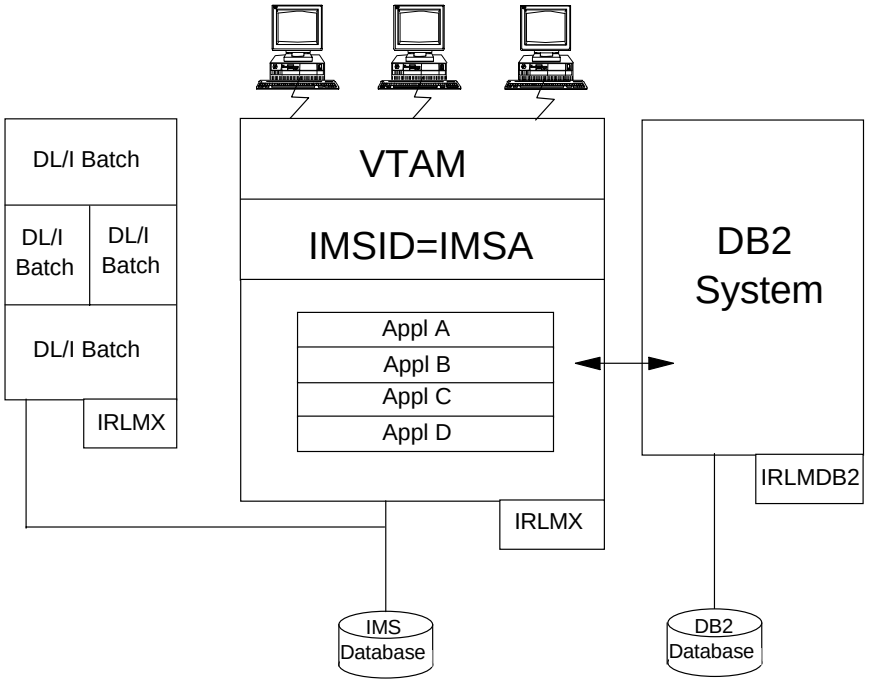


Figure 23. Sample 1: Single IMS System Environment

Figure 23 represents a front-end/back-end IMS system, where all of the required resources (such as terminal network, applications, transactions, multiple MPP regions, and BMP regions) are defined within one IMS system to process the total workload.

In general, the IMS system and application service levels are acceptable but the IMS implementation has some disadvantages in establishing a single point of failure. The constraints within a single IMS system configuration are:

- Implementing changes in system software or hardware or by modified or new application sets often affects the reliability and availability of the IMS online system and the application workload. Failures often produce a total outage and Service Level Agreements cannot be met.
- When new versions or releases of application programs are added to the production systems, the transaction response times deviate from normal, and the system can become instable and unbalanced.
- Network failures are not automatically recovered. In the sysplex environment, the user can log on to another IMS in the sysplex after an IMS failure.

Response times suffer due to CPU bottlenecks, especially in cases where high peak transaction volumes occur during the day shift or at month end. In a single CPU environment, some actions can be taken:

- Change priorities or stop less important transactions.
- Stop or reduce execution of non-IMS workload during critical periods.
- Install larger system hardware to support peak workloads.

Most customers are aware of these facts and set up their objectives to improve their availability and serviceability.

4.3.1.1 Sample 1: Alternative Shared Queues Configuration

Figure 24 presents an overview of an alternative shared-queues environment, migrated from a single IMS system.

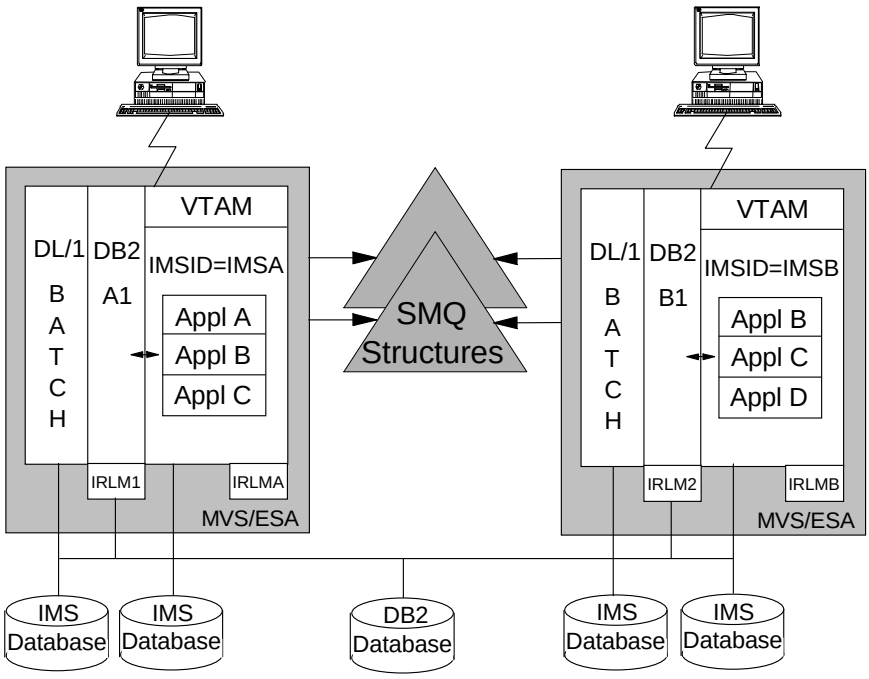


Figure 24. Sample 1: Alternative Shared Queues Configuration

The alternative configuration with shared queues has two IMS systems in a shared-queues group. Both have links to the coupling facility that shares the IMS message queues among them. The application sets (Appl B + Appl C) are cloned in both IMSA and IMSB. Assuming that Appl A and Appl D still require processing isolation (for whatever reason), they are still restricted to execute in a single system.

You may have cloned the entire Stage 1 system definition deck and isolated processing by assigning individual transaction classes to the particular MPRs or explicitly stopped those transaction codes for processing in that IMS system.

Data sharing for Appl B and Appl C has been implemented.

The terminal network may be replicated on each IMS system. The user does not have to know on which system their application runs. Shared queues ensures that the transactions run only on the system where they can run. Frequently the terminal network is split according to where the applications have to be processed.

We have assumed that DB2 data sharing has also been implemented. If you have not implemented data sharing, then the application assignment to each IMS system differs. The following migration steps need to be considered:

1. Analyze individual applications for existing affinities.
2. Assign priorities according to business importance.

3. Analyze the efforts and costs for alternatives.
4. Implement data sharing.
5. Set up the shared-queues environment according to your plan.

4.3.1.2 Advantages

In addition to greater flexibility and reduced In addition to greater flexibility and reduced single points of failure, the benefit of this shared queues configuration is in distributing the transaction workload to another system when it cannot be processed by one IMS system. Transaction peaks are absorbed by processing applications in multiple IMS systems in a shared-queues group.

Other benefits of this shared queues configuration are:

- Periodic transaction peaks can be handled by both IMS systems according to their available CPU resources.
- Planned or unplanned outages have reduced impact on the availability of other application systems.
- If one of the IMS system fails, applications are still available on the remaining IMS system.
- It is possible to increase your computing power in smaller increments than previously possible. Systems running on smaller CPUs can be added, and the workload is easily shared across the sysplex.
- Meeting future capacity requirements and workload growth is less complex and more simply implemented. You only have to extend the IMS sysplex by cloning another IMS component to it.

4.3.1.3 Disadvantages

The main disadvantages of this shared queues configuration include:

- Additional hardware must be installed.
- A Parallel Sysplex must be enabled.
- IMS data sharing must be implemented.
- Existing affinities must be eliminated.
- Enhanced Parallel Sysplex and shared-queues skills are required.

4.3.2 Sample 2: Multiple Horizontal Partitioning of IMS Systems

The multisystem IMS configuration shown in Figure 25 on page 46 is based on the following assumptions:

- The terminal network is divided between two single IMS systems.
- IMS application system resources are replicated or cloned on each IMS system. Therefore users have to know to which IMS system they connect.
- IMS data sharing is active.
- DB2 databases are replicated for access by each IMS system.
- A Parallel Sysplex has not been installed.

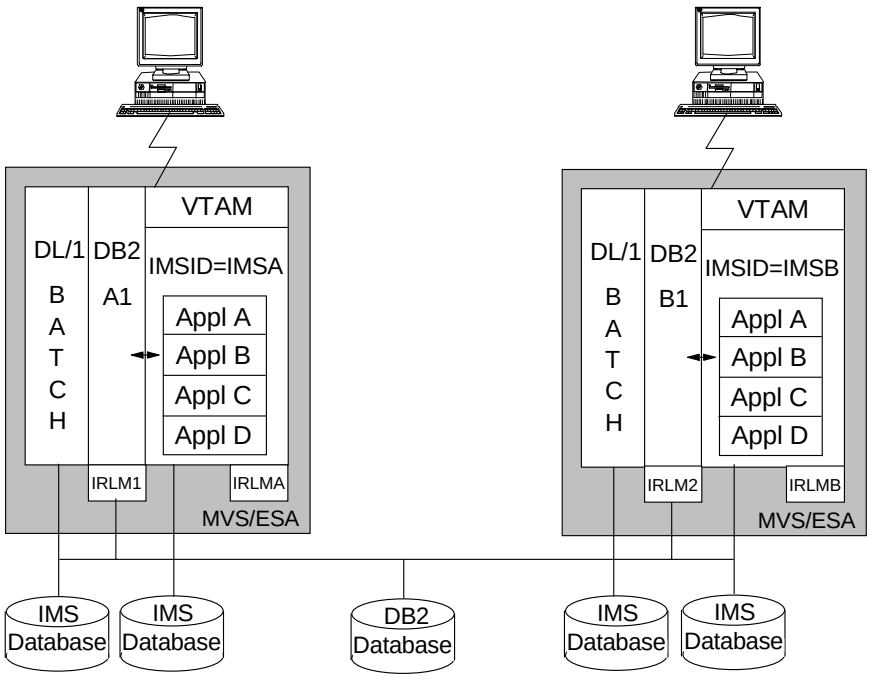


Figure 25. Sample 2: Multiple Horizontal Partitioning of IMS Systems

With horizontal partitioning, replicated IMS With horizontal partitioning, replicated IMS instances are to run on each IMS image. The implementation, however, has some weaknesses:

- The total amount of workload to each IMS system is dependent on the number of user signons and logons.
- If the IMS system fails and no session manager directs logon, half of the production workload cannot be further processed.
- If one IMS system fails, its messages are delivered only after it is restarted successfully. No further processing is possible for its users (unless they log on to the other IMS system).
- Future workload growth is limited to the single CPU capacity.
- In the event of bottlenecks, transaction workloads cannot be automatically offloaded to the other IMS.

4.3.2.1 Sample 2: Alternative Shared Queues Configuration

Figure 26 on page 47 shows an alternative shared-queues environment migrated from a multiple Horizontal partitioned IMS systems environment. The proposed IMS configuration has the Parallel Sysplex environment in production and has implemented shared queues.

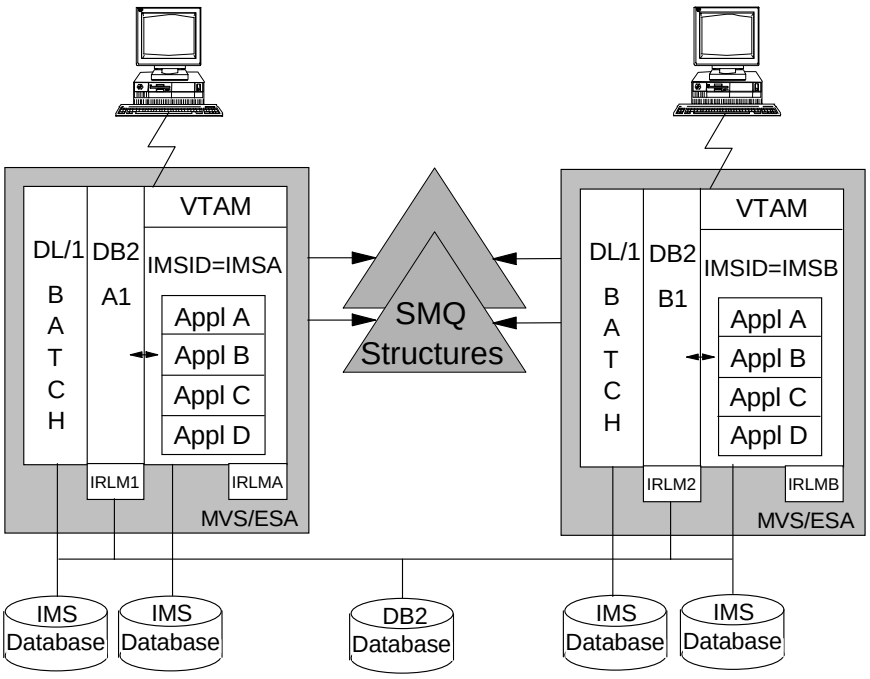


Figure 26. Sample 2: Alternative Shared Queues Configuration

Instead of block-level data sharing (BLDS) with pass the buck (PTB) notifications for lock requests and buffer invalidations between the integrated resource lock managers (IRLMs), data sharing and message sharing are set up in coupling facility structures. The SHMSG and LGMSG queue data sets in the single IMS systems are replaced by the shared queues and messages entered from either subsystem can be processed by any of the two IMS systems in the sysplex.

Nonlocal processing takes place when the originating IMS system does not have the available resources to process the transaction locally. The effect is improved flexibility in distributing and processing transactions in a single view of multiple IMS systems.

Migration to this shared-queues environment is somewhat more complex. Most application system resource definitions are already cloned in systems IMSA and IMSB. For fallback compatibility, keep the initial migration steps as simple as possible. In a subsequent consolidation, you can also clone your terminal network definitions. VTAM generic resources offers even more flexibility. The total network could dynamically establish new terminal sessions across the total IMS sysplex. Transaction processing continues on the surviving IMS system when system failures occur in the other IMS system.

4.3.2.2 Advantages

The major benefits achieved with this shared queues IMS configuration include:

- Improved workload balancing across both IMS systems. The workload is not dependent on its statically assigned is not dependent on its statically assigned terminal network.
- Incremental growth in CPU capacity is based on both IMS systems.
- Parallelism for application processing is extended across IMS boundaries.

- Higher availability for user signons and logons and their applications with the VTAM generic resources feature. If failures occur in a single IMS system, the user can log on to the surviving IMS.

4.3.2.3 Disadvantages

The only disadvantages of this configuration are the additional required coupling facility hardware and the implementation of Parallel Sysplex.

There may be an impact on existing operational procedures.

4.3.3 Sample 3: Multiple Vertical Partitioning of IMS Systems with MSC

The sample configuration in Figure 27 on page 49 is a complex design because of the high transaction volumes. These transaction volumes exceed the capacity of a single CPU for the aggregate of the applications. Each CPU is assigned a single IMS system processing an individual set of applications. The terminal network environment is subdivided in two parts and defined in each front-end/back-end IMS system (IMSA and IMSB).

Because the workload needs to be distributed across multiple IMS systems, MSC links are defined in each IMS system to facilitate communication across any IMS pair of systems in the IMS sysplex.

A message can flow to another IMS system in this IMS complex according to its routing definitions. The example presents only two back-end IMS systems. You can imagine that assigning other work, such as new application sets or the consolidation of printer output services, may be implemented by the addition of IMS systems.

Each IMS system has its own nonshared database sets with a few exceptions where databases must be shared among the four IMS systems. DB2 data is also not shared. All IMS production systems are already running in a Parallel Sysplex environment.

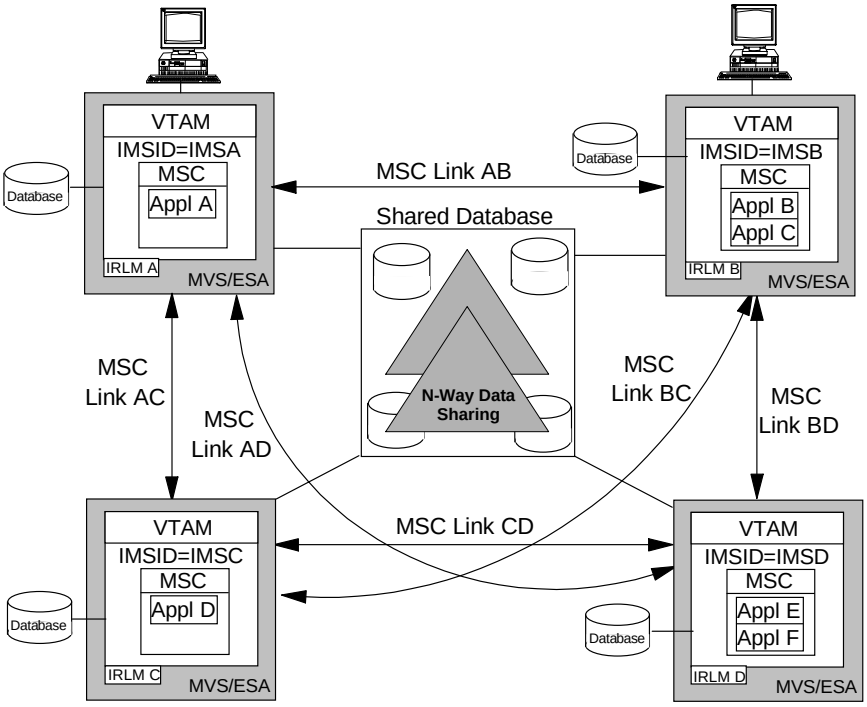


Figure 27. Sample 3: Multiple Horizontally Partitioning of IMS Systems with MSC

There are two major concerns in this environment. When the transaction volume for a single IMS When the transaction volume for a single IMS systems exceeds its hardware capacity, an additional IMS system with MSC links must be established. The complexity increases with application growth and new application sets. New dedicated application IMS systems increase the effort in maintaining and operating the IMS sysplex, as well as requiring more staff to support it.

The configuration is inflexible for assigning on-demand work to other IMS systems. When new applications need to be installed and the required CPU resources are not available on a single IMS, application transfers to other existing IMS subsystems become a difficult and complex task.

Additionally, the high MSC message traffic (MSC link transmission and IMS logging) also has performance impacts on transaction response times and single points of failure. Each IMS system is an atomic part, and its work cannot be taken over by another IMS system.

4.3.3.1 Sample 3: Alternative Shared Queues Configuration

Figure 28 on page 50 shows an alternative shared queues configuration migrated from multiple vertical partitioned IMS systems with MSC.

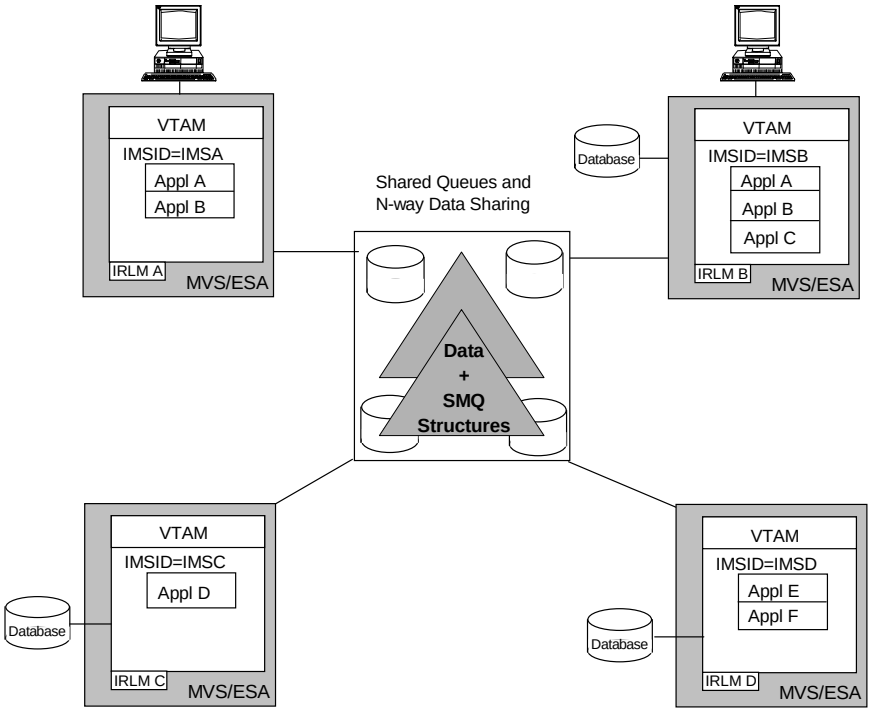


Figure 28. Sample 3: Alternative Shared Queues Configuration

This sample IMS sysplex configuration is less complex than Sample 2. All existing MSC links have been replaced by the shared queues. Transactions received in IMSA and IMSB that cannot be processed locally are stored in the shared queue structures for application processing in the IMSID=IMSC or IMSID=IMSD system.

Not all applications in this sample can be cloned for processing in any IMS system in the shared-queues group in one migration step. Therefore we extend the limited data sharing for APPL A and APPL B for both front-end IMS systems to allow both applications to be processed by the IMSA and IMSB systems.

Obviously the development of this alternative shared queues and the target configuration is based on hypothetical assumptions. But at least these objectives should be satisfied:

- MSC overhead and costs are reduced, and shared queues provide more flexibility.
- Long term planning means application subsets can be made available for processing in any IMS member in the IMS sysplex.
- Improved granularity of processing capacity within the IMS sysplex
- Fewer single points of failure

4.3.3.2 Advantages

The major advantages of this alternative shared queues configuration are:

- Eventually you can replace all of the current MSC links by setting up the shared-queues environment. Initially, keep the MSC definitions defined for fallback. The elimination of active MSC links reduces MSC transmission and logging overhead in the current IMS sysplex. When the new shared queues IMS sysplex is stabilized, expand its usage to add on new IMS members in the shared-queues group.

- Cloning the IMS system definitions in all IMS systems reduces the complexity of system maintenance and operations.
- You can create greater flexibility by extending the data sharing level for more applications, thereby achieving more parallel application processing across multiple IMS participants.
- Future workload demands can be managed by simply adding new IMS systems to the sysplex. the sysplex.
- Automatic workload balancing across the sysplex avoids overloading IMS systems.

4.3.3.3 Disadvantages

The constraints are the manpower effort of planning and implementing shared queues and a detailed analysis of affinities within existing applications. Thorough testing and a detailed migration plan must be set up for this complex environment.

4.3.4 Sample 4: Hybrid IMS Systems Environment with Remote MSC

The hybrid environment consists of multiple IMS DB/DC systems, where some applications run on different IMS systems and other applications are processed only in a single IMS system.

The sample configuration (Figure 29 on page 52) represents two cloned IMS systems with an additional, remotely located IMS system that has its own application subset. The local IMS systems have BLDS without a coupling facility installed. Applications R1 and R2 are processed in one remote IMS system. Remote processing is done either by transferring transactions from the IMSA and IMSB system or by transaction entry through its own network. The automatic transaction and message routing is done by MSC links across the IMS system complex.

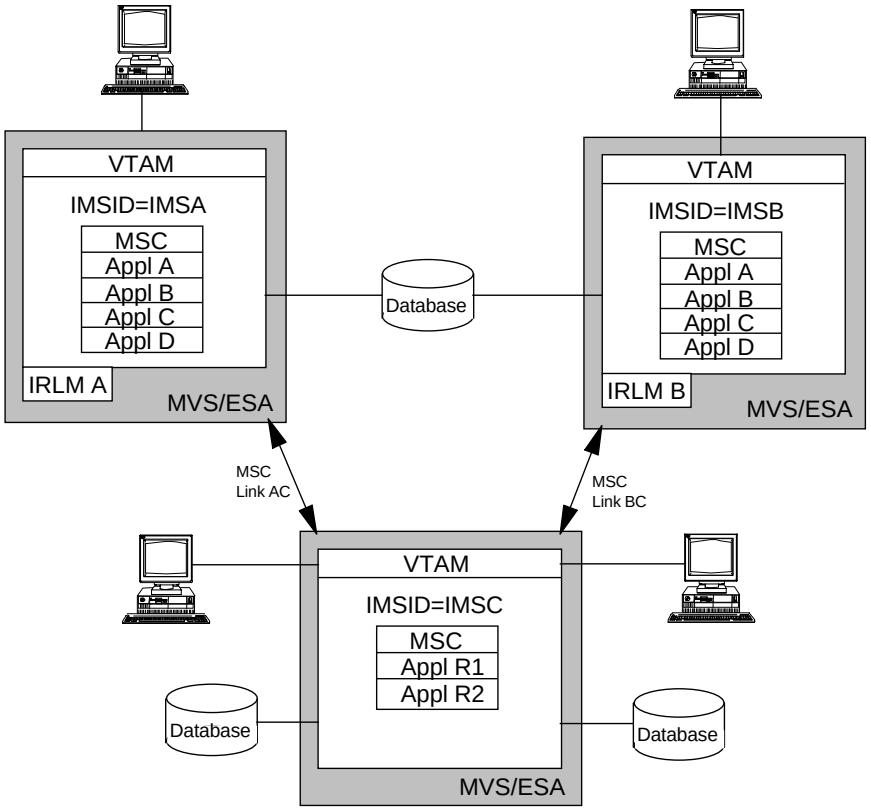


Figure 29. Sample 4: Hybrid IMS Systems Environment with Remote MSC

In this configuration application processing is shared between the two local IMS systems. The workload that each IMS system processes is dependent on its network activities. MSC routing between IMSA and IMSB is not defined, which implies unavailability of one IMS system when it fails.

We also assume that the total capacities of IMSA and IMSB are close to their limits.

4.3.4.1 Sample 4: Alternative Shared Queue Configuration

Figure 30 on page 53 shows an alternative shared-queues environment migrated from a hybrid IMS systems environment with remote MSC.

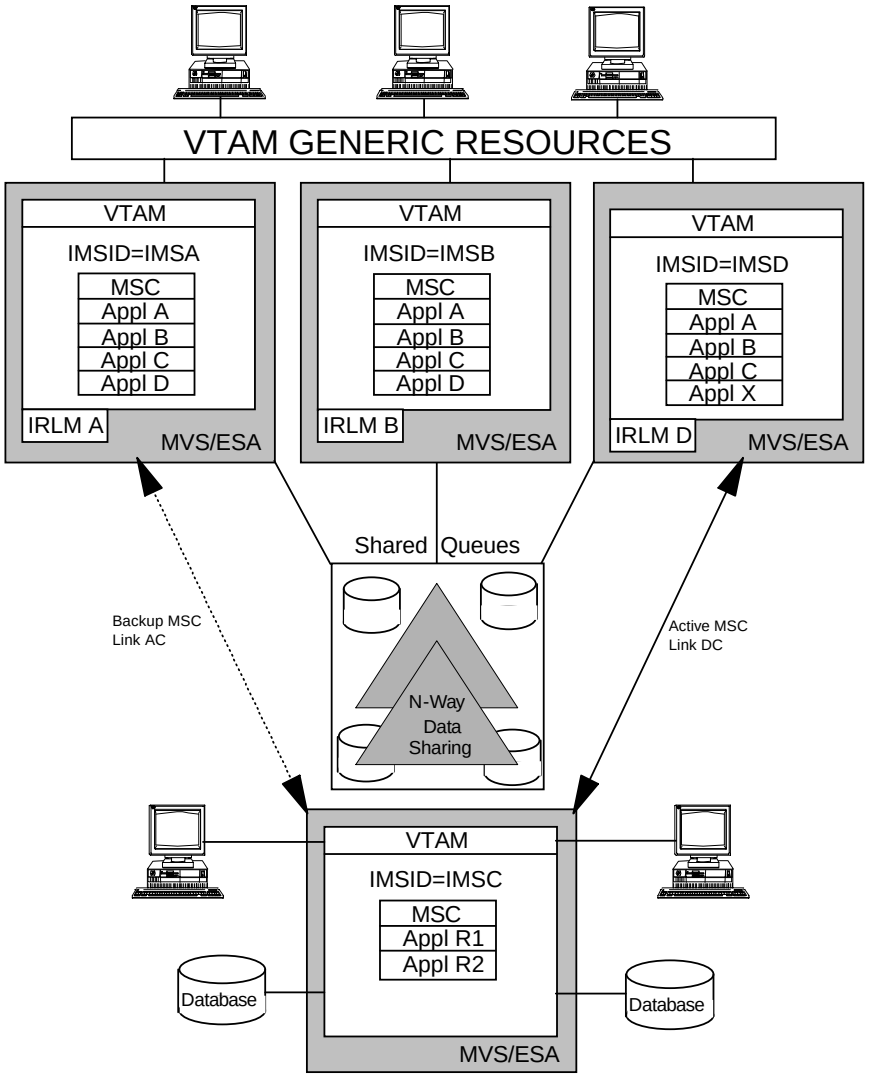


Figure 30. Sample 4: Alternative Shared Queues Configuration

On the basis of the existing constraints and the load forecast for the two local IMS systems, a shared queues implementation provides many improvements.

As the existing system definitions are already cloned for IMSA and IMSB, it is not difficult or complex to set up a third IMS system. The following activities are necessary to implement dynamic application balancing to any local IMS system:

1. Set up data sharing with the coupling facility.
2. Implement shared queues to distribute the transaction workload.
3. Implement VTAM generic resources to manage sessions dynamically.

4.3.4.2 Advantages

This alternative shared queues configuration provides the following benefits:

- The implementation and workload takeover are less complex with additional CPU hardware.
- With VTAM generic resources, there is much greater flexibility, in the case of system failures

- No extra MSC link required for third CPU. MSC links are shared in the shared-queues group.
- MSC input from the remote IMS system can be processed by any IMS system in the shared-queues group.
- Any IMS in the shared-queues group can send MSC output to the remote IMS system.

4.3.4.3 Disadvantages

The constraints are the manpower effort of planning and implementing shared queues and a detailed analysis of affinities within existing applications. Thorough testing and a detailed migration plan must be set up for this complex environment.

4.3.5 Conclusions

New design considerations are possible with shared queues. The objective is to optimize IMS workload throughput and performance within a single-image view of a Parallel Sysplex. Single points of failure are reduced by taking over workload in any other IMS system in an IMS sysplex. The maximum benefits are achieved for a shared-queues environment within a Parallel Sysplex, when

1. Data sharing for all IMS and DB2 databases is implemented.
2. All application and terminal affinities are removed.
3. Any application can be executed in any IMS system.
4. Any terminal can access any IMS system in the shared-queues group.
5. All participating IMS systems in the IMS sysplex contain identical IMS system definitions (cloned).
6. Each IMS member acts as a front-end/back-end IMS system. Whenever required IMS resources are available, application processing occurs locally. In all other, cases any IMS system in the shared-queues group can process the transactions.
7. No MSC extensions exist except for links to physically remote IMS systems.

Chapter 5. Implementing Shared Queues

The implementation of shared queues an IMS system requires additions and changes to your IMS startup parameters and IMS proclib members, the definition of some new data sets, and the definition of some structures in your coupling facility. In this chapter we describe what you need to set up to implement a shared-queues environment for your IMS system.

The steps that you have to perform to implement shared queues are outlined in 5.3, “Steps to Implement IMS Shared Queues” on page 62 and detailed in subsequent sections.

See 5.2, “Policies, Structures, and Data Sets Used for Shared Queues” on page 59 for a list of all the policies, structures, and data sets that you need to define for shared queues, as well as where they are referenced.

We also include information about falling back to the use of local message queues in 5.17, “Fallback to Local Message Queues” on page 87, and the activities you can perform to consolidate your IMS system definitions in 5.18, “Consolidating the IMS System Definitions” on page 87.

MVS definitions such as the coupling facility structures and the data sets required are included in this section for completeness. In many organizations, these definitions are created by the MVS systems programmers, in cooperation with the IMS systems programmers.

We assume that you have all the necessary OS/390 sysplex hardware and software in place to start your customization of IMS for shared queues. See *OS/390 Parallel Sysplex Configuration*, Volumes 1, 2, and 3 (SG24-2075, SG24-2076, and SG24-2077), and *OS/390 Setting Up a Sysplex*, GC28-1779 for information in how to set up a Parallel Sysplex.

5.1 Components Required

In this section we describe all of the components required to manage messages in a shared-queues environment (see Figure 31 on page 56). MSGQ stands for the full function structures, and EMHQ stands for Fast Path structures.

The shared queues components consist of:

- Coupling facility structures
  - MSGQ structures (primary and optional overflow)
  - EMHQ structures (primary and optional overflow)
  - MVS logging structures (MSGQ and optional EMHQ logging structure)
- CQS address space
  - One per IMS
  - Interface between IMS and shared queue structures
- CQS checkpoint data sets
  - Two per CQS (one for MSGQ and one for EMHQ)

  - Used for CQS restart
- Structure recovery data sets
  - Two pair per shared queues group (MSGQ and EMHQ)
  - Used with MVS log stream to recover structures in coupling facility
- MVS log streams
  - Two per IMS (MSGQ and optional EMHQ)
  - CQS logs MSGQ and EMHQ activities into the log streams
  - All CQSs use the same log streams
  - The log streams are implemented as:
    - MVS logging structures (MSGQ and optional EMHQ logging structure)
    - Structure offload MVS log data sets (MSGQ and optional EMHQ)
  - Used for structure recovery and CQS restart
- Optional MVS staging log data set

The key design goal of shared queues is to cache all incoming and outgoing messages in a set of list structures located in the coupling facility. The CQS manages and maintains for its client IMS all of the appropriate work for message processing, message logging, and checkpointing data required for system restart and recovery tasks.

The CQS logging function for shared queues is provided by the MVS system logger. Whenever any failures in an IMS sysplex occur, IMS with its CQS using MVS system functions takes control to provide the overall system and message integrity for the complete IMS sysplex.

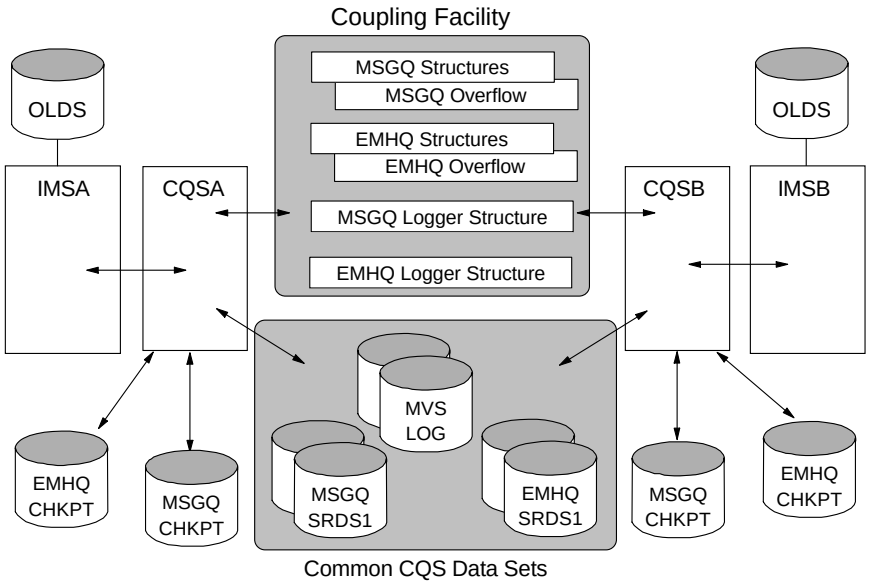


Figure 31. Components of a Shared-Queues Environment

5.1.1 Coupling Facility Structures

In a shared-queues environment, at least two list structures are defined, one MSGQ list structure to hold the full function messages and another to hold its CQS logging data produced by each IMS system in a MSGQ logging structure. When the FPCTRL macro is included in any of the IMS system definitions in the shared-queues group, an additional EMHQ list structure must be allocated to hold the corresponding EMH messages. Their logging data is stored in the EMHQ logging structure. Optionally you can define message overflow structures for both full function and Fast Path processing.

5.1.1.1 MSGQ and EMHQ Structures

The list structure is an area of storage in a coupling facility that enables multisystem applications to share data organized as a set of queues. The structure consists of a set of lists. These structures are defined in the Coupling Facility Resource Management (CFRM) policy. It is a set of rules and actions that systems in an MVS sysplex can follow when using MVS services.

The shared queues are represented by a structure pair which consists of a primary and an overflow structure. The overflow list structure is optional. The overflow structure holds shared queues that are moved off the primary list structure when it has reached a predefined threshold.

Full function application processing requires the definition of its own set of list structures. In a general view, the list structures are organized as a set of queue types. Each queue type represents a different type of work processing, such as `TRAN`, `LTERM`, `OTMA`, and `APPC` ready queues. To avoid contention or long chains in queue types, eleven list header anchor points are used to reference the first list entry in the queue. The list entries represent the message assignment or destination , such as `TRANX` or `LTERMY` or `LTERM7` or `TRAN1`, and their linked message data (composed of number of data elements).

5.1.1.2 MVS Logging Structure

For full function and/or Fast Path message processing with shared queues, each CQS in the shared-queues group logs with the MVS System Logger. The system logger merges a stream of log records written by multiple CQSs in a shared-queues group into a single log stream and writes it to the MVS logging structure in the coupling facility. Fast Path logging data is written to the Fast Path MVS logging structure. When no MVS staging log data sets are defined, a copy of the logged data is kept in a separate dataspace. The logging structure and other required external data sets ensure the recoverability and integrity for all full function and Fast Path messages in a shared-queues environment. In case of failures standard restart and recovery procedures are provided to ensure that full integrity and fast availability continue in processing applications within IMS.

5.1.2 Common Queue Server

The CQS is a generalized queuing facility to manage data objects (messages) in the shared queue. It is the defined interface for each IMS to communicate with shared queues. A CQS runs in a separate address space and is normally started up by its client—a single IMS TM system—automatically. Several different tasks are performed by the CQS on behalf of its clients. Some services provided by the CQS are:

- Notifies IMS when work exists for it

- Writes CQS system checkpoint information to a MSGQ or EMHQ checkpoint data set
- Writes structure checkpoint information to SRDS to recover shared queues list structures
- Manages the overflow and structure recovery process for message queue list structures
- Provides services for IMS, such as structure copy and structure recovery

IMS is always notified when the CQS becomes available or unavailable. For further details refer to Chapter 2, “Common Queue Server” on page 7.

5.1.3 CQS Checkpoint Data Sets

The CQS system checkpoint data sets (MSGQ CHKPT and EMHQ CHKPT) are two separate VSAM ESDSs that are dynamically allocated during CQS initialization. Whenever CQS system checkpoints are issued, status information is stored in these data sets. The status information is used during CQS restart.

The checkpoints are written when:

- A defined number of CQS log records have been written to the log stream in MVS logger structures
- The following IMS command is issued:

  ```
  /CQCHKPT SYSTEM
  ```

- After a structure checkpoint
- At CQS initialization and termination

5.1.4 Structure Recovery Data Sets

In case of coupling facility and structure failures or total loss of all CQSs connected to the structure, an SRDS is required to rebuild the structure in the coupling facility. SRDSs are shared across all CQS address spaces. There are two pairs per shared-queues group, one pair for checkpointing data from the full function list structures and another pair for copying the data objects from the Fast Path list structures. The VSAM defined data sets are written by an explicit IMS command or when the MVS log data set becomes full. The size of each pair must be at least the total size of the primary and overflow list structure (MSGQ Primary + MSGQ Overflow / EMHQ Primary + Overflow). One of the pair is dynamically allocated when a structure checkpoint is requested.

5.1.5 MVS Log Data Set

The MVS log streams consist of the MVS logger structure in the coupling facility and an external set of log data sets. On request the MVS System Logger address space dynamically allocates the required data sets; these are defined as VSAM linear data sets. When the MSGQ or EHMQ MVS logging structure (which contains the merged log stream from all CQSs) is filled to its threshold, the MVS system logger starts offload processing while continue writing the logger structures in the coupling facility. The oldest log data is written to the MVS log data set. It also deletes the copied data in the log structures because there is no need to keep them.

5.1.6 MVS Staging Log Data Set

In general, a copy of the log stream is written by the MVS system logger (one per MVS) to its own logger data space before it is written to the logging structure in the coupling facility. If no MVS staging log data sets are defined, recovery of a logging structure is done by each participating MVS system logger and its logger data space to rebuild the mixed log stream.

If there is a single point of failure (coupling facility and logger data space lost), MVS staging log data sets are used to rebuild the logging structure. The local MVS system logger writes the CQS log stream to its local MVS staging log data set instead of to a dataspace. The MVS staging log data set should be on DASD and accessible by all MVS system loggers in the shared-queues group.

You should carefully design your implementation to ensure the requisite level of system availability in the event of failure.

5.2 Policies, Structures, and Data Sets Used for Shared Queues

There are a number of coupling facility policies, structures, and data sets that need to be set up for a shared-queues environment. Figure 32 shows their location in the sysplex. The numbers shown in Figure 32 are used throughout this chapter.

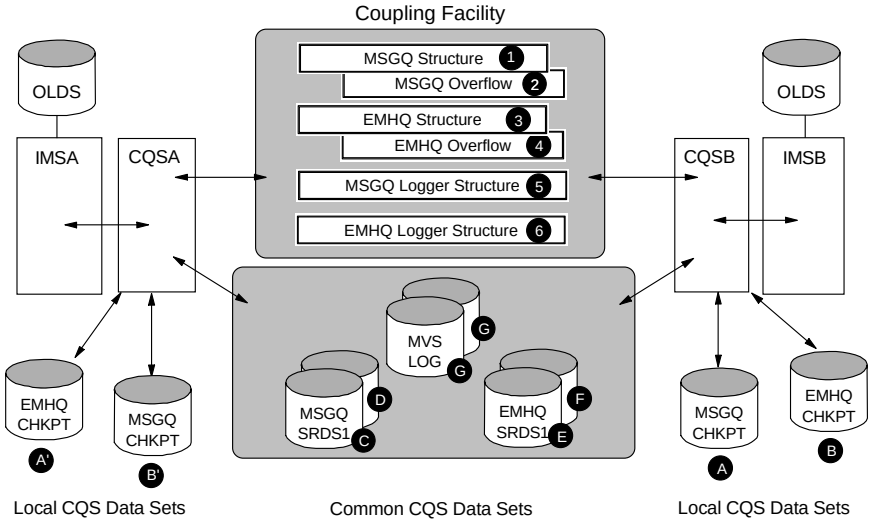


Figure 32. Structures and Data Sets in a Shared-Queues Environment

1 full function Message Queue Structures

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.2, “Defining the MSGQ Structures” on page 69).

Referenced: This structure is referenced in the CQS local definitions, which are specified in the CQSSLxxx proclib member (see 5.9.2.2, “Specifying the CQS Local Structure Definitions (CQSSLxxx)” on page 78).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

2 full function Message Queue Overflow Structure

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.2, “Defining the MSGQ Structures” on page 69).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

3 Fast Path Message Queue Structure

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.3, “Defining the EMHQ Structures” on page 70).

Referenced: This structure is referenced in the CQS local definitions, which are specified in the CQSSLxxx proclib member (see 5.9.2.2, “Specifying the CQS Local Structure Definitions (CQSSLxxx)” on page 78).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

4 Fast Path Message Queue Overflow Structure

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.3, “Defining the EMHQ Structures” on page 70).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

5 Full Function Logger Structure

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.4, “Defining the MVS Log Stream Structures” on page 70).

Referenced: This structure is referenced in the definition of the log stream for the LOGR policy (See 5.8.2.2, “Defining the Log Stream Structure” on page 72).

6 Fast Path Logger Structure

Defined: This structure is defined *Defined:* This structure is defined in the CFRM policy (see 5.8.1.4, “Defining the MVS Log Stream Structures” on page 70).

Referenced: This structure is referenced in the definition of the log stream for the LOGR policy (See 5.8.2.2, “Defining the Log Stream Structure” on page 72).

7 Full Function Log Stream

Defined: The log stream is defined in the LOGR Policy (see 5.8.2.2, “Defining the Log Stream Structure” on page 72).

Referenced: The log stream is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

8 Fast Path Log Stream

Defined: The log stream is defined in the LOGR policy (see 5.8.2.2, “Defining the Log Stream Structure” on page 72).

Referenced: The log stream is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

A Full Function CQS Checkpoint Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.1, “Allocating the CQS Checkpoint Data Sets” on page 64).

Referenced: This data set is referenced in the CQS local definitions, which are specified in the CQSSLxxx proclib member (see 5.9.2.2, “Specifying the CQS Local Structure Definitions (CQSSLxxx)” on page 78).

B Fast Path CQS Checkpoint Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.1, “Allocating the CQS Checkpoint Data Sets” on page 64).

Referenced: This data set is referenced in the CQS local definitions, which are specified in the CQSSLxxx proclib member (see 5.9.2.2, “Specifying the CQS Local Structure Definitions (CQSSLxxx)” on page 78).

C First Full Function Structure Recovery Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.2, “Allocating the Structure Recovery Data Sets” on page 64).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

D Second Full Function Structure Recovery Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.2, “Allocating the Structure Recovery Data Sets” on page 64).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

E First Fast Path Structure Recovery Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.2, “Allocating the Structure Recovery Data Sets” on page 64).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

F Second Fast Path Structure Recovery Data Set

Defined: You have to create this data set with IDCAMS (see 5.7.2, “Allocating the Structure Recovery Data Sets” on page 64).

Referenced: This structure is referenced in the CQS global definitions, which are specified in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

G MVS System Logger Staging Data Sets

Defined: If used, these data sets are dynamically created by the MVS system logger (see 5.7.4, “Allocating the MVS System Logger Offload Data Sets” on page 66).

Referenced: These data sets are used for a short time only and do not have to be backed up.

H MVS System Logger Offload Data Sets

Defined: These data sets are dynamically created by the MVS system logger when offloading data from the logger structure (see 5.7.4, “Allocating the MVS System Logger Offload Data Sets” on page 66).

Referenced: These data sets may be needed in the event of a structure failure and should be backed up in the same way as IMS logging data sets.

5.3 Steps to Implement IMS Shared Queues

The following steps must be completed before you start the CQS and use shared queues:

1. Install IMS software, as described in 5.4, “Install IMS Software” on page 63.
2. Review your IMS system definitions, as described in 5.5, “Review IMS System Definitions” on page 63.
3. Define the IMS and CQS execution data sets, as described in 5.6, “Define IMS Data Sets” on page 63 and 5.7, “Allocating the CQS Data Sets” on page 63.
4. Define the necessary structures in the CFRM policy, as described in 5.8.1, “Defining a CFRM Policy” on page 67.
5. Define the MVS log stream in the MVS system logger (LOGR) policy. See 5.8.2, “Defining the MVS Log Stream” on page 71.
6. Implement the CFRM and LOGR policies. See 5.8.3, “Implementing CFRM and LOGR Policies” on page 73.
7. Customize the CQS environment. See 5.9.2, “Specifying the CQS Parameters” on page 78.
8. Customize the IMS environment. See 5.9.1.1, “Specifying the IMS Control Region Execution Parameters (DFSPBxxx)” on page 75.
9. Authorize connections to CQS structures. See 5.15, “Authorizing Connections to CQS” on page 85.
10. Update the program properties table (PPT). See 5.14, “Updating the MVS Program Properties Table” on page 85.

11. Activate the CFRM policy. See 5.16, “Preparing to Start IMS and CQS” on page 86.
12. Start IMS (and CQS). See 7.1, “Operating Procedures” on page 99.

5.4 Install IMS Software

The BPE and CQS code modules are delivered as part of the IMS installation software. They are link-edited into the RESLIB library at install time when the JCLIN job stream is run.

5.5 Review IMS System Definitions

The shared queues function is activated with the IMS startup parameters. No changes are required in your IMS system definition. New parameters have been added to the IMS startup parameters to override system definition macros that are included in the IMS system definition. This removes the dependency of shared queues implementation on IMS system definition. Implementing shared queues does not require any additions to the IMS system definition, nor should any existing macros be removed.

A new address space is required in a shared queues environment. The CQS is defined as a separate address space and uses coupling facility structures for the shared queues. Refer to Figure 53 on page 90 for a detailed diagram of all parameters and how they fit into the IMS and CQS address spaces.

5.6 Define IMS Data Sets

The short and long message queue data sets and the queue blocks data set (SHMSG, LGMSG, and QBLKS, respectively) are not required with shared queues. We recommend, however, that you leave these data sets in the IMS procedure to ensure that all parameters have been set correctly. Most of the new parameters in the DFSPBxxx proclib member default to the data set control block (DCB) of the queue data sets if not specified in the MSGQUEUE macro in the IMS system definition. With the data sets specified in the IMS procedure, fallback from shared queues is almost as easy as changing one parameter in the DFSPBxxx proclib member.

5.7 Allocating the CQS Data Sets

CQS has some local data sets, as well as shared data sets for logging and checkpointing information. This discussion is logically divided into local and shared data sets. Some of the data sets can be defined by the IMS systems programmer, but most have to be defined by the MVS system programmer. There are many data sets that have to do with the logging of the information stored in the coupling facility, and responsibility therefore should lie within the MVS arena.

The local data sets that need to be defined are:

- MSGQ checkpoint data set
- EMHQ checkpoint data set
- MVS system logger staging data set

Two of each of the shared data sets need to be defined:

- MSGQ structure recovery data sets
- EMHQ structure recovery data sets
- MVS logger DASD log data sets (offload data sets)

5.7.1 Allocating the CQS Checkpoint Data Sets

The CQS checkpoint data sets contain information used in the restart of the CQS. The data sets contain the log tokens in the MVS log stream of the CQS checkpoints. There is a separate data set for EMHQ checkpoint information and MSGQ checkpoint information. The data sets can be small, and they are VSAM ESDSs. The CHKPTDSN subparameter in the STRUCTURE parameter of the CQSSLxxx proclib member is where the name of the data set is defined. The CQS dynamically allocates the checkpoint data sets when a CQS checkpoint is requested.

Figure 33 shows a sample definition for the allocation of the CQS MSGQ checkpoint data set. The EMHQ checkpoint data set would be defined in the same way.

The checkpoint data sets are local to the CQS and therefore do not have to be shared across the sysplex.

```
DEFINE CLUSTER
   (NAME(CQS1.MSGQ.CHKPT)  -                  A
   TRK(1 1)                -
   VOL(IMSSQV)             -
   NONINDEXED              -
   SHAREOPTIONS(3,2)       -
   RECSZ(505,505)          -
   REUSE                   -
   CISZ(512) )
DEFINE CLUSTER
   (NAME(CQS1.EMHQ.CHKPT)  -                  B
   TRK(1 1)                -
   VOL(IMSSQV)             -
   NONINDEXED              -
   SHAREOPTIONS(3,2)       -
   RECSZ(505,505)          -
   REUSE                   -
   CISZ(512) )
```

Figure 33. Checkpoint Data Set Definition

5.7.2 Allocating the Structure Recovery Data Sets

SRDSs are dynamically allocated when a structure checkpoint is requested, and they are dynamically deallocated when the structure checkpoint has been completed. SRDSs are VSAM ESDSs. They must be large enough to hold the contents of both the primary and overflow message queue structures. If EMHQ is used, two pairs of SRDSs must be allocated; one to be used for the recovery of the MSGQ structure and one for the recovery of the EMHQ structure.

SRDSs are allocated in pairs. Each structure has two SRDSs, as the CQS keeps two sets of structure checkpoints. The CQS keeps the last two structure checkpoints to ensure that recovery can take place even if one of the data sets cannot be used. If the data set containing the last structure checkpoint is unusable, the CQS uses the second last SRDS to recover from.

Because SRDSs can be read and updated by any of the CQSs in the shared-queues group, they must be shared across the sysplex.

Figure 34 shows a sample definition for the allocation of the CQS SRDS.

```
DEFINE CLUSTER
   (NAME(CQS.MSGQ.SRDS1)           -          C
   TRK(100 10)                     -
   VOL(IMSSMQ)                     -
   NONINDEXED                      -
   SHAREOPTIONS(3,2)               -
   RECSZ(32761,32761)              -
   REUSE                           -
   CISZ(32768) )                   -
 DATA                              -
   (NAME(CQS.MSGQ.SRDS1.DATA)      -
    CONTROLINTERVALSIZE(32768))
DEFINE CLUSTER
   (NAME(CQS.MSGQ.SRDS2)           -          D
   TRK(100 10)                     -
   VOL(IMSSMQ)                     -
   NONINDEXED                      -
   SHAREOPTIONS(3,2)               -
   RECSZ(32761,32761)              -
   REUSE                           -
   CISZ(32768) )                   -
 DATA                              -
   (NAME(CQS.MSGQ.SRDS2.DATA)      -
    CONTROLINTERVALSIZE(32768))
DEFINE CLUSTER
   (NAME(CQS.EMHQ.SRDS1)           -          E
   TRK(100 10)                     -
   VOL(IMSSMQ)                     -
   NONINDEXED                      -
   SHAREOPTIONS(3,2)               -
   RECSZ(32761,32761)              -
   REUSE                           -
   CISZ(32768) )                   -
 DATA                              -
   (NAME(CQS.EMHQ.SRDS1.DATA)      -
    CONTROLINTERVALSIZE(32768))
```

Figure 34 (Part 1 of 2). Structure Recovery Data Set Allocation

```
DEFINE CLUSTER
   (NAME(CQS.EMHQ.SRDS2)          -          F
   TRK(100 10)                    -
   VOL(IMSSMQ)                    -
   NONINDEXED                     -
   SHAREOPTIONS(3,2)              -
   RECSZ(32761,32761)             -
   REUSE                          -
   CISZ(32768) )                  -
 DATA                             -
   (NAME(CQS.EMHQ.SRDS2.DATA)     -
    CONTROLINTERVALSIZE(32768))
```

Figure 34 (Part 2 of 2). Structure Recovery Data Set Allocation

5.7.3 Allocating the MVS Logger Staging Log Data Sets

The staging log data set is used to back up information that has been stored in the coupling facility but not yet logged to the DASD log data sets (also called the offload data sets). The staging log data set is optional for a nonvolatile coupling facility.

Carefully review whether you want to use staging log data sets. They can reduce the potential data loss in the event of a catastrophic failure, but they may also affect logger system performance.

We recommend that you plan for the use of staging log data sets, even if you do not intend to use them for duplexing the storage of the interim log data. The MVS system logger automatically allocates a staging log data set for a log stream under specific error conditions (see *OS/390 Setting Up a Sysplex*, GC28-1779, for more information).

The staging log data sets are local to the CQS and therefore do not have to be shared across the sysplex.

5.7.4 Allocating the MVS System Logger Offload Data Sets

The MVS system logger offload data sets are implicitly defined in the LOGR policy with the DEFINE LOGSTREAM command. A sample LOGR policy is shown in Figure 38 on page 72. For details of the definition of the LOGR policy, see 5.8.2.1, “Defining the Logging Structure” on page 71 or *OS/390 Setting Up a Sysplex*, GC28-1779.

Because the MVS system logger offload data sets can be read and updated by any of the IMS systems in the shared-queues group, they must be shared across the sysplex.

5.8 Defining Coupling Facility Structures

All structures to be used in a Parallel Sysplex All structures to be used in a Parallel Sysplex must be defined in the CFRM policy. In a shared-queues environment, you have to define structures for:

- Message queue structure pair (primary and overflow)
- EMHQ structure pair (primary and overflow)
- MVS logger structure for the ff message queues
- MVS logger structure for the fast path message queues

IMS needs access to at least one coupling facility structure for the full function message queue. If the IMS definition includes Fast Path, a Fast Path message queue must be defined. Overflow structures for both of the structures can and should be defined, in case the primary structure fills up. See 2.8.1, “Overflow Processing” on page 16.

5.8.1 Defining a CFRM Policy

To enable shared queues, you must define a CFRM To enable shared queues, you must define a CFRM policy. The CFRM policy holds information about the structures used for the MSGQ, EMHQ, and system logger.

Use the couple data set format utility (IXCL1DSU) to format the couple data set, and the MVS administration utility (IXCMIAPU) to define the CFRM policy into the couple data set. The couple data set must be accessible to all members of the sysplex.

The active CFRM policy definition may be updated with the new structures for the CQS, but the policy must be reactivated to implement any changes.

The parameters used to define a coupling facility list structure are given below. See 5.8.3, “Implementing CFRM and LOGR Policies” on page 73 for the JCL used to run the MVS administration utility. More detail can be found in *OS/390 Setting Up a Sysplex*, GC28-1779.

NAME Name of the coupling facility structure to be created

SIZE Maximum size of the structure to be allocated, in units of 1 KB (1024 bytes)

INITSIZE Represents the initial amount of space allocated for the structure, in units of 1 KB (1024 bytes). This parameter is dependent on the release level of the coupling facility structure.

Carefully choose the initial size of the structure. It may be preferable to avoid expansion of the structure while your IMS system is active, by either avoiding the INITSIZE parameter or setting it to the same value as the SIZE parameter. A structure should have a fixed size, and be allocated upfront, first to ensure that the capacity exists on the coupling facility, and second to eliminate unnecessary overhead in extending the coupling facility structure.

The INITSIZE parameter value should be chosen in conjunction with the STRMIN value, specified in the CQSSGxxx proclib member. STRMIN does not override the INITSIZE parameter. The structure is allocated as defined in the CFRM policy if space on the coupling facility allows. STRMIN is used to ensure that a minimum structure size is allocated by MVS. Ensure that a structure of this minimum size is usable in your environment.

REBUILDPERCENT Determines the level of loss of connectivity at which structure rebuild takes place

When the REBUILDPERCENT parameter is set to 25, an attempt to rebuild the structure on a candidate alternative coupling facility is made when 25% of the connectivity to the structure has been lost. This percentage is based on a weighting factor. Refer to *OS/390 Setting Up a Sysplex*, GC28-1779, for a detailed explanation of this parameter.

We advise avoiding structure rebuilds at critical times. It is better to schedule structure rebuilds (if possible) at a time when the other IMS systems in the same shared-queues environment can maintain the expected level of service. A structure rebuild is an expensive overhead in terms of length of time for structure quiesce.

PREFLIST Determines the preference list of coupling facilities. The structure being defined may be allocated on any of the coupling facilities defined in the preference list.

EXCLLIST Determines the exclusion list for the structure. If possible, the structure being defined is not allocated on a coupling facility that already contains one of the structures in the exclusion list.

5.8.1.1 Sizing Coupling Facility Structures

There are guidelines for sizing structures in the coupling facility in Appendix B of the *OS/390 PR/SM Planning Guide*, GA22-7236. Several formulae are described there, and you need the following information to correctly size structures used by the CQS with IMS Version 6:

MDLES 120

LELX 1

LC 192

LTEC 192

R_le 1

R_data (OBJAVGSZ+24)/512

OBJAVGSZ is defined in the CQSSGxxx proclib member (see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79).

Notes:

1. These parameters are release-dependent and may change in future releases of IMS.
2. Control space is contained within the structure. The larger the structure, the larger the control space.
3. A structure dump shows you the maximum EMCs (maximum queue registrations).

Having defined a structure, you can use the `/CQQUERY STATISTICS` command to show the maximum number of list entries and maximum number of data elements for the structure. This is equivalent to the maximum number of data objects (messages and transactions) that can be stored in the structure.

5.8.1.2 Defining the MSGQ Structures

The MSGQ structures are defined as part of the CFRM policy.

The MSGQ structure should be defined with an overflow structure to ensure continuous availability in the event the primary structure is filled. The structure is defined with the parameters described in the 5.8.1, “Defining a CFRM Policy” on page 67. Figure 35 shows a sample policy for the MSGQ structure and the overflow MSGQ structure.

The MSGQ structure name must be the same name as specified in the STRNAME parameter in the CQSSGxxx proclib member for the primary MSGQ structure. You should also define a structure for the message queue overflow. The overflow structure name must match the value of the OVFLWSTR parameter in the CQSSGxxx proclib member.

```
DATA TYPE(CFRM)
:
DEFINE POLICY NAME(POLICY1)
   :
   STRUCTURE NAME(STRMSGQ01)           1
             SIZE(100000)
             INITSIZE(100000)
             REBUILDPERCENT(10)
             PREFLIST(CF01 CF02)
             :
             :
   STRUCTURE NAME(STRMSGQ01OVFLW)      2
             SIZE(40000)
             REBUILDPERCENT(10)
             PREFLIST(CF01 CF02)
```

Figure 35. Structure Definition for a Full Function Message Queue

5.8.1.3 Defining the EMHQ Structures

The EMHQ structures are defined as part of the CFRM policy.

The EMHQ structure must be defined if the IMS system is Fast Path capable. The EMHQ structure should be defined with an overflow structure to ensure continuous availability in the event the primary structure is filled. The structure is defined with the parameters described in 5.8.1, “Defining a CFRM Policy” on page 67.
Figure 36 shows a sample policy for the EMHQ structure and the overflow EMHQ structure.

This EMHQ structure name must be the same name as specified in the STRNAME parameter in the CQSSGxxx proclib member for the primary EMHQ structure. You should also define a structure for the EMHQ overflow. The overflow structure name must match the value of the OVFLWSTR parameter in the CQSSGxxx proclib member.

```
DATA TYPE(CFRM)
:
DEFINE POLICY NAME(POLICY1)
   :
   STRUCTURE NAME(STREMHQ01)          3
             SIZE(100000)
             INITSIZE(100000)
             REBUILDPERCENT(10)
             PREFLIST(CF01 CF02)
             :
             :
   STRUCTURE NAME(STREMHQ01OVFLW)     4
             SIZE(40000)
             REBUILDPERCENT(10)
             PREFLIST(CF01 CF02)
```

Figure 36. Structure Definition for a Fast Path EMH Queue

5.8.1.4 Defining the MVS Log Stream Structures

The MVS system logger structures for CQS log streams are defined in the same way as the CQS queue structures. Figure 37 on page 71 shows a sample definition of the structure to which the system logger writes CQS messages. Use the descriptions in 5.8.1, “Defining a CFRM Policy” on page 67 as a guide to defining the log stream coupling facility structures.

A separate structure needs to be defined for the MVS system logger, and its name must match the LOGNAME parameter in the CQSSGxxx proclib member.

```
DATA TYPE(CFRM)
   :
   DEFINE POLICY NAME(POLICY1)
    :
    STRUCTURE NAME(MSGQLOGSTR)          5
          SIZE(50000)
          REBUILDPERCENT(10)
          PREFLIST(CF01 CF02)
          :
    STRUCTURE NAME(EMHQLOGSTR)          6
          SIZE(50000)
          REBUILDPERCENT(10)
          PREFLIST(CF01 CF02)
          :
```

Figure 37. CFRM Policy Definition for MVS Log Stream Structures

5.8.2 Defining the MVS Log Stream

CQS uses the MVS system logger to log all recoverable information. The MVS system logger makes use of a single log stream in which all CQSs in a shared-queues environment store information. A CFRM structure, where all logging information is placed, must be defined. The LOGR policy is used to define the log streams associated with recovery, and it associates the log stream with a log structure. These resources are described in the sections that follow.

There are two sets of logging structures, one for ff message queues, and one for Fast Path message queues if Fast Path is defined in the IMS system definition.

5.8.2.1 Defining the Logging Structure

IMS and the CQS require the logging structure in the event of a recovery of messages on the shared queues. The MVS system logger merges all CQS log records into one log stream. The MVS log stream is defined in the LOGR policy. The parameters required to define the MVS log streams are described below.

LOGSTREAM Requests that an entry for a log stream be defined in the LOGR policy

NAME Name of the MVS log stream to which the CQS requests the MVS system logger to log. This parameter must be the same as the LOGNAME parameter in the CQSSGxxx proclib member.

STG_DUPLEX Represents whether to log to a staging log data set as well as to the MVS log stream structure. This parameter is used with the DUPLEXMODE parameter to determine the scope of duplex logging. Evaluate whether you want to use duplex logging (for enhanced availability).

DUPLEXMODE Represents conditional duplexing scenarios. If this parameter is set to COND and the STG_DUPLEX parameter is set to Y, the staging log data set or duplexing of logging is performed only if the coupling facility is volatile (has no battery backup). If the parameter is set to UNCOND, the log is always written to the data set, not to the data space.

STG_SIZE Size of the staging log data sets in 4 KB increments

LS_SIZE Size of the offload data set in 4 KB increments

HIGHOFFLOAD Percentage full at which the logger structure is offloaded to the offload data sets

LOWOFFLOAD Represents how much of the structure is offloaded each time the HIGHOFFLOAD percentage is reached

We recommend setting this percentage to 0% so that each time the structure is offloaded, the entire structure that has been used is removed from the structure.

Other parameters are also included in the definition of the MVS log stream but are not necessary in this discussion. For more information, contact your MVS systems programmer or refer to *OS/390 Setting Up a Sysplex*, GC28-1779.

Figure 38 shows an extract of the LOGR policy definition with some of the parameters that have been discussed in this section. This figure shows the definition of the MVS log streams that the CQSs in a shared-queues environment use for recovery purposes.

```
DATA TYPE(LOGR)
   :
   :
   DEFINE LOGSTREAM NAME(CQGRP01.MSGQ.LOG)          7
            STRUCTNAME(MSGQLOGSTR)                  5
            DUPLEXMODE(COND)
            STG_SIZE(125000)
            LS_SIZE(125000)
            HIGHOFFLOAD(50)
            LOWOFFLOAD(0)
            :
            :
   DEFINE LOGSTREAM NAME(CQGRP01.EMHQ.LOG)          8
            STRUCTNAME(XMHQLOGSTR)                  6
            DUPLEXMODE(COND)
            STG_SIZE(125000)
            LS_SIZE(125000)
            HIGHOFFLOAD(50)
            LOWOFFLOAD(0)
            :
            :
```

Figure 38. Log Stream Definition in the LOGR Policy

5.8.2.2 Defining the Log Stream Structure

The MVS log stream structure must be defined in the LOGR policy. The parameters to be used are detailed below.

STRUCTNAME Name of the log structure defined in the CFRM policy

AVGBUFSIZE The average size of the buffer that the CQS writes to the MVS logstream

MAXBUFSIZE The maximum size of the buffer that the CQS writes to the MVS log stream. This size must be less than or equal to 64 KB and should be at least 32 KB.

LOGSNUM Number of MVS log streams that can share the structure. In a CQS shared-queues environment, this number must be 1.

Figure 39 shows the description of the CFRM structure that the MVS logger uses in logging CQS message queue and EMHQ information.

```
DATA TYPE(LOGR)
   :
   :
   DEFINE STRUCTURE NAME(MSGQLOGSTR)          5
            AVGBUFSIZE(4096)
            MAXBUFSIZE(65536)
            LOGSNUM(1)
            :
            :
   DEFINE STRUCTURE NAME(XMHQLOGSTR)          6
            AVGBUFSIZE(4096)
            MAXBUFSIZE(65536)
            LOGSNUM(1)
            :
            :
```

Figure 39. LOGR Structure Description

5.8.3 Implementing CFRM and LOGR Policies

The CFRM and LOGR policies must be converted from descriptive parameters into an executable form from which they can be activated. Therefore you must update the CFRM and LOGR policy couple data sets with the MVS administration utility (IXCMIAPU). Each CFRM and LOGR policy is stored in a separate data set. The couple data sets are updated with the execution of this utility, but the structures are not created until the policy is activated and someone connects to the structure. The changes can be made to the active policy, but the policy must be reactivated to allocate the structures and implement the changes. See 5.11, “Activating the CFRM Policy” on page 83 and *OS/390 Setting Up a Sysplex*, GC28-1779, for more information. Figure 40 on page 74 shows the JCL required to implement the CFRM policy.

```
//POLICY   EXEC PGM=IXCMIAPU
//STEPLIB  DD   DISP=SHR,DSN=SYS1.MIGLIB
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
DATA TYPE(CFRM)
DEFINE POLICY NAME(POLICY1)
   STRUCTURE NAME(STRMSGQ01) ...                1

   STRUCTURE NAME(STRMSGQ01OVFLW) ...           2

   STRUCTURE NAME(STREMHQ01) ...                3

   STRUCTURE NAME(STREMHQ01OVFLW) ...           4

   STRUCTURE NAME(MSGQLOGSTR) ...               5

   STRUCTURE NAME(XMHQLOGSTR) ...               6
```

Figure 40. Sample CFRM Policy Implementation

5.9 Tailoring the IMS System for Shared Queues

The following IMS proclib members must be either updated or created for a shared-queues environment:

DFSPBxxx	IMS execution and startup parameters
DFSDCxxx	Primary and secondary master terminal override parameters
DFSSQxxx	IMS shared queues definitions and startup parameters
CQSIPxxx	CQS initialization parameters
CQSSLxxx	CQS local structure shared queues parameters
CQSSGxxx	CQS global structure shared queues parameters

The proclib members and parameters can be divided into two sections, one for IMS execution-time parameters and one for CQS execution-time parameters. Among the IMS execution time parameters are parameters that relate to shared queues and parameters that affect the existing IMS queue buffer pool management structure.

5.9.1 Specifying the IMS Parameters

The following IMS proclib members for shared queues must be changed.

- IMS control region execution parameters (DFSPBxxx)
- IMS data communications parameters (DFSDCxxx)
- IMS shared queues and CQS parameters (DFSSQxxx)

The SHAREDQ parameter in the DFSPBxxx proclib member is used to activate shared queues. If this parameter is not specified, or is specified as null, the other parameters describing the shared queues environment are ignored.

5.9.1.1 Specifying the IMS Control Region Execution Parameters (DFSPBxxx)

The IMS control region execution parameters are specified in IMS proclib member DFSPBxxx. The changes for a shared-queues environment can be grouped into shared queue parameters, IMS queue buffer pool parameters, and the parameters pointing to the IMS data communications parameters. The parameters required for shared queues are: QBUFSZ, QBUFMAX, QBUFHITH, QBUFLWTH, QBUFPCTX QBUFPCTX, LGMSGSZ, SHMSGSZ, and DC.

Shared Queue Parameters:

SHAREDQ Specifies the three digit suffix of the DFSSQxxx proclib member.

The existence of this parameter determines whether IMS runs with shared queues enabled or not. If the parameter is blank, or undefined, shared queues is not enabled. If the parameter is specified, it provides the name of the proclib member containing the CQS connection information.

IMS Queue Buffer Pool Parameters: Several new or changed parameters in the execution parameters for IMS relate to the IMS queue buffers and the IMS queue buffer pool. These parameters have been added or changed to relate directly to shared queues unless otherwise stated.

QBUF Initial number of queue buffers allocated to the queue pool. If this parameter is not specified, it defaults to the value of the BUFFER parameter in the MSGQUEUE macro of the IMS system definition.

The number of queue buffers can vary when IMS uses shared queues. The pool is automatically expanded and compressed when certain thresholds, described below, have been reached.

QBUFSZ Size of the buffers in the message queue pool

The size specified should be at least the size of the record length of the long message record. If this parameter is not specified, it defaults to the DCB size of the allocated LGMSG data set.

QBUFMAX Maximum number of queue buffers in the message queue pool that IMS can expand to, if expansion of the queue buffer pool is needed

SHMSGSZ Size in bytes of a short message record. If this parameter is not specified, it defaults to the DCB parameter is not specified, it defaults to the DCB size of the allocated SHMSG data set.

LGMSGSZ Size in bytes of a long message record. If this parameter is not specified, it defaults to the DCB parameter is not specified, it defaults to the DCB size of the allocated LGMSG data set.

QBUFHITH High threshold percentage at which the message queue buffer pool is dynamically expanded. The default value is 80%.

QBUFLWTH Low threshold percentage at which the message queue buffer pool is dynamically compressed. The default value is 50%.

QBUFPCTX Amount by which the message queue buffer pool is expanded when the QBUFHITH percentage is reached. The value is a percentage of the initially allocated queue buffers (QBUF) and not the size of the buffer pool at the time the threshold is reached. The default value is 20%.

Other Parameters QTL and QTU are IMS message queue parameters, but have no effect on shared queues. EXVR specifies whether the message queue pool buffers are to be long term page fixed or not.

IMS Data Communications Parameters

DC Specifies the suffix of the DFSDCxxx proclib member containing the overrides for the data communications parameters.

Figure 41 show a sample portion of a DFSPBxxx member used to activate shared queues.

```
 :
 :
DC=001
SHAREDQ=001
QBUF=500
QBUFSIZE=6720
QBUFMAX=2000
SHMSGSZ=336
LGMSGSZ=3360
QBUFHITH=80
QBUFLWTH=50
QBUFPCTX=20
 :
 :
```

Figure 41. Sample DFSPBxxx Proclib Member

5.9.1.2 Specifying the IMS Data Communications Parameters (DFSDCxxx)

The DFSDCxxx proclib member is used to override the The DFSDCxxx proclib member is used to override the data communications definitions that are generated with the IMS system definition. Thus IMS systems in a shared-queues group can be cloned, without duplicating any definitions. Some definitions in an IMS system must be unique (such as the MTO), and these assignments can be overridden to make them unique at IMS startup time.

PMTO=node Node name to override the node name defined to the primary master terminal (PMTO).

PMTO1-PMTO8=(lterm<,MASTER>) specifies the first to the eighth LTERM names of the PMTO. These LTERM names must be unique across all IMSs in the same VTAM generic resource group.

PMTOG=lterm Generic LTERM name of the PMTO. This should be the same name for all IMSs in the same VTAM generic resource group.

SMTO=node, SMTO1-SMPTO8, SMTOG specifies similar parameters for the secondary MTO.

TRUNC=Y/N System default for scratch pad area (SPA) truncations for IMS conversational transactions.

Refer to Chapter 10, “Conversational Programs” on page 125 for more information on the truncation of the SPA.

Figure 42 shows a sample of the IMS data communications parameters.

```
PMTO=SC02189D
PMTOG=MASTER
```

Figure 42. Sample IMS Data Communications Parameters (DFSDCxxx)

5.9.1.3 Specifying the IMS Shared Queues and CQS Parameters (DFSSQxxx)

The DFSSQxxx proclib member specifies parameters related to the shared queue and the CQS address space. The parameters can be grouped as local and global.

Specifying the Local Parameters

CQS Specifies either the CQS procedure name that can be started by IMS during IMS initialization or an MVS START command to allow the user to specify how to start CQS

CQSSSN Local CQS subsystem name to which the local IMS must connect to enable the local IMS to have access to the shared queues.

WAITRBLD Represents whether or not access to the EMHQ structure is quiesced during its recovery processing.

Specifying the Global Parameters: The parameters listed in this section should have the same values for all CQSSLxxx proclib members relating to CQSs in the same shared-queues environment.

EMHQ Primary structure name for the EMH shared queues in the coupling facility.

MSGQ Primary structure name for the full function shared queue in the coupling facility.

This name must not be the same as the EMHQ name.

SQGROUP The IMS XCF group name, DFSxxxxx, that IMS connects to at startup time to enable shared queues processing. startup time to enable shared queues processing.

Figure 43 on page 78 shows a sample DFSSQxxx proclib member that has both Full Function and Fast Path shared queue structures defined.

```
CQS=CQSIVP1
CQSSSN=CQS1
EMHQ=STREMHQ01
MSGQ=STRMSGQ01
SQGROUP=SHQ1
WAITRBLD=Y
```

Figure 43. Shared Queues and CQS Address Space Parameters (DFSSQxxx)

5.9.2 Specifying the CQS Parameters

Three members in the IMS proclib are used as parameters for CQS initialization (CQSIPxxx), CQS local structure definitions (CQSSLxxx), and CQS global structure definitions (CQSSGxxx).

5.9.2.1 Specifying the CQS Initialization Parameters (CQSIPxxx)

The CQSIPxxx proclib member specifies the parameters used in the initialization of the CQS address space. The ARMRST and SSN parameters must be unique among the CQSs in the same shared-queues environment, and the CQSGROUP parameter must be the same for all CQSs in the same shared-queues environment.

ARMRST Specifies whether or not the MVS Automatic Restart Manager (ARM) should restart the CQS address space after an abend

SSN Subsystem name of the CQS address space

CQSGROUP XCF group to which all CQSs in the same shared-queues environment connect at startup. The XCF group name is CQSxxxxx.

STRDEFG Specifies the suffix of the CQSSGxxx proclib member that defines the CQS global structure parameters.

STRDEFL Specifies the suffix of the CQSSLxxx proclib member that defines the CQS local structure parameters.

Figure 44 shows sample CQS initialization parameters.

```
ARMRST=Y
SSN=CQSIVP1
STRDEFG=001
STRDEFL=001
CQSGROUP=CQS6
```

Figure 44. Sample CQSIPxxx Proclib Member

5.9.2.2 Specifying the CQS Local Structure Definitions (CQSSLxxx)

The CQSSLxxx proclib member is used to identify CQS checkpoint data sets for the MSGQ structures and, if defined, the EMHQ structures. The CQS dynamically allocates the checkpoint data sets when a CQS system is started. There should be one CQSSLxxx member for each CQS in the shared-queues group.

There must be a description for each structure used by CQS. In the case of shared queues, there are either one or two primary structures that CQS connects to; the MSGQ structure and the EMHQ structure.

The parameters discussed below are all subparameters of the STRUCTURE keyword.

STRNAME Name of the structure to which the CQS connects

CHKPTDSN Checkpoint data set name used by the CQS for writing checkpoint information. It is allocated dynamically at CQS startup.

SYSCHKPT Number of log records that the CQS writes before taking a system-generated checkpoint

Figure 45 shows a sample structure of the CQSSLxxx proclib member related to an IMS system that has Fast Path defined.

```
                  1                   A
STRUCTURE (STRNAME=STRMSGQ01,CHKPTDSN=IMS.CQS1.QMSG.CHKPT,SYSCHKPT=5000)
STRUCTURE (STRNAME=STREMHQ01,CHKPTDSN=IMS.CQS1.QEMH.CHKPT,SYSCHKPT=5000)
                  3                   B
```

Figure 45. CQS Local Definitions (CQSSLxxx)

5.9.2.3 Specifying the CQS Global Structure Definition (CQSSGxxx)

The CQSSGxxx proclib member defines the global CQS parameters that are related to one or more coupling facility structures. Each CQS in a shared-queues environment must use the same proclib member, or an exact copy of the proclib member. We recommend, for the sake of simplicity, that all CQSs in a shared-queues environment use the same CQSSGxxx proclib member.

The parameters discussed below are all subparameters of the STRUCTURE keyword.

STRNAME Name of the structure to which the CQS connects

SRDSDSN1 Name of the first SRDS

This data set name is used when a structure checkpoint is requested. The CQS dynamically allocates this data set at structure checkpoint time.

SRDSDSN2 Name of the second SRDS that is dynamically allocated when a structure checkpoint is requested

The CQS flip-flops between the two SRDSs as the structure checkpoints are taken. The SRDSs are common to all CQSs in the same shared-queues environment.

LOGNAME Name of the MVS log stream that the CQS uses to record all information related to the structure. All of the CQS address spaces in the same shared-queues environment log to the same MVS log stream identified by the LOGNAME parameter.

This name must be the same as defined in the LOGR policy in MVS.

OBJAVGSZ Average size of the data object that is placed on a queue in the structure. This parameter is used to determine the list element to data element (LE/DE) ratio.

It is better to make the OBJAVGSZ too small rather than too large because overflow structure processing is based on the percentage of data elements in the structure, not on the list entries. If a structure fills up with list entries, the structure is full, and no overflow processing is carried out.

OVFLWMAX Percentage full value of data elements in the structure, above which value overflow processing is performed

OVFLWSTR Name of the structure that the CQS uses for overflow processing

The CQS moves selected queues from the primary structure to the overflow structure with this name if the OVFLWMAX percentage is reached. If this parameter is not specified, overflow processing does not take place, and the CQS rejects any requests to add data elements to the queue that has been selected for overflow processing.

STRMIN Minimum size of the primary coupling facility structure that is allocated if the coupling facility cannot allocate the initial size of the structure as defined in the CFRM policy. The parameter defaults to 0. STRMIN does not change the size of the structure. STRMIN *should* be specified. MVS may not be able to allocate a full-size structure. Use this parameter to ensure that you have a usable minimum size for the structure.

Figure 46 shows a sample of the CQSSGxxx proclib member describing the structures to be defined for shared queues. The IMS systems in this shared-queues environment have Fast Path defined in the IMS system definition and are making use of overflow queue structures for their full function and Fast Path message queues.

```
STRUCTURE
(
STRNAME=STRMSGQ01,                    1
OVFLWSTR=STRMSGQ01OVFLW,              2
SRDSDSN1=IMS.CQS.MSGQ.SRDS1,          C
SRDSDSN2=IMS.CQS.MSGQ.SRDS2,          D
LOGNAME=SYSLOG.MSGQ01.LOG,            7
STRMIN=5000,
OBJAVGSZ=500,
OVFLWMAX=50
)
```

Figure 46. CQS Global Definitions (CQSSGxxx)

```
STRUCTURE
(
STRNAME=STREMHQ01,                    3
OVFLWSTR=STREMHQ01OVFLW,              4
SRDSDSN1=IMS.CQS.EMHQ.SRDS1,          E
SRDSDSN2=IMS.CQS.EMHQ.SRDS2,          F
LOGNAME=SYSLOG.EMHQ01.LOG,            8
STRMIN=5000,
OBJAVGSZ=2K,
OVFLWMAX=50
)
```

Figure 46. CQS Global Definitions (CQSSGxxx)

5.10 Specifying the Base Primitive Environment Parameters

In general, BPE is a hidden software layer, and you do not have to specify anything to use it. There are, however, some parameters that you can specify for problem determination trace information and CQS exit routines.

The CQS specifies any BPE options, using the `BPECFG` startup parameter. This parameter points to a BPE configuration proclib member, which specifies various internal tracing options and CQS exit routines. These options can determine the language for messages, internal trace table levels and sizes, as well as exits that can be used during processing using BPE. The parameters are not discussed in detail because the member generated when IMS is installed is usually adequate. For further information, see the *IMS/ESA Common Queue Server Guide and Reference*, LY37-3730.

5.10.1 Specifying the BPE Configuration Parameters (BPECFGxx)

LANG Specifies the language used for error messages. The only language currently supported is *ENU*, which is support for U.S. English.

TRCLEV Specifies the type, level, product and size of trace table pages in the table.

EXITMBR Specifies the proclib member in which the exits are described and the product with which the exits are associated.

Figure 47 on page 82 shows a sample BPE configuration member.

```
*************************************************************
* CONFIGURATION FILE FOR BPE WITH CQS
*************************************************************

 LANG=ENU                         /* LANGUAGE FOR MESSAGES   */
                                  /* (ENU = U.S. ENGLISH)    */
#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(AWE,HIGH,BPE)             /* AWE SERVER TRACE        */
TRCLEV=(CBS,MEDIUM,BPE)           /* CONTROL BLK SRVCS TRACE */
TRCLEV=(DISP,HIGH,BPE,PAGES=12)   /* DISPATCHER TRACE WITH   */
                                  /* 12 PAGES (48K BYTES)    */
TRCLEV=(LATC,LOW,BPE)             /* LATCH TRACE             */
TRCLEV=(SSRV,HIGH,BPE)            /* GEN SYS SERVICES TRACE  */
TRCLEV=(STG,LOW,BPE)              /* STORAGE TRACE           */
TRCLEV=(USRX,MEDIUM,BPE)          /* USER EXIT TRACE         */
#
# DEFINITIONS FOR CQS TRACES
#

TRCLEV=(CQS,HIGH,CQS)             /* CQS GENERAL TRACE       */
TRCLEV=(STR,MEDIUM,CQS)           /* CQS STRUCTURE TRACE     */
TRCLEV=(INTF,HIGH,CQS)            /* CQS INTERFACE TRACE     */

#
# USER EXIT LIST PROCLIB MEMBER SPECIFICATION
#

EXITMBR=(CQSEXIT0,CQS)            /* SPECIFY PROCLIB DATASET */
                                  /* MEMBER CQSEXIT0 AS CQS'S*/
                                  /* USER EXIT LIST MEMBER   */
```

Figure 47. Sample BPE Configuration Member

5.10.2 Specifying the BPE User Exit Parameters

The BPE user exit parameters are specified in a proclib member whose name is defined with the EXITMBR parameter in the BPECFGxx proclib member. In the sample BPECFGxx proclib member shown in Figure 47, it is called CQSEXIT0.

BPE supplies five exit types. These types are listed below with the name of the type and a brief description.

CLNTCONN Client connection

INITTERM Initialization and termination

OVERFLOW The overflow exit type can determine which queues to exclude from overflow processing.

STRSTAT Structure statistics

STREVENT This exit is called for various structure events and enables you to gather statistics related to the structure. It is called when there is a new connection to the structure, at structure checkpoint, structure rebuild, structure overflow, and when the status of the structure changes (such

as becoming available after a loss, structure failure, or when CQS loses its connection).

The parameters that can be specified in the BPE user exit list are:

TYPE The type of exit being described. This can be one of the five types discussed above.

EXITS Details the list of exits to be called for the indicated type of exit

ABLIM Specifies the abend limit for the exit type being described. When the limit is reached, the exit is no longer called. The default for this parameter is 1.

Figure 48 shows a sample of an exit member for CQS.

```
************************************************************
* SAMPLE USER EXIT FOR CQS
************************************************************

*      DEFINE CLIENT CONNECTION EXIT : XLCLCON0

EXITDEF(TYPE=CLNTCONN,EXITS=(XLCLCON0))

*      DEFINE INITIALIZATION AND TERMINATION EXITS
*           : MYCQSIT0, OEMCQIT0

EXITDEF(TYPE=INITTERM,EXITS=(MYCQSIT0,OEMCQIT0))

*      DEFINE OVERFLOW EXIT : OVERFLO1, OVERFLO2 AND OVERFLO3

EXITDEF(TYPE=OVERFLOW,EXITS=(OVERFLO1,OVERFLO2,OVERFLO3))


*      DEFINE STRUCTURE STATISTICS EXIT STRSTAT0

EXITDEF(TYPE=STRSTAT,EXITS=(STRSTAT0))

*      DEFINE STRUCTURE EVENT EXIT STREVNT0

EXITDEF(TYPE=STREVENT,EXITS=(STREVNT0),ABLIM=0)
```

Figure 48. Sample BPE User Exit Member

5.11 Activating the CFRM Policy

The SETXCF command is used to activate a new CFRM policy. To swap the CFRM policy from the old policy to the new policy, issue the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=POLICY1
```

5.12 Creating a Started Task for CQS

The CQS runs as its own address space, on the same MVS system as the associated IMS system. The CQS has execution parameters that need to be specified in the JCL. The execution parameters specified in the JCL override the proclib members specified in 5.9.2, “Specifying the CQS Parameters” on page 78.

The execution parameters that can be overridden in the CQS JCL are not discussed in detail here (see 5.9.2, “Specifying the CQS Parameters” on page 78). These parameters are:

- ARMRST
- CQSGROUP
- SSN
- STRDEFG
- STRDEFL

The execution parameters that can only be specified in the JCL are detailed below.

BPECFG Specifies the name of the BPE configuration proclib member. Ensure that the necessary proclib members are in your proclib library. They are generated from the IMS system definition by default, and no customization is necessary for the implementation of shared queues.

CQSINIT Specifies the suffix of the CQSIPxxx proclib member.

Figure 49 shows a sample CQS procedure.

```
//CQS      PROC REGION=3000K,
//              SOUT=T,
//              BPECFG=BPECONFG,
//              CQSINIT=001,
//              CQSGROUP=,
//              ARMRST=Y
//*
//CQSPROC  EXEC PGM=CQSINIT0,RGN=&REGION,
//              PARM='BPECFG=&BPECFG,CQSINIT=&CQSINIT'
//STEPLIB  DD   DSN=IMS610.RESLIB,DISP=SHR
//PROCLIB  DD   DSN=IMS610.PROCLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=&SOUT
//SYSUDUMP DD   SYSOUT=&SOUT
```

Figure 49. Sample CQS Procedure

5.13 Defining Subsystem Names in MVS

Figure 50 on page 85 shows a sample of the MVS subsystem name entry for CQS. This member is defined in member IEFSSNxx in SYS1.PARMLIB.

```
SUBSYS SUBNAME(CQS1)      /* IMS V6 COMMON QUEUE SERVER  */
SUBSYS SUBNAME(CQS2)      /* IMS V6 COMMON QUEUE SERVER  */
```

Figure 50. Sample Subsystem Name Entries for CQS

5.14 Updating the MVS Program Properties Table

An entry for the CQS must be added to the MVS PPT. This is probably done by your MVS systems programmer, depending on the standards for your environment. The PPT entry describes the properties of the CQS program. It defines whether the job running the program can be canceled with the MVS CANCEL command, the storage key in which the program is authorized to run, as well as several other properties. Figure 51 shows a sample of the MVS PPT entry for the CQS.

```
PPT PGMNAME(CQSINIT0)
         CANCEL
         KEY(7)
         NOSWAP
         NOPRIV
         DSI
         PASS
         SYST
         AFF(NONE)
         NOPREF
```

Figure 51. Sample Program Properties Table Entry for the CQS

5.15 Authorizing Connections to CQS

If security checking on the coupling facility structures is deemed necessary, the coupling facility structures can be defined to the security product at your installation, and the security administrator can create profiles to control access to the CQS coupling facility structures.

If RACF is the security product being used, the security administrator can define FACILITY profiles for the CQS coupling facility structures. The profile name must be CQSSTR concatenated with the CQS structure name as defined in the CQSSGxxx and CQSSLxxx proclib members.

The CQS issues a RACROUTE REQUEST=AUTH call when a client attempts to connect to a shared queue structure using CQS. A client uses the CQSCONN request to the CQS when attempting to connect to a shared queues structure. If a profile is not defined for the CQS shared queue structure, the structure is not protected. The overflow structures for IMS need not be defined because the primary and overflow structures are considered a pair and as such are treated as one unit. Figure 52 on page 86 shows an example of the RACF definitions for a CQS MSGQ structure named STRMSGQ01 and an EMHQ structure named STREMHQ01.

```
RDEFINE FACILITY CQSSTR.STRMSGQ01 UACC(NONE)
PERMIT CQSSTR.STRMSGQ01 CLASS(FACILITY) ID(IMSGRP) ACCESS(UPDATE)
SETROPTS CLASSACT(FACILITY)

RDEFINE FACILITY CQSSTR.STREMHQ01 UACC(NONE)
PERMIT CQSSTR.STREMHQ01 CLASS(FACILITY) ID(IMSGRP) ACCESS(UPDATE)
SETROPTS CLASSACT(FACILITY)
```

Figure 52. Sample RACF Definitions for CQS Security

5.16 Preparing to Start IMS and CQS

Before starting IMS and hence the CQS, make sure you have completed steps 1 through 9 as described in 5.3, “Steps to Implement IMS Shared Queues” on page 62. Issue this XCF command:

```
D XCF,STR
```

to check that all of the necessary coupling facility structures have been defined. The structures should have a status of `NOT ALLOCATED` if the CQS has not previously accessed the structure.

Follow these steps for each IMS to be activated in the same shared-queues group:

1. Ensure that all implementation steps have been followed.
2. Shut down IMS.
3. Update the IMS JCL to use the new DFSPBxxx member.
4. Coldstart IMS.

5.16.1 IMS Initialization

The following events occur during IMS initialization in a shared-queues environment:

1. The CQS is started if it is not already active.
2. IMS connects to the CQS.
3. IMS connects to the MSGQ and EMHQ structures.
4. IMS resynchronizes the structure queues with the CQS.
5. IMS registers its interest in all statically defined transactions.

5.16.2 IMS Termination

The following events occur during IMS termination in a shared-queues environment:

1. IMS disconnects from the MSGQ and EMHQ structures.
2. The CQS deregisters client interest in all queues on structures.
3. IMS deregisters from the CQS interface.

The CQS must be available for IMS to terminate normally. If the CQS is not available the `/CHECKPOINT SHUTDOWN` command is rejected. If the CQS fails during a shutdown checkpoint, shutdown waits until the CQS is restarted. The MVS

MODIFY command must be used to terminate IMS if the CQS is down and IMS must be terminated immediately.

5.17 Fallback to Local Message Queues

To go from a shared-queues environment to a local message queue, IMS needs to be brought down normally. The SHAREDQ parameter must be deleted from DFSPBxxx or set to null. IMS must then be coldstarted. If there are messages on the shared queues, it is preferable to allow all of the messages to process before the system is reverted back to local message queue processing. Keep in mind that a coldstart of an IMS with local message queues causes any messages in the shared queues to be ignored. The messages are still in the coupling facility; It is just that a non-shared-queue IMS cannot access them.

You can use IMS Message Requeuer Version 2 to remove messages from the shared queues and then use the output to repopulate the local message queues.

To facilitate the conversion back to local message queue processing, it is important that any changes made to the IMS system definition, or any parameter changes that affect processing outside a shared-queues environment, be identified. These changes should be identified before the implementation of shared queues. If these recommendations are followed, fallback is simple.

1. Bring down IMS normally.
2. Update the DFSPBxxx proclib member.

   Either remove the `SHAREDQ` parameter, or set `SHAREDQ=,`.
3. Coldstart IMS

Remember to clean up after the shared-queues environment has been backed out. Delete the structures and the data sets used by the CQS. Follow these steps:

1. Delete coupling facility structures for the EMHQ, MSGQ, and log stream, unless you are intending to requeue these messages with IMS Message Requeuer Version 2.
2. Update the CFRM and LOGR policies.
3. Delete the CQS SRDS and CHKPT data sets.
4. Delete the MVS log data sets.

5.18 Consolidating the IMS System Definitions

Once the implementation of shared queues is complete, there are many things that you can do to clean up the environment and ensure efficient processing and exploitation of the sharing of message queues.

It is important to note each change that is made during consolidation. If parameters are indeed removed, they and their values must be restored if your environment is ever converted back from shared queues to local message queues. The entire environment must be restored to its original state.

5.18.1 IMS System Definition

The IMS system definition can be changed after the implementation of shared queues. In the sections that follow we describe cloning your IMS system definitions, removing MSC link definitions, and the MSGQUEUE macro.

5.18.1.1 Cloning IMS System Definition Decks

IMS/ESA Version 6 assists in the consolidation of multiple IMS system definitions into a single IMS system definition by allowing you to specify as startup options any parameters that need to be unique across multiple IMS systems.

The IMS system definitions of the IMS systems in the shared-queues environment should ultimately be merged, so that all IMS system definition decks are cloned. In a shared-queues environment, cloning uses the advantages of shared queues to the fullest. The advantages of cloned IMS system definition decks are:

- Each transaction can run on any IMS system in the shared-queues group.
- Additional IMS and CQS systems can be started in the sysplex and within the same shared-queues group to assist in message processing during heavy workload periods.
- The complexity of the IMS system definition is reduced by decreasing the number of IMS system definitions that need to be maintained and performed.
- Continuous operations of the IMS environment. If an IMS system fails or is shut down, the other IMS systems in the shared-queues group can take over the entire workload, and not just a common subsection.

If it is not desirable to have all transactions processing on all IMS systems in a shared-queues group, the location of messages being processed can be handled by region distribution among the IMS systems in the shared-queues group. Another way is to assign transactions to a class that is not serviced by any region in the local IMS system. The `/ASSIGN` command can be automated in Timer Controlled Facility (TCF) at IMS startup.

5.18.1.2 Removing MSC Link Definitions within a Shared-Queues Group

MSC links cannot be started between IMS systems in the same shared-queues group. The MSC link definitions are still used in a shared-queues environment to determine the sysids of the shared-queues group. If the IMS system definition decks are cloned, all local and remote transaction definitions need to be merged, and references to local sysids in the shared-queues group need to be removed. The MSC definition macros (MSNAME, MSLINK, MSPLINK) can be removed from the IMS system definition for the MSC links between IMS systems in the same shared-queues group.

If it is necessary to keep the functionality of MSC, with some transactions processing on one IMS system and others on another, it can be done outside the IMS system definition process. Ensure that regions are started only on the IMS system on which the transaction should run, and that those transaction classes match the classes in the regions.

5.18.1.3 Specifying the MSGQUEUE Macro

The MSGQUEUE macro *cannot* be removed from the IMS system definition. Although all of the parameters specified in the macro are either invalid in a shared-queues environment or overridden in a proclib member at IMS startup, the shared queues function is not an IMS system definition parameter. There is no way of knowing at IMS system generation time whether the IMS system is going to use shared queues. The characteristics for the message queue data sets are still used during IMS system generation. The MSGQUEUE macro still provides defaults for LGMSG LRECL, SHMSG LRECL and QBUF size, even though they can be overridden by parameters in the proclib members.

5.18.2 IMS Initialization Parameters

Some proclib member parameters do not need to be specified after the implementation of shared queues. These parameters can be removed, to simplify IMS control region execution and data communications proclib members.

5.18.2.1 IMS Control Region Execution Parameters (DFSPBxxx)

The SPA pool (SPAP) parameter was used before the implementation of shared queues. The SPAP is not used with shared queues as the SPA is kept on the message queue both locally and on the shared queue structure. The SPAP size parameter can be removed from the DFSPBxxx proclib member.

The DEADQ parameter was used for storing messages over a certain age that were queued to dynamic LTERMs (ETO). With the implementation of shared queues, IMS no longer has any knowledge of the length of time a message has been on the LTERM ready queue. The DLQT parameter is therefore no longer needed in an ETO and shared-queues environment and can be removed from the proclib member.

5.18.2.2 IMS Data Communications Parameters (DFSDCxxx)

If the IMS system definition is cloned for each IMS system, it is important to define unique master and secondary terminals for each IMS in the shared-queues group. These can be specified in the DFSDCxxx proclib member. The NODE and LTERM must already be defined in the IMS system definition, but not as master or secondary terminals. The NODE and LTERM names can then be used in the proclib member to be assigned as master and secondary terminals at IMS startup.

5.18.3 IMS Startup JCL

The long and short message queue data sets (LGMSG and SHMSG data sets) are used to store the long and short messages on the queue, to prevent the message queue buffer pool from becoming full. These data sets are not used in a shared-queues environment, and therefore the DD cards in the IMS startup JCL can be deleted, as can the data sets. It is important to check that the QBUFSZ, SHMSGSZ, and LGMSGSZ parameters are specified in the DFSPBxxx proclib member. The default values of these parameters are the DCB size of the three data sets. If these parameters are not specified, IMS uses default values, and there may be performance implications for your IMS system.

5.19 Parameters and Components in a Shared-Queues Environment

Figure 53 shows the relationships among the parameters and components in a shared-queues environment.

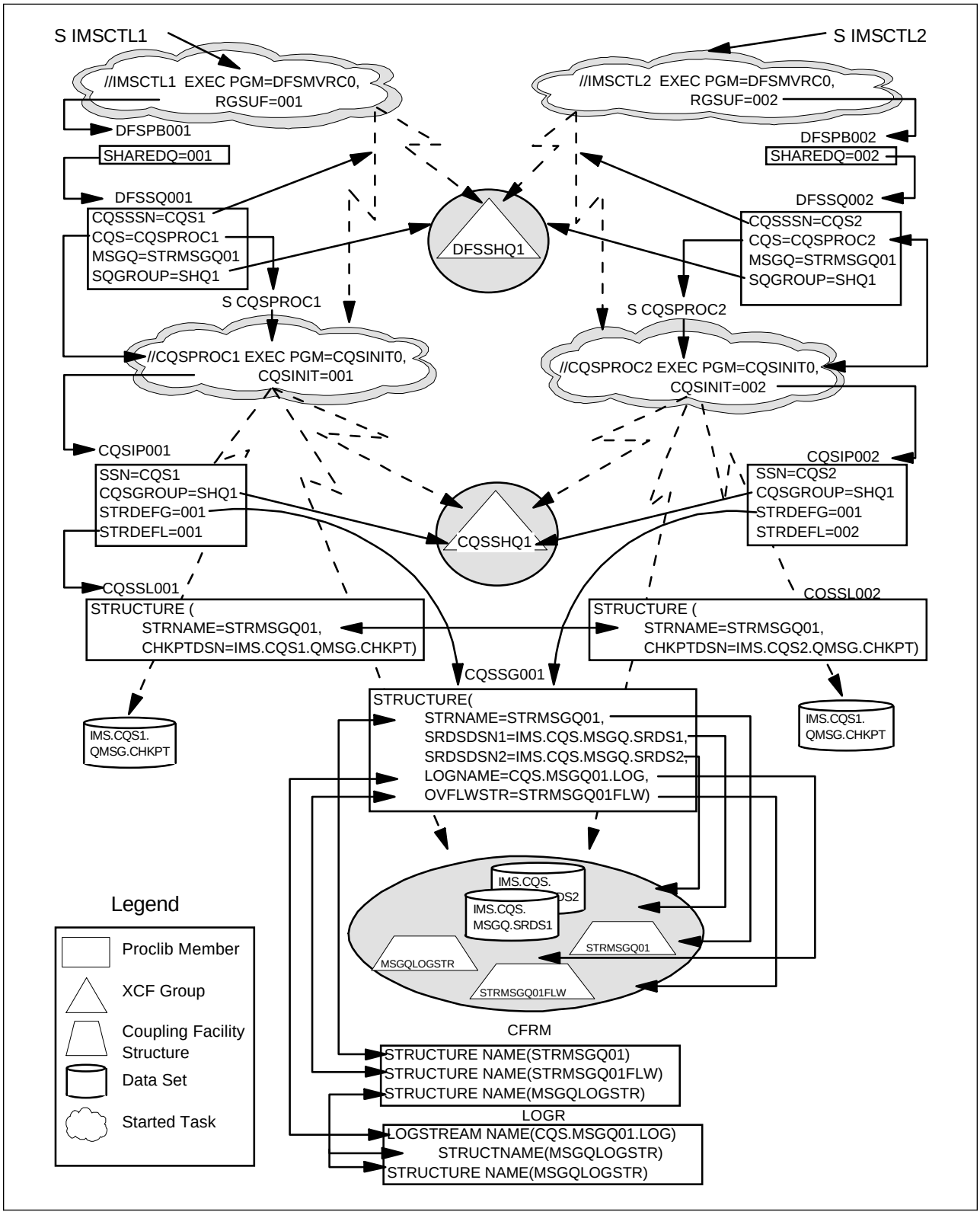


Figure 53. Parameters and Components in a Shared-Queues Environment

Chapter 6. Operations Management

Operating within a shared-queues environment requires consideration of many issues. In this chapter we highlight the issues and discuss what you can do to maximize the availability of the IMS systems. We cover the following topics:

- Recovery scenarios
- Structure maintenance
- Availability considerations

6.1 Recovery Scenarios

The shared queues function introduces new situations that require recovery. In this section we highlight some of those situations, detail what occurs in each, and explain the operational procedures to perform if recovery is required.

6.1.1 CQS Abnormal Termination

One CQS is the interface to the shared queue structures for one IMS. If a CQS is unavailable for any reason, the IMS that it serves cannot access the shared queue structures. In most cases this means that the IMS is effectively unavailable. Batch work that does not require shared queues or Fast Path transactions that have an SPC of local continue to process, because neither of these functions accesses the shared queues. Fast Path transactions that have an SPC of local first continue to process if the resources are available in the local IMS. All full function transactions are required to be put on the MSGQ structure, so while CQS is unavailable they cannot be processed.

If the CQS abnormally ends, you need to restart it as soon as possible in order to return to normal operations. You also need to restart the CQS as soon as possible to avoid having to coldstart it if two structure checkpoints occur before you try to restart it.

If the CQS abnormally ends, transactions that IMS is processing at the time of the CQS abend are moved to the lock queue of the list structure. They remain on the lock queue until the CQS that abended is restarted and the indoubt units of work are resolved. Resolution is done when IMS reconnects to the CQS and resynchronization processing is performed. While the transactions are on the lock queue, they cannot be processed by any other IMS within the shared-queues environment. If you are using VTAM generic resources, if the transaction is either a full function response mode or a Fast Path type, and the user that entered the transaction is connected to another IMS system, the user waits until the CQS is restarted and the message is processed before their session is unlocked. The session can be reset by the MTO.

When the CQS performs overflow processing or moves suspended transactions from the suspend queue for processing, the messages are first moved to the move queue on the list structure. If the CQS abnormally ends while this processing is being performed, there is a chance that a message could be on the move queue. The message remains on the move queue until another CQS takes over the overflow process. While the message is on the move queue, no other IMS within the shared-queues group can process it.

The CQS can be placed under MVS/ESA Automatic Restart Manager (ARM) control or an automated operations package. When the CQS is restarted, a CQS restart event is issued, causing IMS to attempt to connect to the CQS. The CQS and IMS then resynchronize indoubt units of work. IMS then reregisters interest in the queues.

While the CQS is unavailable, IMS will not shut down by the `/CHECKPOINT FREEZE` command. The command is rejected with this message:

```
DFS1975 COMMAND REJECTED AS CQS IS NOT AVAILABLE
```

If necessary, you can shut down IMS, using the MVS modify command.

6.1.2 IMS Abnormal Termination

If IMS abnormally ends, problems that could occur are similar to those that occur when the CQS abnormally ends.

Messages that were being processed by the IMS system when it abnormally ended remain on the lock queue until the IMS system is restarted and the resynchronization process with the CQS is completed. While the transactions are on the lock queue, they cannot be processed by any other IMS within the shared-queues environment. If you are using VTAM generic resources, if the transaction is either a full function response mode or a Fast Path type, and the user that entered the transaction is connected to another IMS system, the user waits until IMS is restarted and the message is processed before the session is unlocked. The session could also be reset by the MTO.

When the IMS system is restarted, it performs emergency restart processing. This involves reconnecting to the CQS and then resynchronizing indoubt units of work with the CQS. The IMS system then reregisters interest in the queues. If the IMS system is coldstarted instead of emergency restarted, the messages that were on the lock queue when it abended are moved to the cold queue. IMS is notified of which messages have been moved to the cold queue by the CQS. When messages are placed on the cold queue, they cannot be processed by any IMS system. The messages remain there until either the shared queues structure is deleted or the messages are requeued using the IMS Message Requeuer (or a similar product).

6.1.3 Message Queue Structures

In this section we describe structure rebuild, structure recovery, structure deletion, and lost connectivity to a structure.

6.1.3.1 Structure Rebuild

The MVS XES component notifies structure users if a rebuild is required. Notification is performed by the issuing of events. When the CQSs are notified through an XES event that a rebuild of a structure is required, each CQS issues a connect call, which causes MVS to either allocate a new structure or connect to the new structure. The first CQS to issue the connect call causes the new structure to be allocated. Both the new structure and the old structure exist while the rebuild process is being performed. Each CQS issues a quiesce to the old structure. The CQSs then determine which CQS is going to coordinate the rebuild process.

Each CQS checks whether it has full connectivity to the structure and can copy the old structure to the new structure. If any CQS can perform a copy, it places its

name in the control queue and coordinates the rebuild of the structure. This CQS is known as the master CQS. If no CQS has full connectivity to the structure, a structure recovery is required.

XES decides in which coupling facility the new structure is to be allocated. The coupling facility where XES allocates the new structure must be listed in the PREFLIST parameter in the CFRM policy. Many factors affect which coupling facility is chosen. Refer to the *OS/390 MVS Programming: Sysplex Services Guide*, GC28-1771, for more information.

If a structure checkpoint or overflow threshold processing is in progress, and a structure rebuild is initiated, the structure rebuild aborts. The CQS retries and aborts rebuild again until the structure checkpoint or overflow threshold process completes.

During rebuild processing, there is an XES protocol that is to be followed. A rebuild is divided into many steps. XES requires that all CQSs connected to the new structure follow the protocol in order for the rebuild to complete. If the protocol is not followed, XES does not proceed to the next step of the rebuild process. The most common rule is that all CQSs must confirm that it is OK to proceed to the next step of the rebuild process. If for some reason a CQS is unable to confirm, the process is halted. If the rebuild process halts because a CQS has not issued a confirm to a step, there is no command that assists you in determining which CQS it is. The recommendation to get around this situation is to issue the MVS cancel command to CQSs other than the master CQS. You can determine which is the master CQS by looking in the CQS log. When the problem CQS is canceled, it is disconnected from the structure, which results in its not being required to confirm each step.

A rebuild can be initiated because of:

- A structure or coupling facility failure
- A structure deletion
- A defined percentage of connectivity being lost
- An operator command

6.1.3.2 Structure Recovery

The master CQS performs a recovery of the shared queue structure. This involves repopulating the structure, using the SRDS (if it is valid), and then applying updates, using the MVS log stream.

6.1.3.3 Structure Deletion

The structure can be deleted by this operator command:

```
SETXCF FORCE,STRUCTURE=strname
```

MVS accepts this command only if there are no CQSs connected to the structure. If the structure is deleted, the first CQS to attempt to connect to the structure causes XES to allocate a new structure. The CQS then performs a recovery if an SRDS exists, using both the SRDS and the MVS log stream. If the SRDS does not exist, no rebuild is required, and an empty structure is used.

6.1.3.4 Connectivity Lost

In the CFRM policy there is a REBUILDPERCENT parameter, which indicates how much connectivity to the structure may be lost before a rebuild automatically occurs. If the rebuild parameter is set to 50, when 50% or more of the connections to the structure cannot connect, a rebuild occurs. Using this example, if you have five CQSs (with equal weightings of 20%) connected to the structure and three lose connectivity, 60% of the connections would be lost and consequently a rebuild would occur. You can define a different weighting on each system (which is necessary if you want to use REBUILDPERCENT). More information about weightings in the sysplex failure management (SFM) policy can be found in *OS/390 Setting Up a Sysplex*, GC28-1779. If a rebuild is to occur, one of the CQSs that still has connectivity is the master CQS and copies the structure contents from the old structure to the new structure. If none of the CQSs has connectivity with the old structure, a structure recovery must be performed, using both the SRDS and the MVS log stream.

6.2 Structure Maintenance

From time to time a shared queue structure may require maintenance because of maintenance performed on the coupling facility machine or the alteration of parameters in the CFRM policy.

To perform maintenance on a coupling facility without severely impacting the running environment, the structures on that coupling facility must be moved. Assuming that you have the capacity on another coupling facility that is listed in the PREFLIST parameter within the CFRM policy for this structure, the structure can be copied to the new coupling facility with minimal impact. To perform a copy, at least one CQS must be active. While the copy is being performed, the structure is quiesced. All data objects are copied to the new structure.

To perform a structure copy to a new coupling facility, issue this MVS command:

```
SETXCF START,REBUILD,STRNM=structurename,LOCATION=OTHER
```

The LOCATION=OTHER parameter on this command moves the structure to another coupling facility in the PREFLIST of the CFRM policy. Without this parameter, the first coupling facility in the PREFLIST is used.

Changes to the CFRM policy can be implemented with little impact to the running environment. Typical changes to the CFRM policy are:

- Changing the size of the structure
- Changing the PREFLIST when a new coupling facility is added to the sysplex
- Changing the value of the REBUILDPERCENT parameter

Policy changes are implemented by updating the policy and then starting it by issuing this MVS command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=policyname
```

When the new policy has been started, you can trigger a rebuild by issuing this MVS command:

```
SETXCF START,REBUILD,STRNM=structurename
```

If you want the new version of the structure to be on a different coupling facility, put the LOCATION=OTHER parameter on the end of the rebuild command.

For further information about the SETXCF command, see 7.3.3, “XCF Commands” on page 108.

6.3 Copying or Recovering Structures

MVS supports structure rebuild to rebuild data on a structure. Structure rebuild is processed by the CQS either as a structure copy or a structure recovery. If at least one CQS in the shared-queues group has access to the structure requiring a rebuild, a structure copy is performed, otherwise a structure recovery is performed. To perform a rebuild, at least one CQS must be running. While a rebuild is being performed, access to the structure is quiesced. For an EMHQ structure, the time that the structure is quiesced is affected by the setting of the WAITRBLD parameter. For more information about the WAITRBLD parameter, see 5.9.1.3, “Specifying the IMS Shared Queues and CQS Parameters (DFSSQxxx)” on page 77. The structure rebuild can be initiated by an MVS operator command, by the CQS, or by MVS in the following situations:

- While at least one CQS is running, an MVS operator can initiate a structure rebuild to copy or recover queues, using the following command:

  ```
  SETXCF START,REBUILD,STRNAME=strname,LOCATION=NORMAL|OTHER
  ```

- The CQS initiates a structure rebuild if, during CQS initialization, it detects an empty structure and a valid SRDS.
- MVS initiates a structure rebuild if your installation defines a rebuild threshold for loss of connectivity, and that threshold is reached. The rebuild threshold is defined by the REBUILDPERCENT parameter in the CFRM policy and the in the CFRM policy and the WEIGHT parameter in the Sysplex Failure Management (SFM) policy. For more details on the rebuild threshold, see 5.8.1, “Defining a CFRM Policy” on page 67.
- If structure copy aborts because of a CQS failure and no other CQS can determine whether the failed CQS is the master, the rebuild starts over as a structure recovery.

6.3.1 Structure Copy

A structure copy can be initiated to make changes to the location or sizing of the structure. It can also be initiated by MVS if the loss of connectivity to a structure exceeds the rebuild threshold. If a structure copy is required for either the MSGQ or EMHQ structures, one of the CQSs that has connectivity with the structure performs the master role. A structure copy copies all data objects, both recoverable and nonrecoverable, from the current structure to a new structure.

6.3.2 Structure Recovery

A structure recovery is performed when the coupling facility fails, a structure has been damaged, or a structure has been deleted by an operator. If a structure recovery is required for either MSGQ or EMHQ structures, one of the CQSs takes the role of master and copies the most recent SRDS into the new structure. When this copy has been completed, the CQS applies all log updates since the SRDS was created. Because recoverable data objects are the only objects that were written to the SRDS, only the recoverable objects are written to the new structure.

If the SRDS is empty because no structure checkpoint was taken, the structure is rebuilt by applying all of the log updates since the structure was created.

6.4 Deleting a Structure

deletes A coldstart performed on a non-shared queues IMS deletes the contents of the message queue data sets. With shared queues, a coldstart of IMS does not affect the contents of the MSGQ and EMHQ structures. The only way to delete the contents of the MSGQ and EMHQ structures is to delete the active structures by issuing this command:

```
SETXCF FORCE,STR,STRNAME=strname
```

and then delete the contents of the SRDSs.

To coldstart:

- Delete and reallocate the SRDS
- Delete the structure

The CQS coldstarts a structure if both the structure and the SRDS are empty.

If a structure checkpoint has been taken, the CQS rebuilds the structure. If the SRDS had been initialized previously but no structure checkpoint was taken, the CQS rebuilds from the log.

6.5 Availability Considerations

In this section we discuss the issues to take into account to achieve minimal impact on your online environment in a shared-queues environment.

The issues to carefully consider are:

- Frequency of structure checkpoints
- Number of coupling facilities in the sysplex
- Sizing of shared queue structures

If either IMS or the CQS abnormally terminates, transactions that were being processed by IMS at the time of the abend cannot recommence processing until both IMS and the CQS are operational and have resolved indoubt units of work.

6.5.1 Frequency of Structure Checkpoints

A structure checkpoint copies the contents of a structure pair (primary and overflow structures) to an SRDS. The SRDS is required for structure recovery. While the structure checkpoint is processing, access to the shared queue structure is quiesced during a portion of the checkpoint process. The CQSs connected to the structure cannot put any new messages or get any messages from the structure for some of the duration of the structure checkpoint. The frequency of structure checkpoints affects the duration of a structure recovery. The less frequently a structure checkpoint is performed, the greater the amount of CQS log stream needs to be read during a recovery. We recommend that a structure checkpoint be performed on a daily basis during quiet times, when the impact of quiescing the structure is minimized. Some customers are taking structure checkpoints each hour without noticing an impact on their system performance.

6.5.2 Number of Coupling Facilities

A minimum of one coupling facility is needed to implement shared queues. In a production environment, we recommend that you have at least two coupling facilities. The second coupling facility is required when the coupling facility that is holding the shared queue structures fails, and you need another coupling facility in which to rebuild your structures. We also recommend, from a performance aspect, that the coupling facility that is holding the CQS log stream be nonvolatile. If it is nonvolatile, the duplexing of the CQS log stream can be stored in a data space rather than on DASD. There is nothing stopping you from having a volatile coupling facility and duplexing to the data space, but it is a risk. Performance is achieved without incurring the extra cost of the nonvolatile coupling facility, but you cannot rebuild your CQS log stream if a power failure takes out both your coupling facility and one of the MVS systems where the data space was being stored.

6.5.3 Sizing of Shared Queue Structures

If the shared queue structures are not large enough, performance of shared queues is impacted. Initially, it is affected by overflow processing. A performance impact may be noticed when this occurs. Apart from the processing that is required to identify and move the queues to the overflow structure, the structure is also quiesced for part of the time while a structure checkpoint is taken. A structure checkpoint is always taken when overflow processing has completed. The real impact of undersizing a structure is felt when it fills up. When this occurs, requests to put an entry on the structure are rejected. IMS continually attempts to put the message on the structure, and it is not successful until a message is taken off the structure.

Chapter 7. Operating IMS in a Shared-Queues Environment

In this chapter we discuss the operational changes required for your IMS systems when you move from a stand-alone to a shared-queues environment. We consider:

- Operating procedures
- Implementing online change
- New commands
- Displaying the transaction queue

7.1 Operating Procedures

Some operations in a shared-queues environment differ from operations in a traditional local message queue environment. Additional processes must be performed at IMS startup and shutdown. Operating procedures must be set up for the CQS as well as for maintaining the structures used to hold the message queues and the logging information. We discuss the following procedures in this section:

- Starting IMS
- Stopping IMS
- Starting the CQS
- Stopping the CQS
- CQS failures
- CQS system checkpoints
- CQS structure checkpoints
- Deleting CQS coupling facility structures
- Resizing CQS coupling facility structures

7.1.1 Starting IMS

IMS startup is performed as usual. The change introduced with a shared-queues environment is that IMS starts the CQS in addition to starting database recovery control (DBRC) and the DL/I separate address space (DLISAS), if applicable. If its CQS is already active, IMS does attempt to start it. IMS identifies itself to the CQS subsystem as part of startup.

7.1.1.1 Coldstarting IMS

Coldstarting IMS resets all of the control blocks in the local IMS system but has no effect on the message queues in a shared-queues environment. Message queues are maintained in the coupling facility structure and are no longer related to a specific IMS system. IMS starts the CQS if it is not already active and connects to the CQS at initialization time.

If IMS was shut down cleanly, before the coldstart was issued, there is no effect on the shared queues. If IMS had failed to shut down successfully and was then cold started, any interactions between IMS and the CQS that were indoubt would be

moved from the lock queue to the cold queue and would remain in the cold queue until the coupling facility structure is deleted.

An example of an indoubt process is a transaction that was successfully bid for by the local IMS, and the transaction was then placed on the lock queue, while the local IMS system was processing the transaction. If IMS abended at this point and did not notify the CQS of the completed processing of the transaction, the transaction would be called an *indoubt process*. IMS/ESA Message Requeuer can be used to move the messages off the cold queue and place them on the shared queues to be processed.

7.1.1.2 Warmstarting IMS

A warm start of IMS has no effect on the shared queues. Again, IMS starts the CQS, if not already active, and connects to it at initialization time. A warm start implies the successful normal shutdown of IMS, which includes a normal shutdown of the CQS. IMS and the CQS are always resynchronized.

7.1.1.3 IMS Emergency Restart

IMS and the CQS are independent subsystems. If IMS fails, the CQS is not terminated. After you issue the `/ERESTART` command, IMS restarts. At restart time, IMS connects to the CQS and attempts to resynchronize with the CQS to resolve any indoubt processes. IMS and the CQS use their log records to resolve any indoubt processes, and this results in messages either being taken off the shared queue, if the messages have been successfully processed, or being moved from the lock queue to the appropriate transaction ready queue to enable processing of the message to be initiated.

7.1.2 Stopping IMS

To shut down IMS in a shared-queues environment, issue the command `/CHECKPOINT FREEZE`. Read 7.3.2, “CQS Commands” on page 108 for the different options of the commands to stop the CQS. After the `/CHECKPOINT` command has been issued, use the `/DISPLAY SHUTDOWN STATUS` command to check that IMS is shutting down successfully and is not waiting for resources.

Figure 54 shows an example of the `/DISPLAY SHUTDOWN STATUS` command where IMS is expecting the CQS to be active, but it is inactive. In this example, CQS job CQSIVP1 must be restarted before IMS can shut down.

```
/DISPLAY SHUTDOWN STATUS

DFS000I     TERMINAL  USER     STATUS   IVP1
DFS000I        1-   1          INPUT  IN PROCESS   IVP1
DFS000I     TERMINAL  USER     STATUS   IVP1
DFS000I       NO OUTPUTTING LINES   IVP1
DFS000I         MSG-IN    1     MSG-OUT   0   IVP1
DFS000I       MASTER ACTIVE   IVP1
DFS000I     IMSLU=N/A.N/A            #APPC-CONV=         0  DISABLED   I
DFS000I     OTMA PHASE=0    IVP1
DFS000I     CQS JOBNAME = CQSIVP1  NOT AVAILABLE    IVP1
DFS000I     *97296/201711*   IVP1
```

Figure 54. Shutdown Status Display Showing Wait CQS Unavailable

If the CQS is inactive when the /CHECKPOINT command is issued, the command is rejected with the following message:

```
DFS1937I IMS SHUTDOWN WAITING FOR CQS subsystem_name.
```

IMS shutdown cannot complete if the CQS is inactive.

During normal IMS shutdown, IMS tells the CQS to shut down. If there is no work inflight, the CQS shuts down. If the CQS does not shut down, you can stop it by issuing an MVS STOP command. If you want to leave the CQS running after IMS shutdown, you can use the NOCQSHUT keyword on the /CHECKPOINT command. (See 7.3.1, "IMS Commands" on page 106 for details of this command.)

7.1.2.1 Automation Hints and Tips

Update the IMS shutdown procedure to check that issuing the /CHECKPOINT command was successful.

Update the IMS shutdown procedures to ensure that IMS does not wait during shutdown for the CQS to be started. If message DFS1937I is present, start the CQS pointed to by the CQS JOBNAME field.

Update the IMS shutdown procedures to check that the CQS has been stopped within the IMS shutdown processing; otherwise, it may be necessary to automate the stopping of the CQS address space.

7.1.3 Starting the CQS

The CQS is started at IMS startup if it is not already active. If you require the CQS to be active before IMS is started, issue the MVS START command. Once IMS has been started and has initialized or restarted, it connects to the CQS subsystem.

The CQS always attempts to warmstart, unless it cannot find the system checkpoint from which to start. If the CQS cannot find the checkpoint location in the checkpoint data set, it reads its local entry on the shared queues to look for the checkpoint location. If the CQS finds the checkpoint information, it issues a write to operator with reply (WTOR) to confirm the use of that log token. The operator can either confirm the use of that log token, suggest an alternative log token, coldstart the CQS, or cancel the CQS. The message identifier for the WTOR is:

```
CQS0031A CONFIRM CQS RESTART FOR STRUCTURE structurename,
           FROM CHECKPOINT LOGTOKEN logtoken
```

or

```
CQS0032A ENTER CHECKPOINT LOGTOKEN FOR CQS RESTART FOR STRUCTURE structurename
```

During a restart of a CQS system, the CQS reads the log records from the last CQS system checkpoint (remember, system checkpoints are local to the CQS), restores the environment for committed data, and backs out uncommitted data objects. If the CQS was shut down normally, a CQS system checkpoint would have been taken at shutdown time, and the environment can be restored from the checkpoint information. If the CQS abended, and the last system checkpoint was taken a long time ago, the last system checkpoint would be read from the log, and all subsequent log records for that CQS would have to be read and applied to the CQS environment to restore the environment to just before the abend.

The CQS purges log records that are older than two structure checkpoints. You must ensure than no CQS address space is inactive for that length of time, or the CQS cannot successfully warmstart. (It can always coldstart.) You can reduce the possibility of this situation by following the correct operating procedures for the system and structure checkpoints. The IMS startup procedures must be updated to ensure that the automated procedures and the operators are aware of the CQS WTORs that may occur at startup. Refer to 7.1.6, “CQS System Checkpoints” on page 103 and 7.1.7, “CQS Structure Checkpoints” on page 104 for a detailed explanation of system and structure checkpoints.

7.1.3.1 Forcing a Cold Start of the CQS

It is possible to force a cold start of the CQS by deleting and redefining the CQS MSGQ and EMHQ checkpoint data sets. When the CQS restarts, it will be unable to find the log token pointing to the checkpoint information in the MVS log stream. The following message is issued:

```
CQS0032A ENTER CHECKPOINT LOGTOKEN FOR CQS RESTART FOR STRUCTURE structurename
```

The operator can reply either `COLD`, `CANCEL`, or `logtoken`. If the operator replies `COLD`, the CQS will be brought up with a cold start.

If only the MSGQ checkpoint data set is deleted and redefined, the EMHQ control blocks in the CQS are still restored with the checkpoint and log stream information. It is therefore essential to delete and redefine both the MSGQ and EMHQ checkpoint data sets.

7.1.4 Stopping the CQS

If IMS is being brought down normally, the CQS is told to shut down. If there is no work in progress, the CQS shuts down.

The CQS is brought down during the normal shutdown of the IMS system with which it is associated. If an IMS failure occurs, the CQS is not shut down because it is independent of IMS. IMS is a client, and the CQS is the server.

Typically, the CQS should not be shut down while the associated IMS is active. If it is necessary to stop the CQS, you can cancel it, using the MVS `CANCEL` command. If the CQS is stopped while IMS is active, IMS no longer has access to the shared queues until the CQS is restarted. Any work that was in progress on that IMS remains on the lock queue and is not available to any IMS system in the same shared-queues group for processing until the CQS has restarted and determined the status of the messages in the lock queue.

If the CQS is not brought down by IMS at IMS normal shutdown time, you can issue an MVS `STOP` command to stop the CQS.

7.1.5 CQS Failures

A failure of the CQS does not imply that IMS will abend. IMS stays active across a failure in the CQS.

If the CQS is defined to ARM, and the CQS fails for some reason, it is automatically restarted. You must ensure that the CQS is restarted on the same MVS system as its client IMS system.

If the CQS is not defined to ARM, issue the start command for the CQS from the MVS master console. If the procedure name for the CQS started task is CQSPROC, the MVS command to start the CQS is:

```
S CQSPROC
```

Once the CQS has been restarted, if IMS did not abend, IMS connects to the CQS system and attempts to resynchronize with CQS. If the resynchronization is successful, IMS begins processing the shared queues. IMS initiates the connection to the CQS and the resynchronization. The CQS will not read any logs if the resynchronization is unsuccessful.

After the CQS has failed, the client IMS cannot access the shared queue until the CQS has been restarted. IMS waits until the CQS is active and the connection between IMS and the CQS has been reestablished before processing shared queues. IMS should still be able to process Fast Path messages if the `LOCAL ONLY` or `LOCAL FIRST` option has been specified. Recovery of Fast Path messages is not necessary, so access to the shared queue structure on the coupling facility is not necessary. IMS cannot process full function messages, because the CQS and the coupling facility are needed to place the transaction on a shared queue.

7.1.6 CQS System Checkpoints

A system checkpoint is necessary for the CQS to log restart and recovery information.

The CQS performs a system checkpoint in the following situations:

- Automatically, based on the CQS log volume. The SYSCHKPT parameter in the CQSSLxxx proclib member defines how many CQS log records are to be written before a checkpoint is issued.
- Manually if the `/CQCHKPT SYSTEM` command is entered
- At the completion of a successful structure checkpoint
- At CQS shutdown
- At the completion of a CQS restart
- As part of resynchronization between IMS and CQS

To initiate a CQS system checkpoint, issue either of the following commands:

```
/CQCHKPT SHAREDQ SYSTEM ALL
```

or

```
/CQCHKPT SHAREDQ SYSTEM structurename
```

These commands are local to the CQS system and should be issued on each CQS system. Processing is not quiesced during CQS system checkpointing.

7.1.7 CQS Structure Checkpoints

The CQS uses structure checkpoint information to repopulate the coupling facility structure in the event of a structure recovery. The information in the structure checkpoint is used to recover the shared queue.

The CQS performs structure checkpoints in the following situations:

- When the MVS log stream approaches a full status
- After a successful structure rebuild (unless it was a structure copy initiated by an operator)
- After a successful overflow threshold process
- After the `/CQCHKPT SHAREDQ STRUCTURE structurename` command is entered to any CQS in the shared-queues group
- At CQS shutdown if the `/CQSET SHUTDOWNSHAREDQ` command had been previously issued

The command that can be issued from IMS is either the `/CQCHKPT SHAREDQ STRUCTURE ALL` or `/CQCHKPT SHAREDQ STRUCTURE structurename` command.

We recommend that a structure checkpoint take place at least once each day, to reduce the number of log records that are kept by the MVS logger. At structure checkpoint time, log records that are older than two structure checkpoints are deleted. Some customers are taking structure checkpoints every hour without noticing an impact on system performance.

During structure checkpoint, the structure being checkpointed is quiesced. Therefore processing of shared queues cannot take place while the checkpoint is in progress. You must ensure that structure checkpoints occur at controlled intervals, to lessen the impact to online processing..

7.1.8 Deleting CQS Coupling Facility Structures

Deleting the CQS coupling facility structures removes all messages that are on the shared queues. It is equivalent to a cold start of IMS in a local queue environment. If IMS/ESA Message Requeuer or a similar product is not available at your site, deleting the coupling facility structures is the only way to remove messages that are on the cold queue. Remember that the CQS attempts to recover the messages if it finds an empty structure at startup time. Therefore, all structure recovery information in the SRDSs must be removed before the CQS is started. You can remove the information by deleting and redefining both CQS SRDSs. Once the SRDSs have been deleted, there is no way of restoring the message queues.

If IMS/ESA Message Requeuer is available for use, it would be better to offload all the messages on the message queue to an external data set before deleting and redefining the SRDSs. With that approach, messages can be placed back onto the message queue if necessary.

The command to delete a coupling facility structure is:

```
SETXCF FORCE,STRUCTURE,STRNAME=structurename
```

Remember to delete the MSGQ and EMHQ structures and their overflow structures if defined. All CQSs that access the coupling facility structures must be stopped.

The SETXCF FORCE command cannot be used while connections to the coupling facility structures exist.

You can use the SETXCF FORCE command to delete failed persistent connections. For example, if a CQS abended, you do not want to bring it back up, and you want to delete the structure.

In summary, the steps to be followed to delete the CQS coupling facility structures are:

1. Bring down all IMS systems in the shared-queues environment normally.
2. Ensure that all CQS systems in the shared-queues environment have been stopped.
3. Delete the coupling facility structures.
4. Delete and redefine both CQS SRDSs.
5. Start each IMS system (and each CQS) in the shared-queues environment.

7.1.9 Resizing CQS Coupling Facility Structures

If the shared queue structures are becoming too full, and it is possible for the structures to run out of space, you may have to alter the coupling facility structure sizes. The CFRM and LOGR policies have to be updated, and the relevant policy activated. The changes remain pending until the structure is REBUILT. You issue a SETXCF command to initiate a structure rebuild. It is important to remember that the structure is quiesced during some stages of the structure rebuild. This has an impact on every IMS system in the shared-queues group that accesses the structure.

The sequence of events for resizing the structure are:

1. Update the CFRM and LOGR policies.
2. Stop the old CFRM policy.
3. Activate the new CFRM policy.
4. Initiate a structure rebuild or structure copy by issuing this command:

   ```
   SETXCF START,REBUILD,STRNAME=structurename
   ```

5. Monitor the structures to ensure that the sizing changes were performed correctly.

7.2 Implementing Online Change

IMS does not coordinate online change across a sysplex or in a shared-queues environment. We recommend that the control blocks relating to transactions and PSBs in a shared-queues environment always be the same. Therefore online change has to be performed on each IMS system in the same shared-queues group at the same time.

Online change is less prone to error if:

- The IMS system definition decks are cloned
- MATRIX and MODBLKS are shared (if identical)
- ACBLIB and FMTLIB are shared

- MODSTAT IDs are identical

Because the commands used to implement an online change are not global, there is a recommended sequence of events for implementing an online change in a shared-queues environment:

1. Ensure that all systems in the shared-queues group are active on the same MODSTAT libraries. Issue `/DISPLAY MODIFY` on each IMS to ensure that the active libraries are the same, if shared.
2. Copy the updated libraries to the inactive libraries to be changed.
3. Enter `/MODIFY PREPARE` on each IMS system in the shared-queues group.
4. Enter `/DISPLAY MODIFY` on each IMS system in the shared-queues group. Any work in progress must be resolved on each system before this step is complete.
5. Enter `/MODIFY COMMIT` on each IMS system in the shared-queues group.

If the online change fails, you must reverse the change on each IMS system in the shared-queues group by issuing `/MODIFY ABORT` on each IMS where the change has not already been committed. If the change has been committed, issue another `/MODIFY PREPARE` and `/MODIFY COMMIT` pair to revert back to before the change was implemented.

If online change implementation does not follow the recommended procedure, it is possible to have inconsistencies between IMS systems in the same shared-queues group. Consider the scenario when a transaction is entered on the local IMS but can process on another IMS system. The responses received for that transaction can be different depending on the IMS system in which it was processed and the stage of implementing the online change on all IMS systems.

7.3 New Commands

In this section we discuss the new commands that are used to operate IMS in a shared-queues environment. These commands are grouped into three categories:

- IMS commands
- CQS commands
- XCF commands

We end the discussion with a review of the IMS commands that are not effective in a shared-queues environment. Throughout this section, a *local IMS* refers to the IMS from which the command was entered, and a *local CQS* refers to the CQS connected to the IMS from which the command was entered.

7.3.1 IMS Commands

For more information about these commands, see the *IMS/ESA Operator's Reference*, SC26-8742.

- `/CHECKPOINT FREEZE NOCQSSHUT`

  Shuts down IMS without shutting down the CQS.

- `/DISPLAY QCNT TRAN|LTERM|BALG|APPC|OTMA|REMOTE MSGAGE n`

Displays the queue count of the resource type in the shared queue environment. The MSGAGE parameter restricts the display to resources that have had messages queued for more than *n* days. Using a MSGAGE of 0 displays all resources that have messages queued.

- `/DEQUEUE TRAN transaction PURGE|PURGE1`

  Dequeues transactions from the transaction ready queue. TRAN is a new keyword on the dequeue command. The transaction must be stopped in order for it to be dequeued.

- `/DEQUEUE SUSPEND`

  Moves all transactions on the transaction suspend queue to the transaction ready queue. It also resets the "suspend" status of the transaction on the local IMS only. A `/START TRANSACTION` must be issued on every other IMS system in the same shared-queues environment to reset the status of the transaction.

- `/DISPLAY CQS`

  Displays information about the CQS to which IMS is or was connected.

- `/DISPLAY OVERFLOWQ STRUCTURE ALL|structurename`

  Displays a list of queue names that are in overflow status for the coupling facility structure displayed

- `/DISPLAY TMEMBER|TRAN|NODE|MSNAME|LUNAME|LTERM|LINK|LINE xxxxx QCNT`

  This DISPLAY command has a new parameter, QCNT, to give the global queue count of the resource specified in the shared-queues environment. Without the QCNT parameter, the display is of the same format with shared queues, but the ENQ and DEQ counts will be the same, and the QCNT will be zero.

- `/DISPLAY STRUCTURE ALL|structurename`

  Displays the status of one or more structure types

- `/TRACE SET ON|OFF TABLE QMGR|SQTT`

  The TRACE command has two new table traces; the queue manager trace, QMGR, and the shared-queues trace, SQTT.

- `/DISPLAY TRACE TABLE`

  The information displayed includes entries for QMGR and SQTT.

- `/ASSIGN` and `/MSASSIGN`

  Any /ASSIGN and /MSASSIGN commands must be entered at every IMS system in the same shared-queues environment to ensure that the control blocks are synchronized across all IMS systems.

- `/DISPLAY SHUTDOWN STATUS`

  The information displayed includes the CQS name that IMS must connect to, if the CQS is not active; otherwise the shutdown hangs.

- `/MODIFY PREPARE|ABORT|COMMIT`

  Care must be taken with these commands to ensure that each IMS system in a shared-queues environment always has the same definitions for the same transactions. In a shared-queues environment, the control blocks must be synchronized for each IMS system. The PREPARE and ABORT commands

can now be used with an automated operator interface, but COMMIT is still not supported.

7.3.2 CQS Commands

CQS commands are issued in the same way as IMS commands:

- From the IMS console
- The IMS outstanding reply
- From a terminal logged on to IMS

For more information about the commands that can be issued from IMS to the CQS, refer to the *IMS/ESA Operator's Reference*, SC26-8742.

- `/CQCHKPT SHAREDQ STRUCTURE ALL|structurename`

  Initiates a structure checkpoint of all structures or a specified structure. The structure checkpoint takes a copy of the recoverable data objects in the structure.

- `/CQCHKPT SYSTEM STRUCTURE ALL|structurename`

  Initiates a system checkpoint on the local CQS. The system checkpoint takes a copy of the control blocks and tables in the local CQS address space necessary for restart and dumps it to the MVS log stream. The location of the checkpoint information is logged in the local CQS checkpoint data set.

- `/CQSET SHUTDOWN SHAREDQ ON|OFF STRUCTURE ALL|structurename`

  Indicates to CQS that a structure checkpoint should be taken at CQS shutdown time. The command has an ON and an OFF parameter. This command, if issued, should only be issued on one CQS in each shared-queues environment. If a shutdown of more than one CQS in the shared-queues environment is scheduled, and a shutdown checkpoint has been scheduled in more than one CQS, more than one checkpoint of the structure is taken. This increases the time taken for the CQS and, therefore, IMS to shut down, as well as causing a hang in the IMS systems that are still active in that shared queues environment.

- `/CQQUERY STATISTICS STRUCTURE ALL|structurename`

  Displays information about all or specified structure types, with fields showing the List Entries (LE) and Data Elements (DE) used for each structure. The maximum and current number in use are displayed. The maximum number of list entries is equivalent to the maximum number of data objects that can be put on the structure.

7.3.3 XCF Commands

There are several XCF commands that you may need to use in a shared-queues environment. For more information about these commands, see *OS/390 MVS System Commands*, GC28-1781.

- `D CF`

  Displays information about the coupling facilities connected to the MVS

- `D XCF,STRUCTURE`

  Displays a list of coupling facility structures and their status

- `D XCF,STRUCTURE,STRNAME=structure`

Displays information about the specific structure

- `D XCF,POLICY`

  Displays status information about the policies defined in the sysplex

- `SETXCF START,POLICY,TYPE=CFRM,POLNAME=policy`

  Activates a new policy

- `SETXCF START,ALTER,STRNAME=structure`

  Initiates the altering of a specific structure on the current coupling facility

- `SETXCF START,REBUILD,STRNAME=structure`

  Initiates the rebuild of a specific structure on the current coupling facility

- `SETXCF START,REBUILD,STRNAME=structure,LOCATION=OTHER`

  Initiates the rebuild of a specific structure on another coupling facility defined in the PREFLIST parameter in the CFRM structure definition

- `SETXCF FORCE,STRUCTURE,STRNAME=structure`

  Deletes a specific structure in the coupling facility. The connections to the structure must be inactive for the command to be successful.

7.3.4 Commands Not Effective with Shared Queues

Several IMS commands should be used only in a non-shared-queues environment:

- `/DISPLAY TRANSACTION Q`

  Displays the queues on the local IMS system. The display always shows *NO QUEUES*.

- `/DISPLAY Q`

  Displays the queues on the local IMS system. The display always shows *NO QUEUES*.

- `/NRESTART BUILDQ FORMAT SM|LM|QC`

  The SM, LM, and QC parameters are ignored. The formatting of the queue data sets is not supported in a shared-queues environment.

- `/CHECKPOINT SNAPQ|DUMPQ|PURGE`

  The SNAPQ, DUMPQ, and PURGE commands do not cause a dump or purge of the message queues. The `/CQSET` command must be used to initiate a checkpoint of the message queue structures at shutdown. DUMPQ and PURGE still cause IMS to be shut down but without having any effect on the shared queues.

- `/ERESTART BUILDQ`

  environment because the queue data sets are no longer used.

7.4 Displaying the Transaction Queue

The examples in this section show the new commands to display the transaction queue and their output.

Figure 55 on page 110 shows the output of the command to display the transactions with a queue count of greater than zero at the current time. This

display shows the number of transactions on the shared queue for full function and Fast Path transactions, by transaction name. This command is the equivalent of the /DISPLAY TRAN Q command in an IMS system with local message queues.

```
/DISPLAY QCNT TRAN MSGAGE 0

Response:
  QUEUENAME          QCNT-TOTAL     QCNT-AGED    TSTMP-OLD    TSTMP-NEW
 IVTNO                        7             7    97294/175153 97295/13410
 IVTNV                        2             2    97295/134711 97295/13471
 *97295/142433*
```

Figure 55. Displaying the Transaction Queue: Count of Zero at Current Time

Figure 56 shows the output of the command to display the transactions with a queue count greater than zero on the local IMS system. In a shared-queues environment, the output should show *NO QUEUES*.

```
/DISPLAY Q TRANSACTION

Response:
      CLS     PTY     MSG CT TRAN     PSBNAME
     *NO QUEUES*
     *97295/142331*
```

Figure 56. Displaying the Transaction Queue: Count of Zero on Local IMS

Figure 57 shows a similar display to Figure 56. The queue count should always be zero, because in a shared-queues environment there are no local queues.

```
/DISPLAY Q

Response:
    CLS CT  PTY CT  MSG CT TRAN CT
   *NO QUEUES*
   *97295/143727*
```

Figure 57. Displaying the Transaction Queue in a Shared-Queues Environment

Figure 58 shows the output of the command to display a specific transaction in a shared-queues environment. The display shows the global count of all messages queued to that transaction in the shared queue.

```
/DISPLAY TRAN IVTNO QCNT

Response:
   TRAN          GBLQCT
  IVTNO              7
  *97295/143208*
```

Figure 58. Output of Display Transaction Command: Global Queue Count

Figure 59 on page 111 shows the output of the command to display a specific transaction on a local system. The QCT should always be 0 because the transactions are not queued in the local queue buffer pool but in the shared queue structure on the coupling facility. The ENQCT shows the number of transactions that have arrived in the local IMS system. The enqueue count (ENQCT) of a transaction increases by 1 when the transaction is entered on the local IMS, and increases by 1 if the transaction is queued on the shared queue and taken off the shared queue to be processed by the local IMS.

```
/DISPLAY TRAN IVTNO
Response:
   TRAN     CLS ENQCT   QCT   LCT  PLCT CP NP LP SEGSZ SEGNO PARLM    RC
  IVTNO      1    25     0 65535 65535  1  1  1     0     0  NONE     0
      PSBNAME: DFSIVP1
      STATUS: LOCK
  *97295/142601*
```

Figure 59. Output of Display Transaction Command on a Local System

Chapter 8. Preparing Your IMS Environment for Shared Queues

In this chapter we introduce the changes you may need to make to your IMS environment when you implement shared queues.

8.1 Planning for Shared Queues

The implementation of shared queues involves a lot of changes in your current IMS application environment. Before you can start planning, you must have understood the major design and functionality goals and their implications for your existing IMS system implementation.

Today in many customer IMS installations there are functional dependencies, affinities created by the limit of hardware capacity, the terminal network, or the data accessed by the application system. In general, affinities are frequently a single point of failure and reduce customer expectations of having their applications system continuously available. Some affinities can be removed easily, some may need further investigation to implement changes in their applications, and others may force you to install new software releases and/or hardware before setting up a shared queues project.

8.2 Local Processing

The major approach of shared queues is to assign work to any IMS that has the resources to process it. When feasible, however, transactions should be processed locally with minimal intervention to shared queues. If a dependent region is waiting and eligible to process the transaction, the entire message fits in one queue buffer, and the transaction is not serial, processing the transaction locally is the best option. Executing the transaction workload in an optimal manner is the highest priority goal. If an IMS does not have the resources to process a particular transaction, another IMS in the shared-queues group can process it. This design approach guarantees achieving the best performance in a shared-queues environment.

8.3 Affinities

When a transaction must run in a specific, single IMS environment in order to run successfully, we say it has affinity status; that is, the transaction has an affinity for that IMS. A good example is a device that keeps data and is not shared for access by other MVS systems. Some examples of affinities are applications that:

- Access IMS databases that are not shared
- Access main storage databases (MSDBs)
- Access data entry databases (DEDBs) with sequential dependent segments (SDEPs). These databases cannot be shared until IMS/ESA Version 6.
- Access DEDBs with the virtual storage option (VSO). These databases cannot be shared until IMS/ESA Version 6.
- Access DB2 databases that are not defined as shareable

- Access data in storage (incore data)
- Access devices or files that have physical connectivity affinities
- Have terminals physically connected to a particular IMS system to process transactions allowed to be entered from these terminals
- Serialize application processing by MAXRGN=1 or SERIAL defined transactions
- Have APPC or OTMA transactions, which are only processed by that IMS system where entered

Obviously not all affinities can be eliminated immediately. Some require further investigation, others can be solved quickly. Affinities need not be a bottleneck for the implementation of shared queues. Initially, you may create and develop an IMS sysplex with just those applications that you have identified as eligible for shared-queues processing. Subsequently, based on your experience with shared queues in an IMS sysplex, you may continue to further integrate other applications after removing their affinities or simply isolating them within your IMS sysplex.

It may be an advantage to start with a subpart of your application systems and their shared databases to realize the advantages and benefits to your IMS subsystem installation.

For more information, see Chapter 12, “Significant Status and Affinities” on page 141.

8.4 IMS Queue Manager

The traditional IMS queue manager manages the short and long message queue data sets and the QBLKS data set. Although the data sets are not used to back up messages in a shared-queues environment, the MSGQUEUE macro is still required, to facilitate fallback and reduce necessary changes to the IMS system definition. Messages that remain in the local system are held in the local queue buffers. The size has to be reevaluated with shared queues even though external data sets are never used. A message on the local queue is always recoverable from the queue checkpoints.

For more information, see 5.6, “Define IMS Data Sets” on page 63.

8.5 Transaction Processing

When a local transaction is received in the queue buffers and a dependent region is available on the local IMS, the input transaction is placed directly on the lock queue in the shared queues. The dependent region schedules and receives the input transaction from the queue buffers. If an associated MPP cannot process the transaction, the input is placed on the transaction ready queue. Scheduling in a shared-queues environment ignores references to queue counts, therefore:

- The PARLIM count is ignored but still must be coded.
- The MAXRGN limit is honored.
- Limit priority is not honored.

8.6 Serial Transaction Processing

Serial transactions are processed only by the IMS that places the message on the queue. Serialization is in effect only within a single IMS and not across the IMS sysplex. If you require serialization across the IMS sysplex, several options are possible.

For more information, see Chapter 15, “Serial Transactions” on page 151.

8.7 Transaction Destinations

Transactions must be defined in all of the IMS systems that can process them. You can define transactions in all IMS systems with a message class that has no active dependent region. This approach enables you to enter transactions but have them processed in some system other than in this shared-queues group.

When transactions are not defined, IMS can specify the destination name dynamically through the output creation exit routine (DFSINSX0) to execute only in a shared group environment. See 8.18, “Undefined Resources” on page 118 for more information.

8.8 LTERM Destinations (Static)

To place an output message on the shared LTERM queue when ETO is not enabled, the LTERM must be defined in at least one IMS system.

8.9 Multiple LTERM Names (Static)

Several problems arise in a shared-queues environment when the same LTERM is defined in multiple IMS systems. In a single IMS, the DC definitions are automatically consistent (validation by IMS sysgen). IMS does not have a global mechanism for ensuring LTERM names are unique across multiple IMS systems in a shared-queues environment. It is the user's responsibility to avoid or handle those conflicts.

The objective in your installation should be to make your terminal definitions consistent *across* the shared-queues group.

A simple and safe solution is to create one front-end IMS containing all terminal definitions, with just one CQS registering interest in the LTERMs and only one IMS being notified of the existence of output messages. The single point of failure may be reduced by using XRF to back up the front end.

Another alternative for creating unique LTERM names in a shared-queues group is to use the terminal signon user exit (DFSSGNX0). During signon the exit replaces the LTERM name with a name that is unique in the sysplex.

See Chapter 14, “Duplicate LTERM Names and Output Security” on page 149 for more information.

8.10 LTERM Destinations (ETO)

The use of shared queues may not be compatible with some ETO implementations. Some ETO users may not have unique LTERM names across a shared-queues environment. In such cases you can:

- Add a suffix to a common set of characters to make the LTERM names for a given user unique.

  or

- Associate the LTERM name with a user structure, where the LTERM name is selected by a table keyed by userid.

As with static LTERMs, prevent the creation of the *same active* dynamic LTERM name within the shared-queues group.

Assign a unique suffix to each IMS or use some other method for naming ETO LTERM destinations. Reset significant status with the terminal signoff user exit (DFSSGFX0) to cause user structures to be deleted at signoff.

Refer to Chapter 9, “Extended Terminal Option” on page 121 for further details.

8.11 Shared Printer and ETO Autologon

In a shared-queues environment, multiple IMS systems may compete to deliver output from a specific queue (LTERM) name. This happens when a shared printer is defined in multiple systems, is active in more than one system, or is subject to autologon from two or more IMS systems. Ensure in all cases that the LTERM name or the LUNAME (AUTOLOGON=Y) is correct. Investigate in more detail through the output creation exit routine (DFSINSX0).

For more information, see 9.5, “Shared Printers and AUTOLOGON” on page 123.

8.12 Remote Transactions and LTERMs

Remote transactions and remote LTERMs do not have to be defined in the originating or local IMS system in the shared-queues group, but they *must* be known to the destination IMS (MSC-linked remote IMS). The output creation exit routine, DFSINSX0, designates the input message as a local or remote transaction or LTERM.

For more information, see Chapter 11, “Multiple Systems Coupling” on page 135 and 10.6, “Multiple Systems Coupling” on page 134.

8.13 Conversational Processing

In a shared-queues environment, the conversation status is local to the IMS system that initiates the conversation. Any IMS system that registers interest in this transaction can process any iteration of the conversation. The message response is of register interest by the local IMS that is in conversational mode.

Note: OTMA and APPC conversations are always processed locally.

The SPA is saved as the first segment of a message on shared queues; the SPA pool is no longer used. This change affects the current size of the queue buffers used in the local IMS.

Whenever a user logs off during a conversation and logs on to another IMS, the new session begins a new conversation; the original is conversation is suspended. When a user logs on again to the IMS that has maintained the conversational control block (CCB), that IMS continues the suspended conversation.

For more information, see Chapter 10, “Conversational Programs” on page 125.

8.14 Suspended Transactions

When a transaction is suspended, it is placed on the transaction suspend queue in the coupling facility. To make such transactions available for processing, you must issue the `/DEQUEUE SUSPEND` command, which moves all transactions to either the transaction ready queue or the transaction serial queue.

The suspend status is reset locally where the `/DEQUEUE` command was entered. To reset the suspend status across the sysplex, you must issue the `/START TRANSACTION` command on each IMS system.

8.15 AOI Transactions

All AOI transactions scheduled by the AOI exit routine should be defined as serial to ensure local processing. Placing an AOI transaction on the shared queues allows any IMS to process it, regardless of whether the command applies to that IMS system or not.

8.16 APPC and OTMA Transactions

APPC and OTMA transactions are processed locally by the IMS system that received the transaction. The first input message segment is queued in either the MSGQ or EMHQ structure on the transaction ready queue. Subsequent message segments are queued on the transaction staging queue for recoverability. The first output message segment is stored on the APPC or OTMA ready queue. All second to nth message segments from APPC or OTMA are queued on the LTERM staging queue.

Output messages must be delivered by the IMS system that originally received the input message.

For more information, see Chapter 16, “APPC and OTMA Processing in a Shared-Queues Environment” on page 155.

8.17 IMS Dead Letter Queue

A dead letter queue (DEADQ) is a queue where the output messages have not been accessed for a specified time. In a shared-queues environment, IMS maintains a dead letter queue only for local user structures. For the shared queues on the coupling facility, it is necessary to issue a new command

```
/DISPLAY QCNT MSGAGE
```

to display potential dead letter queues. This command displays global queue information for a specified resource type.

For more information, see 9.3, “IMS Dead Letter Queue” on page 121.

8.18 Undefined Resources

Any messages or transactions received in a local IMS must find their destinations before they are put on the shared queues. If the destination is not defined locally (IMS does not know which resources are defined in other IMSs), it can be defined dynamically through the output creation exit, DFSINSX0.

ISRT to IOPCB is considered to be a valid destination, and ISRT to an undefined ALTPCB destination is rejected when ETO is not active. With ETO, invoke the DFSINSX0 exit to define a message as an LTERM or transaction.

For more information, see Chapter 13, “Undefined Resources” on page 145.

8.19 Fast Path Options

The Fast Path input edit routine (DBFHAGU0) determines whether Fast Path messages are eligible for IMS sysplex processing. This routine assigns one of three sysplex processing codes to the message to decide where it is to be processed. The receiving message is either processed locally (LOCAL ONLY), prioritized for local processing first when an IFP region for the particular transaction is available (LOCAL FIRST), or put to the shared queue for GLOBAL ONLY processing. LOCAL ONLY restricts shared-queue processing. LOCAL FIRST places the input message on the EMHQ structure, and the local IMS system does not have the capacity to process it. GLOBAL ONLY specifies that the message is always stored to the EMHQ coupling facility structure.

For more information, see 3.2, “Processing a Fast Path Transaction” on page 28.

8.20 Shared Queues and MSC

Messages destined for SYSIDs in the same shared-queues group are put on the shared queues instead of being sent across the MSC link. MSC links can be defined between systems in the same shared-queues group for backup purposes, but they cannot be started.

IMS systems in a shared-queues environment can communicate with each other through MSC only if they are *not* in the same shared-queues group.

Messages can be received through MSC from a remote IMS system outside a shared-queues group. They are placed on the shared queues and can be processed by any member of the shared-queues group.

Messages can be sent to IMS systems outside the shared-queues group. They are sent by whichever IMS has the active link.

Each IMS in a shared-queues group *must* be genned with the MSC function (minimum of one macro each: MSPLINK, MSLINK, and MSNAME) if messages are sent through MSC to an IMS system outside the shared-queues group.

For more information, see Chapter 11, “Multiple Systems Coupling” on page 135, and 10.4.2, “Multiple Systems Coupling” on page 131, and 10.6, “Multiple Systems Coupling” on page 134.

8.21 IMS Online Change

IMS does not coordinate online changes across a sysplex. To simplify the process, we recommend running with a cloned configuration and sharing libraries (ACBLIB, MODBLKS, MATRIX, FORMAT) across the sysplex. MODSTAT IDs must be the same across the sysplex. A procedure is described in 7.2, “Implementing Online Change” on page 105.

Chapter 9. Extended Terminal Option

In this chapter we describe the processing in a shared-queues environment using ETO and dynamic terminals. Terminals created dynamically behave differently from static terminals, and the differences are more pronounced in a shared-queues environment. These differences and their impact must be understood, so that you can cater for any problems or inconsistencies in your planning for shared queues.

9.1 Shared Queue Types Used with Dynamic LTERMs

The shared queue types that can be used during processing of messages involving dynamic terminals are the same as those used for static terminals, namely:

- LTERM ready queue on MSGQ structure
- LTERM staging queue on MSGQ structure
- LTERM ready queue on EMHQ structure

IMS registers interest only in the following queue types:

- LTERM ready queue on MSGQ structure
- LTERM ready queue on EMHQ structure

9.2 How IMS Registers Interest in Dynamic LTERMs

IMS registers interest in the LTERM name created when the user signs on. IMS registers interest in the LTERM ready queue on the MSGQ structure when the user signs on. For Fast Path, IMS registers interest in the LTERM ready queue on the EHMQ structure when the first Fast Path transaction is entered.

IMS deregisters interest in the LTERM on the LTERM ready queue on both the MSGQ and EMHQ structures when the user signs off or IMS terminates.

9.3 IMS Dead Letter Queue

Before the implementation of shared queues, the dead letter queue was used to store messages queued to dynamic LTERMs for more than a defined length of time. The length of time was specified with the `DLQT` parameter in the DFSPBxxx proclib member. The command `/DISPLAY USER DEADQ` displayed the messages (dead letters) that had exceeded the defined length of time. The messages could be dequeued by issuing the `/DEQUEUE USER` command, thus deleting the messages off the message queue.

LTERMs are dynamic control blocks created on the local IMS system where signon occurred. In a shared-queues environment, IMS does not have access to the number of messages queued to a local LTERM. The dead letter queue therefore does not exist in a shared-queues environment. If shared queues is not implemented, the `/DISPLAY USER DEADQ` works as before.

A similar concept to the dead letter queue has been introduced to IMS with shared queues. This is the `MSGAGE` keyword on the `/DISPLAY QCNT LTERM` command. To check the number of messages on the equivalent of a DEADQ, issue the command

with MSGAGE equal to the defined limit of the DEADQ in your environment (see Figure 60 on page 122).

The DISPLAY QCNT LTERM MSGAGE 5 command is explained in 7.4, “Displaying the Transaction Queue” on page 109. For more detailed information, see Chapter 7, “Operating IMS in a Shared-Queues Environment” on page 99, and the *IMS/ESA Operator's Reference*, SC26-8742.

```
/DISPLAY QCNT LTERM MSGAGE 5

 QUEUENAME          QCNT-TOTAL     QCNT-AGED    TSTMP-OLD    TSTMP-NEW
USER6                       16            16   97306/121516 97317/081926
USER7                        4             1   97317/081257 97321/124017
USER1                        3             3   97317/233414 97317/152708
USER2                        9             7   97312/151011 97319/191411
USER3                       25             5   97309/175013 97322/135213
USER4                       12             9   97307/100019 97320/120915
*97322/151236*
```

Figure 60. Sample /DISPLAY QCNT LTERM MSGAGE Command

9.4 Duplicate LTERM Names

IMS registers interest in dynamic LTERMs when they are created at signon time. If a user signs on to many IMSs in a shared-queues group, and the DFSSGNX0 signon exit is not used, the LTERMs may be created with the same name. The name defaults to the userid that signed on. If the user has signed on to the same IMS system more than once or signed on to more than one IMS system in the shared-queues group, inconsistencies in processing can occur. More than one IMS system has registered an interest in the LTERM, and the responses created by a user on one node may be received by a user on another node. There can be no guarantee that the responses received by the duplicate LTERMs are the responses related to the input message that was entered at the terminal.

Consider this example: LTERMA is created on both IMSA and IMSB because USERA signed on to both IMS systems from two different sessions. TRANA is entered on IMSA, and TRANB is entered on IMSB. Both responses are queued to LTERMA on the LTERM ready queue. Both IMSA and IMSB have registered interest in LTERMA on the LTERM ready queue. If TRANA completes processing before TRANB, the first response on LTERMA queue is TRANA's. This response should be delivered to LTERMA on IMSA because that is the LTERM that issued the transaction. If IMSB is the first IMS to dequeue a message from the LTERMA queue, LTERMA on IMSB receives the response to TRANA.

IMS cannot cross-check LTERM names across IMS systems within a shared-queues environment. Signon exit DFSSGNX0 can determine whether the same LTERM name exists within that IMS and should be tailored to alter the LTERM name in some way to ensure that the LTERM name is unique. Duplicate LTERM names are a restriction in IMS shared queues that is not enforced. It is the IMS system programmer's responsibility to ensure that the processing results in a shared-queues environment using ETO are consistent.

To ensure that dynamic LTERM names are unique within a shared-queues group, it may be necessary to implement an LTERM naming standard that has an IMS identifier in the name. It may also be necessary to prevent multiple signons to a single IMS system. If multiple signon is required, a unique LTERM name must be generated.

Duplicate LTERM names can be created in a shared-queues environment in the following situations:

- Same user signs on to multiple IMS systems in a shared-queues group.
- The signon exit DFSSGNX0 routine generates the same LTERM name for more than one user.
- Multiple signons by one user are allowed.
- The Signon exit DFSSGNX0 generates the same LTERM name for a user on each IMS system to which the user signs on.

Possible solutions:

- Enhance signon exit DFSSGNX0 to generate IMS-specific LTERM names.
- Enhance signon exit DFSSGNX0 to reject multiple signons to a single IMS.
- Force signon through one IMS system only.
- Use static terminal definitions only.
- Have a terminal-only front-end IMS system.
- Force each user to log on to only one IMS system.
- Create the LTERM name from the node.

If an application program has a dependency between terminals and related printers, the printer can be defined with an LTERM name that has a similar name to the related LTERM, but with a portion of the name identifying the LTERM as a printer. A user (`USER1`) signs on to IMS, and an LTERM is created with the name of `USER1`. If a transaction is entered for an application that has a dependency to a related printer, the LTERM name for the printer can be `USER1P`, with the `P` identifying the LTERM as a printer.

Duplicate LTERM names are discussed more generally in Chapter 14, “Duplicate LTERM Names and Output Security” on page 149. Refer to that chapter for a more detailed description of the restrictions and possible solutions for duplicate LTERM names.

9.5 Shared Printers and AUTOLOGON

If a printer is shared across IMS systems, the AUTOLOGON parameter could be used to automatically connect the printer to an IMS system when messages get inserted to the printer's LTERM on that IMS. The AUTOLOGON parameter is associated with a node name. The AUTOLOGON option can be used only in an ETO environment. Messages are inserted to an LTERM, and the LTERM name is associated to a node name. Multiple LTERMs are defined to the same node. This creates the environment for sharing the node.

When a message is inserted to the LTERM, the user structure is created by the output creation exit (DFSINSX0) if the structure does not already exist. IMS checks

whether the node is already allocated by VTAM to another system. If it is, IMS queues the user structure to a user waiting queue. As soon as the node becomes available, VTAM switches the node to the IMS system with a user waiting queue, and the user structure is allocated to the node. The queued message is delivered to the node. If the node was not active, IMS would create the user structure, allocate it to the node, and VTAM would automatically log on the node to IMS.

If the node is defined to many IMS systems in a shared-queues group, and messages were being inserted to LTERMs associated with the AUTOLOGON node, the node would be switching excessively from one IMS system to the other. In a shared-queues environment, each IMS still maintains a user waiting queue, but only for messages inserted locally. While the messages are sent to the same ultimate destination, the LTERM names are different, and the messages are queued to different LTERMs in the shared queue. This process negates the benefit of shared queues and creates unnecessary overhead.

If the same LTERM name is used for the AUTOLOGON node in different IMSs in the same shared-queues group, only one IMS system would have registered interest in the LTERM because the node can only be allocated to one system at a time. In this case, responses created by both IMS systems are queued to the same LTERM queue on the shared queue, and the IMS with a registered interest in the LTERM dequeues most of the messages. Switching is reduced, but the user structure has still been created on all IMS systems sending output to that destination, and the node is switched when the queue becomes empty.

Possible solutions are:

- Create a generic LTERM for all LTERMs that specify the same AUTOLOGON node parameters. In this way, all messages are delivered to the IMS system that has registered interest in the generic LTERM at the time the node was allocated to that IMS. Use the output creation exit to create the generic LTERM.
- Use the output creation exit to allocate the LTERMs to different AUTOLOGON nodes for each IMS system. The printer is not switched between IMS systems because the destinations are different printers.

Chapter 10. Conversational Programs

A *conversational program* is an MPP that processes transactions made up of several steps, each consisting of a connected series of terminal-to-program-to-terminal interactions. Conversational processing is used when one transaction contains several parts. It does not process the entire transaction at the same time. Information is stored between iterations in the SPA.

Processing conversational transactions across the shared-queues environment, where successive iterations of the conversation may be processed on separate IMSs, requires certain changes.

In this chapter we describe the changes in conversational processing and message flow for the shared-queues environment and explain the impact of the changes on both the local message queue pool and shared queue structures.

In this chapter, we use the following terminology:

front-end IMS. The IMS to which the terminal was connected when the initiating conversational transaction was entered.

back-end IMS. Any IMS, other than the front-end IMS, that is in the same shared-queues environment as the front-end IMS.

10.1 Scratch Pad Area

During each iteration of a conversational transaction, the MPP processing that iteration may need information gathered during previous iterations of the conversation. This information is stored in the SPA.

The SPA must be accessible to each IMS in the shared-queues group if conversational transactions are to be executed on any IMS system. You achieve accessibility by storing the SPA in the coupling facility structure. The first segment of the message on the shared queue contains the SPA. In previous releases of IMS, the SPA was stored in a pool in the local IMS called the SPA pool. The SPA pool is not used in IMS/ESA Version 6.

Conversations are defined at the transaction level in the IMS system definition. The TRANSACT macro uses the `SPA` parameter to specify the size of the SPA. If this parameter is present, the transaction is considered conversational. When a conversational transaction is entered at a terminal, IMS schedules the program associated with the transaction. During its first GU call, IMS passes the program a SPA of the size specified in the IMS system definition.

In a shared-queues environment, once the transaction is entered at the terminal, IMS determines that the transaction is conversational and creates a SPA of the size specified in the IMS system definition. The SPA is initialized to binary zeros and placed on the local message queue as the first segment. The input message is the second to *nth* segments. The message is then placed on the transaction ready queue.

When an IMS region is available to process the transaction, the MPP processes the transaction in the usual manner. (See 10.4.3, “Sample Conversational Processing

with the STRUNC and RTRUNC Options" on page 131 for a detailed explanation of conversational processing in a shared-queues environment). Once the response has been inserted to a destination, it is queued on the LTERM ready queue on the coupling facility structure. The IMS that has registered interest in the LTERM reads the response from the LTERM ready queue, causing the response to be written to the local queue buffer, and the message is moved to the lock queue on the coupling facility structure. IMS sends the response to the required destination. The response remains on the lock queue and in the local message queue buffer until the next input message has been received.

Care must be taken when sizing the shared-queue structure and the local message queue pool before implementing shared queues. The response message and SPA are kept on the shared queue and the queue buffer until the next iteration of the conversation. If the time between iterations is long, and the volume of conversational transactions is high, the message queue buffer pool and shared queue structure must be large enough to allow for the response messages.

SPAs are stored in the local message queue pool and on the CQS coupling facility structure. Because the SPA is placed in a message queue buffer, you must consider the size of the SPAs in your environment and the volume of conversational transactions in your system when determining the specification of long and short message sizes, to ensure optimal use of the local message queue pool buffers.

If the volume of conversational transactions in your environment is high, the structure size you choose must take into account the size of the SPA when determining the size of messages stored in your system in the following places:

- The message queue buffer pool of the local IMS system
- The shared queue structure
- The queue buffer pool of the IMSs processing the iterations of the conversation.

10.2 Terminal and User Affinities

Although a conversational transaction can process on any IMS system within a shared-queues environment, the conversational state of the initiating terminal is held only in the local IMS. The conversational state information is held in a CCB that is associated with either a node or a user, depending on whether the terminal is statically defined or created dynamically with ETO. Therefore, the conversational state of a terminal can only be maintained on the IMS system that entered the initial conversational transaction. If a conversation is in progress, and the user gets disconnected from the IMS system, the front-end IMS system still maintains the conversational state for that node.

If the user is disconnected before receiving the output message and after the input message is placed on the shared queue and logs on to a different IMS, that IMS attempts to send the conversational response to the terminal. The alternative IMS system does not know that the terminal was in a conversational state. The CCB does not exist for that LTERM, but the response message has a SPA in the first segment. IMS calls the conversational abnormal termination exit (DFSCONE0), which may, depending on how the exit is coded, delete the message. The user can then start another conversation on that IMS system. If the user logs on again

to the original IMS, the session hangs, because the conversational status is "waiting for output," and the output has already been deleted by the exit. The user should issue the /EXIT command to terminate the conversation.

The conversation in reality is complete because the user has received the message, but the original IMS system has no knowledge of this fact. You must be aware of that when moving to a shared-queues environment, as it differs from earlier releases of IMS.

If VTAM generic resources are being used, the node has an affinity to the original IMS. Because of this affinity, the node can connect only to that IMS, so conversational integrity is maintained.

10.3 Conversational Message Flow

In this section we describe the interactions involved in processing a conversation across a shared-queues environment and illustrate them in Figure 61 through Figure 66 on page 130.

A user logs on to IMSA, using a statically defined terminal, and issues TRANA, which is defined as conversational. TRANA is defined to both IMSA and IMSB. Both IMSA and IMSB have registered interest in TRANA and TRANB. IMSA registered interest in LTERMZ when the terminal logged on to IMSA.

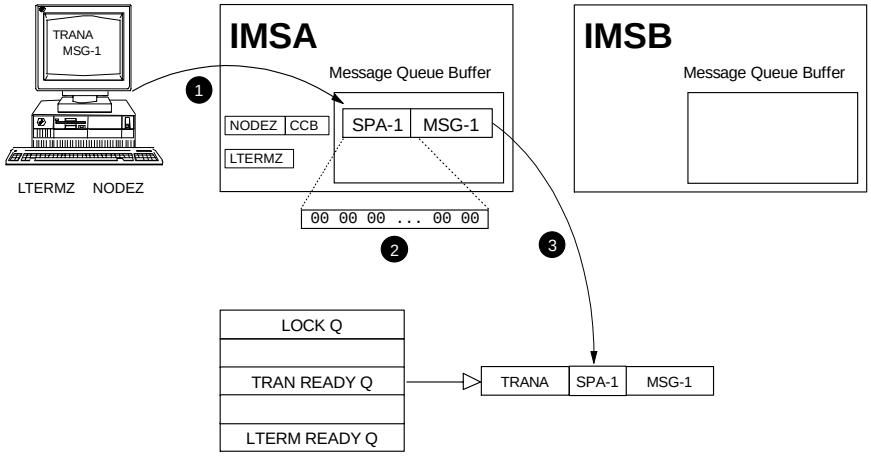


Figure 61. Conversational Message Flow (Part 1)

Here are the steps involved in a conversational message flow:

1. The user issues transaction TRANA MSG-1 from LTERMZ / NODEZ.

   TRANA is defined as conversational in IMSA.

   A CCB is created on IMSA to record the status of the conversation. The CCB is anchored off NODEZ because the terminal is statically defined.

2. IMSA creates a SPA (SPA-1) and initializes it to binary zeros. SPA-1 is inserted as the first segment of the input message in the message queue buffer pool. MSG-1 is the second segment of the message (and subsequent segments of the message if necessary).

3. IMS passes the message to the CQS, and the message SPA-1 MSG-1 is queued to the transaction ready queue.

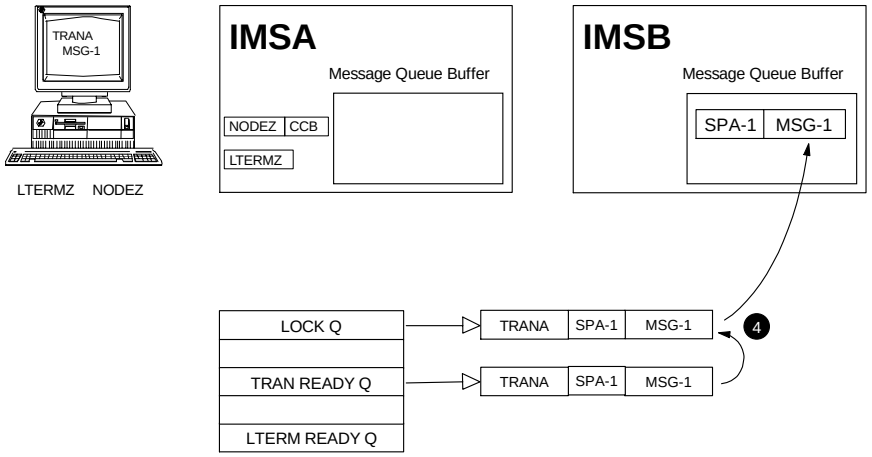


Figure 62. Conversational Message Flow (Part 2)

4. Since IMSB has registered interest in TRANA, it is notified that the queue has changed from empty to non-empty. Assuming IMSB has the resources available, CQS passes the message (SPA-1 MSG1) from the transaction ready queue into a message queue buffer in IMSBs message queue buffer pool. The input message, SPA-1 MSG-1, is moved from the transaction ready queue to the lock queue, to prevent other IMSs from trying to process this message.

   SPA-1 MSG-1 is processed by IMSB, and the program does a deferred message switch to TRANB, and inserts a transaction code of TRANB into the SPA.

   The updated SPA (SPA-2) and the response (MSG-2) are inserted onto the message queue buffer pool on IMSB.

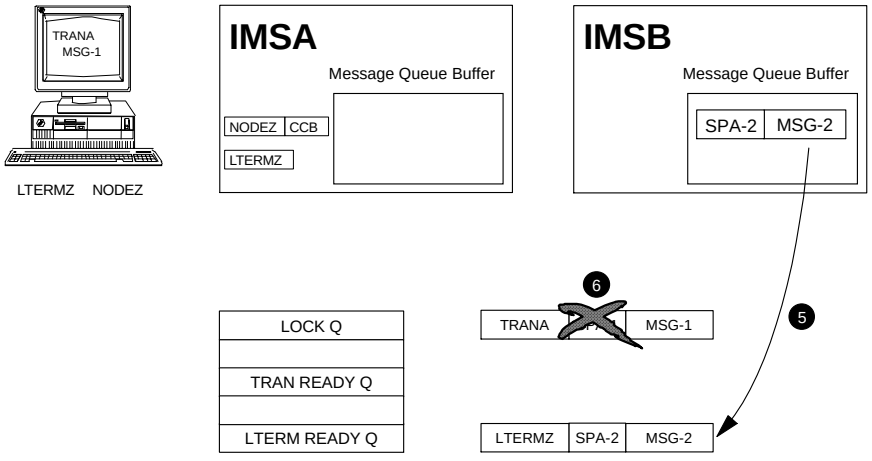


Figure 63. Conversational Message Flow (Part 3)

5. CQS moves the message to the LTERM ready queue as a multisegment message.
6. The previous message, SPA-1 MSG-1, is deleted from the lock queue. IMSA is notified that there is work on the LTERM ready queue, as it has a registered interest in this LTERM.

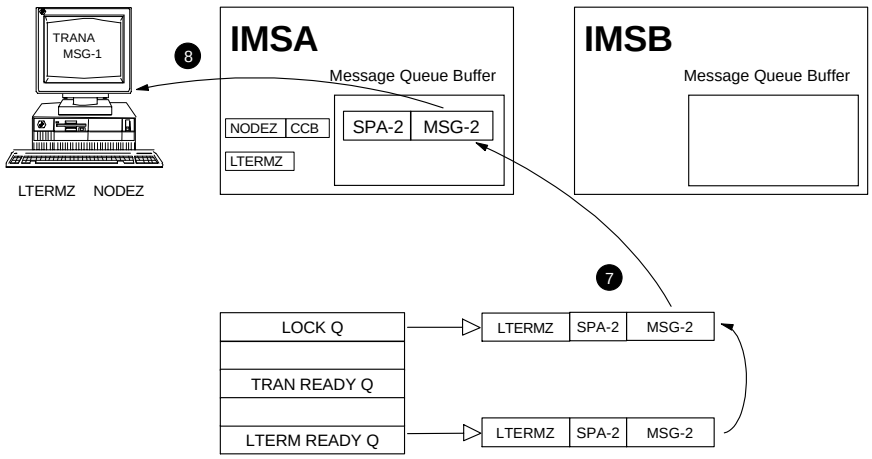


Figure 64. Conversational Message Flow (Part 4)

7. IMSA reads the message from the shared queue into the local message queue buffers, and the SPA and output message (`SPA-2 MSG-2`) are moved to the lock queue.

   IMSA stores the location of the message in the lock queue in its terminal control block.

8. The output message (`MSG-2`) is sent to `LTERMZ`.

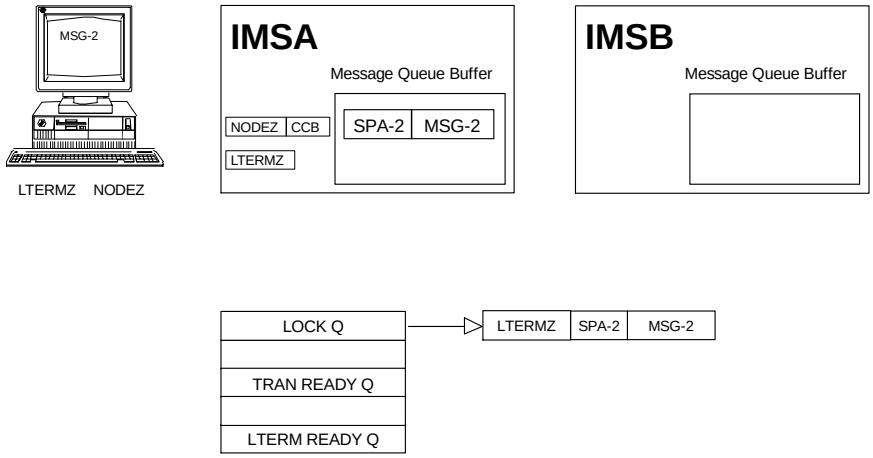


Figure 65. Conversational Message Flow (Part 5 of 6)

The SPA and output message (`SPA-2 MSG-2`) remain in the message queue buffer pool in IMS and on the lock queue in the coupling facility until the next iteration of the conversation.

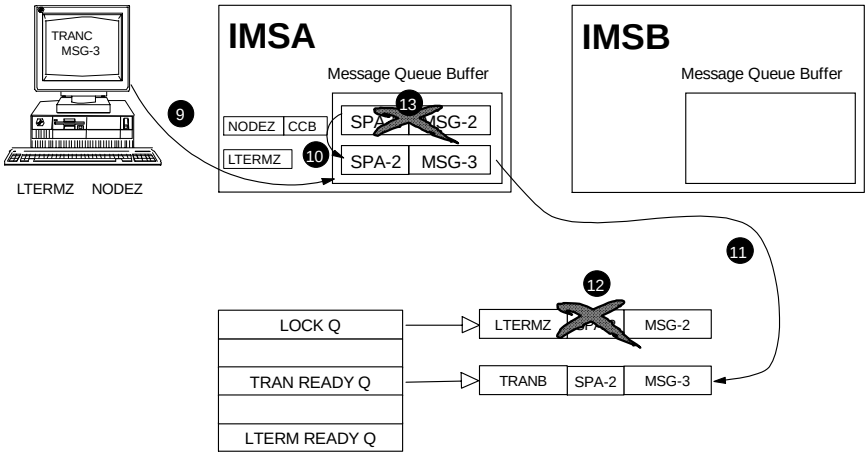


Figure 66. Conversational Message Flow (Part 6)

9. LTERMZ sends another iteration of the conversation to IMSA. The CCB is read to find the location of SPA-2 in the message queue buffer.
10. The SPA (SPA-2) is again inserted as the first segment of the message.
11. The SPA and new message (SPA-2 MSG-3) are placed on the transaction ready queue.
12. The output message (SPA-2 MSG-2) on the lock queue is deleted.
13. The IMS queue buffer containing the output message (SPA-2 MSG-2) is freed.

Processing continues from step 4 for subsequent iterations of this conversation.

10.4 Program Switches within Conversations and the Truncation of SPA

An application can switch a conversation to a new transaction, either immediately or deferred. An immediate message switch is performed by using an insert to the alternate PCB, and a deferred message switch is performed by using an insert to the IOPCB. The destination of the switch can be either a transaction in the local shared-queues environment or a remote transaction that is routed through MSC. The deferred switch is performed after the response has been sent, that is, the next input message sent is routed to the new program, and the new program is responsible for issuing the response message.

If a program switch is performed in IMS/ESA Version 6, and the transaction related to the switching program has a larger SPA than the transaction related to the switched-to program, the larger SPA is placed on the shared queue, but the SPA presented to the switched-to program is the size of the SPA defined to its related transaction. The SPA is not truncated in the message queue buffer or in the shared queue, but it is truncated, if necessary, when passed to the next program. The SPA inserted in the shared queue is made up of the new SPA, with the portion of the SPA that was truncated appended to it.

With IMS/ESA Version 6, an option exists to prevent the truncation of the SPA data. The next transaction in the conversational iteration receives a SPA of the size defined for it. The portion of the SPA that was truncated is restored onto the SPA returned from the switched-to transaction. 10.4.3, “Sample Conversational

Processing with the STRUNC and RTRUNC Options" on page 131 describes SPA truncation with the STRUNC and RTRUNC options.

10.4.1 Defining SPA Truncation Options

The system default is to save the truncated SPA across iterations of the conversation. You can override this default by using the TRUNC parameter in the DFSDCxxx proclib member, or by specifying the truncation options on individual TRANSACT macros. The format is:

```
TRANSACT TRANCode, SPA=(size,STRUNC|RTRUNC)
```

When a conversation is initiated, and on each program switch, the truncated data option is checked. When the truncated data option has been set, it remains set for the life of the conversation, or until the truncation option is reset by a program switch. For more information about setting the default for the truncated data option, see 5.9.1.2, "Specifying the IMS Data Communications Parameters (DFSDCxxx)" on page 76, and *IMS/ESA Installation Volume 2: System Definition and Tailoring*, GC26-8737.

10.4.2 Multiple Systems Coupling

The SPA require special consideration if any of the iterations of a conversation are processed remotely (through MSC). The SPA is passed to the remote system along with the message. If the receiving IMS system is not IMS/ESA Version 6, it may receive more data than it expected. In these circumstances, you should specify the RTRUNC option.

10.4.3 Sample Conversational Processing with the STRUNC and RTRUNC Options

Below we explain the difference in the handling of the SPA for a transaction defined with RTRUNC and a transaction using STRUNC (either explicitly or as the default). Use Figure 67 on page 132 and Figure 68 on page 133 as a reference in the explanation following.

10.4.3.1 STRUNC Option

Figure 67 on page 132 illustrates the use of the SPA when the STRUNC option has been specified:

1. PROGRAMA receives a SPA of 100 bytes that has been initialized to binary zeros. At the end of processing, PROGRAMA inserts a SPA of 100 bytes to the message queue with a transaction code of TranB.
2. TranB has been defined with a SPA size of 60. Because STRUNC has been defaulted to or specified on the TRANSACT macro for TranB, the entire SPA of 100 bytes is placed on the shared queue. When PROGRAMB issues the first GU, the SPA received is the first 60 bytes of the SPA on the shared queue.
3. On completion of PROGRAMB, the saved SPA of 40 bytes is appended to the 60-byte SPA returned by PROGRAMB. PROGRAMB inserts the SPA with a transaction code of TRANC. TRANC is defined with a SPA of 150 bytes.
4. The SPA presented to PROGRAMC at GU consists of 150 bytes:
   - The first 60 bytes are the SPA inserted by PROGRAMB.
   - The next 40 bytes are the remaining 40 bytes of the SPA inserted by PROGRAMA.

- The last 50 bytes are initialized to binary zeros.

5. After ProgramC has processed the transaction, the SPA stored is 150 bytes.

If the STRUNC option is being used, the size of the SPA kept in the shared queue is the size of the largest SPA of all the transactions that are processed in a single conversation. It is important to remember this when sizing the local message queue buffer pool, as well as the shared queue coupling facility structures.

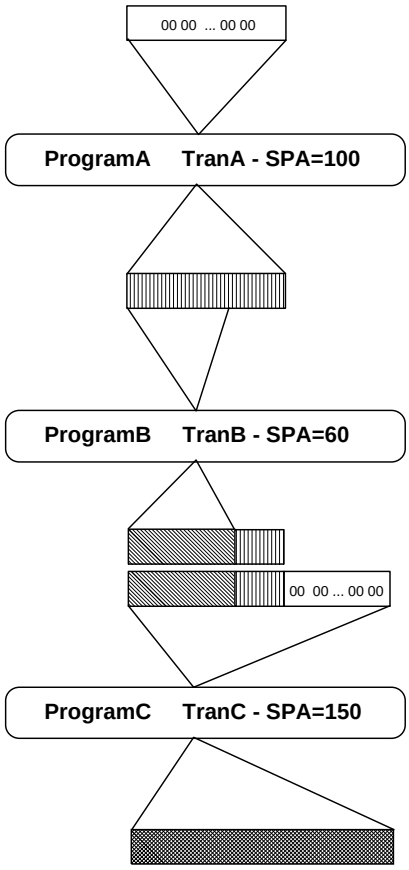


Figure 67. Truncation of SPA with the STRUNC Option

10.4.3.2 RTRUNC Option

Figure 68 on page 133 illustrates the use of the SPA when the RTRUNC option has been specified.

1. PROGRAMA receives a SPA of 100 bytes of binary zeros.
2. When the SPA is inserted back to the message queue with a transaction code of TranB and the RTRUNC option is specified, the last 40 bytes of the SPA are discarded. Only the first 60 bytes are placed on the shared queue. PROGRAMB receives a SPA of 60 bytes with the GU.
3. ProgramB places the updated SPA onto the message queue with a transaction code of TRANC. Because TRANC has a SPA of 150 bytes, the last 90 bytes are initialized to binary zeros. The SPA passed to PROGRAMC has 60 bytes of information inserted by PROGRAMB and 90 bytes of binary zeros.
4. The SPA stored by ProgramC is 150 bytes.

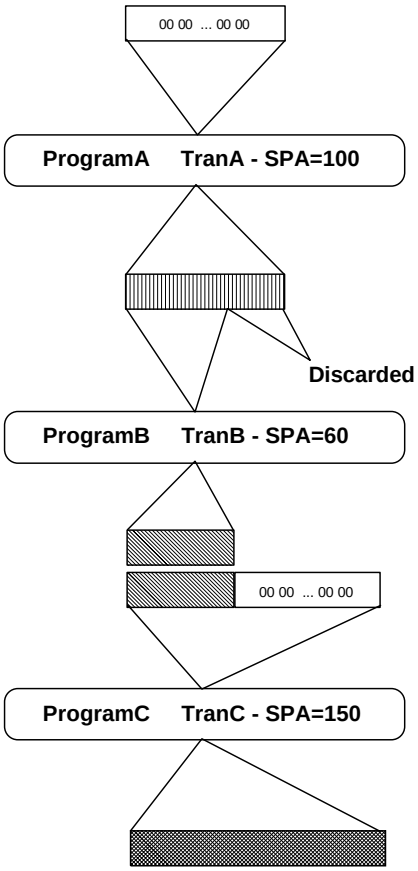


Figure 68. Truncation of SPA with the RTRUNC Option

10.5 Terminating a Conversation

A conversation can be terminated by an application program or by the user. An application ends a conversation by inserting a SPA with a blank transaction code or by issuing a deferred transaction switch to a transaction that is not defined as conversational. When the SPA is inserted as part of the response, the front-end IMS processes the response by deleting the CCB, sending the response to the terminal and deleting the output queued on the lock queue and in the local queue buffer pool.

The user can exit a conversation by entering the `/EXIT` command while in conversation. This command is accepted only when entered on a terminal logged on to the local IMS (the IMS system on which the conversation was initiated). The termination processing that follows depends on whether the SPA is available to the front-end IMS or not. The SPA may not be available to the front-end IMS if the terminal has entered the next input but not yet received a response. The SPA is unavailable to the front-end IMS if the SPA and message are either:

- On the shared queue waiting to be processed
- Executing on another IMS system in the shared-queues environment

  or
- Executing on a remote IMS system connected through MSC

The SPA is available to the front-end IMS if the response has been sent to the terminal, but the next input message has not been entered.

If the SPA is available to the front-end IMS, IMS accesses the SPA through the CCB and calls the conversational abnormal termination exit (DFSCONE0), passing the SPA as a parameter to the exit. The default exit deletes the CCB, removes the response from the lock queue, and releases the local message queue buffer containing the SPA.

If the SPA is not available to IMS, IMS still calls DFSCONE0, but with a dummy SPA. The default exit terminates the conversation by deleting the CCB. The following message is sent to the terminal:

```
DFS577I EXIT COMPLETED. TRANSACTION STILL ACTIVE.
```

The transaction is still processed. When the transaction has finished processing, the response is inserted onto the shared queue. The local IMS attempts to process the response, but the CCB does not exist. IMS once again drives the conversational abnormal termination exit, DFSCONE0, and deletes the response from the lock queue and the local queue buffer pool. The transaction has been processed, but the output was deleted and never sent to the terminal.

The DFSCONE0 can be customized to perform a different sequence of events from those described. For more information about this exit, see Appendix C, “IMS Exits” on page 169, or the *IMS/ESA Customization Guide*, SC26-8732.

10.6 Multiple Systems Coupling

Conversational transactions that are defined as remote (MSC) in the shared-queues environment are also placed on the shared queue and routed to the remote IMS for processing. Care should be taken to ensure that the SPA passed with a message is not larger than the receiving system is expecting.

Chapter 11. Multiple Systems Coupling

IMS/ESA Version 6 enhances the MSC facilities available for sharing messages across many different IMS systems in a Parallel Sysplex. In this chapter we describe the changes to MSC when shared queues are used.

Before we begin, let us review some terminology. A *shared-queues group* is the name given to a number of IMS systems that share a common message queue structure located in the coupling facility. The IMS systems are called *front-end* or *back-end* systems. The front-end system is the IMS system that receives the input from the terminal. The back-end system is the system that processes the input. If a message stays within one IMS system in a shared-queues group, that IMS system is both the front-end and back-end system.

A message is said to be processed in a *local system* when the message stays within that system. All IMSs within the same shared-queues group are considered to be local to each other, in terms of MSC, because the message does not get passed outside the group through an MSC link. The *remote system* is the IMS system outside the shared-queues group, linked through MSC, that processes the message. An *intermediate system* is a system through which the message is sent to reach the final remote system.

11.1 Message Routing in a Shared-Queues Environment

Messages between IMS systems that are in the same shared-queues group are sent through shared queues. MSC links can be defined between systems in the same shared-queues group but cannot be started. MSC link definitions between IMS systems in the same shared-queues group would be for either fallback purposes, as a first step of migrating to shared queues, or for isolating where a transaction can run. The IMS systems must be shut down and coldstarted in non-shared-queues mode in order to start the links.

11.1.1 System Identifier Table

Each IMS system within a shared-queues group maintains a table that contains a list of the system identifiers (sysids) the shared-queues group knows about. Each sysid that is known to the shared-queues group has a value that is either local or remote. Local indicates that the message can be handled by an IMS system within the shared-queues group. Remote indicates that the message needs to be sent across an MSC link to another IMS system. If the sysid has a value of local, the message is put on either the transaction ready queue or the LTERM ready queue. If the sysid has a value that is remote, the message is either put on the transaction ready queue or queued to the appropriate MSNAME on the remote ready queue. For details on the queue on which a message is placed, see 11.1.3, “MSC Processing Scenarios” on page 136.

The sysids associated with each IMS system are in the MSNAME macros in the IMS system definition. If `MSNAME SYSID=(4,3)` is in the IMS system definition, the local sysid is 3 and the remote sysid is 4. When an IMS system joins a shared-queues group, it sends a list of its local sysids and a list of remote sysids to which it is connected. All of the IMSs that are currently in the shared-queues group respond with a list of their local sysids and a list of remote sysids to which they are

connected. This ensures that all IMSs in the shared-queues group have the same sysid table.

11.1.2 Registering Interest in MSC Links

Each IMS system within the shared-queues group that has an active MSC link needs to be notified when there are messages that need to be sent across the MSC link to a remote system. This notification is achieved by the IMS system registering an interest in the MSNAME that is associated with the MSC link. In a shared-queues environment, message responses that are being sent to a remote MSC system are queued to the MSNAME on the remote ready queue. When this IMS system is playing the role of the intermediate system, both input and output messages are placed in the remote ready queue on the MSNAME queue. An MSNAME queue is anchored from the remote ready queue.

When an MSNAME queue goes from empty to nonempty, the IMS system that has a registered interest in that queue is notified by its CQS that there is work to be processed. Only one IMS system can have a registered interest in any one MSNAME queue.

When the MSC link becomes active, IMS registers interest in the MSNAME on the remote ready queue. IMS also registers interest in all transactions that are defined with a remote destination of the sysid for that MSNAME. IMS deregisters interest in the MSNAME and the transactions that are defined with a remote destination of the sysid for that MSNAME when the MSC link becomes inactive.

11.1.3 MSC Processing Scenarios

In this section we cover several processing scenarios in an MSC configuration. We explain the role the sysid table plays in determining on which queue type the message is placed.

Figure 69 on page 137 shows three IMS systems in a shared-queues group that are connected by MSC links to two stand-alone IMS systems and five transaction definitions and the MSNAME macros that define the MSC links.

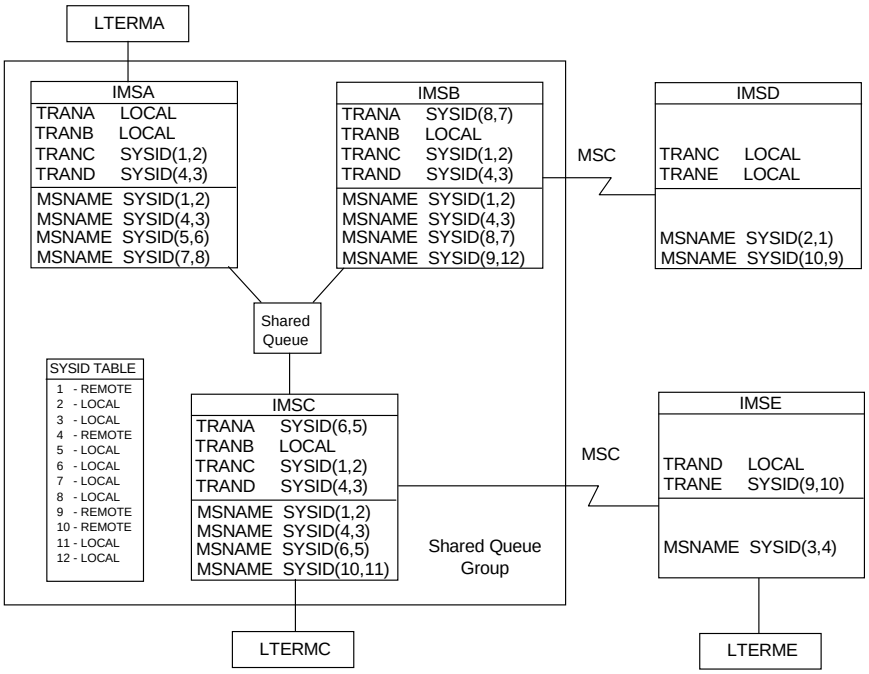


Figure 69. Message Routing in an MSC Shared-Queues Group

The sysid table in the shared-queues group lists which sysids are known to the shared-queues group and whether they are local or remote to the shared-queues group. This scenario demonstrates how you could move from three IMS systems that were previously connected by MSC links to a shared-queues environment with little change to the IMS system definitions.

When MSC links are used to connect the IMS systems in the shared-queues group, any TRANA transactions received by IMSB and IMSC are sent to IMSA through the MSC links. When they are in a shared-queues group, any TRANA transactions are put on the the transaction ready queue. Because only IMSA has registered an interest in TRANA, these transactions are processed by IMSA. The IMS system definition has not been cloned for the IMS systems in the shared-queues group, because TRANA is defined as local in IMSA and remote in the other IMS systems within the shared-queues group.

All messages that need to be passed among IMS systems in the shared-queues group are passed by the front-end IMS system putting the message on the shared queue and the back-end IMS system reading it off the shared queue. This process removes the requirement for MSC links between IMS systems that are in a shared-queues group. Using the configuration in Figure 69, we can map the flow of message routing using various connection paths:

- **Local transaction in an MSC front-end system**

  If LTERMA enters a message for TRANA, it is placed on the transaction ready queue for TRANA. Because IMSA has a registered interest in this queue, the transaction can be processed in the IMSA front-end system. IMSA is both the front-end and the back-end system.

- **Local transaction in an MSC front-end and back-end system**

If LTERMA enters a message for TRANB, IMSA places the message on the transaction ready queue for TRANB, where it can be processed by either IMSA, IMSB, or IMSC.

- **Remote transaction in an MSC shared queues front-end system sent out through a back-end system to a remote non-shared-queues system**

  If LTERMA enters a message for TRAND, IMSA places the message on the transaction ready queue. IMSC registered an interest in TRAND when the MSC link between IMSC and IMSE became active. IMSC is notified that the TRAND queue has gone from empty to nonempty. IMSC reads the message off the transaction ready queue and sends it over to IMSE through the MSC link. IMSE processes the message and sends the response back to IMSC. IMSC places the response on the LTERM ready queue. IMSA is notified that the LTERMA queue has gone from empty to nonempty. IMSA reads the response of the LTERM ready queue and sends it to LTERMA.

- **Local transaction in a back-end system**

  If LTERMC enters a message for TRANA, it is placed on the transaction ready queue for TRANA. The TRANSACT macro for TRANA implies that the message is going to be sent across an MSC link, but that is not the case because the remote sysid is within the same shared-queues group. IMSA reads the message off the transaction ready queue and processes the message. IMSA then puts the response on the LTERM ready queue. IMSC reads the response off the LTERM ready queue and sends it to LTERMC.

- **Remote transaction from a non-shared-queues IMS, through a shared queues IMS intermediate system, to a remote non-shared-queues IMS**

  If LTERME enters a message for TRANE, IMSE determines that it is a transaction that needs to be run on a remote system. It sends it across the MSC link to IMSC. IMSC looks at the sysid table and determines that sysid 9 is a remote sysid. IMSC then queues the message on the remote ready queue for MSNAME 9. IMSB is the only IMS system that has a registered interest in MSNAME 9. IMSB reads the message off the remote ready queue and sends it across the MSC link to IMSD. IMSD has TRANE defined as local so it processes the message and sends the response back across the MSC link to IMSB. IMSB looks at the sysid table and determines that sysid 10 is a remote sysid. IMSB then queues the message on the remote ready queue for MSNAME 10. IMSC is the only system that has a registered interest in MSNAME 10. IMSC reads the message off the remote ready queue and sends it across the MSC link to IMSE. IMSE then sends the response back to LTERME. Note that if an IMS in the shared-queues group is an intermediate system, both the input and the output messages are placed on the remote ready queue.

To support MSC in a shared-queues environment, the MSC feature must be in the IMS system definition for all IMSs in the shared-queues group. There is no check to enforce this at IMS initialization, or when new IMSs join the shared-queues group. It must be set up beforehand. To include MSC in your IMS system definition, define at least one MSC link (that is, MSPLINK, MSLINK, and MSNAME macros). The link does not necessarily have to be started. Also, if the partner IMS is in the same shared-queues group, link restart fails for the partner with this message:

```
DFS2149I PARTNER IN SAME SHARED QUEUES GROUP - RESTART ABORTED
```

Figure 69 on page 137 is an example of IMS systems in the same shared-queues group that have been defined with MSC links among them. IMSA has been defined with a link between both IMSB and IMSC. The link cannot be started but is required in order for the IMS system definition to work. Having TRANA defined with a `SYSID(6,5)` in IMSC requires that there be an MSC definition with the same sysids on the MSNAME macro, in order for the IMS system definition to complete successfully.

As IMS systems join the shared-queues group, IMS performs an MSC internal table rebuild to reflect a common view of the sysid table for the shared-queues group. If an IMS system that is in a shared-queues group shuts down, a bit is set in the table to indicate that is currently not available.

11.2 /MSVERIFY Support

Externally, there is no change to /MSVERIFY support in a shared-queues environment. Any IMS in a shared-queues group connected to a remote IMS can issue or receive an /MSVERIFY request to or from the remote IMS. However, IMSs within the shared-queues group that do not have active MSC link connections cannot send or receive /MSVERIFY requests.

11.3 Directed Routing

Directed routing is an MSC function whose use is restricted in a shared-queues environment. MSC directed routing enables an application program to specify the system name (MSNAME) and destination within that system for a message to an LTERM or an application program. With directed routing, the remote destination, an LTERM or an application program, does not have to be defined in the local system. When an application program requests a directed routing call, the message is queued to the MSNAME that it specifies. MSC then sends the message to the remote system.

Within a shared-queues environment, directed routing does not work if you are sending the message to an IMS system that is in the same shared-queues group because the message that the application program sends would be queued on the remote ready queue to an MSNAME in which no IMS system has registered interest. An IMS system registers interest in an MSNAME only if it has an MSC link active. An MSC link cannot be started between IMS systems that are in the same shared-queues group. The application program that performed the directed routing call is returned a status code that indicates the request has been successful but the message will never be delivered to the requested system.

Directed routing still works among any IMS systems connected through an active MSC link such as:

- Non-shared-queues IMS systems
- An IMS system in a shared-queues group and an IMS system in a different shared-queues group
- An IMS system in a shared-queues group and a non-shared-queues IMS system

Application changes need to be made if directed routing is being used among IMS systems that you plan to move into an IMS shared-queues environment.

11.4 MSC Program Routing Exit (DFSCMPR0)

Use of the MSC program routing exit is restricted in a shared-queues environment. The MSC program routing exit can change the destination of a message destined for either an LTERM or a program. The remote destination, an LTERM or a program, does not have to be defined in the local system. The message is queued to the MSNAME that it specifies. MSC then sends the message to the remote system.

Within a shared-queues environment, the MSC program routing exit does not work if you are sending the message to an IMS system that is in the same shared-queues group because the message would be queued on the remote ready queue to an MSNAME in which no IMS system has registered interest. An IMS system only registers interest in an MSNAME if it has an MSC link active. An MSC link cannot be started among IMS systems that are in the same shared-queues group.

The MSC program routing exit still works among any IMS systems connected through an active MSC link such as:

- Non-shared-queues IMS systems
- An IMS system in a shared-queues group and an IMS system in a different shared-queues group
- An IMS system in a shared-queues group and a non-shared-queues IMS system

Application changes need to be made if the MSC program routing exit is being used among IMS systems that you plan to move into an IMS shared-queues environment.

Appendix C, “IMS Exits” on page 169 describes the changes that may need to be made to the MSC exits.

Chapter 12. Significant Status and Affinities

A significant status indicates that a terminal is active in a processing cycle. IMS maintains the status of the terminal in the terminal control blocks. Thus IMS can restore the status of the terminal if the terminal is disconnected either by logoff or session termination. The terminal status is restored when the terminal reconnects to that IMS system. In a shared-queues environment, where a terminal can connect to any IMS system in the same shared-queues group, significant status can become an issue.

Significant status indicates that the processing performed at the terminal is not logically complete and that the terminal is still considered to be active in terms of a processing cycle. Significant status is local to an IMS system, because the status of a terminal is kept in the terminal control blocks in that IMS system. Significant status cannot be transferred with logon to different IMS systems in a shared-queues group.

A terminal is considered to have a significant status in the local IMS system when it is in:

- Response mode
- Conversational mode
- Exclusive mode (`/EXCLUSIVE`)
- Test mode (`/TEST`)
- Test MFS mode (`/TEST MFS`)

or has a preset destination.

If a terminal has a significant status in an IMS system, it is said to have an affinity for that IMS system. An affinity is created the first time an LU connects to an IMS system and is usually deleted at session termination. Affinities can remain after session termination, if the terminal has a significant status in the IMS to which it was connected. Affinities operate differently for static and dynamic terminals.

In this chapter we cover the issues that need to be taken into account before implementing a shared-queues environment.

12.1 Static and Dynamic Terminal Affinities

Static terminal affinities are always associated with a node. When a user logs on to an IMS system from a node that has an affinity to it, the terminal is restored to the same processing mode or cycle that was active the last time that node was in session with that IMS system.

If multiple users have access to the same node, no output security exists and there may be inconsistencies in processing. Consider the situation where `USERA` logs on to IMSA, issues `/TEST MFS`, and begins to enter transactions. If the terminal is disconnected from IMSA, and `USERA` moves away from the terminal, `USERB` may log on to IMSA from the same terminal, and not only possibly get the responses from `USERA`'s processing, but also have the significant status restored to `TEST MFS` mode.

In this situation the processing of the transactions looks different and may even act differently.

Dynamic terminals have a significant status associated with a node for the time that the terminal is connected to IMS. At signoff, the significant status is moved from the node control block to the user control block. When the user next logs on to the same IMS system, the user control block is restored to the terminal control block of the terminal to which the user has logged on. The user is then in the same processing mode or cycle as the previous session.

12.2 VTAM Generic Resources

VTAM generic resources are similar to shared queues in that the former distribute sessions, and the latter balance workload across IMS systems that are in a shared processing environment. Shared processing means that the IMS environments are in an MVS sysplex, the IMS systems are engaged in data sharing, and all IMS systems are capable of processing the entire IMS workload. VTAM generic resources provide the ability to distribute the IMS data communication sessions across IMS systems while providing some mechanisms to protect terminal affinities and terminals with a significant status in IMS. Shared queues provide the ability to share processing workloads across numerous IMS systems in the same shared-queues group. Together, VTAM generic resources and shared queues provide workload management across IMS systems in the same VTAM generic resources and shared-queues group.

VTAM generic resources are not discussed in detail. We discuss only those issues relating to processing in a shared-queues environment.

12.2.1 System Affinities Using VTAM Generic Resources

An affinity is created the first time a LU connects to an IMS system using a generic VTAM ACB for IMS. These affinities are usually deleted at session termination. Affinities can remain after session termination, if the terminal has a significant status in the IMS to which it was connected. Significant status indicates that whatever processing was being performed at the terminal is not logically complete, and that the terminal is still considered to be active in terms of a processing cycle. Significant status is local to an IMS system, since the status of a terminal is kept in the terminal control blocks in that IMS system.

If a significant status exists on an IMS system for a terminal, that terminal must log on to the same IMS system in order to complete the processing that resulted in the significant status; that is, the terminal has an affinity to an IMS system if that IMS system has a significant status for that terminal.

12.2.2 Static Terminals Using VTAM Generic Resources

VTAM generic resources create an affinity for a statically defined terminal at initial logon to the IMS system. If the terminal is disconnected, and the user logs on to that terminal again using the generic IMS application control block (ACB), the terminal again connects to that IMS. If VTAM generic resources are used, VTAM creates an affinity for a terminal to a specific IMS system, if the terminal has a significant status in that IMS system. Even if that IMS system has failed, VTAM generic resources do not allow the terminal to connect to any other IMS system in

the VTAM generic resources group, and the user cannot use the terminal to access any IMS in the VTAM generic resources until the IMS system is restarted. If users log on to IMS directly without using the generic IMS ACB, they can still process transactions, but this will violate the system affinities.

If it is necessary to ensure that the terminal can operate at all times whether or not affinity with an IMS exists, the signoff exit routine (DFSSGFX0) or the logoff exit routine (DFSLGFX0) can be customized to reset the status of the terminal.

12.2.3 Dynamic Terminals using VTAM Generic Resources

For an ETO terminal, significant status is moved from the terminal control block to the user control block at signoff time, and the terminal is not considered to have an affinity to an IMS system. The user is said to have an affinity. Because VTAM generic resources maintain terminal affinities only, the terminal's affinity to the IMS system is removed. The terminal can therefore establish a session to any IMS system in the VTAM generic resources group, and not necessarily the IMS system to which it was previously connected. The affinity associated with the user remains on the original IMS until the user logs on to that IMS later, which can potentially cause confusion.

12.3 Deleting Significant Status and Affinity

It is possible to force affinity deletion by using the signoff exit (DFSSGFX0) or the logoff exit (DFSLGFX0). These exits apply to both terminal and user affinities.

If you are using VTAM generic resources, you can delete affinities when you shut down IMS, using the `LEAVEGR` option on the `/CHECKPOINT SHUTDOWN` command or by a coldstart of IMS.

12.4 In Summary

You must consider the relevance of significant status in your environment and decide whether logons to the same IMS system should be forced or not. If users are aware of the IMS ACB name or can log on directly to specific IMS systems, controlling the problem is more difficult. Users must be restricted to logging on to only one IMS system, and they have to accept that if that IMS system in the shared-queues group is not active, no service is available to them. If this solution is not possible in your environment, it may be necessary to customize the logon and logoff or signon and signoff exits, to ensure that significant status is deleted.

For more information about:

- Signon and signoff exits, refer to Appendix C, “IMS Exits” on page 169.
- Customization of the exits, refer to the *IMS/ESA Customization Guide*, SC26-8732
- Significant status specific to conversational processing, refer to Chapter 10, “Conversational Programs” on page 125

Chapter 13. Undefined Resources

Undefined resources may exist in a shared-queues environment because the IMS system definition is not cloned for all IMS systems in the shared-queues group. A transaction may be defined to one IMS system and not to another IMS system in the shared-queues group. Before shared queries, if a transaction was entered from a terminal that was logged on to an IMS system where the transaction was not defined, the message would be rejected. With shared queues it is possible to use an exit to determine whether to dynamically allocate that resource to enable the message to be placed on the shared queue and be processed by one of the IMS systems in the shared-queues group to which it is defined.

In this chapter we discuss undefined resources and the output creation exit (DFSINSX0) to explain those situations where the exit could be useful.

13.1 Valid Destinations

When a message is placed in the local IMS message queue buffer pool, it is first verified to determine whether the destination exists. That destination could be a transaction, a terminal, or an MSNAME. In a shared-queues environment, it may not be possible to clone the IMS system definition, and not all resources are defined in the all IMS systems.

The destination may be valid in the shared-queues group because it is defined somewhere in the shared-queues group, but an IMS system cannot determine whether a resource is defined on another IMS system in the same group, and thus cannot determine that the destination is valid. If the destination is valid within the shared-queues group, and the message is placed on the shared queue, it becomes available to all IMS systems in the shared-queues group, and one of the IMS systems can process it.

If the destination of the message cannot be verified using the IMS system definition in a shared-queues environment, the output creation exit can be called to verify the destination of the message.

13.2 Output Creation Exit

The output creation exit (DFSINSX0) is called at find-destination time if the destination cannot be validated on the local IMS system in a shared-queues environment. In releases before IMS/ESA Version 6, DFSINSX0 was called only if `ETO=Y` was defined, to validate terminals. Now, with IMS/ESA Version 6 and `SHAREDQ` equal to a value, DFSINSX0 is always called if it is included in the IMS system.

Figure 70 on page 146 shows the processing that occurs with undefined resources and the DFSINSX0.

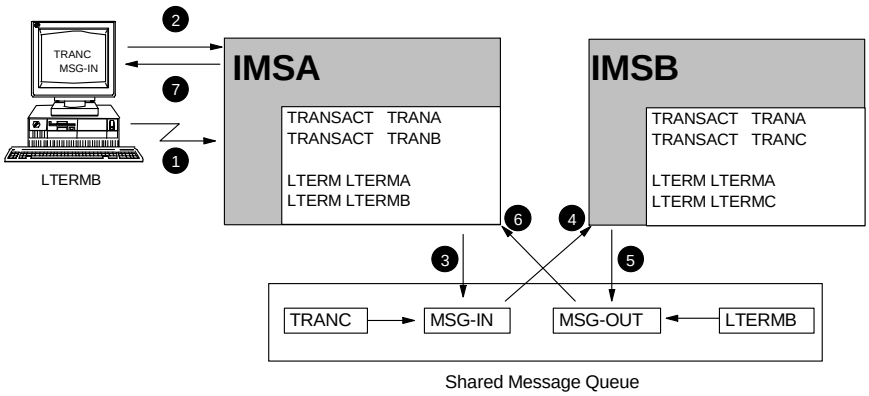


Figure 70. Processing with Undefined Resources

IMSA registers interest in TRANA and TRANB at startup. IMSB registers interest in TRANA and TRANC at startup.

1. LTERMB logs on to IMSA.

   IMSA registers interest in LTERMB.

2. LTERMB issues TRANC MSG-IN.

   IMSA runs the FINDDEST routine to validate the destination of the message. IMSA does not have TRANC defined. DFSINSX0 is driven to further validate the transaction. If DFSINSX0 validates the transaction code, a dynamic transaction control block is created on the local IMS, and normal transaction processing resumes. The transaction control block is used only for message delivery, not message processing.

3. IMSA places the message on the shared queue.

4. IMSB is notified of the message and removes it from the queue. IMSB processes the message and performs an insert to IOPCB (LTERMB) with output message MSG-OUT. The IOPCB destination is considered valid.

5. MSG-OUT is placed on the shared queue for LTERMB.

6. IMSA is notified of the message for LTERMB and removes MSG-OUT from the queue.

7. IMSA sends MSG-OUT to LTERMB.

An insert to an undefined IOPCB destination is considered valid, whereas an insert to an undefined ALT=PCB is considered invalid. If ETO is active, DFSINSX0 is called to validate the destination. If the destination validated, the dynamic terminal structure is created by IMS.

13.3 In Summary

If IMS system definitions are not cloned in a shared-queues environment and it is required that messages be accepted as input from IMS systems that do not have the transaction defined, the transaction definitions can be dynamically created. Thus messages can be placed on the shared queue and be processed by an IMS system with the transaction defined. The same is possible for LTERMs.

To reduce the number of exits in your IMS environment and simplify maintenance of the environment, we recommend that you clone the IMS system definitions.

For more information about DFSINSX0, refer to the *IMS/ESA Customization Guide*, SC26-8732.

Chapter 14. Duplicate LTERM Names and Output Security

Output security is a term that describes the situation where responses are sent to the user or node that originated the response. If a response is received on a node, or user, and the user is not supposed to receive the output, output security is not available. Duplicate LTERM names in a shared-queues environment can result in an environment having no output security. A node may receive messages destined for an LTERM, where the message was created by another node with the same LTERM name in the same shared-queues group.

When two nodes log on to IMS systems in the shared-queues group and have the same LTERM name, two IMS systems in the same shared-queues group register interest in the same LTERM name queue. Any messages on the LTERM queue can be sent to either node, regardless of which node initiated the response.

If terminals are defined statically, and the IMS system definitions are cloned for each IMS system in the shared-queues group, then no problems exist. There is a many-to-one relationship between LTERMs and nodes. Because only one IMS session can be established for each node at one time, no two IMS systems in a shared-queues group can have registered interest in the same LTERM name.

In a single IMS system, checking for duplicate LTERMS is performed when the IMS system definition is run. In a shared-queues environment, IMS has no way of coordinating the checking of duplicate LTERM names across the IMS systems in a shared-queues group. If IMS system definitions are not cloned for all IMS systems in the shared-queues group, it is important to remove or change any definitions where the same LTERM name has a different node name in two separate IMS system definitions.

In an ETO environment, the problem is more complex, depending on the naming standards implemented for LTERMs. Refer to Chapter 9, “Extended Terminal Option” on page 121 for more specific information about ETO terminals in a shared-queues environment.

A simple and safe solution is to create one front-end IMS with all terminal definitions for the entire shared-queues group. All logons to the shared-queues group are through the front-end IMS. Only one IMS system registers interest in the LTERMs, and only one IMS system is notified of the existence of output messages. The IMS system definition checks for duplicate LTERMs when the generation is performed, and no duplicate LTERMs exist in the shared-queues group. You can reduce the single point of failure by using XRF as backup to the front-end IMS system, or VTAM generic resources to enable unnoticeable switching to another IMS system with the same terminal definitions in the event of a front-end IMS failure. VTAM generic resources can be used to balance the number of sessions across the VTAM generic resource group.

It must be stressed, however, that this solution does not exploit all of the advantages of a shared-queues environment. The environment cannot exploit local-only processing, and the terminal load is not balanced. The continuous availability advantage of shared queues is also not achieved.

Customers with an ETO environment are familiar with node name User Descriptor. When this descriptor is used (or when the signon user exit DFSSGNX0 output

parameters are used), IMS replaces the user and LTERM names with the node name. The construct node name User Descriptor was introduced to provide a similar environment to static terminals.

Chapter 15. Serial Transactions

In this chapter we describe the processing of serial transactions, transactions that have SERIAL=YES coded on the TRANSACT macro in the IMS system definition. Serial transactions are processed differently from other transactions in a shared-queues environment, as we explain below.

15.1 Shared Queue Types Used

The shared queue types used during processing of serial transactions are:

- Transaction serial queue
- Transaction staging queue
- LTERM ready queue
- LTERM staging queue

IMS does not register interest in either of the staging queues. The relevant staging queues are used only during serial transaction processing if the input or output messages span a message queue buffer. The staging queues are used to hold the second through nth message queue buffers of the message.

A serial transaction message is entered at a terminal or switched to during a program-to-program switch. IMS determines from the IMS system definition that the transaction is serial. The message is placed on the transaction serial queue with a queue name of the transaction code and the IMSID of the local IMS system. The transaction is serialized within the IMS system, but not within the shared-queues environment. The message is placed on the shared queue to provide recoverability of the message. A serial transaction is processed in the same order and fashion as serial transactions in a non-shared-queues environment.

15.2 How IMS Registers Interest in Serial Transactions

When an IMS system in a shared-queues environment is started, IMS registers interest in all transactions that are statically defined. If a transaction is defined as serial, IMS registers interest in the queue name of the transaction code concatenated with the IMSID. If TRANA is defined as SERIAL=YES on IMSA, then IMSA registers interest in the transaction by registering interest in the queue name of TRANAbbbIMSA where b represents a blank.

15.3 Scope of Processing

A serial transaction can be processed only on the local IMS system. The queue name on which a serial transaction message is placed contains the IMSID of the local system. The local IMS system is the only system in the shared-queues group that has a registered interest in the transaction.

15.4 Suspended Serial Transactions

If a serial transaction abends with a U3303, for instance, IMS typically USTOPs the transaction and places the message on the suspend queue. In a shared-queues environment, the transaction is USTOPped, but the message is placed back on the transaction serial queue. The message is inserted into the front of the queue, thereby maintaining the sequence of messages on the queue. IMS deregisters interest in the transaction when the transaction is USTOPped. As soon as the transaction has been restarted or the `/DEQUEUE SUSPEND` command has been issued, IMS once again registers interest and attempts to schedule the messages.

15.5 Serializing Transactions in a Shared-Queues Group

Serial transaction processing is not serialized across IMS systems in the shared-queues group, and serial transactions cannot be processed by any IMS system in the shared-queues group other than the local IMS. Therefore if a serial transaction is queuing on one IMS system, no other IMS system in the same shared-queues group can assist in processing the messages. This negates important benefits of implementing shared queues.

It is necessary to determine the reasons for defining transactions as serial. You have to know whether the transactions must process serially per IMS system, serially per shared-queues group, sequentially per IMS system, or sequentially per shared-queues group. If the reason for serialization is a constraint due to MVS resources, the messages can be processed serially per IMS and not per shared-queues group. If the reason is due to database contention when sequential processing might be necessary to ensure that transactions are processed in the order in which they enter IMS or the shared-queues group.

A transaction must be defined as serial only if sequential processing, in a serial fashion, of transactions is necessary. Serialization requirements can be managed in several different ways (see below). If a transaction needs to process serially across a shared-queues group, the TRANSACT macro definition of `SERIAL=YES` is not sufficient.

Below we discuss various solutions to serializing processing in a shared-queues environment. If the transactions need to process in the sequence in which they arrive in the IMS system, `SERIAL=YES` is the only method of ensuring that that happens. The disadvantage is that no other IMS can assist in the processing of the messages if they start queuing under a heavy workload.

Transactions that are defined as serial are processed only on the local IMS system in a shared-queues group, and the transactions are serialized in that IMS system only. At any one time, there can be up to *n* `TRANA`s processing in the shared-queues group if the shared-queues group comprises *n* IMS systems.

If there is a dependency to ensure that at any one time only one of each specific transaction is being processed in the shared-queues group, many different solutions could be implemented. If `TRANA` is a transaction defined as `SERIAL=YES`, consider the solutions listed below.

It is important to remember that local transactions (not defined as `SERIAL`) have higher priority than transactions entered on another IMS in the shared-queues group if the local IMS has a dependent region available to process the transaction.

Therefore if MSGA and MSGB for TRANA were entered on IMSA and IMSB, respectively, at almost the same time, MSGA would be processed first if the single region ensuring serialization was started on IMSA and was waiting for work. If TRANA had messages on the transaction ready queue, and a dependent region happened to be available when a local message for the transaction was entered, the local message would be processed before any other messages already queued for that transaction.

- **Define TRANA as SERIAL=YES in only one IMS in the shared-queues group**.

  The transaction can only be entered and processed on the IMS system in which it is defined (unless the output creation exit, DFSINSX0, is used; see Appendix C, "IMS Exits" on page 169). Other IMS systems cannot assist in workload balancing.

  If the DFSINSX0 is used, it cannot direct a message for a transaction on another IMS that is defined as SERIAL=YES.

- **Define TRANA as LOCAL on one IMS and REMOTE on all other IMSs in the shared-queues group, removing the SERIAL=YES parameter**

  TRANA can be entered on any IMS system in the shared-queues group, and all TRANA messages are routed to and processed on a single IMS system, namely the IMS system that has the system only, the IMS system that has the transaction defined as local. The transactions are routed by using shared queues because all SYSIDs defined to each IMS in the shared-queues group are seen as local. However, only the IMS system with the transaction defined as local registers interest and therefore can process the transactions. It is possible that transactions may not be processed in the order in which they were entered because of the local first processing option.

  If all remote transactions were defined in the IMS system definition with the same APPLCTN macro as the local transaction, the /MSASSIGN command can be used to change the processing of the transaction from one IMS to another in the same shared-queues group. This might be necessary to ensure continuous availability of service in the event that the processing IMS is brought down for scheduled maintenance.

  Refer to Chapter 11, "Multiple Systems Coupling" on page 135 for information about MSC in a shared-queues environment.

- **Define TRANA on all IMS systems, with only one dependent region in shared-queues group**

  Remove the SERIAL=YES parameter in the IMS system definition. Start only one dependent region in the shared-queues group that is capable of processing the transaction. If necessary, change the class of the transaction to a unique class to isolate the transaction from other transactions and ensure that it is the only transaction to be processed in a region. One and only one region must be started on any IMS system in the shared-queues group.

  You can also achieve serial processing by using the MAXRGN parameter if changing the class is not a solution for your environment. MAXRGN=1 ensures that only one transaction is active in a processing region at one time. It is necessary to ensure that all regions in which the transaction is eligible to process are started on only one IMS system in the shared-queues group. The scope of the MAXRGN parameter is within the local IMS system where the transaction can process and not the across the shared-queues group.

If the volume of serial transactions is low in your environment, it may be possible to use the same unique class for all serial transactions, to better utilize the one message region. If the workload is too high for one region to handle, it is possible to use both the unique class and the `MAXRGN=1` parameter and start multiple copies of the same region on a single IMS system.

- **Define `TRANA` as REMOTE on all IMS systems in the shared-queues group and LOCAL on an IMS outside the shared-queues group**

  This solution is similar to the second solution discussed above. The difference is that the MSC link between the shared-queues group and the remote IMS system must be defined and started in order for the messages to be accepted.

Chapter 16. APPC and OTMA Processing in a Shared-Queues Environment

In this chapter we describe the processing of APPC and OTMA transactions in a shared-queues environment. Because APPC and OTMA transactions are processed differently from other transactions in a shared-queues environment, attention needs to be payed to this topic if your installation has high volumes of APPC and OTMA transactions.

16.1 Shared Queue Types Used

The shared queue types used during the processing of APPC and OTMA transactions are:

- Transaction ready queue
- OTMA ready queue
- APPC ready queue
- LTERM staging queue

The OTMA and APPC ready queues are similar and are therefore discussed together. It is important to remember, however, that they are in fact two separate queue types in the shared-queues structure. The transaction ready queue contains the input messages for the OTMA and APPC transactions, and the OTMA ready queue and APPC ready queue contain the output messages of OTMA and APPC transactions only. The LTERM staging queue is used to store the second to nth queue buffer of any message that is over one queue buffer long.

16.2 How IMS Registers Interest in APPC and OTMA Transactions

When an IMS system in a shared-queues environment is started up, IMS registers interest in all transactions that are statically defined. IMS registers interest for a transaction by the queue name of the transaction code. For OTMA- and APPC-initiated transactions, IMS registers interest in the OTMA or APPC transaction the first time the transaction is received from an OTMA or APPC device. The term *device* is used in this section to describe the source of an APPC or OTMA transaction and the destination of the response from an APPC or OTMA transaction.

The queue name in which IMS registers interest is the transaction code with the IMSID concatenated. If the IMS has an IMSID of `IMSA`, and `TRANA` has entered IMS from an OTMA device, IMS registers interest in the OTMA transaction by using the queue name of `TRANA bbbIMSA`, where `b` represents a blank. This is the same for APPC transactions.

16.3 Scope of OTMA and APPC Transaction Processing

Any OTMA or APPC transaction that enters an IMS system in a shared-queues environment can be processed only by that IMS, that is, the local IMS system. The exception to this is an APPC transaction that can be defined as remote and is sent through MSC (or shared queues) to a back-end IMS system for processing.

Both OTMA and APPC processing use MVS services. Restrictions in MVS do not allow OTMA and APPC transactions to be processed on any IMS system other than the local IMS.

16.4 OTMA and APPC Response Processing

Transactions destined for an APPC or OTMA device are placed, respectively, on the APPC ready queue or the OTMA ready queue, and are processed by IMS. The response message is placed on a queue with a queue name of `timestamp + <IMSID>`. IMS keeps track of the messages in the OTMA ready queue and the APPC ready queue and retrieves messages from those queues using the unique queue name. IMS does not register interest in the queue or in the device.

16.5 Special Considerations

The local processing restriction of APPC and OTMA transactions must be kept in mind when implementing shared queues. If the volume of APPC or OTMA transactions is high, the implementation of shared queues may not ensure the balancing of workload across IMSs in the same shared-queues environment. Another IMS system in the same shared-queues environment cannot process the transactions if the local IMS cannot handle the load.

The responses from OTMA and APPC transactions are also restricted to the local IMS system. If the local IMS system fails or for some reason cannot process the output messages, no other IMS system in the same shared-queues environment can process the responses. Only the local IMS system can retrieve the responses from the shared queue.

Chapter 17. Performance and Tuning

The shared queues function was developed to assist IBM customers whose production environment is growing beyond the processing capability of a single processor. Data sharing, the starting point of extending processing capability across multiple systems, was introduced with IMS/VS Version 1.3 for 2 sharing systems and increased by IMS/ESA Version 5 to 32 sharing systems. MSC can be used to route the messages among multiple IMS systems. Any increased CPU cost is covered by the extra CPU capacity available through additional IMSs. Extra CPU cost is balanced by additional

- Availability of service
- Flexibility of options
- Overall capacity.

In this chapter we discuss several ways of achieving better performance in a shared-queues environment.

17.1 Reduce Logging

The shared queue is an overhead in terms of additional work that needs to be done. Refer to Appendix B, “Log Records” on page 163 for a comparison of the amount of logging that was performed with local message queues and now with shared queues.

17.2 Reduce Accesses to the Coupling Facility

Accesses to the coupling facility are overheads that are not accrued with local message queues. The more accesses that can be avoided, the better the response time of the transactions. Accesses to the coupling facility can be avoided by making full use of the Local First option for Fast Path transaction processing. You must ensure that the region occupancy percentages are low enough to cater for the average load expected on the local IMS system.

Full function transactions process locally if resources (message processing regions) are available for immediate message processing. Full function transactions then have fewer accesses to the coupling facility because the input message is placed directly onto the lock queue. The processes avoided are:

- Placing a message on a transaction ready queue
- Notifying each IMS with registered interest of the message in the queue
- Moving a message from the transaction ready queue to the lock queue

In the case of Fast Path input messages, all accesses to the coupling facility are avoided before transaction processing.

17.3 Eliminate Overflow Processing

Overflow processing causes the coupling facility structure to be quiesced when message queues are moved from the primary structure to the overflow structure. Once the selected message queues have been moved, a structure checkpoint is taken. This may cause a delay, which should be avoided especially during peak periods. It is better to allocate an oversized primary structure than to perform overflow processing. Overflow processing should be avoided if possible. It is better to monitor the size of the structure and the number of messages on the queue than to have overflow processing.

17.4 Perform Structure Checkpoints During Low-Impact Periods

Structure checkpoints have an impact when processing in a shared-queues environment. The structure checkpoint can take a long time, relative to the typical transaction processing time. The structure is also quiesced during a portion of that interval. The length of time taken to take a structure checkpoint is determined by the number of messages as well as their distribution in the structure. The structure checkpoint is a parallel process with one stream for each queue type. If one queue on a queue type is large, parallel processing of the queue types offers no advantage in time.

17.5 Ensure Frequent Checkpoint Intervals

Frequent structure checkpoints reduce the number of log records that have to be kept in the MVS log stream and the MVS staging log data set. A restart of an abended CQS is faster if the number of log records to be read is low. Frequent checkpointing also ensures that the size of the MVS staging log data set can be kept to a minimum. The higher the volume of transactions using shared queues, the higher the volume of logging information, and more DASD space is required to hold the logging information.

17.6 Ensure Balanced Network Load

It is preferable to implement some form of network load balancing among the IMS systems in a shared-queues group to ensure that local first processing can be exploited to the fullest. See Chapter 14, “Duplicate LTERM Names and Output Security” on page 149 for information about duplicate LTERM names and Chapter 12, “Significant Status and Affinities” on page 141 for information about significant status.

Appendix A. Transaction Flow

This appendix describes the flow of an IMS transaction, from the time the user logs on to IMS through to the receipt of the transaction output to the terminal. We describe this flow for a Full Function transaction in a local queue environment, and for a Fast Path transaction in a shared-queues environment.

A.1 Full Function Transaction Flow in Local Message Queue Environment

1. Logon
   - User logs on to the `IMSA` VTAM ACB on static terminal `NODE1`.
2. Signon (optional for static LTERM)
   - User signs on using userid USER1.
3. Enter transaction
   - User enters `TRANA MSG-IN`.
4. Data transfer
   - VTAM transfers message to `IMSA`.
5. Input message processing
   - Input message is placed in HIOP pool.
   - Basic edit or MFS editing is performed.
   - Message `MSG-IN` is edited into `IMSA`'s message queue buffer pool.
6. Find destination
   - Call FINDDEST to determine whether the transaction code is valid. If it is not, reject `MSG-IN`.
7. Input queuing
   - Queue `MSG-IN` to SMB to make `MSG-IN` eligible for scheduling.
8. Scheduling
   - `MSG-IN` is moved to the IMS message queue buffer pool if the message is on MSGQ data set.
   - PSB is loaded into PSB, PSBW, and DPSB pool from ACBLIB, if necessary.
   - DMB is loaded into DMBP from ACBLIB, if necessary.
9. Program load/program initialization
   - Program is loaded into scheduled region.
     or
   - Program is initialized.
10. Message queue GU
    - Read `MSG-IN` into region.
11. Program execution

- PGMA executes in region.

12. Output message insert
    - IMSA inserts output message MSG-OUT to message queue buffer pool.
    - IMSA queues MSG-OUT onto the CNT for LTERMA.
13. Syncpoint processing
    - IMSA deletes MSG-IN from the message queue buffer pool.
    - IMSA posts CLB NODE1 as having a message waiting.
14. Program termination (depends on processing options)
    - PGMA ends, depending on processing options.
15. Output message processing
    - Retrieve MSG-OUT from IMSA's message queue buffer pool.
    - Perform MFS output processing, if necessary.
    - MSG-OUT edited into HIOP.
16. Data transfer
    - Message is transferred by VTAM to the node.
17. Output queue processing
    - Acknowledgment is received.
    - MSG-OUT is deleted off message queue buffer pool.

A.2 Fast Path Transaction Flow in a Shared-Queues Environment

1. IMS startup or restart
    - IMSA registers interest in all static transactions that are not defined as Fast Path.
    - IMSA registers interest in TRANB.
2. Logon
    - User logs on to the IMSA VTAM ACB on static terminal NODE1
3. Register interest in LTERM
    - NODE1 is statically defined in IMSA as having an LTERM name of LTERM1.
    - IMSA registers interest with CQSA in LTERMA on the shared queue.
4. Signon (optional)
    - User signs on using userid USER1.
5. Enter transaction
    - User enters TRANA MSG-IN.
6. Data transfer
    - VTAM transfers message (TRANA MSG-IN) to IMSA.
7. Input message processing
    - Input message is placed in CIOP pool.

- Basic edit or MFS editing is performed using MFS pool.
- Message is queued to IMSA's message queue buffer pool.

8. Insert to the CQS
   - IMS queue manager issues a CQS PUT request for the message MSG-IN.
9. CQS notification
   - The CQS notifies IMSA that the TRANA queue has gone from empty to nonempty (assume TRANA queue empty).
10. Scheduling
    - PSB is loaded into PSB, PSBW, and DPSB pool from ACBLIB.
    - DMB is loaded into DMBP from ACBLIB.
11. Read message from CQS
    - IMSA issues a READ to CQSA for TRANA.
    - CQSA passes the message (MSG-IN) to IMSA.
    - CQSA moves the message (MSG-IN) onto the lock queue.
    - IMSA places MSG-IN onto message queue buffer pool.
    - Queue MSG-IN to SMB to make MSG-IN eligible for scheduling.
12. Program load/program initialization
    - Program is loaded into scheduled region.
      or
    - Program is initialized.
13. Message queue GU
    - Read message into region or program.
14. Program execution
    - The transaction is processed by the IFP.
15. Output message insert
    - IMSA inserts MSG-OUT to IMSA message queue buffer pool.
16. Syncpoint processing
    - IMSA deletes message (MSG-IN) from the message queue buffer pool.
    - IMSA notifies CQSA to delete message (MSG-IN).
    - CQSA deletes message (MSG-IN) from lock queue.
    - IMSA issues PUT to CQSA of message (MSG-OUT) for LTERMA.
    - CQSA PUTs message (MSG-OUT) on LTERM ready queue.
17. Program termination (depends on parameters)
    - PGMA ends, depending on processing options.
18. CQS notification
    - CQSA notifies IMSA that the LTERM ready queue for LTERMA has gone from empty to nonempty.
19. Read output message

- IMSA requests (reads) MSG-OUT from the shared queue.
- CQSA passes message MSG-OUT to IMSA.
- CQSA moves message MSG-OUT onto the lock queue.
- IMSA places MSG-OUT on IMSA's message queue buffer pool.
- IMSA queues MSG-OUT onto the CNT for LTERMA.
- IMSA posts CLB NODE1 as having a message waiting.

20. Output message processing
    - IMSA reads message MSG-OUT off IMSA's message queue buffer pool.
    - IMSA performs MFS output processing if necessary.
    - Message MSG-OUT is moved to CIOP pool.
21. Data transfer
    - Message is transferred by VTAM to the node.
22. Output queue processing
    - Acknowledgment is received.
    - IMSA notifies CQSA to delete MSG-OUT.
    - IMSA deletes MSG-OUT from IMSA's message queue buffer pool.
    - CQSA deletes MSG-OUT from the lock queue.

Appendix B. Log Records

In a shared-queues environment, the introduction of the CQS provides a new set of log records. The CQS log records are written to an MVS log stream by the MVS system logger. The CQS log records are required for:

- Structure recovery.
- CQS restart.
- resynchronizing IMS and the CQS.

In addition to the new CQS log records, many changes have occurred to IMS log records. All IMS log records now contain the universal coordinated time (UTC) value, which assists in merging log records into chronological sequence. Queue manager and EMH log records have had a unit of work (UOW) identifier added to them. Both the UTC and UOW identifier greatly facilitate tracing a message within the logs.

B.1 CQS Log Records

The CQS passes the CQS log records to the MVS system logger, which stores them on a logging structure. When the logging structure reaches a defined threshold, the CQS log records are offloaded to a log data set. The CQS log records are deleted when they are no longer required as input for a structure recovery. The logging structure and the log data sets form the log stream. There is a separate log stream for the MSGQ structure and the EMHQ structure.

The CQS log record DSECTs can be obtained by assembling the CQSLGREC macro with parameter TYPE=ALL. Table 1 lists the CQS log records. For more details on the CQS log records, see *IMS/ESA Common Queue Server Guide and Reference*, LY37-3730.

Table 1. CQS Log Records

03	Connect to Structure
04	Disconnect from Structure
07	Put Message on Client Queue
08	Read Message and Move to Lock Queue
0B	Move Message to Another Queue
0D	Delete Message from Structure
10	CQS Shutdown
32	System Checkpoint
40	Structure Initialization
42	Structure Checkpoint
43	Structure Rebuild
44	Overflow Processing
60	Structure Statistics

B.2 Tracking Messages

In a shared-queues environment, messages are tracked by a unique UOW identifier. The UOW identifier is used to track the originating message as well as all messages spawned from that message. IMS QMGR, IMS EMH, and some CQS (types 07, 08, 0B and 0D) log records contain the UOW identifier. IMS log records have a UOW identifier that is 34 bytes long. The CQS log records have a UOW identifier that is 32 bytes long. The UOW identifier comprises the originating IMSID, store clock (STCK) value when the message arrived, processing IMSID, STCK value when it was processed, and a flag to indicate which action was performed on the shared queue structure. The flag indicates either a Get or a Put. The originating IMSID and the STCK value of when the message arrived will be the same for the life of the message. Figure 71 shows the layout of the UOW identifier.

Originating System ID		Processing System ID		Flags
Originating IMSID (8)	Token (8) STCK	Processing IMSID (8)	Token (8) STCK	(2)

Figure 71. Unit of Work Format

The UOW field changes as a transaction goes through different stages of processing. Figure 72 on page 165 demonstrates how the first 16 bytes of the UOW remain the same for the life of the message. The remaining parts of the UOW change for each stage in the transaction's life. Here is what occurs at each stage of the message flow shown in Figure 72 on page 165:

1. MSG1 arrives in IMSA from LTERMA. IMSA puts MSG1 on the MSGQ structure. The originating system message identifier is now set.

1. PGMA reads MSG1 from the MSGQ structure.

2. PGMA performs a program-to-program message switch. The switched-to transaction, MSG2, is put on the MSGQ structure.

2. PGMB reads MSG2 from the MSGQ structure and processes it.

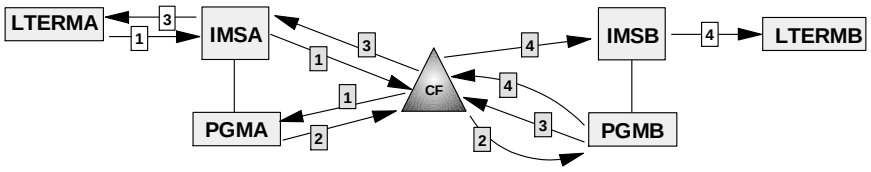
3. PGMB sends a response back to the originating terminal by putting MSG3 on the MSGQ structure, which is to be sent to LTERMA.

4. PGMB also puts MSG4 output on the MSGQ structure, which is to be sent to LTERMB. MSG4 is sent by using the alternate I/O PCB.

3. IMSA reads MSG3 from the MSGQ structure and sends it to LTERMA.

3. IMSB reads MSG4 from the MSGQ structure and sends it to LTERMB.

The UOW identifiers enable the easy identification of all messages that relate to the one transaction.



1	IMSA	T1			P
1	IMSA	T1			R
2	IMSA	T1	IMSA	T2	P
2	IMSA	T1	IMSA	T2	R
3	IMSA	T1	IMSB	T3	P
4	IMSA	T1	IMSB	T4	P
3	IMSA	T1	IMSB	T3	R
4	IMSA	T1	IMSB	T4	R

Figure 72. Unit of Work Identifiers

B.3 Log Sequence

In a non-shared-queues environment, IMS performs the logging to the OLDS data sets. In a shared-queues environment, the logging is performed to both the online data sets (OLDS) and the MVS log stream. In the sections that follow we detail which log records are created during the life of a transaction and where they are placed. We cover full function and Fast Path transactions in both a non-shared-queues environment and a shared-queues environment.

B.3.1 Non-Shared-Queues Environment

In a non-shared-queues environment, for a non-MSC-run transaction, a single IMS system performs all of the logging to the OLDS. Any analysis of how a transaction performed requires looking at only the local IMS system's OLDS or system log data sets (SLDSs). Table 2 shows the log records that IMS creates when a full function transaction is processed. Table 3 on page 166 shows the log records that IMS creates when a Fast Path transaction is processed.

Table 2 (Page 1 of 2). Log Sequence for a Full Function Transaction without Shared Queues

01	Input Message Received
35	Enq Input Message on SMB
31	GU Input Message by Application
03	Insert Output Message
35	Enq Output Message on SMB or CNT
37	Synch Point
33	Free Input Message Buffer
31	GU Output Message by DC
36	Dequeue Output Message

Table 2 (Page 2 of 2). Log Sequence for a Full Function Transaction without Shared Queues	
33	Free Output Message Buffer

Table 3. Log Sequence for a Fast Path Transaction without Shared Queues	
5901	Input Message Received
5903	Output Message Sent
5937	Sync Point
5936	Dequeue Output Message

B.3.2 Shared Queues Environment

In a shared-queues environment, for a non-MSC-run transaction, one or more IMS systems perform logging to their local OLDS. In addition to this, logging is performed by each CQS to the MVS log stream. A transaction can be traced between the front-end IMS system and back-end IMS systems by the UOW field that has been added to all queue manager and EMH log records.

Table 4 shows the log records that are created for a full function transaction that processes on a back-end IMS system. Table 5 on page 167 shows the log records that are created for a Fast Path transaction that processes on a back-end IMS system. In both tables, IMSA is the front-end IMS and IMSB is the back-end IMS. You can see from the tables that more logging is performed in a shared-queues environment than in a non-shared-queues environment.

Table 4 (Page 1 of 2). Log Sequence for a Full Function Transaction with Shared Queues					
IMSA Log Records		**CQS Log Records**		**IMSB Log Records**	
01	Input MSG received				
35	Enq input MSG on SMB				
		07	PUT input MSG		
33	Free input MSG				
		08	READ input MSG		
				01	Prefix input MSG
				31	GU input MSG
				03	Insert output MSG
				35	Enq output MSG to QBLK
				37	Synch point
		07	PUT output MSG		
		0D	DELETE input MSG		
				33	Free input MSG buffer
				33	Free output MSG buffer

Table 4 (Page 2 of 2). Log Sequence for a Full Function Transaction with Shared Queues					
		08	Read output MSG		
03	Prefix output MSG				
35	Enq output MSG to CNT				
31	GU output MSG				
36	Dequeue output MSG				
		0D	DELETE output MSG		
33	Free output MSG buffer				

If the Fast Path transaction is processed by the front-end IMS system, without accessing the EMHQ structure, the log records produced are the same as those for a Fast Path transaction in a non-shared-queues environment. This situation always occurs if the transaction is defined with a sysplex processing code of *local only* and will occur if the transaction is defined with a sysplex processing code of *local first* and the transaction is successfully scheduled on the local IMS system.

Table 5. Log Sequence for a Fast Path Transaction with Shared Queues					
IMSA Log Records		**CQS Log Records**		**IMSB Log Records**	
5911	Input MSG PUT on EMHQ structure				
		07	PUT input MSG		
		08	READ input MSG		
				5901	READ input MSG
				5903	Insert output MSG
				5937	Synch point
				5916	PUT output MSG on EMHQ structure
		07	PUT output MSG		
		0D	DELETE output MSG		
		08	READ output MSG		
5936	Dequeue output MSG				
		0D	DELETE output MSG		

B.4 Messages Moved to the Cold Queue

If an IMS system is emergency restarted with either the COLDCOMM or the COLDSYS parameters, the CQS moves any messages that an IMS was processing at the time of an abend to the cold queue. The locked messages are also moved to the cold queue if CQS was coldstarted, regardless of what IMS did. The messages were on the lock queue at the time of the IMS abend. The messages are moved to the cold queue, when IMS indicates to the CQS during its resynchronization that it has been brought up cold. The CQS informs IMS of the messages moved to the cold queue. IMS creates an X'6740' log record for each message that is moved to the cold queue. X'6740' is a new IMS log record for shared queues. For more information about the cold queue, see 2.3.1, “Private Queue Types” on page 11.

B.5 Printing CQS Log Records

Print the CQS log records from the MVS log stream with the IMS File Select and Formatting Print utility (DFSERA10) with exit routine CQSERA30. Figure 73 shows the JCL required to print the CQS log records from an MVS log stream. This JCL causes the MVS system logger to invoke the default log stream subsystem exit routine, IXGSEXIT, to copy the log records. The exit routine returns a maximum of 32760 bytes of data for each log record even though the CQS may create larger records. You can specify the name of a different exit routine, if necessary.

```
//CQSERA10 JOB   MSGLEVEL=1,MSGCLASS=A,CLASS=K
//STEP1    EXEC  PGM=DFSERA10
//STEPLIB  DD    DISP=SHR,DSN=IMS.RESLIB
//SYSPRINT DD    SYSOUT=A
//TRPUNCH  DD    SYSOUT=A,DCB=BLKSIZE=80
//SYSUT1   DD    DSN=SYSLOG.MSGQ01.LOG,
//               SUBSYS=(LOGR,IXGEXIT),
//               DCB=(BLKSIZE=32760)
//SYSIN    DD  *
CONTROL   CNTL H=EOF
OPTION    PRINT EXITR=CQSERA30
END
//
```

Figure 73. JCL to Print CQS Log Records

The data set name that is specified in `SYSUT1` points to the CQS log stream name that is specified in the CQSSGxxx proclib member. For more information about the CQSSGxxx proclib member, see 5.9.2.3, “Specifying the CQS Global Structure Definition (CQSSGxxx)” on page 79.

Appendix C. IMS Exits

The IMS exits are described in the *IMS/ESA Customization Guide*, SC26-8732. Table 6 summarizes the changes that may be required to IMS exits in a shared-queues environment.

Table 6 (Page 1 of 6). IMS Exits

	Description	**Remarks**
DBFHAGU0	Fast Path Input Edit/Routing	This exit determines whether a Fast Path potential or Fast Path exclusive transaction should be processed by Fast Path EMH. You can specify how a message is processed sysplexwide: **Local first** This is the default. The message is processed locally, if an IFP region is available on the local IMS subsystem. If not, the message is passed to the EMH queue structure. **Local only** The message is not written to the EMH queue structure. The message is processed on the local IMS subsystem. **Global only** The message is placed on the EMH structure. The application program must be active on at least one of the sharing IMS subsystems. If not, the message is discarded and an error message is issued. To avoid unsuccessful scheduling, we recommend using *local first* .
DBFHC40 / DBFHDC44	Data Entry Database Randomizing	These exits are not affected by the implementation of shared queues.
DBFLHSH0	Data Entry Database Resource Name Hash	This exit is not affected by the implementation of shared queues.
DBFUMSE1	Data Entry Database Sequential Dependent Scan Utility (DBFUMSE1) (DBFUMSE1)	This exit is not affected by the implementation of shared queues.
DFSAOE00	Type 2 Automated Operator	All messages queued to an automated operator application must be defined as serial, so that they remain on the IMS subsystem in which they are queued. Callable services, which can be used to send a message to an automation program, go to local storage, not the coupling facility.
DFSAOUE0	Type 1 Automated Operator	This exit may generate transactions for execution. Make sure that these transactions are executed locally (for example, by defining them as SERIAL). A transaction not defined as SERIAL can be processed on any IMS subsystem that has the transaction defined.
DFSBSEX0	Build Security Environment	This exit is not affected by the implementation of shared queues.

Table 6 (Page 2 of 6). IMS Exits		
	Description	**Remarks**
DFSCCMD0	Command Authorization	The CTB address is not passed in a shared-queues environment.
DFSCKWD0	IMS Command Language Modification Facility	In this exit you can alter the keyword table for commands. Check this module against the new /DISPLAY and /CQSxxx commands.
DFSCMLR0 / DFSCMLR1	MSC Link Receive Routing	These exits can reroute messages to different local transactions. MSC must be included in the IMS. Rerouting to another IMS subsystem with a shared-queues group is not possible.
DFSCMPR0	MSC Program Routing	This exit is invoked for change calls only if MSC is defined. If MSC is defined, it is invoked for all transactions, not only those to remote destinations.
DFSCMPX0	Segment Edit/Compression	This exit is not affected by the implementation of shared queues.
DFSCMTR0	Terminal Routing	Whenever possible, use the input message routing exit (DFSNPRT0).
DFSCMTU0	User Message Table	Check this exit against the new messages concerning the CQS and the shared queues.
DFSCMUX0	Message Control/Error	This exit is invoked for the `/DEQUEUE` command for LTERM, node, MSNAME, luname, tpname, tmember, and tpipe but is not invoked for the `/DEQUEUE` command for transactions.
DFSCNTE0	Message Switching (Input)	This exit is not affected by the implementation of shared queues.
DFSCONE0	Conversational Abnormal Termination	In a shared-queues environment, IMS cannot pass a conversational SPA to DFSCONE0 after the input message is queued and before the output message is received. If an /EXIT, /START NODE, LINE, or USER command is entered before the output message is received, the conversation terminates, but the SPA is not passed to the exit routine. If the conversation terminates in this manner, the exit is called a second time when the output message is received, and IMS passes the most up-to-date SPA to the exit.
DFSCSGN0	Sign On/Off Security	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSCSMB0	Transaction Code (Input) Edit	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSCTRN0	Transaction Authorization	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSCTSE0	Security Reverification	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSCTTO0	Physical Terminal (Output) Edit	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSFDOT0	Dump Override Table	This exit is not affected by the implementation of shared queues.

Table 6 (Page 3 of 6). IMS Exits

	Description	Remarks
DFSFEBJ0	Front-End Switch	This exit allows you to keep the input terminal in response mode while it is waiting for the reply from the processing system for messages entered in an IMS system by a front-end switchable VTAM node and processed in another system (for example, a remote IMS). If you use the front-end switch (FES), make sure that the LTERM names used to route the messages are unique across all IMS subsystems in a shared-queues environment.
DFSFLGX0	Logger	This exit is not affected by the implementation of shared queues.
DFSFTFX0	Log Filter	This exit is not affected by the implementation of shared queues.
DFSGMSG0	Greeting Messages	This exit is not affected by the implementation of shared queues.
DFSGPIX0	Global Physical Terminal (Input) Edit	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSHDC40	HDAM Randomizing	This exit is not affected by the implementation of shared queues.
DFSINSX0	Output Creation	This exit is called in an ETO (ETO=Y) and/or shared-queues (SHAREDQ=name) environment. It can be used to validate an unknown destination for a message. When the exit indicates that the destination is a transaction, the transaction must be known locally in order to place the transaction on the shared queue. The exit does not allow the transaction to be executed locally. It supports the specification of transaction attributes such as response, conversational (SPA size), Fast Path, and local/remote. We recommend defining all transactions statically. It might be difficult for the exit to determine whether the unknown destination is a valid transaction, an LTERM, or just invalid. When the exit indicates that the destination is an LTERM, the LTERM must be known locally in order to place the message on a shared queue. For a dynamic LTERM, ETO is required.
DFSINTX0	Initialization	This exit is not affected by the implementation of shared queues.
DFSISIS0	Resource Access Security	This exit is not affected by the implementation of shared queues.
DFSI7770	7770-3 Input	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSLGFX0	Logoff	This exit is not affected by the implementation of shared queues. You can optionally use it to delete significant status for statically defined terminals, which may allow affinity deletion.
DFSLGNX0	Logon	This exit is not affected by the implementation of shared queues.
DFSLUEE0	LU 6.2 Edit	This exit is not affected by the implementation of shared queues.
DFSLULU0	LU 6.2 Destination	This exit is not affected by the implementation of shared queues.
DFSME000	Input Message Field Edit Routine	This exit is not affected by the implementation of shared queues.
DFSME127	Input Message Segment Edit Routine	This exit is not affected by the implementation of shared queues.

Table 6 (Page 4 of 6). IMS Exits		
	Description	**Remarks**
DFSNDMX0	Nondiscardable Messages	This exit provides users with a mechanism to tell IMS what to do with the input message associated with an abended application program. With shared queues, if an invalid destination is specified, IMS invokes the output creation exit.
DFSNPRT0	Input Message Routing	This exit is invoked only if MSC is defined. It is invoked for all transactions, not just those to remote destinations. This exit may request a program-to-program switch to be processed on the local IMS subsystem. There is no need to define the transaction as SERIAL. This exit can be used to avoid scheduling in a remote IMS.
DFSO7770	7770-3 Output Edit Routine	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSPIXT0	Physical Terminal (Input) Edit Routine	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSPPUE0	Partner Product	This exit is not affected by the implementation of shared queues.
DFSPRE60	System Definition Preprocessor (Input Phase)	This exit is not affected by the implementation of shared queues.
DFSPRE70	System Definition Preprocessor (Name Check Complete)	This exit is not affected by the implementation of shared queues.
DFSQSPC0	Queue Space Notification	In a shared queues environment, this exit is called when a QBUFFER is allocated to a data object. The exit cannot examine how full the structures are. A parameter list is passed to the exit. The list includes two new bit settings indicating whether a structure is in overflow mode or not and whether a destination is in overflow mode or not. This exit can use these settings to reject the placement of messages on the shared queue.
DFSSBUX0	Sequential Buffering Initialization	This exit is not affected by the implementation of shared queues.
DFSSGFX0	Sign-Off	This exit is not affected by the implementation of shared queues. You may optionally use this exit to reset significant status for ETO users.
DFSSGNX0	Sign-On	This exit is not affected by the implementation of shared queues.
DFSSIML0	Shared Printer	This exit is not affected by the implementation of shared queues.
DFSSS050	Large SYSGEN Sort/Split Input	This exit is not affected by the implementation of shared queues.
DFSSS060	Large SYSGEN Sort/Split Output	This exit is not affected by the implementation of shared queues.
DFSS7770	7770-3 Sign-on	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFSTXIT0	Time-Controlled Operations	This exit is not affected by the implementation of shared queues.
DFSYDRU0	OTMA Destination Resolution	This exit is not affected by the implementation of shared queues.

Table 6 (Page 5 of 6). IMS Exits		
	Description	**Remarks**
DFSYIOE0	OTMA Input/Output Edit	This exit is not affected by the implementation of shared queues.
DFSYPRX0	OTMA Prerouting	This exit is not affected by the implementation of shared queues.
DFS29800	2972/2980 Input Edit Routine	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DFS36010	4701 Transaction Input Edit Routine	The CTB address is not passed in a shared-queues environment, so register 7 is always zero.
DSPBUFFS	Buffer Size Specification Facility	This exit is not affected by the implementation of shared queues.
DSPCEXT0	RECON I/O	This exit is not affected by the implementation of shared queues.

Appendix D. Special Notices

This publication is intended to help management and technical professionals including system programmers, systems and database administrators, and planners, formulate an implementation plan for using shared queues, a new feature of IMS/ESA Version 6. The information in this publication is not intended as the specification of any programming interfaces that are provided by the IBM Information Management System (IMS), Transaction and Database Server for System/390, Program Number 5655-158. See the PUBLICATIONS section of the IBM Programming Announcement for IMS/ESA Version 6 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

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, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to

the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AS/400
BookManager
CICS/ESA
DB2
DFSMSdss
IBM
IMS
IMS/ESA
MQSeries
MVS
MVS/ESA
OS/390
Parallel Sysplex
PR/SM
RACF
RS/6000
SP
System/360
System/390
VTAM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 179.

- *IMS/ESA Version 6 Guide*, SG24-2228
- *IMS/ESA Data Sharing in a Parallel Sysplex*, SG24-4303
- *IMS/ESA Sysplex Data Sharing: An Implementation Case Study*, SG24-4831
- *IMS/ESA Multiple Systems Coupling in a Parallel Sysplex*, SG24-4750
- *OS/390 MVS Parallel Sysplex Capacity Planning*, SG24-4680
- *Parallel Sysplex Coupling Facility Online Monitor: Installation and Users Guide*, SG24-5153

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

E.3 Other Publications

These publications are also relevant as further information sources:

- *IMS/ESA Common Queue Server Guide and Reference*, LY37-3730 (Available only to licensed customers)
- *IMS/ESA Customization Guide*, SC26-8732
- *IMS/ESA Installation Volume 2: System Definition and Tailoring*, GC26-8737
- *IMS/ESA Operator's Reference*, SC26-8742
- *OS/390 Setting Up a Sysplex*, GC28-1779
- *OS/390 MVS Programming: Assembler Services Guide*, GC28-1762
- *OS/390 MVS Programming: Sysplex Services Guide*, GC28-1771
- *OS/390 MVS System Commands*, GC28-1781

- *OS/390 PR/SM Planning Guide*, GA22-7236

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com/`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/`

- **PUBORDER** — to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com/
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

□ Invoice to customer number

□ Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

Back-end IMS. Any IMS, other than the front-end IMS, that is in the same shared-queues group as the front-end IMS.

base primitive environment (BPE). A system-service layer component of IMS that provides a common set of system services.

checkpoint data set. A local data set that contains CQS system checkpoint information with respect to a group of shared queues.

client. A subsystem that uses the CQS. For example, IMS is a client of the CQS.

cold queue. A CQS private queue type that contains indoubt data objects for a client or a CQS that coldstarted.

common queue server (CQS). A server that receives, maintains, and distributes data objects from a shared queue on a coupling facility list structure for its clients.

coupling facility. A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

CQS restart. Process by which the CQS starts up; either a cold start or a warm start. During a CQS warm start, the CQS environment is restored to the state it was in when CQS terminated. During a CQS cold start, the CQS environment is not restored to a previous state; it is reinitialized.

data object. A piece of client data that is placed on a shared queue through a CQS request.

front-end IMS. The IMS that the terminal was connected to when the initiating conversational transaction was entered.

list structure. A coupling facility data structure that consists of an array of lists and an optional lock table.

log token. A token that identifies a particular log record in the MVS log stream that is used to locate that log record.

master CQS. The CQS that coordinates a sysplexwide task. The other CQSs sharing in the task are participants. If the master CQS fails for any reason, another CQS takes over the role of master and either continues or aborts the task.

overflow structure. An MVS coupling facility list structure that contains shared queues when the primary structure reaches an installation-specified overflow threshold. The overflow structure is optional.

primary structure. An MVS coupling facility list structure that contains shared queues.

queue. A collection of data objects with the same name.

queue name. A 16-byte name used to identify the queue that contains a data object on the shared queues. Multiple data objects may be identified by the same name.

queue type. A logical grouping of queues, either by CQS (private queue types) or the client (client queue types). An example of a CQS private queue type is the lock queue type, in which the CQS groups all locked data objects. An example of a CQS client queue type is the IMS transaction ready queue, in which IMS groups all transaction queues that are ready to be processed.

shared queue. A coupling facility structure, managed by the CQS, that contains data objects that are available to all CQS clients in the sysplex.

structure recovery data set (SRDS). Shared data sets that contain structure checkpoint information for shared queues on a structure pair. There are two SRDSs per structure pair.

sysplex. A set of MVS systems that communicate with each other through hardware components and software services.

unit of work (UOW). To CQS, a client-defined grouping of data objects.

List of Abbreviations

ACB	access control block
AOI	automated operator interface
APPC	Advanced Program-to-Program Communication
APPL	application
APPLCTN	application
APPLID	application identifier
ARM	MVS/ESA Automatic Restart Manager
BLDS	block-level data sharing
BMP	batch message processing region
BPE	base primitive environment
CCB	conversational control block
CF	coupling facility
CFCC	coupling facility control code
CFRM	Coupling Facility Resource Manager
CLB	communication line block
CNT	communication name table
CNTL	control
CPC	central processing complex
CPU	central processing unit
CTB	communication terminal block
DASD	direct access storage device
DB/DC	database/data communications
DBRC	database recovery control
DC	data communication
DCB	data control block
DCCTL	DC control
DD	data set definition
DE	data element
DEDB	data entry database
DEQ	dequeue
DLISAS	DL/I separate address space
DMB	data management block
EMC	event monitor control
EMH	expedited message handling
ENQ	enqueue
ESDS	entry sequenced data set
ESTAE	extended subtask ABEND exit
ETO	Extended Terminal Option (IMS DC)
FE	front end
FF	Full Function
FP	Fast Path
GR	generic resources
GU	get unique
HDAM	hierarchic direct access method
I/O	input/output
IBM	International Business Machines Corporation
IFP	interactive fast path
IMS	Information Management System
IMS/ESA	Information Management System/Enterprise Systems Architecture
IMSA	Information Management System A
IOPCB	input/output program communication block
IRLM	integrated resource lock manager
ISC	intersystem communication
ISRT	insert
ITSO	International Technical Support Organization
JCL	job control language
JCLIN	control statement that precedes job control language input to SMP/E controlled libraries
LCT	local count
LE	list entry
LTERM	logical terminal
LU	logical unit
MFS	message format service
MPP	message processing program
MPR	message processing region
MRQ	IMS Message Requeuer

MSC	multiple systems coupling feature
MSDB	main storage database
MSG	message
MTO	master terminal operator
MVS	Multiple Virtual Storage (IBM System 370 and 390)
MVS/ESA	Multiple Virtual Storage/Enterprise Systems Architecture
OLDS	online log data set
OS	operating system
OTMA	Open Transaction Manager Access
PCB	program communication block
PMTO	primary master terminal
PPT	program properties table
PR/SM	Processor Resource/Systems Manager
PROC	command procedure
PROCLIB	procedure library
PSB	program specification block
PSBNAME	program specification block name
PTB	pass the buck
PTF	physical twin forward pointer
PTF	program temporary fix
QCT	queue count
QMGR	queue manager
RACF	Resource Access Control Facility
RECON	recovery control (data set)
RMT	remote
RRS	Resource Recovery Services/MVS
SDEP	sequential dependent segment
SID	system identifier
SLDS	system log data set
SMB	scheduler message block
SMQ	shared message queue
SPA	scratchpad area
SPC	sysplex processing code
SQ	shared queue
SQG	shared queue group
SRDS	structure recovery data set
SSN	subsystem name
STCK	store clock
SUBSYS	subsystem
SYSID	system identifier
TCF	timer controlled facility
TM	transaction manager
UTC	coordinated universal time
UOW	unit of work
VSAM	Virtual Storage Access Method
VTAM	Virtual Telecommunications Access Method
VTCB	virtual terminal control block
WLR	workload router
WTOR	write to operator with reply
XCF	cross-system coupling facility
XES	cross-system extended services
XRF	extended recovery facility

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

ITSO Redbook Evaluation

IMS/ESA Version 6 Shared Queues
SG24-5088-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-5088-00
Printed in the U.S.A.

IMS/ESA Version 6 Shared Queues

SG24-5088-00