



---

## **IMS Locking** *with Program Isolation or the IRLM*

*Rich Lewis*  
IMS Advanced Technical Support  
IBM Americas

August 2009

---

**IBM**





# Contents

Contents .....	iii
Abstract .....	vi
Trademarks and Service Marks .....	vi
Locking Overview .....	1
Purpose of Locking .....	1
Why a Knowledge of Locking is Needed .....	1
Lock Managers.....	1
Locking Environments.....	2
Online Subsystems.....	2
Block Level Data Sharing.....	2
Database Level Data Sharing.....	2
Resources Locked .....	3
Full Function Databases.....	3
Fast Path Databases.....	3
Lock Levels and Lock Compatibility .....	3
Private Attribute.....	4
Sync Points.....	5
Abends and ROLL, ROLB, and ROLS Calls .....	5
Deadlocks.....	6
Full Function Locking.....	8
Database Record Locks.....	8
Database Record Lock Summary.....	9
Segment Locks.....	10
Segment Lock Summary.....	11
Block Locks .....	11
Block Lock Summary .....	12
Busy Locks.....	12
Busy Lock Summary.....	13
Extend Locks .....	13
Extend Lock Summary .....	13
Data Set Reference Locks .....	14
Data Set Reference Lock Summary .....	14
Command Lock.....	14
Command Lock Summary .....	14
Other Locking .....	15
Locks for the Q Command Code and Get Hold Calls .....	15
Lock Summary Table.....	15
Logical Relationships.....	16
Secondary Indexes .....	18
Fast Path Locking .....	20
Fast Path Lock Manager .....	20
DEDB Locking .....	20

CI Locks.....	20
HSSP and UOWs .....	21
FLD Call Locking for DEDBs.....	22
Q Command Code and DEQ Call.....	22
Segment Level Locking .....	23
Changing Lock Ownership .....	23
MSDB Locking.....	23
RLSE Call .....	25
Area Lock.....	25
Area Lock Summary .....	25
Multiple Area Structure Lock .....	26
Multiple Area Structure Lock Summary.....	26
Command Lock.....	26
Command Lock Summary .....	26
VUNLOAD Lock.....	27
VUNLOAD Lock Summary .....	27
Buffer Overflow (OBA) Lock .....	27
Buffer Overflow (OBA) Lock Summary .....	27
Special Locking Cases .....	29
PROCOPTs of GO, GON and GOT .....	29
PROCOPT of E.....	29
Without Block Level Data Sharing (BLDS).....	29
With Block Level Data Sharing (BLDS) .....	30
HALDB Online Reorganization .....	30
Limiting the Number of Locks Held by a Program .....	31
PI Limit on Number of Waiters .....	32
Lock Timeouts .....	33
IRLM TIMEOUT Value.....	33
IMS LOCKTIME Values.....	33
Deadlocks.....	35
Example of Deadlock between Multiple Programs .....	35
Deadlock Detection Timing.....	36
Choosing a Victim .....	36
INIT STATUS GROUPB Call .....	37
Handling a Deadlock Victim .....	38
Deadlocks with CICS Resources .....	40
Deadlocks with DB2 Resources.....	40
Retained Locks and Lock Rejects.....	42
Database Level Data Sharing.....	43
Virtual Storage Use for Locks .....	44
PI Lock Manager Virtual Storage .....	44
IRLM Virtual Storage .....	44
Design Advice.....	45
Minimize PROCOPT values.....	45
Frequent Sync Points .....	45
Communications within a Sync Interval.....	46

Frequently Updated Records.....	47
Using the RLSE Call.....	47
Using a "Get Lost" Technique .....	48
Using PROCOPT=E to Avoid Checkpointing.....	48
Deadlock Detection.....	49
Locking Traces and Report Programs.....	50
Lock Resource Names .....	50
IMS subsystem ID with IRLM local locking.....	50
Full function locks .....	50
Lock Resource Name Formats.....	51
Fast Path Locks .....	52
IMS Monitor Trace .....	54
PI and Lock Traces .....	54
IMS Monitor (DFSUTR20) .....	55
Reporting of Waits for Space Management.....	56
PI Trace (DFSPIRP0) .....	56
RMF II ILOCK (IRLM Long Lock Detection) Report .....	57
KBLA Deadlock Trace Record Analysis Report (DFSKTDL0) .....	58
KBLA IRLM Lock Trace Analysis Utilities (DFSCLTx0).....	58
File Select and Formatting Print Utility (DFSERA10).....	61
Record Format and Print Module (DFSERA30) .....	61
PI Trace Record Format and Print Module (DFSERA40).....	62
IMS Trace Table Record Format and Print Module (DFSERA60) .....	65
Trace and Report Matrix.....	65
IMS Performance Analyzer for z/OS Reports .....	66
Deadlock Summary.....	66
Deadlock List.....	66
Fast Path DEDB Resource Contention Summary.....	68
Glossary .....	69
Index .....	74

# Abstract

This White Paper is based on *IMS Program Isolation Locking*, GG66-3193, which was published as a IBM Technical Bulletin in December 1990. The document has been extensively revised to include block level data sharing and changes in locking that have been made since 1990. The revisions include coverage of block level data sharing, HALDB, Fast Path Segment Level Locking, the Q command code with DEDBs, the RLSE call, LOCKMAX, LOCKTIME, the Long Lock Report, the KBLA Lock Trace Report, and other changes introduced since 1990.

This white paper is intended to give design advice in the area of locking for installations developing systems that use IMS. The advice applies to both database design and application program design. This advice is given by first presenting an explanation of why and how IMS uses locking and then by explaining its implications for designs. Information on monitoring locking activity is included.

The information in this white paper pertains to IMS Versions 9, 10, and 11. Both IMS TM and DBCTL environments are covered.

I thank Kevin Stewart, Frank Ricchio, Jeff Fontaine, Steve Nathan, and Dave Viguers of IMS Development and Suzie Wendler and Kenny Blackman of Advanced Technical Support for their reviews of drafts of this document and for their assistance with my understanding of IMS locking.

## Trademarks and Service Marks

The following terms, used in this publication, are registered trademarks or service marks of the IBM Corporation in the United States and other countries:

IBM  
CICS  
DB2  
WebSphere  
z/OS

The following terms, used in the publication, have been adopted by the IBM Corporation as trademarks or service marks in the United States and other countries:

IMS

# Locking Overview

## *Purpose of Locking*

IMS uses locking for integrity reasons. Locking isolates database changes made by a program from other concurrently executing programs.

Originally, IMS used the term *Program Isolation* (PI) to describe its locking capability. PI prevents programs from changing data that other programs are accessing and prevents programs from accessing data that other programs have changed but not committed. When a program changes data the isolation occurs until the changes are completed and committed. The isolation typically lasts only for the time required to process one online transaction or for a batch job to reach a checkpoint. The locking provided by PI is limited to one IMS online subsystem. This is either an IMS TM with DB subsystem or a DBCTL subsystem.

Later, IMS added *Block Level Data Sharing* (BLDS). BLDS allows updates by programs in multiple IMS online subsystems and by multiple IMS batch jobs. BLDS added the Internal Resource Lock Manager (IRLM) to IMS. The IRLM is capable of handling locks from multiple IMS subsystems. It is required with BLDS and optionally may be used without BLDS.

## *Why a Knowledge of Locking is Needed*

Locking makes some parts of databases temporarily unavailable to other programs. This can lead to performance problems. Typically locking does not cause performance problems because locks are usually held for a very short time and a very small percentage of resources are locked at any time. On the other hand, it sometimes can have a noticeable effect on the performance of programs that concurrently use the same databases. Database and application designers need to be aware of locking schemes so that they may design these databases and programs to work efficiently. Problems caused by locking are sometimes addressed with database design changes, sometimes with application program changes, sometimes with operational changes, and sometimes with combinations of these.

## *Lock Managers*

IMS has three lock managers: the PI lock manager, the Internal Resource Lock Manager (IRLM), and the Fast Path lock manager. A subsystem never uses both the PI lock manager and the IRLM. One of these two is selected. If Fast Path databases are used, the Fast Path lock manager is also used. A subsystem is an IMS TM/DB system, an IMS DBCTL system, or an IMS batch job.

### **PI Lock Manager**

The PI lock manager can manage only PI locks. It cannot be used with block level data sharing (BLDS). The PI lock manager runs as part of its host subsystem and provides locking services only to the subsystem. Most installations that do not use BLDS choose the PI lock manager.

### **IRLM**

The IRLM can manage lock requests from multiple IMS subsystems or a single subsystem. It also may manage DB2 locks. If BLDS is used, the IRLM must be chosen. The IRLM runs as a separate subsystem in MVS. One IRLM may provide locking services to one or multiple IMS subsystems or one DB2 subsystem. It cannot provide locking services to both IMS and DB2.

**Fast Path Lock Manager**

The Fast Path lock manager is used with Fast Path databases. Fast Path lock requests are first processed by this lock manager. When a lock request conflicts with a previous lock request, the Fast Path lock manager also invokes the other lock manager in the subsystem so that deadlock detection may be done.

## ***Locking Environments***

Locking is used in any environment where multiple application programs may access and update databases concurrently while maintaining data integrity. These include IMS TM with DB subsystems and Database Control (DBCTL) subsystems and IMS BLDS environments.

This document discusses locking for both non-BLDS and BLDS environments.

## **Online Subsystems**

IMS online TM with DB subsystems and DBCTL subsystems require locking to provide integrity. This is required since multiple programs may be updating a database at the same time. Either PI or the IRLM may be used.

## **Block Level Data Sharing**

Block Level Data Sharing (BLDS) adds locks in addition to those for PI. BLDS allows multiple subsystems to share and update databases. BLDS locks are used to provide integrity in this more complex environment. BLDS users must use the Internal Resource Lock Manager (IRLM) to manage IMS locks. When BLDS is used, it does not have to be used for all databases. It is only used for those databases registered to DBRC with SHARELVL(2) or SHARELVL(3). SHARELVL(2) allows sharing by multiple IMS subsystems using the same IRLM. SHARELVL(3) allows sharing by multiple IMS subsystems using multiple IRLMs. Databases which are not registered or which are registered with SHARELVL(1) or SHARELVL(0) do not use BLDS and do not use the locks that are only used by BLDS. SHARELVL(0) does not allow the database to be shared by multiple IMS subsystems. SHARELVL(1) allows restrictive sharing. If one IMS subsystem is allowed to update, others can read but without integrity. If there are no updaters, multiple IMS subsystems may read with integrity.

## **Database Level Data Sharing**

Database Level Data Sharing allows databases to be shared between IMS subsystems, however, updates to a database are allowed in only one subsystem. When updates to a database are allowed in a subsystem, other subsystems may read the database but these reads are without integrity. If no subsystem is allowed to update a database, all subsystems may read it with integrity. Database level data sharing is implemented for a database when it is registered to DBRC with SHARELVL(1).

Database level data sharing may be implemented with either the IRLM or PI as the lock manager for online systems. If IRLM is used, both online subsystems and batch jobs may use it. This provides an advantage for those subsystems which read a database without integrity. The read subsystems can receive notifications of



data set extensions by the updating subsystem. They also may use the Coupling Facility cache structures to receive buffer invalidations. When the IRLM is used with database level data sharing a small number of locks are used in addition to those used with PI locking; however, database level data sharing uses fewer locks than those used with BLDS. The locking differences are explained below under "Database Level Data Sharing" on page 43.

## ***Resources Locked***

### **Full Function Databases**

The basic item that is locked for full function databases is a database record. This is a root segment and all of its dependents. When an application program is positioned in a database record it must hold the lock on the database record.

BLDS adds locks for updated blocks. These are either VSAM CIs or OSAM blocks. The purpose of these locks is to serialize updates in different IMS subsystems.

Other resources are also locked. They will be discussed later in this paper.

### **Fast Path Databases**

The basic item that is locked for Fast Path DEDBs is a CI. When an application program is positioned in a database record it must hold the lock on the CI containing the root. As other CIs for the database record are accessed, locks for them are also obtained.

There is an exception to the locking of CIs for DEDBs. This is segment level locking. It will be discussed under "Segment Level Locking" on page 23.

The basic item locked for the Fast Path MSDBs is a segment.

Other resources are also locked. They will be discussed later in this paper.

## ***Lock Levels and Lock Compatibility***

Locks for IMS resources are obtained at a level. Four levels are used with the PI lock manager. Five levels are used with IRLM. If the resource represented by the lock is already locked by another program, the new lock request may or may not be granted. This depends on the level at which the lock is held and the level requested. The following tables show the lock levels, their names, and the compatibility of locks with other levels. You should be careful when using the lock names because they are sometimes misleading. In fact, sometimes the same names are used for different levels. The first name listed on each row in the "Level Names" column of the table is the one used in reports created by the PI Trace Record Format and Print Module (DFSERA40). More information on this module is given under "Locking Traces and Report Programs" on page 50.

<i>Table 1. Lock Levels</i>		
<b>Level Number</b>		<b>Level Names</b>
<b>PI</b>	<b>IRLM</b>	
1	2	Read or Read Only
Not Used	3	Erase
2	4	Share or Read
3	6	Update, Single Update, or Hold
4	8	Exclusive or Update

A program may request a lock that it already holds. This will never cause a conflict. That is, a request by a user will never conflict with the same user. The compatibility matrix is for lock requests from different programs. A *yes* in the table indicates that the lock request will be granted. A *no* in the table indicates that the lock request will not be granted immediately. The requestor will have to wait until the holder has released the lock.

<i>Table 2. Lock Compatibility Matrix Using PI Level Numbers</i>					
		<b>Level Requested</b>			
		1	2	3	4
<b>Level Held</b>	1	yes	yes	yes	no
	2	yes	yes	no	no
	3	yes	no	no	no
	4	no	no	no	no

<i>Table 3. Lock Compatibility Matrix for IRLM</i>						
		<b>Level Requested</b>				
		2	3	4	6	8
<b>Level Held</b>	2	yes	yes	yes	yes	no
	3	yes	yes	no	no	no
	4	yes	no	yes	no	no
	6	yes	no	no	no	no
	8	no	no	no	no	no

## ***Private Attribute***

The IRLM includes a capability for locks to be compatible within a subsystem but incompatible across different subsystems. This is done by using the private attribute with a lock. The private attribute is used only with the IRLM and with a data sharing environment. If the private attribute is included in a lock request, the lock will not be granted if there is a holder of the lock in another IMS subsystem. The holder may be at any lock level and the requestor may be at any level. This allows the lock manager to grant concurrent lock requests within a subsystem while preventing concurrent holders in different subsystems.

## Sync Points

The length of time that locks are held by a program affects the performance of other programs requesting the locks. All locks held by a program are released when a synchronization point is reached. Synchronization points are usually called “sync points” and sometimes called “commit points”. Some locks are released at other times, but frequent sync points are usually needed to provide good system performance. A sync point occurs when an application program commits the work that it has done. Sync points are created by the conditions shown in the following table.

<i>Table 4. Sync Points</i>	
<b>Program Type</b>	<b>Sync Point Condition</b>
IMS MPP	GU to IOPCB with MODE=SNGL CHKP call Program termination
IMS Message-driven BMP	GU to IOPCB with MODE=SNGL CHKP call Program termination
IMS Non-message-driven BMP	CHKP call SYNC call Program termination
IMS FP EMH (IFP) program	GU to IOPCB with MODE=SNGL CHKP call
IMS JMP	IMSTransaction.commit() GU to IOPCB with MODE=SNGL Program termination
IMS JBP	IMSTransaction.commit() IMSTransaction.checkpoint() Program termination
CICS task	Term command or call CICS sync point Transaction termination
ODBA thread	RRS Commit (ATRCMIT or SRRCMIT)

A GU to the IOPCB with MODE=MULT does not create a sync point. Sync points are created for MODE=MULT applications by program termination, CHKP calls, or Java IMSTransaction commit().

## Abends and ROLL, ROLB, and ROLS Calls

Locks are also released by application program abends and ROLL, ROLB, and ROLS calls.

In IMS online subsystems the abend of an application program results in IMS backing out its database updates and releasing its locks.

Abends of batch (DLI or DBB) jobs which participate in block level data sharing may be dynamically backed out. This only occurs when the BKO=Y parameter is specified, a DASD log is used, and the abend is an IMS pseudo abend. When dynamic backout is invoked for a batch job all of its locks are released. If dynamic backout is not invoked, the locks are released by the Batch Backout (DFSBB000) utility.

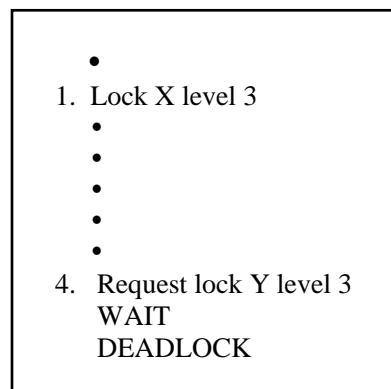
The ROLL, ROLB, and ROLS calls may be used to either abend an application program or backout some or all of its work. The ROLL call creates a U0778 abend. The ROLB call backs out all database updates since the last commit point and returns control to the application program. All locks are released by this processing.

The ROLS call may be used to back out to the last commit point or to an intermediate point which was set by a SETS or SETU call. When the backout is to the last commit point, all locks are released. When the backout is to an intermediate point, no locks are released. In this case, IMS does not have information about which locks are no longer needed. Database record locks and block locks that were required for the backed out updates might still be required since they may have been obtained for processing before the back out point.

## Deadlocks

Occasionally, lock requestors become *deadlocked*. This happens when two requestors are waiting on each other. The following example illustrates such a situation.

Program A



Program B

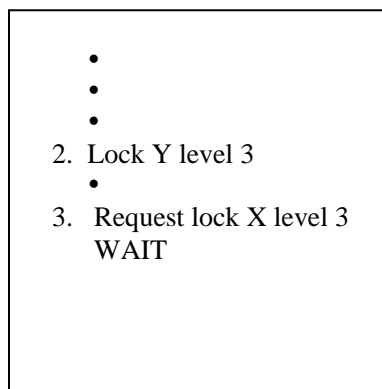


Figure 1. Deadlock example

1. Program A requests a lock on resource X at level 3. The lock is granted.
2. Program B requests a lock on resource Y at level 3. The lock is granted.
3. Program B requests a lock on resource X at level 3. The lock request can not be granted because program A already holds the lock at level 3. Program B must wait.
4. Program A requests a lock on Resource Y at level 3. The lock request cannot be granted because program B already holds the lock at level 3. Program A must wait.

Since both are waiting, neither will give up the locks they hold. Special action is required by the lock manager to resolve this deadlock. The lock manager, either PI or IRLM, will detect the existence of a deadlock. When one is found, one of the participants is selected to be the victim. Its updates are backed out and its locks released. This allows the other participant in the deadlock to continue. The backout and release of locks is done by either abending the program or issuing an internal ROLB call. Some deadlocks involve more than two programs and, possibly, more than two resources. More information on deadlocks is available under “Deadlocks” on page 35.

Deadlock resolution has an interesting implication about the acquisition of locks during backout. Backout processing never requires new locks. If it did, a backout would be exposed to creating a deadlock. This is not allowed since backouts are used to resolve deadlocks and a deadlock created during the resolution of another deadlock would be very difficult to handle.

# Full Function Locking

This section documents the locks used with full function databases.

## Database Record Locks

As mentioned in the Locking Overview section above, the basic item locked for full function databases is a database record. The database record is identified in different ways for the different access methods. The following table shows the resources that are locked.

<i>Table 5. Full Function Database Record Locks</i>	
<b>Access Method</b>	<b>Lock Resource</b>
HISAM	Hashed key of root segment
HIDAM and PHIDAM	RBA of root segment Hashed key of root segment
HDAM and PHDAM	RBA of RAP

The resource that is locked is the value in this table plus an identification of the database and database data set.

For HISAM, IMS hashes the key of the root segment. This produces a resource name that is locked. There are millions of possible values that the hashing algorithm produces. This tends to minimize the possibility of different keys hashing to the same value and producing lock conflicts.

For HIDAM and PHIDAM, the RBA of the root is always used to identify the database record. The root segment resides in the prime HIDAM or PHIDAM database, not the index. The RBA is from this prime HIDAM or PHIDAM database. There are times when the hashed value of the key of the root segment is also used. This is the key in the index. Locking of the hashed key occurs when IMS is either inserting a root segment or erasing it. These are the only times that changes are made to the index. When a record is being inserted into or deleted from the index, IMS locks the hashed key to prevent two programs from adding or deleting the same root segment concurrently.

IMS locks the RBA of the Root Anchor Point (RAP) from which the root is chained for HDAM and PHDAM databases. Since multiple roots may be chained from the same RAP, this is really a lock on one or more database records. When one root is locked, all the roots on the RAP chain are locked.

The level for a lock on a database record depends on the processing option (PROCOPT) of the PCB used for the call that acquires the position in the record. If the PROCOPT allows updates, the lock is acquired at PI level 3 or IRLM level 6. If the PROCOPT does not allow updates, the lock is acquired at PI level 2 or IRLM level 4. Since PI level 3 (IRLM level 6) is not compatible with other PI level 3 or 2 (IRLM level 6 or 4) holders, updaters do not share a database record. Since PI level 2 (IRLM level 4) is compatible with other PI level 2 (IRLM level 4) holders, multiple non-updaters may share a database record.

A program may have update sensitivity to a database, but lock a database record at PI level 2 (IRLM level 4). This occurs when the PSB has multiple PCBs referencing the same database. The level of the lock request is determined by the PROCOPT of the PCB used for the call.

Database record locks are released either when position is moved to another database record, when a sync point is reached, or when a RLSE call is issued. The RLSE call is discussed below. If no updates are done

while the program is positioned in the database record, the lock is released when position for the PCB is moved to another database record. There is one exception to this release. In a BLDS environment with a PROCOPT which includes E, the lock on the database record is not released until the application sync point is reached. Moving position to another database record will not release the lock in this case.

In a PI locking environment the database record lock may be demoted from level 3 to level 1 when an update has been done and position is moved to another database record. This depends on the access method being used and the segment which is updated. If HISAM is being used and there are no updates in the primary logical record, that is, in the logical record holding the root, the lock is demoted to level 1. If there are any updates in the primary logical record, the database record is retained at level 3. For other access methods if the root is updated, the lock is retained at level 3, otherwise it is demoted to level 1. Database record locks held at level 1 or 3 are kept until a sync point is reached. The use of the level 1 locks on database records is explained below under the discussion of segment locks.

The RLSE call may be used to release locks not protecting updates. When the call is issued using a full function PCB, only the locks for that PCB are released. Typically, this is the lock for the current database position. If the PCB references a logical database or a secondary index, there may be multiple database record locks for the current position. RLSE calls that use a Fast Path PCB do not release any full function locks.

HALDB Online Reorganization always requests the database record lock at PI level 3 or IRLM level 6. It processes a partition sequentially and requests the locks on a set of database records at a time. This is a unit of reorganization. It releases these locks at the end of each unit of reorganization.

Database record locks are also used for locking with the Q command code when the IRLM is the lock manager. This is explained on page 15 under "Locks for the Q Command Code and Get Hold Calls."

## Database Record Lock Summary

**Purpose:** Controls access to database records. Database record locks prevent access to uncommitted updates or to database records for which the Q command code is used.

**Locking environments:** Online systems and BLDS

**Resource locked:** Database record. For HISAM and secondary indexes this is a hash value for the key. For HIDAM and PHIDAM this is the location of the root segment. For HDAM and PHDAM this is the location of the RAP from which the root segment is chained.

**Level:** The level depends on the PROCOPT for the PCB used by the call. PROCOPTs allowing updates use PI level 3 or IRLM level 6. PI locks at level 3 may be demoted to level 1 when position is moved to another record. PROCOPT=G uses PI level 2 or IRLM level 4.

**Requested:** When database record is first accessed.

**Released:** If the database record is not updated, the lock is released when position for the PCB is moved to another database record. In a PI locking environment when the root segment (or the primary logical record for HISAM) is not updated, the lock is released when position is moved to another record even when a dependent segment in the database record is updated. Otherwise, the lock is released at application sync point.

## Segment Locks

The choice of lock managers, PI or IRLM, affects locking for dependent segments. If the PI lock manager is used, individual segments are locked when they are updated. Other segments in the database record may be accessed when the updating program moves its position to another database record. If the IRLM is used, individual segments are not locked, instead, the lock on the database record is held until the updates are committed. This locks the entire database record.

The rest of the discussion of the segment locks applies to installations using the PI lock manager.

Locks for segment updates are always obtained at level 3. This prevents two programs from updating a segment concurrently. The locking of root segments and dependent segments are handled differently. Roots are locked with the database record lock. This works well since there is a one-to-one relationship between roots and database records. Dependent segments are locked by their RBA or relative record number (RRN).

Locks for segment updates are always held until a sync point is reached.

The following table shows the resources that are locked for segments.

<i>Table 6. Full Function Segment Locks</i>	
<b>Access Method</b>	<b>Locked Resource</b>
HISAM	Hashed key of root segment RRN of overflow logical record
HIDAM and PHIDAM	RBA of segment
HDAM and PHDAM	RBA of RAP for roots RBA of segment for dependents

For HDAM, PHDAM, HIDAM, and PHIDAM, the RBA of the dependent segment is used to identify the segment when the lock request is made.

HISAM database segments are not individually locked. Instead, IMS treats the primary logical record as a root segment and the overflow logical records as dependent segments. A change to any segment in the primary logical record is protected by the lock of the database record. A change to any segment in an overflow logical record causes the overflow logical record to be locked. Overflow logical records are identified by the database, data set, and RRN in the data set.

PI level 1 locks for database records have a special use. They are used in conjunction with segment locking in a PI locking environment. When a segment is updated, its database record must be locked at level 3. When position is moved to another database record, IMS sometimes demotes the lock on the database record to level 1. This is explained above in the discussion of database record locks. The level 1 lock is used as an indicator to other programs which may establish position in the database record. If the record is locked at level 1, it indicates that there is at least one segment in the record that has been updated. The second program must see if a dependent segment is locked before it can access it. If the database record is not locked, the second program does not have to check for locks of any dependent segments before accessing them. When IMS asks for a lock on a database record, an indication of whether or not the database record is locked at level 1 is returned as part of the lock request. If it is locked at level 1, IMS must test to see if each of its dependents is locked before accessing them. It does this with a special type of request which is sometimes called a test enqueue (TENQ). If a dependent is locked at level 3, the test enqueue causes the requestor to wait. If it is not locked, the test enqueue does not cause a wait. In either case, this test enqueue request does not cause the segment to be locked.



Segment locks are also used for locking with the Q command code. This is explained on page 15 under "Locks for the Q Command Code and Get Hold Calls."

## Segment Lock Summary

**Purpose:** Prevents access to updated dependent segments. It is only used in a PI locking environment.

**Locking environments:** Online systems using the PI lock manager

**Resource locked:** Segment. For HDAM, PHDAM, HIDAM, and PHIDAM this is the location of the segment. For HISAM this is the location of the logical record in which the segment resides.

**Level:** The level is always PI level 3.

**Requested:** When a dependent segment is updated in a PI locking environment.

**Released:** At application sync point.

## Block Locks

Block locks are used only with BLDS. They are used to serialize updates to the same physical block by different subsystems. These are locks on VSAM CIs or OSAM blocks. These block locks are typically requested at IRLM level 4. Since IRLM level 4 is compatible with other level 4 requests, the level does not prevent concurrent holders of the lock. Instead, a different mechanism is used. This is the private attribute. Block locks are always requested with the private attribute. The private attribute applies to the IMS subsystem, not to the individual requestor. Multiple transactions or programs running in the same IMS subsystem may hold the block lock concurrently. Transactions or programs running in different IMS subsystems cannot hold the block lock concurrently. There is a good reason for this handling of block locks. Updaters of a block in the same IMS subsystem will never attempt to update the same part of a block. The database record lock prevents this. They will be updating different database records in the same block. Since all programs and transactions in one IMS subsystem use the same buffer pools, these updates will be done to the same copy of the block. There is no need to serialize these updates. On the other hand, updates made in different IMS subsystems use different buffer pools and, therefore, different copies of the block or CI. These must be serialized. If they were not, updates from one system would overwrite the updates for another when the buffer was written to disk.

All updates to OSAM and VSAM ESDS data sets use level 4 locks. There is special handling of block locks for KSDSs. Replaces and inserts of logical records in a KSDS request a level 4 lock. Deletes of logical records in a KSDS request a level 3 lock. Level 3 locks are compatible with other level 3 requests, but incompatible with level 4. This allows multiple concurrent deletes of records in the same KSDS CI by programs in the same IMS subsystem. It does not allow concurrent deletes with inserts or replaces. This is done to ensure that a backout of a delete will have space available in the CI for the reinsertion of the logical record. If inserts were allowed before the delete was committed, the space might not be available. The second case of special handling for KSDSs is used with CI/CA splits. When a CI or CA split occurs, the lock request for the CI is upgraded to level 6. This is incompatible with all other levels which are used. It prevents any other program or transaction from updating the CI while the split occurs.

HALDB Online Reorganization requests block locks when BLDS is used for the reorganized database. Obviously, the output data sets are updated and there are block locks for these blocks or CIs. There are also

updates to the input data sets and block locks are requested for them. The cursor is written in the second block or CI of the input data set. There is a block lock for this block or CI for every unit of reorganization. When twin backward (TB) pointers are used with PHIDAM, the twin chain between roots must be maintained across the input and output data sets. When pointers are updated in the input data set, block locks are requested for their CIs or blocks. Block locks for HALDB Online Reorganization are released at the end of the unit of reorganization.

## Block Lock Summary

**Purpose:** Serializes updates to an OSAM block or VSAM CI from different IMS subsystems. It also is used to prevent concurrent inserts and deletes to the same KSDS CI within an IMS subsystem and to prevent CI/CA splits concurrent with other updates to the same CI.

**Locking environments:** BLDS

**Resource Locked:** The location of the OSAM block or VSAM CI.

**Level:** IRLM level 4 is always used for OSAM and VSAM ESDSs. IRLM level 4 is used for KSDS record inserts and replaces. IRLM level 3 is used for KSDS record deletes. IRLM level 6 is used for CI/CA splits.

**Attribute:** The private attribute is always used.

**Requested:** When an update occurs for a VSAM CI or OSAM block in a database using BLDS.

**Released:** At application sync point.

## Busy Locks

IMS uses busy locks to serialize some activities to database data sets. These are OPENs, CLOSEs, new block creations, and updates to KSDSs. New block creation is either the use of a new block at the end of an HDAM, PHDAM, HIDAM, or PHIDAM database data set or the addition of a new logical record to a HISAM data set. There is a busy lock for each database data set. The serialization for OPENs and CLOSEs is done to ensure that two programs are not trying to open at the same time. OPENs and CLOSEs will not occur when other programs are accessing the database data set. New block creations are different. They are likely to occur while other programs are using the data set. Before a new block is created, IMS asks for the busy lock on the data set. This does not prevent other use of the data set but it ensures that only one program will be creating new blocks at the end of a data set at any time. Busy locks for OPEN, CLOSE, and new block processing are always requested at PI level 4 or IRLM level 8.

Busy locks for KSDS updates are used to protect against updates to a CI while a CI/CA split is occurring in a block level data sharing environment. Even though different programs might be updating different records in a KSDS CI, there is a potential problem. A CI split due to the insert of a record could cause other records in the CI to be moved to another CI. This problem is avoided by serializing all updates to CIs while an insert is being processed. When an insert to a KSDS is done, a busy lock at IRLM level 8 for the data set is requested. Other updates to KSDSs request the busy lock at IRLM level 2 for the data set.

Busy locks are only held while processing of the KSDS update, new block, OPEN, or CLOSE is being done. They are requested and released as part of one DL/I call or one operation.

## Busy Lock Summary

**Purpose:** Serializes opens and closes of database data sets. Serializes the creation of new blocks in database data sets. Prevents updates to a KSDS in one IMS system while a CI/CA split is being processed in another IMS system in a BLDS environment.

**Locking environments:** Online systems and BLDS

**Resource Locked:** Database data set which is identified by its DMB number and data set number.

**Level:** PI level 4 or IRLM level 8 is used for open, close, and new block creation. IRLM level 8 is used for KSDS inserts. IRLM level 2 is used for KSDS updates other than inserts.

**Requested:** When a database data set is opened or closed or when a new block is created. In a block level data sharing environment, the lock is requested when an update to a KSDS is done.

**Released:** At the end of the operation.

## Extend Locks

Extend locks are used for extending database data sets. This is adding new allocations. Extend locks are used only with the IRLM for databases registered to DBRC with a SHARELVL of 1, 2, or 3. The locks are used to serialize these extensions between IMS systems. Since open and closes should not be done while an extension is in process by another subsystem, the extend lock is also acquired when a data set is opened for update or closed after an open for update. Extend locks are only used in a block level data sharing environment. The locks are requested at IRLM level 2 with the private attribute. The private attribute prevents requestors from different IMS subsystems from holding the lock concurrently.

## Extend Lock Summary

**Purpose:** Serializes extensions of database data sets across multiple IMS subsystems.

**Locking environments:** BLDS and database level data sharing using IRLM

**Resource Locked:** Database data set which is identified by its DMB number and data set number.

**Level:** IRLM level 2 is always used.

**Attribute:** The private attribute is always used.

**Requested:** When a database data set is extended, opened, or closed.

**Released:** At the end of the extend operation.

## **Data Set Reference Locks**

Data set reference locks are used only with the IRLM for databases registered to DBRC with a SHARELVL of 1, 2, or 3. The data set reference lock for a database data set is held by all IMS subsystems which have a data set open. These locks are not used to serialize access to anything. Instead, they are used in conjunction with notifications. For example, when an IMS subsystem extends a data set, it sends a notification to other IMS subsystems which have the data set open. It does this by sending the notification through the IRLM with a reference to the data set reference lock. The IRLMs then send the notification to all holders of this data set reference lock. Since the data set reference lock is held by all IMS subsystems which have the data set open, all of these IMS subsystems receive the notification.

## **Data Set Reference Lock Summary**

**Purpose:** Used for routing notifications to IMS subsystems which have a data set open

**Locking environments:** BLDS and database level data sharing using IRLM

**Resource Locked:** Database data set which is identified by its DMB number and data set number.

**Level:** IRLM level 2 is always used.

**Requested:** When a database data set is opened

**Released:** When a database data set is closed

## **Command Lock**

The command lock is requested by each IMS subsystem using an IRLM. It is requested when the IMS subsystem is started. The lock is requested at a share level. It is held until the subsystem terminates. The lock is not used to serialize access to anything. Instead, it is used in conjunction with notifications that are not associated with a database. For example, type-1 commands with the GLOBAL parameter are sent between IMS systems by using notifications to holders of this lock.

## **Command Lock Summary**

**Purpose:** Used for routing notifications to IMS subsystems

**Locking environments:** IMS subsystems using the IRLM

**Resource Locked:** A "dummy" resource

**Level:** IRLM level 2 is always used.

**Requested:** When an IMS subsystem is started

**Released:** When an IMS subsystem is terminated.

## Other Locking

### Locks for the Q Command Code and Get Hold Calls

The use of the Q command code causes additional locking. The Q command code may be specified when accessing a segment. It ensures that the retrieved segment will not be updated by another program when position is moved to another database record. The Q command does not prevent other application programs from retrieving the segment. Locking for the Q command code differs between the PI lock manager and the IRLM. When the PI lock manager is used and the Q command code is specified, IMS gets the lock for the segment at level 2. For HDAM and PHDAM roots, this is a lock on the RAP. Of course, if the application program also updates a dependent segment, the lock is promoted to level 3. The Q command code may be released by a DEQ call. The DEQ call will cause IMS to release the level 2 lock on segment. If a DEQ call is not used, the locks for the Q command code are released at sync point time. When the IRLM is used individual segments are not locked. Instead, the Q command code causes the database record to be held at IRLM level 4 until sync point time or a DEQ call is issued. If the application also updates a segment in the database record, the level 6 lock is held until sync point.

The use of the Q command code affects the processing of get hold calls. Since the Q command code is used to prevent updates of segments, it must prevent get hold calls from retrieving them. With PI the lock on the segment is used for this. With the IRLM the lock on the database record is sufficient. If there are no Q command codes used, IMS does not need to make a lock request when retrieving a dependent segment unless the database record is locked at level 1 and the PI lock manager is used. If Q command codes are used with the PI lock manager, IMS must see if a segment is locked when processing a get hold call for it. The following describes the processing with PI. When a Q command code is issued for a database, IMS turns on an indicator associated with the database. When this indicator is on any get hold call for the database will cause a special lock request to be issued. This lock request is similar to the one used for accessing dependent segments in database records locked at level 1. It is sometimes called a test enqueue (TENQ). The lock request for get hold calls test to see if the dependent segment is locked at PI level 2. If it is, the get hold call waits. If it is not locked, the requestor does not have to wait. In either case, the get hold call does not cause the segment to be locked. Only update calls and Q command code calls cause dependent segments to be locked. As mentioned above, the indicator that Q command codes have been used is associated with a database. Each database has one indicator. As long as the indicator is turned on, all dependent segment get hold requests in a database require the test for the level 2 lock. The indicator is turned off only when there are no application programs scheduled with intent against the database.

When EXEC DLI commands are used, Q command codes and get hold calls are not explicitly used. They are implied by certain commands. The use of either the LOCKED or LOCKCLASS option with a get function in an EXEC DLI command is equivalent to the use of a Q command code with a call. With EXEC DLI all get processing implies get hold processing. That is, each EXEC DLI command which specifies a function of GN, GNP, or GU is equivalent to a get hold call. If Q command codes or LOCKED or LOCKCLASS options are used in the system, the locking for EXEC DLI gets calls will be affected as explained above for Q command codes and get hold calls.

### Lock Summary Table

The following table summarizes the locks used for full function databases, the levels at which they are held, and the use of each level.

Table 7. Full Function Locks Summary			
Lock Type	Level		Meaning
	PI	IRLM	
Database Record	1	N/A	Updated dependent segment in database record
	2	4	Non-update program positioned in database record or Q command code held for root (PI) or any segment in database record (IRLM)
	3	6	Update program positioned in database record or Updated root segment
Segment	2	N/A	Q command code held for segment
	3		Updated segment
Block	N/A	3	Delete of KSDS record
		4	Update of OSAM block, update of ESDS CI, or insert or replace of KSDS record
		6	CI/CA split for KSDS
Busy	1	2	KSDS non-insert operation
	4	8	OPEN, CLOSE, new block, or KSDS insert being processed
Extend	N/A	2	Data set extension
Data Set Reference	N/A	2	Data set open

## Logical Relationships

The use of logical relationships affects locking. IMS locks database records in physical databases, but logical database records may be comprised of multiple physical database records from one or more physical databases. When a logical database record is accessed, IMS locks the physical database records as it accesses them. The following is an example of this locking.

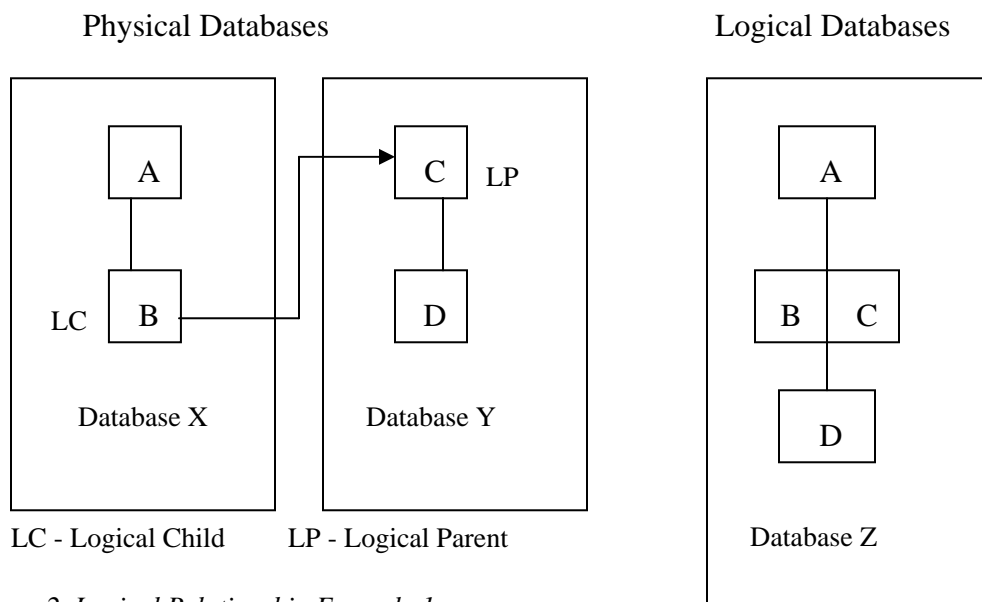


Figure 2. Logical Relationship Example 1

If a program using logical database Z gets segment A, it will get a database record lock for database X. If it gets segment BC, it will get a database record lock for database Y.

The following figure illustrates a slightly more complicated case and the processing needed to lock the database records.

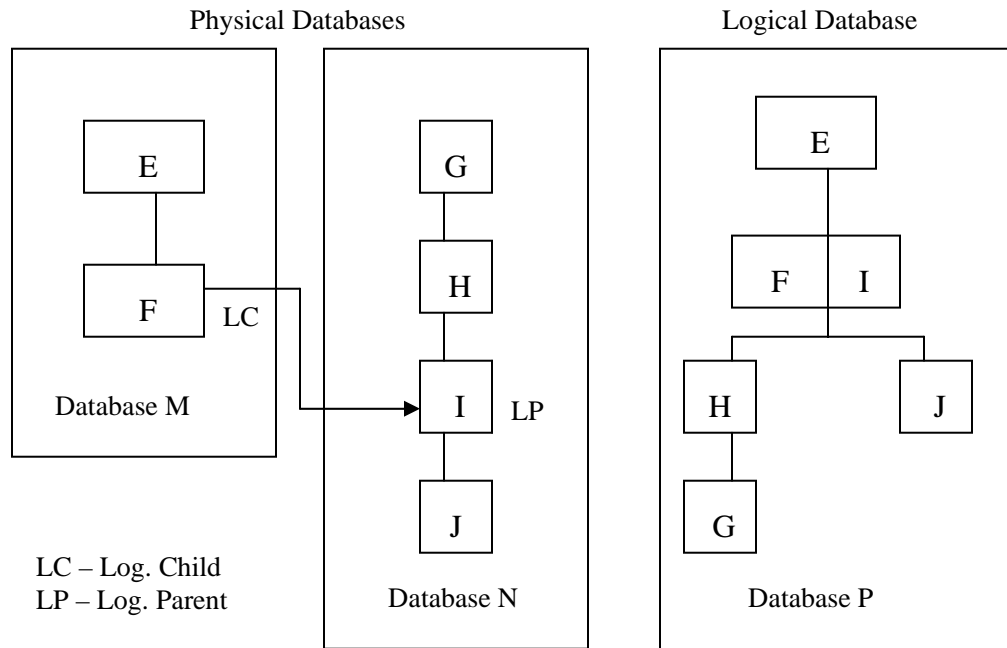


Figure 3. Logical Relationship Example 2

If a program using logical database P gets segment E, it will get a database record lock for database M. If it gets segment FI, it will get a database record lock for database N. This database record lock is associated with root segment G.

IMS may need to access other segments in the database containing the destination parent (segment I). This may be required so that IMS may lock the database record. The processing depends on the type of pointers that are used.

- If symbolic pointers are used for the logical relationship, the root is traversed to get the destination parent segment and the data needed to lock the database record is found on the way to the destination parent.
- If direct pointers are used for the logical relationship, IMS may have to find the resource to lock after reaching the destination parent segment.
  - If HIDAM is used for database N, the root's RBA must be determined. This is the value in the physical parent pointer of the root's child (segment H). Physical parent pointers are used to do this. In our example the physical parent pointer in segment I would be used to reach segment H. The value in the physical parent pointer in segment H would be the RBA of segment G. This RBA is used to lock the database record. PHIDAM simplifies this processing. The HALDB extended pointer set (EPS) in segment F contains the RBA of segment G. This is used to lock the database record.
  - For HDAM databases the RAP must be found. If the concatenated key of the destination parent (segment I) is stored in the logical child (segment F), the root (segment G) key is used as input to

the randomizing routine. The routine is used to find the RBA of the RAP. If the concatenated key of the destination parent is not stored in the logical child segment, physical parent pointers are used to access the root. The root is then used to find the RAP. PHDAM simplifies this. The HALDB extended pointer set (EPS) in segment F contains the RBA of the RAP. This is used to lock the database record.

As the previous discussion illustrates, decisions to use symbolic keys or direct pointers and whether or not to store the concatenated key in the logical child affect the processing that IMS must do to lock database records for HDAM and HIDAM databases. These considerations do not apply to PHDAM and PHIDAM databases.

## Secondary Indexes

Secondary indexes are special databases used for alternative accessing of other databases. There are special locking considerations for these secondary indexes.

The following figure illustrates a database with a secondary index. Segment C is the source segment. That is, the secondary index is based on data in segment C. Segment B is the target segment, that is, the secondary index points to segment B.

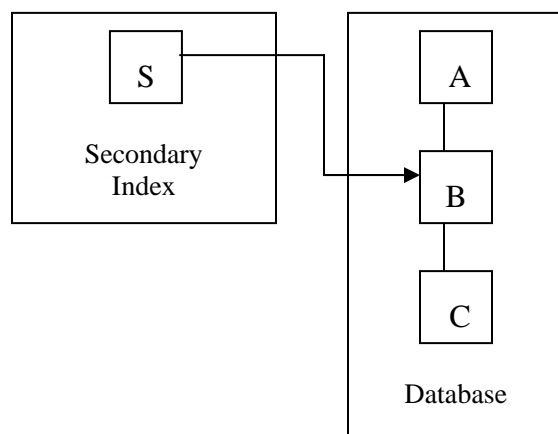


Figure 4. Secondary Index

When an entry is added or deleted in a secondary index, a level 3 database record lock on the secondary index is obtained. The locked resource is the key of the index entry and it is held until a sync point is reached. This is the same way that HISAM databases are locked. If a source segment for a secondary index is replaced and the source field is changed, two changes must be made to the secondary index. One secondary index record must be deleted and another must be inserted. Locks on both of these secondary index database records must be obtained.

Updates to secondary indexes are generally made due to changes in source segments of primary databases; however, secondary indexes also may be processed as databases. When this is done, locking for them is the same as for HISAM databases.

When a secondary index is used to access a database by an alternate processing sequence (PROCSEQ= is specified on the PCB), the database records in the primary database are locked. The same locking that would occur if the database records were accessed through the root of the primary database is done. The secondary index may also be locked. This depends on the pointers in the secondary index and the primary database organization. No lock on the secondary index is requested for some cases. This is true for symbolic pointers to HDAM, for direct pointers to HIDAM roots, and for all HALDB secondary indexes. For all other cases



(direct pointers to HDAM, direct pointers to HIDAM dependents, and symbolic pointers to HISAM and HIDAM), the secondary index entry's database record lock is obtained.

A secondary index may also be used to process qualifications in a segment search argument (SSA). This may be done when INDICES = is specified on the SENSEG statement in the PSB. There is no locking of the secondary index entries when this use is made of the index. Locking of the secondary index is not required because the database record lock in the primary database is held. Since the secondary index can only be modified by changing its source segment in the primary database, the lock on the primary database record is sufficient.

# Fast Path Locking

## *Fast Path Lock Manager*

IMS Fast Path databases have their own lock manager which is used in conjunction with either the PI lock manager or the IRLM. The Fast Path lock manager is used only for Data Entry Databases (DEDBs) and Main Storage Databases (MSDBs). When a lock for one of these databases is needed, a request is made to the Fast Path lock manager. If the lock can be granted, no request is made to the PI lock manager or the IRLM. This tends to save instructions because the Fast Path Lock manager is specialized for locking Fast Path resources. If the lock request must wait, the Fast Path lock manager must check to see if a deadlock situation exists. Since a deadlock could include a full function database resource, the other lock manager, either PI or the IRLM, must be consulted. The other lock manager does all of the deadlock detection processing. To do this processing, it must be aware of all the waiters for locks and all the holders of locks on which other programs are waiting. To give the other lock managers this information, the Fast Path lock manager must do more than just make the last lock request. It must first request the lock for the current holder or holders of the lock. After these requests are processed, the Fast Path Lock manager then requests the lock for the new requestor. This provides the other lock manager with all of the lock information to do deadlock detection processing.

There is an exception to the use of the Fast Path lock manager. It is not used for DEDBs with block level data sharing. BLDS requires that the locks be managed across multiple IMS subsystems. BLDS requires the IRLM to manage all locks. This means that when DEDBs are registered with SHARELVL(2) or SHARELVL(3) and the IRLM is used, the Fast Path lock manager will not be used for locks on these databases.

## *DEDB Locking*

### **CI Locks**

When Data Entry Databases (DEDBs) are used, locks are obtained on CIs in the DEDB Area data sets. When a database record is entered, the CI holding the RAP is locked. When the root and direct dependents in the database record are accessed, additional CIs may have to be processed and each of these CIs is locked. CIs containing sequential dependent segments are not locked. For the CIs containing roots or direct dependent segments, PI level 1 (IRLM level 2) locks are used when the PROCOPT in the PCB does not allow updates. If updates are allowed, PI level 4 (IRLM level 8) locks are used. These locks are not released when position is moved to another database record. They are released or their level is modified at one of three other times.

1. If the Fast Path buffer stealing facility is invoked it may release a lock. The buffer stealing facility is invoked when the program has used all of its normal buffers. The facility will keep a CI locked at PI level 4 (IRLM level 8) if any of the following conditions are met.
  - The CI has been modified
  - The root of the current position is in the CI and the processing intent (PROCOPT) allows updates.
  - If delete or insert of roots is permitted and the root of the database record which precedes the current position on the RAP chain is in the CI.

The facility will keep a CI locked at PI level 1 (IRLM level 2) if it contains the root of the current position and the processing intent does not allow updates. The facility releases locks for all other CIs.

2. If the CI is not updated, the lock on the CI is released at sync point time.
3. If the CI is updated, its lock is released after it is written as part of output thread processing.

DEDB CIs are identified to the lock manager by their RBA, database number, and area number. There is also an indicator that signifies that this lock is a CI lock.

## HSSP and UOWs

When High Speed Sequential Processing (HSSP) is used, the locking for DEDBs is altered. HSSP requests do not lock individual CIs except those in independent overflow (IOVF). Instead, they lock units of work (UOWs). This substantially reduces the number of lock requests. To provide integrity, other concurrently running programs must also lock the UOWs for the DEDB Area against which HSSP is being run. So when HSSP is being used, HSSP requests lock UOWs at level 4 but do not lock CIs in the root addressable part, and other requests using the DEDB lock both CIs and UOWs in the Area. These requests lock the UOWs at PI level 1 (IRLM level 2). PI level 1 (IRLM level 2) locks on UOWs are released when the holder has released all of its locks on CIs in the UOW. HSSP's PI level 4 (IRLM level 8) UOW locks may be thought of as substitutes for locks on CIs in the UOW. These UOW locks are released when HSSP would have released all of its locks on CIs in the UOW if it had obtained locks on CIs. If any CIs have been updated, the lock on the UOW is released at sync point time.

UOW locking is also used by the preload process for DEDB VSO areas when shared VSO is not used. During the preload process, CIs are loaded into the address space by UOW and UOW locks are used for integrity. With shared VSO, CIs are loaded into the Coupling Facility structure one CI at a time and CI locking is used for integrity.

DEDB UOWs are identified to the lock manager by their database number, area number, and RBA. There is also an indicator that signifies that this is a UOW lock.

The following table summarizes DEDB locks.

Table 8. DEDB Locks Summary					
Resource			Level		Released
			PI	IRLM	
Root or DDEP CI	Update	without VSO	4	8	Output thread
		with VSO	4	8	Sync pt.
	No update	with update PCB	4	8	Buffer steal or sync pt.
		with non-update PCB	1	2	Buffer steal or sync pt.
Seq. Dep. CI			No locks		
UOW	HSSP request		4	8	Sync pt. if no updates
					Output thread if updates
	Non-HSSP request with HSSP active		1	2	When locks on CIs in UOW released
Non-HSSP request with HSSP not active		No locks			

## FLD Call Locking for DEDBs

The locking of DEDBs for FLD calls sometimes depends on the VIEW= specification in the PSB. Unfortunately, the IMS publications are incomplete in this area. They document VIEW=MSDB, but do not document VIEW=MSDBL. VIEW=MSDBL indicates that a lock should be requested during FLD call processing. VIEW=MSDB indicates that a lock should not be requested during FLD call processing. In either case, a lock is requested during sync point processing.

When a FLD call with VERIFY is used for DEDBs, the CI may be locked twice. First, if VIEW=MSDBL is specified in the PCB, the CI is locked at PI level 1 (IRLM level 2) for the duration of the call. This lock is released as part of call processing. Second, the CI is locked at PI level 3 (IRLM level 6) and then released during sync point processing. The lock during sync point processing does not depend on the VIEW= specification.

When a FLD call with both VERIFY and CHANGE is used, the CI may be locked twice. First, if VIEW=MSDBL is specified in the PCB, the CI is locked at PI level 1 (IRLM level 2) for the duration of the call. This lock is released as part of the call processing. Second, the CI is locked at PI level 4 (IRLM level 8) and then released during sync point processing. The lock during sync point processing does not depend on the VIEW= specification.

When a FLD call with CHANGE but without a VERIFY specification is used, the CI is not locked during call processing but is locked at PI level 4 (IRLM level 8) and then released during sync point processing.

Table 9. FLD Call Locking for DEDBs Summary				
CALL	VIEW	Lock Level		Lock Duration
		PI	IRLM	
FLD/VERIFY	MSDBL	1	2	During call
		3	6	During sync point
	MSDB or not specified	3	6	During sync point
FLD/VERIFY/CHANGE	MSDBL	1	2	During call
		4	8	During sync point
	MSDB or not specified	4	8	During sync point
FLD/CHANGE	MSDB, MSDBL, or not specified	4	8	During sync point

Fast Path converts some GU calls to FLD calls. This occurs when VIEW=MSDB or VIEW=MSDBL is specified in the PCB and the database is root only. If non-shared VSO is also used, there is no lock during the call with VIEW=MSDB. Locking is only done during sync point processing. Without VSO, with shared VSO, or when VIEW=MSDBL is specified, the lock is also held during the call, but released at the end of the call. In all cases the lock is held during sync point processing.

## Q Command Code and DEQ Call

The Q command code may be used with DEDBs. It causes a lock on a CI to be held until sync point or a DEQ call releases the lock. The level of the lock depends on the PROCOPT of the PCB used for the call. That means that the level of the lock for the CI is the same as it would be without the Q command code. The difference is that the lock will not be released by Fast Path's "buffer stealing" routine. Buffer stealing occurs when the program runs out of buffers. IMS examines the CIs in the programs Fast Path buffers and "steals" the buffers whose CIs are no longer required to be in the buffers. It also releases the locks on these CIs. The Q command code keeps the buffer holding a CI from being stolen and the lock on the CI from being released.

The Q command code is typically used to ensure that no change is made to a segment while a transaction is running.

The DEQ call may be issued with a DEDB PCB. This will cause IMS to release the program's DEDB locks except those for altered CIs and for CIs protecting the current root position. It will release the locks which were obtained with the Q command code unless they are altered or are protecting a current root position. Even though the call is issued with a specific Fast Path PCB, locks acquired using other Fast Path PCBs may be released.

## Segment Level Locking

Fast Path DEDBs have an exception to their normal locking when several conditions are met. This exception is called segment level locking. It is used only when the following conditions are met:

- DEDB is a root-only database
- Area uses VSO, but does not use shared VSO
- Root is defined as fixed length
- Segment edit/compression routine is not defined
- PROCOPT=G or PROCOPT=GR is used

Segment level locking is designed to provide greater concurrency for accesses to these areas. It provides concurrency that is similar to that for MSDBs. This facilitates the conversion of MSDBs to VSO DEDBs.

With segment level locking the CI is still locked, but it is locked at PI level 1 (IRLM level 2) even with PROCOPT=GR. In addition to the CI lock, the individual segment is locked. For get hold (GHU and GHN) calls, the segment lock is at PI level 4 (IRLM level 8). For get calls without hold (GU and GN), the segment lock is at PI level 1 (IRLM level 2) with one exception. When PROCOPT=GR is used and VIEW=MSDB is not specified, get calls without hold (GU and GN) lock the segment at PI level 4 (IRLM level 8).

FLD calls get the segment lock at PI level 4 (IRLM level 8) and release it as part of the call processing. This lock is acquired and released again as part of sync point processing.

When multiple CIs are read to find the root which satisfies the call, each CI is locked. With segment level locking the locks on CIs which do not contain the root satisfying the call are released at the end of the call.

## Changing Lock Ownership

When updates to DEDBs are made by an application, the ownership of the DEDB locks is changed at the end of sync point processing. The ownership is transferred from the application program (PST) to the output threads which write the updates. The locks are released at the end of output thread processing. This is done because the PST will request new locks which are associated with the next transaction that it processes. The locks from the previous unit of work cannot be released before the updates are written by output thread processing. This change of ownership will be seen in any lock traces which are done during this processing.

## MSDB Locking

The locks on Main Storage Databases (MSDBs) are on the segments. Since there are no dependent segments in an MSDB, this is equivalent to locking a database record. The locking scheme for MSDBs is different from

that for full function or DEDB databases. There is generally less locking associated with calls against MSDBs and much of the locking is postponed until sync point processing is done.

When a get hold call is used to access an MSDB, the segment is locked until sync point time. If the processing intent (PROCOPT) includes delete or replace, a PI level 3 (IRLM level 6) lock is requested. If the processing intent does not include delete or replace a PI level 1 (IRLM level 2) lock is requested.

When a get call without a hold is used to access an MSDB, the segment is locked at PI level 1 (IRLM level 2) during the call. The lock is released as part of the call processing.

When a REPL or DLET call is used for an MSDB, a PI level 4 (IRLM level 8) lock is requested during sync point processing. The segment is already locked at PI level 3 (IRLM level 6) due to the get hold call that preceded the update. The PI level 3 (IRLM level 6) lock is promoted to a PI level 4 (IRLM level 8) lock.

When an ISRT call is used for an MSDB, a PI level 3 (IRLM level 6) lock is requested. This lock is promoted to PI level 4 (IRLM level 8) during sync point processing.

FLD calls may be used with MSDBs to verify the values of fields in a segment and to change them. A FLD call may have any mixture of verify and change operations for multiple fields in one segment. In the following discussion, FLD/VERIFY is used to indicate a FLD call with verify but no change operations. FLD/CHANGE is used to indicate a FLD call with change but no verify operation. FLD/VERIFY + CHANGE is used to indicate a FLD call with both verify and change operations.

When a FLD/VERIFY call is used, the segment is locked twice. First, the segment is locked at PI level 1 (IRLM level 2) for the duration of the call. This lock is released as part of call processing. Second, the segment is locked at PI level 3 (IRLM level 6) during sync point processing.

When a FLD/VERIFY + CHANGE call is used, the segment is locked twice. First, the segment is locked at PI level 1 (IRLM level 2) for the duration of the call. This lock is released as part of the call processing. Second, the segment is locked at PI level 4 (IRLM level 8) during sync point processing.

When a FLD/CHANGE call without a VERIFY specification is used, the segment is not locked during call processing but is locked at PI level 4 (IRLM level 8) during sync point processing

MSDB segments are identified to the lock manager by the address of a control block associated with the segment and its MSDB serial number. The control block is either the segment prefix or the ECNT. The following table summarizes MSDB locks.

<i>Table 10. MSDB Locks Summary</i>				
Call Type		Lock Level		Lock Duration
		PI	IRLM	
Get Hold	without PROCOPT = R or D	1	2	From call through sync pt.
	with PROCOPT = R or D	3	6	
Get without Hold		1	2	During call
REPL		4	8	During sync pt.
DLET		4	8	During sync pt.
ISRT		3	6	From call until sync pt.
		4	8	During sync pt.
FLD/VERIFY		1	2	During call
		3	6	During sync pt.
FLD/VERIFY + CHANGE		1	2	During call
		4	8	During sync pt.
FLD/CHANGE		4	8	During sync pt.

## ***RLSE Call***

The RLSE call provides a way to release locks without sync point processing or committing any updates. When the RLSE call is used with a Fast Path PCB, all Fast Path locks which are not protecting updates are released. These locks may be for multiple Fast Path databases. Full function locks are not released when a Fast Path PCB is referenced.

## ***Area Lock***

The area lock is used to serialize several activities for Fast Path DEDB areas. These include open, close, sequential dependent (SDEP) inserts, SDEP utility processing, and many commands. The commands include /START AREA, /START DB, /STOP AREA, /STOP DB, /STOP ADS, /DBR DB, /DBR AREA, and the equivalent UPDATE commands.

## **Area Lock Summary**

**Purpose:** Used to serialize certain activities against an area

**Locking environments:** IMS subsystems with Fast Path

**Resource Locked:** An identification of the area.

**Level:** PI level 3 or IRLM level 6

**Requested:** When the operation begins

**Released:** When the operation ends

## ***Multiple Area Structure Lock***

The multiple area structure lock is used to serialize connect, disconnect, and deletion of structure entries for a multiple area structure with shared VSO. The lock is requested at the read level (ILRM level 2) when IMS connects to or disconnects from a multiple area structure. The lock is requested at the exclusive level (IRLM level 8) when IMS deletes entries from a multiple area structure.

## **Multiple Area Structure Lock Summary**

**Purpose:** Used to serialize connect, disconnect, and entry deletions for a multiple area structure

**Locking environments:** IMS BLDS systems using multiple area structures

**Resource Locked:** An identification of the structure

**Level:** IRLM levels 2 and 8

**Requested:** When the operation begins

**Released:** When the operation ends

## ***Command Lock***

The Fast Path command lock is similar to the full function command lock. It is requested by each IMS subsystem with Fast Path that is using an IRLM. It is requested when the IMS subsystem is started. The lock is requested at a share level. It is held until the subsystem terminates. The lock is not used to serialize access to anything. Instead, it is used in conjunction with notifications that are not associated with a database. For example, type-1 commands with the GLOBAL parameter for Fast Path resources are sent between IMS systems by using notifications to holders of this lock.

## **Command Lock Summary**

**Purpose:** Used for routing notifications to IMS subsystems

**Locking environments:** IMS subsystems with Fast Path that use the IRLM

**Resource Locked:** A "dummy" resource

**Level:** IRLM level 2 is always used.

**Requested:** When an IMS subsystem is started



**Released:** When an IMS subsystem is terminated.

## ***VUNLOAD Lock***

The Fast Path VUNLOAD lock is used to serialize the /VUNload command across multiple IMS systems in a Parallel Sysplex using shared VSO. The serialization is for all /VUNLOAD commands, even when they specify different areas. The lock is also requested for the /START, /STOP, and /DBR commands for shared VSO areas and for open processing for a shared VSO area. This locking also prevents deadlocks between lock processing across the sysplex and latch processing within IMS subsystems.

## **VUNLOAD Lock Summary**

**Purpose:** Used for serializing the /VUNLOAD command across multiple IMS subsystems in order to prevent deadlocks between command processing by different subsystems.

**Locking environments:** IMS subsystems with Fast Path that use the IRLM

**Resource Locked:** A "dummy" resource associated with the /VUNLOAD command

**Level:** IRLM level 8 is used for commands. IRLM level 2 is used for area open processing.

**Requested:** When an IMS subsystem processes a /VUNLOAD, /START, /STOP, or /DBR command for a shared VSO area and when a shared VSO area is opened by an IMS subsystem.

**Released:** When the command or open process completes.

## ***Buffer Overflow (OBA) Lock***

The Fast Path buffer overflow (OBA) lock is used to serialize the use of OBA buffers by a dependent region or thread. Only one region or thread may be using its overflow buffers at any time unless the 64-bit Fast Path buffer manager is used. This buffer manager was introduced in IMS Version 11 and is optional. When the 64-bit Fast Path buffer manager is used, the OBA lock is not used. The OBA lock is requested when a dependent region has used its normal buffer allocation (NBA) and requires more buffers. It is released when the dependent region no longer needs the overflow buffers.

## **Buffer Overflow (OBA) Lock Summary**

**Purpose:** Used for serializing the use of OBA buffers by a dependent region or thread. It is not used with the Fast Path 64-bit buffer manager. With the 64-bit buffer manager multiple regions and threads may be using their overflow buffer allocation concurrently.

**Locking environments:** IMS subsystems with Fast Path which are not using the 64-bit buffer manager.

**Resource Locked:** A "dummy" resource associated with the buffer overflow.

**Level:** IRLM level 8 is always used.

**Requested:** When a dependent region needs overflow buffers.

**Released:** When the dependent region or thread releases its overflow buffers.

## Special Locking Cases

There are some settings for PROCOPT parameters in PSBs which have special effects on locking. These are PROCOPT values of GO, GON, GOT, and E.

### ***PROCOPTs of GO, GON and GOT***

IMS allows a user to specify a “read without integrity” option for a database. This is done by specifying PROCOPT=GO in the PSB. When this PROCOPT is used, no locking is done for the database. GO is not valid for MSDBs. It is valid for full function databases and DEDBs. When PROCOPT=GO is used, other programs in the subsystem may be updating the database. Since they may be inserting, deleting, or replacing segments or their pointers, the lack of locking may cause data integrity problems. This is why PROCOPT=GO is called “read without integrity.”

To limit, but not eliminate the integrity exposure, additional options are available with GO for full function databases. These are GON and GOT. If GON is used and IMS recognizes a potential integrity exposure, such as an invalid pointer, it will return a ‘GG’ status code for the call instead of abending the application program. If GOT is used an additional procedure is sometimes used. This additional procedure includes the use of locks. It is only used for HDAM, PHDAM, HIDAM, and PHIDAM databases and is not used when a secondary index is being used to provide an alternate processing sequence. It also is not used when a logical relationship has been crossed so that the position is not in the same physical database with the root segment of the database record. If GOT is used and IMS recognizes an integrity exposure, the procedure checks to see if there is a lock on the database record being processed. If there is a lock, it waits until the lock is released. When any wait completes, IMS accesses the data again. This does not provide complete integrity, but somewhat decreases the probability of an integrity problem. This testing for a lock that is done with GOT is a special type of lock request that is sometimes called a test enqueue (TENQ). It is the same type of lock request that is used to test for locks on dependent segments when a full function database record is locked at PI level 1. The test enqueue waits for the lock to be released, but does not acquire the lock for the program using PROCOPT=GOT.

### ***PROCOPT of E***

IMS allows users to specify that a program is to have exclusive use of a database or segment types in the database. This is done by specifying an E in a PSB PROCOPT value. It is used in conjunction with other options, such as, G or A. E is only valid for full function databases. Specifying PROCOPT=E on a PCB statement establishes a default for SENGSEG statement PROCOPTs. PROCOPT of E has different implications with and without BLDS.

### **Without Block Level Data Sharing (BLDS)**

If BLDS is not used for a database and a PROCOPT of E is used for the root segment in a database, the program will have exclusive use of the database. No other programs with sensitivity to the database will be scheduled concurrently with this program. Since there can be no conflicting users, there is no locking for the program’s use of the database.

If E is not used for the root segment, but used for dependents, the program will have exclusive use of the specified dependents, but may share the root segments. That is, no other programs with sensitivity to the specified dependent segment will be scheduled concurrently with this program. If the program has exclusive use of a segment, no locking for the segment will be done, however, locks for database records, and other segments will be requested.

## **With Block Level Data Sharing (BLDS)**

If BLDS is used for a database, a PROCOPT of E does not provide exclusive scheduling of the database across the data sharing IMS subsystems. It only provides for exclusive scheduling within an IMS subsystem. This means that locking must always occur to provide integrity across the IMS subsystems. Database record locks are always held until application sync point or an RLSE call. This means that a PROCOPT of E in a BLDS environment does not provide exclusive use of the database or its segment types across the sysplex. Instead, it provides exclusive access to database records from when they are accessed until sync point or an RLSE call.

## ***HALDB Online Reorganization***

HALDB Online Reorganization (OLR) uses the same locks that are used by application programs. These include database record locks, block locks, and busy locks. OLR reorganizes a set of database records at a time. This is called a unit of reorganization (UOR). The database records in a UOR are locked before any copies of the segments are made to the output data sets. OLR never waits for a database record lock while holding other locks after building the minimum UOR of a single record. It does this by making conditional lock requests. If the lock request cannot be granted immediately, OLR regains control and lessens the number of database records in the UOR. Only the database records whose locks have already been obtained remain in the UOR. These database records are copied to the output data sets. At the end of the copies for a UOR, OLR commits. This releases the locks. OLR then moves to the next UOR.

The number of database records in a UOR is dynamically adjusted. OLR attempts to hold a minimal number of locks at any time and to hold them for a short time. It may be observed that OLR rarely, if ever, holds more than 1,000 locks and rarely, if ever, holds them for more than a second. This minimizes the locking impact on concurrently running application programs.

When OLR is executed in a BLDS environment, block locks are requested for the blocks in the output data sets into which segments are copied. Block locks are also requested for the input data set. The cursor is written in the second block or CI of the input data set. There is a block lock for this block or CI for every unit of reorganization. When twin backward (TB) pointers are used with PHIDAM, the twin chain between roots must be maintained across the input and output data sets. When pointers are updated in the input data set, block locks are requested for their CIs or blocks. Block locks are required since concurrently executing application programs may be making inserts or updates of other segments in these blocks.

OLR may be involved in a deadlock. When it is, it is almost always chosen as the victim. As mentioned in "Choosing a Victim" on page 36, OLR has a very low "worth" value. Only Fast Path (IFP) regions have a lower value. This value is used in determining which of the participants in a deadlock is chosen as the victim.

## Limiting the Number of Locks Held by a Program

The LOCKMAX specification may be used to limit the number of locks held by any program instance. If this limit is exceeded, the program abends with U3301. LOCKMAX may be specified in two ways. First, it may be specified on the PSBGEN statement during the PSBGEN process. Second, it may be specified as an execution time parameter for IMS dependent regions (MPP, BMP, JMP, JBP, and IFP) and IMS batch (DLI and DBB) regions. For IMS batch regions, it only applies when block level data sharing is used. If it is specified for a dependent region or IMS batch region, this specification overrides any specification on the PSBGEN statement. The only way that the PSBGEN value is used is when the LOCKMAX value for the region is not specified.

Valid values for LOCKMAX are 0 through 255. A value of 0 indicates that there is no limit to the number of locks that may be held. Other values are in units of 1000. For example, LOCKMAX=5 indicates that the limit is 5000 locks.

Users may determine the maximum number of locks held by a program by examining log records. For each application program sync point, IMS will write either a x'37', x'5937', or x'41' log record with the "high water mark" lock count for the sync interval. The value in the x'41' log record is non-zero only for IMS batch jobs using block level data sharing. The value in the x'37' or x'5937' should be used for all online regions.

The following table shows the macros which may be used to create DSECTs for these log records. The fields containing the "high water mark" log counts are shown.

<i>Table 11. Log records with lock held "high water marks"</i>		
<b>Log Record</b>	<b>Mapping macro</b>	<b>Field</b>
x'37'	DFSXFER	XFERLHLD
x'41'	DFSLOG41	LOG41LKH
x'5937'	DBFLGSYN	SYNCLKS

## PI Limit on Number of Waiters

The PI lock manager limits the number of programs that may be waiting for a lock at any time to 63. If there are 63 waiters and another program makes a lock request that would wait, it is abended with U2478.

Obviously, this should rarely, if ever, occur. For a U2478 abend, the application program is backed out. If it is an MPP or JMP transaction, it is rescheduled. This is similar to the action taken on a U0777 abend for deadlocks.

For APPC CPIC driven application programs and modified standard application programs, the U2478 abend is not issued. Instead, a U0124 abend is issued and the program is not rescheduled.

## Lock Timeouts

PI does not have a lock timeout facility. IMS lock requests never timeout when PI is the lock manager. The IRLM has a lock request timeout facility. It is controlled by the TIMEOUT value for IRLM and the LOCKTIME value for IMS.

### ***IRLM TIMEOUT Value***

The TIMEOUT value for IRLM controls the timing of IRLM time out actions. It defaults to 300 seconds but may be changed with the following command.

```
F irlmproc,SET,TIMEOUT=seconds,imssubsystemname
```

If any lock request waits for this time, the IRLM issues a DXR162I message and drives an IMS exit which creates an SMF record type 79 subtype 15 (79.15). This record contains information about the lock holders and waiters for the resource. By default, no other action is taken. The DXR162I message is:

```
DXR162I irlmx CYCLE NUMBER nnnnnnnnn PROCESSED FOR TIMEOUT.
```

The DXR162I message does not identify the waiter, the holder, or the resource with the lock conflict. These are identified in the IRLM Long Lock Detection Report which is generated from the SMF 79.15 records. Long Lock detection is explained under "RMF II ILOCK (IRLM Long Lock Detection) Report" on page 57.

### ***IMS LOCKTIME Values***

The IMS wait time for the IRLM is defined with the LOCKTIME statement in the DFSVSMxx PROCLIB member for online systems or in the DFSVSAMP DD data set for batch (DLI or DBB) jobs. The format of the statement is:

```
LOCKTIME=(mtime,maction,btime,baction)
```

Where:

*mtime* is the timeout value for MPP, JMP, and IFP regions, CCTL(CICS) and ODBA threads, or system processes. It is specified in seconds from 1 to 32767.

*maction* is either ABEND or STATUS. ABEND indicates that MPP, JMP, IFP, CCTL, and ODBA programs which timeout will abend with U3310. STATUS indicates that these programs which timeout will receive a 'BD' status code for the DL/I call. The default is ABEND.

*btime* is the timeout value for BMP, JBP, DLI, and DBB regions. If *btime* is not specified, the value for *mtime* applies to all regions.

*baction* has the same meaning as *maction* but applies to BMP, JBP, DLI, and DBB regions. If *baction* is not specified, the value for *maction* applies to all regions.

Some system processes, such as commands, ask for locks. If they time out, the process is ended. For example, a command would fail.

The IMS lock timeout capability is related to the IRLM TIMEOUT value for the IMS subsystem. The IRLM TIMEOUT value controls the timing within IRLM. When the IRLM TIMEOUT value is exceeded by a lock request, IRLM informs IMS. IMS then checks its LOCKTIME value to see if it has been exceeded. If it has

not been exceeded, the lock request is not ended. If it has been exceeded, the lock request is ended with either a 'BD' status code being returned for the call or the program being abended with a U3310. Since IMS only checks its LOCKTIME value when IRLM's TIMEOUT value has been exceeded, the IRLM TIMEOUT value must not be larger than the smaller of the two IMS LOCKTIME values. To ensure that this is true, IMS initialization processing communicates with IRLM. It tells IRLM to set its TIMEOUT value to the smaller of its two LOCKTIME values. If IMS does not have a LOCKTIME value specified in DFSVSMxx of DFSVSAMP a default value of 300 seconds is used. The IRLM TIMEOUT value must be a multiple of the IRLM local deadlock detection time. If the requested TIMEOUT value is not a multiple of the deadlock detection time, the IRLM rounds up the TIMEOUT value. This is rarely a problem since deadlock detection times are typically one second or smaller and TIMEOUT values are typically many seconds.

You should be careful when using the "F irlmproc,SET,TIMEOUT=seconds,imssubsystemname" command. The command changes the IRLM TIMEOUT value but does not cause IMS to change its LOCKTIME values. Nevertheless, it could affect the timing of lock timeouts by IMS. If the IRLM TIMEOUT value exceeds an IMS LOCKTIME value, the timeouts of lock requests will occur only after the lock has waited for the IRLM TIMEOUT value. Increasing the TIMEOUT value may cause IMS to time out lock requests later than the time indicated by its LOCKTIME values. Since the IRLM TIMEOUT value is used for the timing of Long Lock detection, the command will always set this timing.

An enhancement to IMS Versions 10 and 11 is planned. The enhancement will allow users to change the IMS LOCKTIME values for an online system with an UPDATE IMS command. The format of the command is:

```
UPDATE IMS SET(LOCKTIME(MSG(mtime),MSGOPT(maction),
                      BMP(btime),BMP(baction),
                      TELLIRLM(Y|N)))
```

The meanings of mtime, maction, btime and baction are the same as in the IMS LOCKTIME statement in the DFSVSMxx PROCLIB member as shown above. TELLIRLM(Y) indicates that the IRLM TIMEOUT value for this IMS subsystem should be set to the lower of *mtime* and *btime*.

Table 12. Lock timer values			
	IRLM DEADLOK value	IRLM TIMEOUT value	IMS LOCKTIME values
Specification	Only the first subparameter is used	Specified to IRLM for an IMS	Specified to IMS; two values; one is for transactions and the other is for batch
		Multiple of DEADLOK time	
Use	Used for deadlock detection	Used for Long Lock reporting	Used for lock wait timeout
		Used to drive IMS for lock wait timeout determination	
Action	When lock waits exceed this time, IRLM determines if they are deadlocked	When lock waits exceed this time, message DXR162I is issued and waits are reported by Long Lock but the waits continue	IMS exit is driven by IRLM when a lock has waited for the IRLM TIMEOUT value; IMS determines if it has waited for the LOCKTIME value



# Deadlocks

An overview of deadlocks was covered under “Deadlocks” on page 6. This section provides more information on deadlocks, including how they are resolved.

The overview section showed a simple example of a deadlock between two programs, but deadlocks may be more complex. They may involve more than two programs. The following examples illustrate this situation.

## Example of Deadlock between Multiple Programs

The following is an example of a deadlock involving three programs.

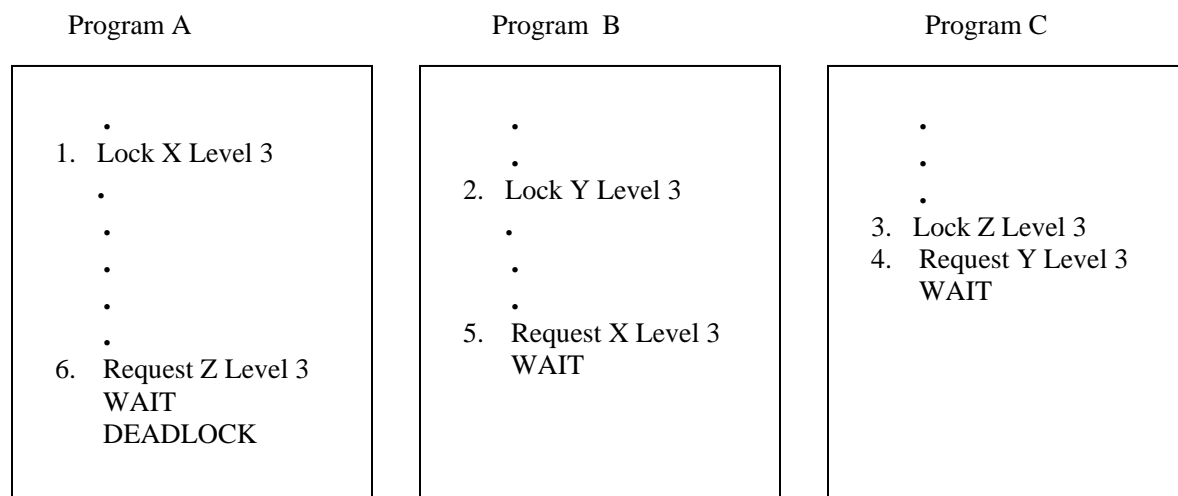


Figure 5. Deadlock Between Multiple Programs

1. Program A requests a lock on resource X at level 3. The lock is granted.
2. Program B requests a lock on resource Y at level 3. The lock is granted.
3. Program C requests a lock on resource Z at level 3. The lock is granted.
4. Program C requests a lock on resource Y at level 3. The lock request cannot be granted because program C already holds the lock at level 3. Program C must wait.
5. Program B requests a lock on resource X at level 3. The lock request cannot be granted because program A already holds the lock at level 3. Program B must wait.

A deadlock does not exist yet. Program A is not waiting. If it releases its lock on resource X, Program B may be given the lock and proceed. It could then release its lock on resource Y, which would allow program C to proceed.

On the other hand, the following could happen.

6. Program A requests a lock on resource Z at level 3. The lock request cannot be granted because program C already holds the lock at level 3. Program A must wait.

Now, all of the programs are waiting and cannot give up their locks. A deadlock exists.

## ***Deadlock Detection Timing***

The two lock managers, PI and the IRLM, use different techniques to trigger their search for deadlocks.

The PI lock manager checks for a deadlock whenever a program's lock request would cause it to wait. Before making the program wait, PI checks to see if the lock request will cause a deadlock.

The IRLM does not check for deadlocks as part of the lock request processing. It has an independent process which is triggered by a user selected time interval. This is the local deadlock cycle which is specified by the first subparameter of the DEADLOCK parameter on the IRLM execution procedure, DXJRPC. Values from 100 to 9999 are interpreted as milliseconds. Values from 1 to 99 are interpreted as seconds. The IRLM only investigates those lock requests which were also waiting at the previous interval, that is, have waited at least the local deadlock cycle time. If one is found, the IRLM will see if it is involved in a deadlock. The other requests in the deadlock do not necessarily have to have waited for the cycle time. This means that no deadlock is detected before the specified interval has elapsed. Some deadlocks may not be detected until a program has waited almost two intervals. A typical value for the local deadlock detection time is 1 second. The second subparameter of the DEADLOCK parameter is sometimes described as the number of local deadlock detection cycles in a global cycle. The IRLM does not use this subparameter. All local cycles are global cycles. If there are multiple IRLMs, they coordinate their local cycles, that is, they make them the same value. Multiple IRLMs check for deadlocks between subsystems at each deadlock cycle.

## ***Choosing a Victim***

When a deadlock occurs, the lock manager selects a program to be a victim. This means that the program's updates will be backed out and its locks released. This clears the deadlock and allows the other program or programs to continue. Either of two things may occur when the victim is an IMS dependent region. It may be terminated with a U0777 abend or an internal ROLB call may be issued and control returned to the program with status code BC or FD. CICS programs receive an ADCD abend. ODBA threads are terminated and the call receives an AIB return code, reason code, and error extension information indicating the deadlock.

In most cases IMS attempts to choose a victim whose backout will cause the least disturbance to the system. For example, in an IMS TM environment it is much easier to handle the abend of a message processing program than the abend of a non-message driven BMP. The system automatically reschedules the MPP but an operator would have to restart the BMP. For this reason, the lock managers have a scheme for choosing the victim which is based on the type of programs that are involved. The lock managers assign a worth value to each participant in the deadlock. This worth value is assigned based on the type of program. The victim is typically the program with the lowest worth value. IMS assigns worth values in the following order with the first in the list having the highest values.

1. Batch (DLI or DBB) program which has done an update or a Fast Path online utility
2. Message driven BMP with MODE=MULT
3. Non-message driven BMP or JBP

4. Message driven BMP with MODE=SNGL
5. CPI-C driven program
6. Batch (DLI or DBB) program which has not done an update
7. CICS task or ODBA thread
8. MPP or JMP with MODE=MULT
9. MPP or JMP with MODE=SNGL
10. HALDB Online Reorganization utility
11. Fast Path (IFP) program

The program types at the top of the list are least likely to be chosen as victims. For example, if a deadlock between a non-message driven BMP (item 3) and a MPP with MODE=SNGL (item 9) occurred, the MPP would be the victim. Not all items in the list apply to all environments. For example, items 2, 4, 5, 8, 9, and 11 do not apply to DBCTL subsystems.

Frequently, a deadlock will occur between two programs that are of the same type. When this occurs, the PI and IRLM lock managers have different algorithms for choosing which program will be the victim. PI chooses the one which has run the shortest time since its last sync point. IRLM chooses the one which has waited the shortest time for its lock request. There are exceptions to the IRLM algorithm. For example, in some cases the IRLM will not choose a program which is processing a message which deadlocked on its previous schedule. This is done to lessen the probability of multiple reoccurrences of the same deadlock. Similarly, if both programs are in IFP regions, the IRLM chooses the one which has run the shortest time since its last sync point.

Even when the program types differ there are exceptions to using the worth value ranking for choosing a victim. When there are multiple programs involved in the deadlock, the lock manager sometimes does not select the program with lowest worth as the victim. It does this for some complex situations. For example, selecting the program with the lowest worth may not eliminate the deadlock. Consider the following example. Program 1 and program 2 both hold a share level lock on resource A. Program 3 holds an exclusive level lock on resource B. Program 1 requests a share level lock on resource B. It waits. Program 2 requests a share level lock on resource B. It waits. Program 3 requests an exclusive level lock on resource A. This creates a deadlock. In this case, program 3 will be chosen as the victim even if its worth value is higher than that of programs 1 and 2. If either program A or program B were chosen as the victim, the deadlock would not be resolved. Instead of making both programs A and B victims, only program C is chosen to be a victim. Another instance when the ranking is not used occurs with a block lock for an insert in a KSDS. If the request for the block lock causes a deadlock, the inserting program will not be chosen as the victim.

## ***INIT STATUS GROUPB Call***

Application programs may issue an INIT call with STATUS GROUPB in the I/O area or the equivalent EXEC DLI ACCEPT STATUSGROUP ('B') command. This call indicates that the program should receive control when it is chosen as the victim in a deadlock. When the deadlock occurs, the DL/I call that creates the deadlock receives a 'BC' status code in the database PCB. IMS backs out its database resources, with the exception of GSAM, to the last commit point. If IMS is the syncpoint coordinator, DB2 database resources are backed out, any persistent MQ input messages are requeued, and any persistent MQ output messages are

backed out. If there are users of the External Subsystem Interface, their updates are also backed out. IMS Transaction Manager output messages issued since the last commit point are backed out unless they were inserted and purged using an EXPRESS=YES PCB. Input messages are returned to the message queue for MPPs, JMPs, and message-driven BMPs. For IFP regions all input messages are returned to balancing group queues.

## ***Handling a Deadlock Victim***

The following explains what happens to the victim in a deadlock when it has not issued the INIT STATUS GROUPB call.

When a victim is chosen, all updates are backed out. The victim program is either abended or it is given an indication that the back out has occurred. The abend is either an IMS U0777 abend for MPP, JMP, IFP, BMP, or JBP regions, a U0123 abend for CPIC driven application programs or modified standard application programs, or a CICS ADCD abend for CICS tasks. An FD status code is returned only when Fast Path databases are being processed by a victim that is a non-message driven BMP or JBP. This is discussed below. A U0777 or U0123 abend will cause a back out of the program to its last sync point. A CICS ADCD abend will cause a back out of the program to its last sync point if dynamic transaction back out (DTB) is specified. This will back out non-IMS resources, such as CICS file control data sets. IMS database updates are always backed out when deadlocks occur. If the program has not created a sync point, the back out is to the beginning of the program's execution.

When an ODBA thread is the victim of a deadlock, the database updates are backed out and the DL/I call completes with AIB "system failure" return code of x'00000108', a "thread termination" reason code of x'00000244' and error extension code x'10000309'. The AIB error extension code is the hexadecimal value for the 777 abend code with the high order bit on. The ODBA thread is terminated. The application program can no longer make calls on the thread.

For IMS TM systems, the application program may have processed one or more input messages since its last sync point. If this is true, the input message or messages for MPP, JMP, or BMP regions are returned to the queue. Multiple input messages will have been processed only if MODE=MULT is used. For a Fast Path region, the input message is retained in its buffer. MPP, JMP, and IFP region transactions are rescheduled automatically unless they are CPIC driven application programs or modified standard application programs. No message is sent to the terminal operator and the operator is unlikely to be aware of the backout and rescheduling.

BMPs, JBPs, and Fast Path utilities must be rescheduled if they are abended.

If HALDB Online Reorganization is the victim of a deadlock, it is automatically restarted by IMS.

When the victim is a CICS task an ADCD abend is created. The task may be restarted. For restart to be done, the installation must specify that it wants both dynamic transaction backout (DTB) in the CICS system and transaction restart for the transaction. Transaction restart is specified by the RESTART parameter when defining the transaction to CICS.

As was mentioned above, not all victims receive abends. There are three exceptions.

The first exception is a program which issues an INIT STATUS GROUPB call. This is described above under "INIT STATUS GROUPB Call" on page 37.

The second exception is for a non-message driven BMP or JBP which has a PCB that references a Fast Path resource. It may create a deadlock situation with a call to either a Fast Path or full function database. If the BMP or JBP is chosen as the victim, it is not abended. Instead, an internal ROLB call is processed to backout the program's processing and the call that caused the deadlock receives an FD status code. The application program is allowed to handle the situation. The program may continue processing or choose to abend itself. Generally, it is preferable to continue since this will avoid the need to restart the job. The call that receives the FD status code does not have to use a Fast Path PCB. If any PCB in the PSB references a DEDB or MSDB, the non-message driven BMP or JBP will get the FD status code instead of the U0777 abend.

The third exception is for deadlocks found during sync point processing. This must involve MSDB resources because locks for DEDBs and full function databases are not requested during sync points. This situation does not apply to CICS tasks or ODBA threads because they do not have access to MSDBs. When deadlocks are found during sync point processing, a U0777 abend is not issued. Instead, an internal ROLB call is issued to back out any updates and release locks. If the program is a BMP or JBP, an FD status code is returned to the call that caused the sync point. This could be a GU to IO PCB, a SYNC call, or a CHKP call. For message driven BMPs, any input messages processed since the last sync point are returned to the queue. BMPs remain scheduled and may retrieve any input messages again. Other types of programs do not get the FD status code. For MPPs and JMPs the input message or messages are returned to the queue. For an IFP region, the input message is retained in its buffer. MPP, JMP, and IFP region input messages are reprocessed automatically. The MPP, JMP, or Fast Path program remains scheduled and will receive the same or another input message as a result of the GU to the IO PCB that caused the sync point. In either case, the situation does not require that the program take any special action.

The following table summarizes the action taken on a deadlock victim when an INIT STATUS GROUPB call has not been issued.

<i>Table 13. Deadlock Actions without INIT STATUS GROUPB call</i>		
<b>Program Type</b>	<b>Deadlock in Sync Point? <sup>1</sup></b>	<b>Action <sup>2</sup></b>
MPP or JMP	Y or N	U0777, input message is reprocessed on reschedule
Message driven BMP	N	U0777, input message is reprocessed when BMP is rescheduled. BMP rescheduling is not automatic.
	Y	Original input message is returned to the GU IO-PCB call which caused the deadlock
Non-message driven BMP or JBP without FP PCB in PSB	N	U0777
Non-message driven BMP or JBP with FP PCB in PSB	Y or N	FD status code
Fast Path (IFP)	Y or N	U0777, input message is reprocessed automatically
CICS DBCTL	N	ADCD, backout and retry depend on CICS specifications
ODBA thread	N	AIB return code, reason code, and error extension codes are set. Thread is terminated and the program can no longer make calls on the thread.

Notes:

1. This can only occur with MSDBs. MSDBs are not supported with CICS or ODBA.
2. The deadlock action always includes a backout

## ***Deadlocks with CICS Resources***

CICS application programs may access CICS resources, such as VSAM data sets, that are not controlled by IMS. CICS uses enqueueing mechanisms to protect these resources. These include both CICS enqueueing mechanisms, which it sometimes calls exclusive control, and VSAM's own exclusive control facilities.

For more information on CICS enqueueing facilities and their use with CICS resources, see the *CICS Recovery and Restart Guide* and the *CICS Application Programming Guide* for the version of CICS you are using.

It is possible for a deadlock to occur which includes both IMS and CICS resources. The following is an example of such a deadlock.

1. Program A requests a lock on IMS resource X. The lock is granted.
2. Program B requests a VSAM record Y, causing an enqueue of Y. The request is granted.
3. Program B requests a lock on IMS resource X. The lock request cannot be granted because program A already holds the lock at an incompatible level. Program B must wait.
4. Program A requests a VSAM record Y, causing an enqueue Y. The request cannot be granted because program B has a conflicting enqueue for Y. Program A must wait.

A deadlock has occurred but IMS's lock manager cannot detect it. It is not aware of the enqueue conflict for the VSAM record. Similarly, CICS enqueueing facility is not aware of the locks on IMS resources; however, the deadlock may be broken by a lock timeout. If the IRLM is the lock manager and the LOCKTIME parameter is used, the IMS lock request may be timed out by IMS. See "Lock Timeouts" on page 33 for an explanation of this time out capability.

CICS can also handle these situations by using a timeout facility. If a CICS task is suspended for longer than a user specified time, the task is timed out and abended. One can view this as CICS deciding that a deadlock must exist and choosing the waiter as a deadlock victim. In the example above, only Program A is waiting on a CICS resource, so it would be abended. This would cause Program A to release its lock on IMS resource X and allow Program B to continue. The time that the program is allowed to wait is specified in the DTIMOUT value for each transaction. This should be specified for all CICS transactions using both IMS and CICS recoverable resources. This should include the mirror tasks and CECI transactions. If DTIMOUT is not specified and the IRLM TIMEOUT facility is not used, there is no timeout facility for the transaction and a deadlock would persist until a system operator abended one of the waiting tasks.

## ***Deadlocks with DB2 Resources***

Deadlocks involving both IMS and DB2 resources may occur. This is similar to the situation with IMS and CICS resources that is discussed above. The following is an example of a deadlock involving IMS and DB2. It could occur in either an IMS TM, CICS, or DBCTL environment.

1. Program A requests a lock on IMS resource X. The lock is granted.
2. Program B requests a lock on DB2 resource Y. The lock is granted.
3. Program B requests a lock on IMS resource X. The lock request cannot be granted because program A already holds the lock at an incompatible level. Program B must wait.

4. Program A requests a lock on DB2 resource Y. The lock request cannot be granted because program B already holds the lock at an incompatible level. Program A must wait.

A deadlock has occurred but the lock managers cannot detect it. IMS and DB2 always use different lock managers. Either IMS uses PI and DB2 uses IRLM, or IMS and DB2 use different IRLMs.

As mentioned above in the "Lock Timeouts" section on page 33, a lock request for an IMS DL/I call may timeout. This would resolve the deadlock. Similarly, DB2 also has a lock timeout function. DB2 lock requests always have a timeout value. Any DB2 lock request that waits for longer than the specified time is assumed to be in deadlock. In our case, Program A would be the selected victim if the DB2 request times out. The DB2 wait time for the IRLM can be defined on the DB2 DSNTIPI installation panel. The default value is 60 seconds. This means that no deadlock between IMS and DB2 resources would be broken by a DB2 timeout before 60 seconds has elapsed.

When a program's DB2 lock request times out, the actions taken depend on the type of program that issued the SQL call. MPPs, JMPs, message driven BMPs, and Fast Path (IFP) programs receive U0777 abends which cause a back out to the last sync point. The input message or messages for a MPP, JMP, or BMP are returned to the queue. For a Fast Path (IFP) region transactions are rescheduled automatically. A message driven BMP must be restarted by an operator. For non-message driven BMPs and JBPs, DB2 updates are backed out, an internal ROLB call is issued to back out the IMS updates, and the SQL call receives a -911 return code. The application program continues after receiving the -911. If the DB2 request that times out was issued by a CICS task, the SQL call receives either a -911 or -913 return code. In either case, the CICS program continues after receiving the return code. A -913 indicates that the SQL statement was unsuccessful but no backout of previous SQL or IMS calls was done. A -911 indicates that all DB2 and IMS updates have been backed out to the last sync point. The program may continue processing after either a -911 or -913 is returned. The setting of the ROLBI parameter in DB2's CICS Resource Control Table (RCT) determines whether the -911 or -913 will be used. If the DB2 request that times out was issued by an ODBA thread, such as a DB2 stored procedure, backout is not done automatically and the SQL call receives a -913 return code. The program should invoke a rollback as soon as possible using SRRBACK or ATRBACK.



## Retained Locks and Lock Rejects

Retained locks are locks held for a failed subsystem. Retained locks are still known to the lock manager, but the holders of these locks are no longer active. The failed subsystem must be restarted before the locks can be released. Requestors of these locks at an incompatible level do not wait. This is reasonable since a wait would likely be for a very long time, possibly many minutes. Instead of waiting these lock requests are rejected.

Retained locks may occur in a BLDS environment. If one of the IMS subsystems fails, its IRLM fails, or the LPAR on which it is running fails, its locks are retained. If only the IMS subsystem fails, its IRLM maintains its locks in a retained status. If the IRLM fails or the LPAR fails, the other IRLMs in the data sharing group maintain the locks in a retained status. With BLDS the coupling facility lock structure contains information about all locks protecting updates. This information is kept in the lock structure record list. If an IRLM fails, the record list for its locks is copied by the other IRLMs in the data sharing group into their storage. This ensures that the locks are always stored in at least two locations. If a lock request is processed and it is incompatible with one of these retained locks, the lock request is rejected.

Retained locks may also occur with DBCTL services using either PI or the IRLM. If the CCTL subsystem, such as CICS, fails, it may be holding locks in an in-doubt status. IMS does not know if CICS will commit or abort, so it cannot back out the updates and release the locks. Neither can it commit the updates and release the locks. IMS maintains the locks in a retained status. Requests for these locks at an incompatible level are rejected. When CICS is restarted and reconnected to IMS, the in-doubt units of work are resolved. They are either committed or aborted. These actions cause the locks to be released.

The effect of a lock reject depends on the use of the INIT STATUS GROUPx call or the EXEC DLI ACCEPT STATUSGROUP('x') command. If either of these is used, lock rejects result in a BA or BB status for the call requesting the lock. The BB status code indicates that updates made since the last commit are backed out. The BA indicates that only the current call is backed out. In either case, the call fails and the program may take other actions. If neither INIT STATUS nor EXEC DLI ACCEPT is used, the application program receiving the lock reject is abended. This is a U3303 abend. For IMS message processing programs, the input message is placed on the suspend queue with the following exceptions. For IMS transactions involved in protected conversations with RRS, the input message is discarded. For APPC CPIC driven application programs and modified standard application programs, the U3303 abend is not issued. Instead, a U0125 abend is issued and the transaction is not rescheduled. For CICS applications, the IMS U3303 abend results in an ADCI abend code.

When a U3303 abend occurs it is accompanied by a DFS3304I message.

```
DFS3304I  IRLM LOCK REQUEST REJECTED. PSB=psb_name DBD=dbd_name
        JOBNAME=job_name RGN=nnn SUBSYSTEM=subsystem
```

<i>psb_name</i>	The name of the PSB issuing the failed call
<i>dbd_name</i>	The name of the database with the retained lock
<i>job_name</i>	The name of the z/OS job receiving the 3303 abend
<i>nnn</i>	The number (decimal) of the PST receiving the 3303 abend
<i>subsystem</i>	The name of the IMS subsystem that holds the lock



## Database Level Data Sharing

Database level data sharing is used when a database is registered to DBRC with SHARELVL(1). SHARELVL(1) ensures that if an IMS subsystem has authorization to update a database no other IMS subsystem will be authorized to update it or to read it with integrity. Authorizations to read without integrity are allowed. SHARELVL(1) also allows multiple IMS subsystems to have concurrent authorizations to read the database with integrity when no subsystem has update authority. Locks do not have to be shared between the subsystems in these situations. Of course, locks are required within an IMS online subsystem.

Even though locks between subsystems are not required for databases using database level data sharing, they can be used. In fact, when the subsystems are using IRLMs in the same data sharing group, a small subset of locks are used. These are data set reference locks and extend locks. These are used so that when an updating IMS subsystem extends a database data set, the other subsystems will receive the new extent information. This avoids abends where the systems using read without integrity attempt to read blocks or CIs in the new extents. The locks do not directly provide the new extent information to the reading subsystems, but they are used in the process. The following explains how this process works.

When the IRLM is used with a SHARELVL of 1, 2, or 3, the database reference lock is acquired when a database data set is opened. When a new extent is created for a database data set, the extending subsystem uses the IRLMs to send information about the new extents to all other holders of the data set reference lock. The information sent is the set of control blocks for the new extent. The receiving subsystems add these control blocks. This allows them to read the blocks or CIs in the new extent. This does not provide full integrity to the subsystems. Instead, it reduces the probability of abends by the readers when the updating subsystem extends a data set.

There is a second advantage to using the IRLM with database level data sharing. In a Parallel Sysplex data sharing environment the use of the IRLM also allows the use of IMS's cache structures in the Coupling Facilities. These structures are used to implement buffer invalidations between IMS subsystems. These invalidations are used to limit the chance that a system reading without integrity will use an old copy of a block or CI which has been updated to a different state in another IMS subsystem. This does not provide full integrity to the subsystems. Instead, it reduces the probability of reading uncommitted updates or following incorrect pointers to invalid data.

## Virtual Storage Use for Locks

Both the PI lock manager and the IRLM keep their information which represents locks in control blocks. This section explains the effects that these control blocks have on the use of virtual storage.

If the storage available for locks is exhausted, the lock manager will abend the first program to request a lock that would exceed the limit. Usually, this will occur when a “runaway” program causes many lock requests to occur without creating a sync point; however, the program receiving the abend will not necessarily be the “runaway” program. It may be any other program in the system that happens to make the request that causes the limit to be exceeded.

For MPP, JMP, IFP, BMP, and JBP regions, if the virtual storage available for locks is exhausted, the application program will be abended and backed out. With the PI lock manager this will be a U0775 abend. With the IRLM it will be a U3300 abend. With either lock manager, CICS tasks receive an ADLA transaction abend and are backed out if dynamic transaction backout (DTB) is specified. In all cases the IMS updates are backed out. MPP, JMP, and message-driven BMP input messages are reprocessed unless a CPIC driven application program or modified standard application program receives the abend.

### ***PI Lock Manager Virtual Storage***

For IMS TM and DBCTL environments, the PI lock manager keeps its control blocks above the 16 megabyte line in the DL/I address space or in ECSA. The DL/I address space is used unless Fast Path databases are included in the system. When Fast Path databases are in the system, all PI control blocks, including those for full function databases are in ECSA.

The maximum amount of virtual storage that may be used for PI control blocks is determined by the PIMAX execution parameter. If this parameter is not specified, the second parameter specified for CORE= on the system definition IMSCTF macro determines the maximum storage for PI control blocks. The specification is in 1K blocks. Of course, the storage may also be limited by the virtual storage available in the region or ECSA.

Each PI lock requires about 48 bytes of virtual storage.

### ***IRLM Virtual Storage***

The IRLM keeps most of its lock control blocks above the 2-gigabyte bar in 64-bit storage of its address space. Each IRLM lock requires about 540 bytes of virtual storage. The space for these control blocks may be limited by the z/OS MEMLIMIT parameter for the job or job step.

## Design Advice

This section provides general advice for designing databases, application programs, and systems to avoid locking problems.

### ***Minimize PROCOPT values***

PROCOPT specifications in PSBs affect potential processing concurrency. For this reason, PROCOPT values should be minimized, that is, PROCOPTs allowing updates should not be specified when updates will not be done. The PROCOPT value determines the level at which full function database record locks and Fast Path DEDB CI locks are requested. These locks may be shared by multiple programs which specify PROCOPT=G. If an update PROCOPT is used, the locks may not be shared. If no updates are going to be done, the PROCOPT should not include update values. This will allow greater concurrency in the system.

Sometimes it may be advisable to generate a PSB with two PCBs for the same database. One PCB would have PROCOPT=G and the other would have PROCOPT=A. This is useful when a program reads frequently used segment occurrences and sometimes updates these or other segment occurrences. If the PCB with PROCOPT=G is used for the reads, the program may share the segment with other concurrently running programs. The PCB with PROCOPT=A may be used for the processing which requires updates. This technique is not required for all programs because most programs do not have lock conflict problems. On the other hand, this technique is useful in addressing some lock conflict problems.

### ***Frequent Sync Points***

Since locks have the potential to prevent other programs from accessing data, systems need to be designed so that locks are not held for excessive lengths of time. Typically, this requires frequent sync points to release the locks. The time between sync points usually varies by program type and these are discussed below.

Holding locks is usually not a problem for online transactions. They usually run for a short time, a second or less, and then reach a sync point. This releases their locks and allows other programs to access the data that they had locked. On the other hand, there are some types of online transactions which may cause problems.

CICS online transactions may use conversational programming which allows the program to wait on input from a terminal user while holding locks. These waits are typically for a much longer time. They may last for 15 seconds, 30 seconds, or even several minutes. Usually, CICS conversational programming is unacceptable when locks are held across communications with terminal users.

Some online transactions do so much processing that they run for minutes. If they hold locks for this amount of time, they may cause problems. Sometimes these programs may be broken into multiple units of work or sync intervals. In IMS TM environments this is usually done by having a transaction do part of the work, do a program-to-program message switch, and then create a sync point (GU to IO-PCB). The next program then continues the work. This next program actually may be another execution of the same program. In CICS environments, the application program may use a SYNCPOINT command or TERM call to terminate one PSB and then schedule another PSB. In either the IMS TM or CICS case, breaking the work into multiple units of work reduces the length of the time any lock is held. Of course, the application designer must consider recovery requirements for the work that is broken into multiple pieces. If the system fails or an application abends while one of these processes is active, the designer must ensure that it is recoverable and restartable.

Batch jobs are usually the ones that cause locking problems because of the lengths of time that they hold locks. These may be either BMPs, JBPs, or ODBA threads, such as DB2 stored procedures which perform batch processing. These batch jobs usually require frequent checkpoints to create sync points. The required frequency of the sync points will vary between jobs. There is a trade-off between the lack of concurrency from holding locks and the overhead of taking checkpoints. Sometimes it is difficult to predict the required frequency because the installation may not be able to predict data reference patterns or because the use of the data may vary from time to time. Since this required frequency is difficult to predict, it is usually desirable to make the checkpoint frequently easily modifiable. This may be done by having the program read the desired checkpoint frequency from a control data set or database record when it begins.

Most installations base checkpoint frequency on elapsed time or the amount of processing done. Processing is usually measured by the number of segments or database records which are read or updated. Checkpoints must be done at a point from which a program may be restarted. Usually, batch programs go through a processing iteration and see if a checkpoint should be taken. That is, they check the elapsed time since the last checkpoint or the number of database records or segments processed. If a checkpoint is not yet needed, another iteration is made and the check repeated. This process is repeated until a checkpoint is taken and the timers or counters are reset.

Almost all batch jobs require sync points, but there can be exceptions. If a batch job does not hold any individual lock for a long time, it probably does not require sync points. Locks for updates are held until a sync point is reached, so update jobs almost always require sync points. Locks that are associated with position in a full function database record are released either at a sync point or when position is moved to another database record. If a batch job does no updates and does not hold position on any database record for a long time, it probably does not require sync points. Remember that IMS can maintain a position for each PCB in the PSB, so programs that use a lot of PCBs usually hold positions on several dataset records.

## Communications within a Sync Interval

Communication flows within a sync interval may delay the sync point. This could prevent the implementation of frequent sync points.

OTMA and APPC may delay sync points while waiting on remote partners to respond to a sync point request. This occurs with OTMA using commit mode 1 (send-then-commit) or with APPC when either uses synclevel=confirm or synclevel=syncpt. In these cases locks are not released until the response from the remote partner is received and processed. Communications delays may cause locking problems in these environments. If locking is a potential problem, it may be advisable to use synclevel=none to avoid these problems.

The use of synchronous callout which was introduced in IMS Version 10 is another way that communications may occur within a sync interval. Synchronous callout allows IMS applications to invoke services from outside the IMS system. The application program waits within its sync interval for the response. Obviously, a slow response could cause locks to be held a long time. Synchronous callout is invoked with the DL/I ICAL call. Users may specify a timeout value for the call. If the response is not received within the timeout value, the callout request is terminated and control is returned to the application program with a return code of x'0100' and reason code of x'0104' in the AIB control block. The application program can then do any further processing and commit or back out. The default timeout value is 10 seconds, but the application may set any other value in the 4-byte field. Times are specified in units of 0.01 seconds. All ICALs should set the appropriate time out value and include logic to handle the timeout.

## ***Frequently Updated Records***

Frequently updated records often create locking problems. When a segment is updated, the lock protecting the update is held until a sync point is reached. This means that no other program may access the data. For high volume systems it is important to avoid such “single thread” access. There are several designs that may lead to frequent updated segments. Two of them are particularly significant.

The first design is one that uses control segments. A control segment is one that contains control information that many programs access. If these programs update the data or access it using a PCB with update processing intent, it is likely to cause a problem. An example of this kind of segment is one which contains the next sequential number to assign, such as an invoice number. As each new invoice is processed, the program gets the number and increments it. In such a system, only one program could be assigning numbers at any time. There are several techniques to avoid or minimize the locking conflict. First, it may be sufficient to delay the retrieval and update of the number until the program is about to create a sync point. This will minimize the time that the lock is held and allow more transactions to process in a time period. Second, it may be necessary to use several series of numbers. For example, there may be several "next invoice numbers" stored in different database records. Different types of invoices would get their numbers from the different database records. Third, it may be necessary not to use sequential numbers. A randomizing technique may be used to choose a number. If a duplicate is created, another number would have to be chosen but with large enough numbers this could be minimized. Of course, a combination of these techniques may also be used. Besides sequential numbers control segments may have totals from different processes or from other segments. A segment might keep totals from all terminal operations in the system. Such a segment is very likely to cause a locking problem. It is more preferable to keep totals for each operator in separate segments and combine them when a grand total is needed.

A second design that may lead to frequently updated records is a database with few roots. When a program is positioned in a database record while using an update PCB, no other program may be positioned in the record. When there are few records, this often leads to contention for database records. When a program has updated a segment and moved to another database record, other programs may not be able to process updated database record. If the IRLM is being used, no other program may enter the database record before the updater reaches a sync point. If the PI lock manager is being used, there may be more concurrency but updated segments or pointers may prevent programs from accessing other segments in the database record. When HDAM is being used, few RAPs have the same effect as few roots because IMS locks RAPs, not roots, to lock database records. Database administrators should be aware of the danger of implementing databases with few database records or HDAM RAPs.

## ***Using the RLSE Call***

The solution to some locking problems requires that a program give up its lock on a full function database record or Fast Path CI. This will allow other programs to access the record. It may be desirable to give up the lock without requiring the original program to reach a sync point. Of course, the original program cannot have updated the database record or CI because this would cause the lock to be held until a sync point is reached. To give up the database record lock or the CI lock the program may issue a RLSE call. The RLSE call uses a database PCB. For full function PCBs the call releases the database record lock held for the position of this PCB when there are no uncommitted updates in the record. For Fast Path PCBs the call releases all Fast Path locks for unmodified data. These locks may be for multiple Fast Path databases.

## ***Using a "Get Lost" Technique***

Before the introduction of the RLSE call in IMS Version 9, some installations used a "get lost" technique to release locks held by a program. To give up the database record lock, the program either used the PCB to get position on another database record or issued a call which resulted in no lock being held. This type of movement off of a database record is sometimes called a "Get Lost" technique. If a program needs to move from a database record and not cause further lock conflicts it may move to a database record that will not be accessed by any other program. This generally requires that a special root segment be reserved for this program. The root may be associated with a user or terminal operator to ensure that no two concurrently executing programs try to "get lost" on the same database record. If there are no such reserved roots, a special technique may be used. This special call is a get unique (GU) call with a segment search argument (SSA) that requests a root with a key equal to high values (X'FF...'). This will result in no lock being held for a database record for this PCB. A 'GE' status code will be returned on the call. With the introduction of the RLSE call, the get lost is not required.

## ***Using PROCOPT=E to Avoid Checkpointing***

An explanation of PROCOPT=E appears under "PROCOPT of E" on page 29. Specifying PROCOPT=E for a root segment avoids locking for the database. This may eliminate the need for checkpoints in a special circumstance.

Some installations have batch IMS jobs with no checkpoints. They would like to run these jobs as BMPs under either IMS TM or DBCTL. This would give them two advantages. First, logging for the job would go to the online system's log instead of a separate log for the batch job. The installation could produce and manage fewer logs when using BMPs. Second, dynamic backout would be invoked for all jobs abends, not the subset that is backed out when BKO=Y is specified for a batch job. This can simplify operating procedures. On the other hand, BMPs usually need frequent checkpoints to release locks. Locks need to be released for two reasons.

1. Other programs may want access to the resources that are locked. When PROCOPT=E is used, this access will not be granted because other programs that could access the resources will not be scheduled concurrently with the PROCOPT=E program. This means that PROCOPT=E cannot be used, with or without checkpoints, if concurrent use of the database is desired.
2. Locks need to be released to free virtual storage. Even though this storage is above the line, it is limited. If the limit is exceeded, the application program will be abended. An IMS U0775 or U3300 abend or CICS ADLA abend will occur. This is explained under "Virtual Storage Use for Locks" on page 44. Since PROCOPT=E avoids locking, checkpoints are not needed to free this virtual storage. PROCOPT=E may be used to avoid these abends when checkpoints are not taken.

In summary, PROCOPT=E may be used to run batch jobs as BMPs without adding checkpoints to release locks if concurrent access to the database by other programs is not needed. The use of PROCOPT=E eliminates locking and, therefore, the abends that would occur if virtual storage limits for locks were exceeded.

If the implementation of block level data sharing is planned, be careful about the use of PROCOPT=E to avoid locking. When block level data sharing is used, locks are acquired for the database. PROCOPT=E forces exclusive use of the database only in the IMS system where the PROCOPT=E job executes. Other systems still have access to the database. When multiple systems share a database, locks are required to provide integrity across the systems.

## ***Deadlock Detection***

When a deadlock occurs, some time is required to detect and handle the deadlock. Since the detection time is part of the time required to process a lock request, this may affect response times in a system. The importance of this depends on the frequency of deadlocks in the system.

The choice of lock manager and the use of DB2 or CICS resources may effect this detection time. If the PI lock manager is used and no DB2 or CICS resources are involved, the deadlock will be detected as soon as a call creates it. The deadlock may be broken and resolved before a noticeable effect occurs. Deadlock detection is different with the IRLM. The IRLM only detects deadlocks after they have existed for at least the deadlock detection cycle. The minimum cycle time is 100 milliseconds and maximum is 99 seconds. A deadlock detection cycle time of a few seconds or more may make deadlocks more noticeable. Typically, installations use IRLM deadlock detection cycle times of one second or less. If DB2 or CICS resources are involved with IMS resources, a deadlock is only broken by a timeout in CICS or DB2. The time specified for the timeout cannot be so short that it would lead to timeouts when no problem exists; therefore, almost any deadlock between IMS and either DB2 or CICS resources will have a noticeable effect on response times.



# Locking Traces and Report Programs

IMS has traces and report programs that may be used to monitor locking activity. This section discusses those traces and the information in the report programs. Instructions for executing the report programs are included in the *IMS System Utilities* publication.

## Lock Resource Names

When IMS requests a lock from PI or the IRLM, it requests the lock on a resource name. This is a string of bytes that uniquely identifies the resource and type of lock. For example, a lock on a database record in a HIDAM database would include the RBA of the root segment and the identity of the database and data set in which it resides. These lock resource names often appear in lock traces and lock reports. The following explains the resource names used for the different types of locks.

## IMS subsystem ID with IRLM local locking

When the IRLM is used for local locking, the subsystem ID is added to lock resource name. Local locking is locking without data sharing. Adding the subsystem ID allows an IRLM to service multiple IMS systems where not all databases are shared. Locks only need to protect resources within an IMS subsystem. Adding the subsystem ID to the lock resource name prevents lock conflicts between local locks within different IMS subsystems which otherwise would have the same resource name.

## Full function locks

In full function lock resource names the DMB# identifies a database. When the database is registered and the IRLM is used, the DMB# is the global DMB# from the RECON database record. When PI is used or when the IRLM is used but the database is not registered, the DMB# is the local DMB#. This is the relative number of the DDIR control block for the database in the IMS online system.

In lock resource names the DCB# identifies the data set within the database. It is one byte. For a full function non-HALDB database this is the same number that is used to assign database data sets to buffer subpools in the DFSVSMxx member. The DCB# is 1 for primary indexes, unique secondary indexes, the primary data set for non-unique secondary indexes and the data set containing root segments for HISAM databases. The DCB# is 2 for overflow data sets in non-unique secondary indexes and HISAM databases. The DCB# is 1 for the data set containing HDAM or PHIDAM roots. If there are multiple data set groups for HDAM or HIDAM, successive data set groups use the successive DCB numbers. HALDB uses different schemes for different database types. The DCB# for PSINDEX database data sets is always 1. The following tables show the DCB# that is associated with each data set in PHDAM and PHIDAM databases. When the DDNAME letter is Y or M through V, the DCB# also has the x'80' bit turned on. For KSDSs the RBA address has the low order bit on.

Table 14. DCB numbers used with PHDAM databases												
DDNAME letter	A/M	L		B/N	C/O	D/P	E/Q	F/R	G/S	H/T	I/U	J/V
DCB#	1	2 <sup>1</sup>	3 <sup>2</sup>	4	5	6	7	8	9	10	11	12

1. Index component

2. Data component



Table 15. DCB numbers used with PHIDAM databases														
DDNAME letter	A/M	L		X/Y		B/N	C/O	D/P	E/Q	F/R	G/S	H/T	I/U	J/V
DCB#	1	2 <sup>1</sup>	3 <sup>2</sup>	4 <sup>1</sup>	5 <sup>2</sup>	6	7	8	9	10	11	12	13	14

1. Index component

2. Data component

Lock resource names for HALDB include the partition ID number when the IRLM is used with SHARELEVEL 2 or 3 databases. This is added to the end of the lock resource name. The partition ID is needed to create a unique resource name since multiple partitions in the same database could have identical DCB numbers and RBA values. The partition ID is not used with PI or with SHARELVL 0 or 1 databases since each partition has its own DDIR control block and the relative number of the DDIR is used instead of a global DMB#.

## Lock Resource Name Formats

PI always uses eight-byte lock resource names. Lock resource name lengths vary with the IRLM. The lock resource names used with the IRLM begin with a one-byte length field. The length includes the one-byte length field. This length field is not shown in the formats which follow.

### Database Record Lock

for HDAM or PHDAM using OSAM or ESDS

RBA of RAP	DMB# <sup>2</sup>	DCB#	C'P' <sup>3</sup>	<sup>1</sup>
------------	-------------------	------	-------------------	--------------

for HIDAM or PHIDAM using OSAM or ESDS

RBA of root segment	DMB# <sup>2</sup>	DCB#	C'P' <sup>3</sup>	<sup>1</sup>
---------------------	-------------------	------	-------------------	--------------

for KSDS

Hashed value of root key	DMB# <sup>2</sup>	DCB#	C'P' <sup>3</sup>	<sup>1</sup>
--------------------------	-------------------	------	-------------------	--------------

### Segment Lock

for HDAM or PHDAM dependent segments and HIDAM or PHIDAM segments

RBA of segment	DMB# <sup>2</sup>	DCB#	X'40'	<sup>1</sup>
----------------	-------------------	------	-------	--------------

for HDAM or PHDAM root segments

RBA of RAP	DMB# <sup>2</sup>	DCB#	X'40'	<sup>1</sup>
------------	-------------------	------	-------	--------------

for KSDS

Hashed value of root key	DMB# <sup>2</sup>	DCB#	X'40'	<sup>1</sup>
--------------------------	-------------------	------	-------	--------------

for HISAM ESDS

RRN of the logical record	DMB# <sup>2</sup>	DCB#	X'40'	<sup>1</sup>
---------------------------	-------------------	------	-------	--------------

## Block Lock

RBA of block or CI	DMB# <sup>2</sup>	DCB#	C'P' <sup>3</sup>	<sup>1</sup>
--------------------	-------------------	------	-------------------	--------------

## Busy Lock

X'FF'	C'ZID'	DMB# <sup>2</sup>	DCB#	C'B' <sup>3</sup>	<sup>1</sup>
-------	--------	-------------------	------	-------------------	--------------

## Extend Lock

X'FF'	C'XID'	DMB# <sup>2</sup>	DCB#	C'P'	<sup>1</sup>
-------	--------	-------------------	------	------	--------------

## Data Set Reference Lock

X'FF'	C'DID'	DMB# <sup>2</sup>	DCB#	C'B'	<sup>1</sup>
-------	--------	-------------------	------	------	--------------

## Command Lock

X'FF'	C'CID'	X'7FFF'	X'FF'	C'B'
-------	--------	---------	-------	------

### Notes:

1. For SHAREVL 2 and 3 databases if the IRLM is used and the database is HALDB, the partition ID (2 bytes) is added to the end of the lock resource name.
2. If the database is not registered or if PI is used, the DMB# is the local number as determined by the definitions in the IMS subsystem and the high order bit is off. If the IRLM is used and the database is registered in the RECONs, the DMB# is the global number from the RECONs and has the high order bit turned on. There is one exception to this. If the database is HALDB and registered at SHAREVL 0 or 1, the DMB# is the local number as determined by the definition in the IMS subsystem and the high order bit is off.
3. This is a blank (x'40) with local locking. Local locking is used with the PI lock manager and with the IRLM when the database is not registered with a SHAREVL of 2 or 3.

## Fast Path Locks

In lock resource names the area identification includes either the global DMCB# or the DMAC number. The global DMCB# is used when the database is registered and the IRLM is used. When the DEDB is not registered or PI is used, the DMAC number is used. This is a number assigned by the IMS system.

## Fast Path CI Lock

X'00'	RBA <sup>1</sup>	Area identification <sup>2</sup>	C'F'
-------	------------------	----------------------------------	------

### Fast Path Segment Lock<sup>3</sup>

B'10'	Low order 30 bits of CI RBA	Area identification <sup>2</sup>	C'F'
-------	-----------------------------	----------------------------------	------

### Fast Path UOW Lock

X'E4'	RBA of first CI in UOW <sup>1</sup>	Area identification <sup>2</sup>	C'F'
-------	-------------------------------------	----------------------------------	------

### Fast Path Area Lock with IRLM

X'F8'	Database Name (8 bytes)	Area Name (8 bytes)	C'F'
-------	-------------------------	---------------------	------

### Fast Path Area Lock with PI

X'F8'	C'AID'	Area identification <sup>2</sup>	C'F'
-------	--------	----------------------------------	------

### Fast Path Buffer Overflow (OBA) Lock<sup>3</sup>

X'F2'	Address of ESCDMOBU <sup>4</sup>	X'FFFF'	C'F'
-------	----------------------------------	---------	------

### Fast Path MSDB Lock<sup>3</sup>

X'F1'	Address of MSDB Control Word <sup>5</sup>	MSDB Serial #	C'F'
-------	---	---------------	------

### Fast Path VUNLOAD Lock

X'F9'	C'VUNLOAD	' (16 bytes)	C'F'
-------	-----------	--------------	------

### Fast Path Multiple Area Structure Lock

X'FA'	Structure Name (16 bytes)	C'F'
-------	---------------------------	------

### Fast Path Command Lock

X'FF'	C' CID'	x'7FFFFFF'	C'F'
-------	---------	------------	------

#### Notes:

1. This is the high order 3 bytes of the CI RBA
2. With PI the area identification is a 2-byte DMAC number and a byte of x'00'. With IRLM the area identification is the 2-byte DMCB number followed by the area number. The area number is one byte

when the number of areas in the database is 240 or less. Otherwise, it is two bytes and the C'F' suffix follows the two bytes.

3. When the IRLM is used, the 2-byte subsystem ID is added to the end of the resource name. This prevents conflicts between locks in different IMS subsystems. This applies to systems with and without data sharing since these resources cannot participate in data sharing.
4. This is the address of the ESCDMOBU. The content of ESCDMOBU identifies the current owner of the OBA lock.
5. The MSDB Control Word is an address which uniquely identifies a record in an MSDB.

## ***IMS Monitor Trace***

The IMS Monitor Trace is used to collect information that is reported by the IMS Monitor report program. It collects information on waits for locks when either PI or the IRLM is used as the lock manager.

The IMS Monitor trace is available in either an IMS TM or DBCTL environment. It is turned on with either of the following commands.

1. /TRACE SET ON MONITOR ALL
2. /TRACE SET ON MONITOR APDS

The trace information is collected on the IMS Monitor data set which is processed by the IMS Monitor report program (DFSUTR20).

## ***PI and Lock Traces***

IMS has two closely associated traces of lock information. They are the PI trace and the lock trace. These traces are written in the same table in virtual storage and optionally written to the IMS log or an external trace data set. All events traced by the PI trace are also traced by the lock trace. The additional events in the lock trace appear when the IRLM is used. This includes information about waits for locks. The PI trace is written to the log or trace data set when any of the first three commands in the following list are issued in an IMS TM or DBCTL environment.

The first two commands are equivalent since LOG is the default when OPTION is used. ALL adds wait times to the PI trace. The keyword TIME may be used after OPTION. It requests that the wait time be included in the trace record, but it does not cause the trace to be written to the log or external trace data set. The fourth command causes the lock trace to be written to the log or external trace data set. It includes tracing of wait times.

1. /TRACE SET ON PI OPTION
2. /TRACE SET ON PI OPTION LOG
3. /TRACE SET ON PI OPTION ALL
4. /TRACE SET ON TABLE LOCK OPTION LOG

The trace records written by the PI and lock traces are the following.

- Written by PI and lock trace.
  - X'CA' – Entry for a DLI call. Used with PI lock manager and IRLM.
  - X'CA' – Lock request using PI lock manager
  - X'CB' – Lock request waited with PI lock manager.
  - X'C7' – Deadlock.
  - X'C8' – Lock request using IRLM. Entry into lock manager.
  - X'C9' – Lock request using IRLM. Exit from lock manager.
  - X'CC' – Exit from lock request handler. Used with PI and IRLM.
- Written only by lock trace.
  - X'C8' – Lock request suspended by IRLM. Beginning of wait.
  - X'C8' – Lock request resumed by IRLM. End of wait
  - X'D0' – Data sharing notify sent.
  - X'D2' – IRLM status exit driven

The lock trace information is described in the IMS Diagnosis publications under "DL/I Trace." These publications are *IMS Version 9 Diagnosis and Reference*, *IMS Version 10 Diagnosis Reference*, and *IMS Version 11 Diagnosis*. CSECTs for these and other trace records may be generated by assembling the following macro from IMS's MACLIB (SDFS MAC).

IDLIVSAM TRACENT

There is no tracing by the Fast Path Lock Manager. Locks for Fast Path resources are only traced when a wait is required and another lock manager is called.

## ***IMS Monitor (DFSUTR20)***

The IMS monitor is available in IMS TM and DBCTL environments. The IMS Monitor reports are created from IMS Monitor trace records. For lock IWAITs, IMS writes a record when the wait begins and another when the wait ends. The report uses these records to report on the elapsed time for these waits. Lock wait times are shown in two of the IMS Monitor reports, the Program I/O report and the Region IWAIT report. Deadlocks are shown in the Deadlock Event Summary section following the heading "\*\*\*Reports\*\*\*".

The program I/O report shows IWAITs by PCB within programs. The programs are identified by their PSBs. IWAITs are identified by the characters "PI" in the DDN/FUNC column. "PI" is used even when IRLM is the lock manager. This is followed by the physical database's DBD name and the segment code for the segment being processed. If PI waits occur for different segments, there will be a line in the report for each segment. The number of waits, the mean wait times, and the maximum wait time are reported for each segment in each

PCB for the program. This report is an excellent source for use in determining if locking is a problem for a program.

The following is an example of the reporting of two lock waits for database RZCMA001.

<u>PSBNAME</u>	<u>PCB NAME</u>	<u>IWAITS</u>	<u>TOTAL</u>	<u>MEAN</u>	<u>MAXIMUM</u>	<u>DDN/FUNC</u>	<u>MODULE</u>
	RZCMA001	2	3419	1709	1991	PI RZCMA001...	1

The Region IWAIT report shows IWAITS by region. It accumulates the IWAITS for all programs that execute in the region. It is similar to the Program I/O report. Waits for locks are identified by the characters "PI" in the FUNCTION column under the DL/I CALLS section of the report. As in the program I/O report, "PI" is used even when IRLM is the lock manager. The characters "PI" are followed by the physical database's DBD name and the segment code for the segment being processed. The number of waits, the mean wait times, and the maximum wait time are reported for each segment in the region.

The following is an example of the reporting of waits for locks for databases SMWLJ001 and RZCMA001 in region 45.

```

IMS MONITOR    *** REGION IWAIT ***
                .....IWAIT TIME.....
**REGION      45 OCCURRENCES      TOTAL      MEAN      MAXIMUM      FUNCTION  MODULE
DL/I CALLS
                16      20959      1309      4696      PI=SMWLJ001...1
                19      48901      2573      26494     PI=RZCMA001...1

```

The Deadlock Event Summary report may be used to determine how long it took to backout a program when it was the victim in one or more deadlocks. The elapsed time for the backouts is listed in the MEAN ELAPSED TIME column for the call with level code '00' and a blank status code. This is the call for which the victim in the deadlock was waiting. The level code of '00' is used to indicate the call by the victim that caused the deadlock.

## Reporting of Waits for Space Management

Occasionally the report program will include lines which appear to be lock IWAITS but actually are not lock waits. Instead, these are waits for space management which are handled by a latch mechanism, not locking. When one of these latch waits occurs, IMS writes lock IWAIT monitor records; therefore, these latch waits are reported as if they were lock IWAITS. You can recognize these waits in the report since they include a zero for the segment number in the report. For example, you might see a line such as:

<u>**REGION</u>	<u>15 OCCURRENCES</u>	<u>TOTAL</u>	<u>MEAN</u>	<u>MAXIMUM</u>	<u>FUNCTION</u>	<u>MODULE</u>
<u>DL/I CALLS</u>	1	172	172	172	PI=RZCMA001...	0

These space management latch waits never cause deadlocks since the holder never asks for a lock or another space management latch while holding one of these latches.

## PI Trace (DFSPIRP0)

The Program Isolation Trace Report Utility is used to report waits for locks when the PI trace has been written to an IMS log or external trace data set for a system using the PI lock manager. Although the trace will include records for every request made to the PI lock manager, the report will only include those requests that had waits. For each of these requests, the report lists the following:

- Resource requested. This is the DBD name, DCB number, and 4-byte resource ID. The 4-byte ID is either the RBA, RRN, or hashed key.
- Time of the request.
- Elapsed time of wait except for Fast Path resources.
- Names of requesting and holding PSBs

The report is sorted by database, data set, and resource ID. The report also includes the total number of waits for each resource.

This report is particularly useful for finding locking problems when the PI lock manager is used.

## ***RMF II ILOCK (IRLM Long Lock Detection) Report***

The RMF II ILOCK report is available when IRLM is used as the lock manager. It uses information from SMF records to gather information about lock requests that have waited longer than the IRLM TIMEOUT value. When these "long locks" occur, the report lists the holder of the locks and the waiters. The IRLM TIMEOUT value defaults to 300 seconds but may be set to another value with the following command:

```
F irlmproc,SET,TIMEOUT=seconds,imssubsystemname
```

See "Lock Timeouts" on page 33 for information on using this command and its relationship to IMS lock timeouts.

The ILOCK report requires the writing of SMF type 79 subtype 15 records. These may be specified with the following z/OS command:

```
S  RMF , , , ( SMFBUF ( RECTYPE ( 79 ( 15 ) ) ) )
```

The SMF records are written by IMS when its timeout exit routine is driven by IRLM. IRLM drives this exit routine when a lock request has waited longer than the number of seconds specified for its TIMEOUT value. This value is set during IMS initialization. IMS provides its LOCKTIME value from the DFSVSMxx PROCLIB member for online subsystems or the DFSVSAMP DD data set for batch jobs. If IMS does not have a LOCKTIME value specified in DFSVSMxx or DFSVSAMP, a default value of 300 seconds is used. The TIMEOUT value may be modified with the F irlmproc,SET,TIMEOUT=time,imssubsystemname command.

Reporting is generated by issuing the RMF Monitor II ILOCK ALL command.

In the ILOCK report a lock holder or waiter is identified as one of the following:

**BLOCKER:** This is a program that holds a lock for which another program is waiting.

**TOP BLOCKER:** This is a BLOCKER which is not waiting on a lock.

**WAITER:** This is a program that is waiting for a lock.

BLOCKER/WAITER: This is a program that is both a blocker and a waiter.

The following is a sample report.

RMF - ILOCK IRLM Long Lock Detection							Line 1 of 15
Command ==>							Scroll ==> HALF
CPU= 37/ 35 UIC=2540 PR= 0							System= RMF5 Total
State	Type	Lock_Name	PSB_Name	Elap_Time	CICS_ID		
	IMS_ID	Recovery_Token	PST# Trx/Job	Wait_Time	DB/Area		
-----							
CF Structure ACOXLOCK at 07/28/2006 13:02:10 Deadlock Cycle 00002EC7							
-----							
TOP	BMP	09C943CFA7800101D700000000000000	DFSSAMB1	00:06:04			
BLOCKER	ACO3	ACO3 0000000300000000	0006 IRLMTOPZ				
-----							
TOP	BMP	09C3614505800101D700000000000000	DFSSAMB1	00:06:09			
BLOCKER	ACO1	ACO1 0000000600000000	0006 IRLMTOPA				
-----							
WAITER	BMP	09C3614505800101D700000000000000	DFSSAMB2				
	ACO2	ACO2 0000000800000000	0007 IRLMWT1	00:05:52	DI21PART		
-----							
WAITER	BMP	09C943CFA7800101D700000000000000	DFSSAMB7				
	ACO2	ACO2 0000000900000000	0008 IRLMWT2	00:05:42	DI21PART		
-----							

The ILOCK report is documented in the *z/OS RMF Report Analysis* and the *z/OS RMF User's Guide* publications.

## ***KBLA Deadlock Trace Record Analysis Report (DFSKTDL0)***

The KBLA (Knowledge Based Log Analysis) Deadlock Trace Record Analysis Report utility (DFSKTDL0) formats and summarizes data extracted from IMS x'67FF' log records which are written when deadlocks occur. It produces a summary report, a victim report, and a detail report. This utility does not read trace records, it only processes x'67FF' log records which do not require tracing.

The summary report includes the number of deadlocks on the log and summaries of deadlocks by hour, IMS system, state, lock type, database, PSB, lock name, and RBA. This information is useful in understanding if deadlocks are a problem and identifying the high volume deadlocks and their cause.

The victim report shows the participants in each deadlock and which participant was chosen as the victim.

The detail report provides more detailed information about each deadlock. This includes the holders of locks, the locked resources, the levels of locks, database names, PST numbers, and IMS calls that produced the lock requests.

## ***KBLA IRLM Lock Trace Analysis Utilities (DFSKLTX0)***

The KBLA (Knowledge Based Log Analysis) Lock Trace Analysis utilities may be used to report on lock waits when using the IRLM. As the name implies, these utilities produce reports by processing IRLM lock



traces. There are three utilities. The first is the KBLA Lock Analysis Control File Creation utility (DFSCLTA0). It reads the RECONs and produces a control file. This file is used to match global DMB numbers in the lock trace to their database names so that the names may be used in the report. The second utility is the KBLA Lock Trace Analysis Reduction utility (DFSCLTB0) utility. It reads the lock trace records and produces an intermediate file. This file contains data on lock requests which resulted in waits as well as detailed information on all lock requests. This file is read by the KBLA Lock Trace Detailed Print Program (DFSCLTC0) to produce its reports. These utilities may be invoked either through the KBLA IRLM Lock Analysis panel or by creating and running your own JCL.

DFSCLTC0 has options to limit reporting to a subset of databases, PSTs (dependent regions or threads), or to a minimum wait time. There are three standard output reports. These reports list the output in database name order, wait time order, and request completion order. The execution of these utilities is documented in the *IMS System Utilities* manual.

The following is a sample summary report:

Suspended IRLM Lock Requests Summary Report - Wait Time Order Page 001							
Trace Date = 01/12/2005 Trace Start Time = 16:01:47 Trace End Time = 16:06:26							
Trace Elapsed Time (secs) = 278							
Trace Input DSN = IMS.ISA1.DFSTRA01							
Database Name	DS Id	Lock Req Count	Wait Count	Not Int Count	Total Time	Average Time	Maximum Time
BFLMSGY3	01	8628	115	110	9.198	0.079	2.76
BFLMSGY7	01	8452	102	98	4.813	0.047	4.36
BFLMSGP	01	15862	181	169	4.401	0.024	0.64
BFLSUMP	01	3929	40	37	3.703	0.092	2.39
BCMTLRD	09	1153	1	1	3.400	3.400	3.40

The wait count includes internal latch waits. The "Not Int Count" column is the count of "not internal" waits. These are lock waits and the count does not include internal latch waits.

The following is a sample detailed report:

Suspended IRLM Lock Requests Report - Req Comp Order														Page 0043
Trace Date = 01/12/2005 DSN = IMS.ISA1.DFSTRA01														
Lock Request	Start Time	End Time	Elapsed	Type	Num	Type	DB	DS	RBA/HASH	S	Flag	--IRLM--	Call	Trace
												RCFB	TRAC	Time Seq#
16:06:09.723	16:06:09.724	0.000	F	100	BIDP	4	BCICINYL	01	099DE001	P	CPR	0000	08C0	ISRT 001 16:06:09.690 0975
16:06:09.727	16:06:09.727	0.004	F	100	BIDP	4	BCICINYL	01	099DE001	P	CPKF	0000	08C0	ISRT 001 16:06:09.690 0C98
16:06:09.567	16:06:09.952	0.385	G	067	FPCI	8	BCWTRMD	08	00024CE0	F	K	0440	08F0	F073
16:06:10.170	16:06:10.170	0.004	G	067	BIDP	4	BAGTX1P	01	32117800	P	CPKF	0840	08F0	ISRT 001 16:06:10.170 8B69
16:06:10.209	16:06:10.242	0.032	G	100	FPCI	8	BGLACAD	06	005203A0	F	K	0440	08F0	9A67
16:06:10.354	16:06:10.354	0.004	L	122	FPCI	8	BCWRDAD	10	00053AE0	F	K	0440	2080	D030
16:06:10.397	16:06:10.398	0.001	L	122	FPCI	8	BCWRDAD	11	00143820	F	K	0440	2080	DFDE
16:06:10.438	16:06:10.438	0.000	L	122	FPCI	8	BCWRDAD	13	0009E000	F	K	0440	2080	EB9D
16:06:10.959	16:06:10.992	0.032	L	038	BIDP	6	BCWTRPP	01	0412E804	P	PKF	0000	2080	ISRT 001 16:06:10.959 BBD8
16:06:11.011	16:06:11.012	0.001	L	122	FPCI	8	BCWRDAD	11	00168360	F	K	0440	2080	D79D

## ***File Select and Formatting Print Utility (DFSERA10)***

This File Select and Formatting Print Utility (DFSERA10) is used to print and produce reports from IMS logs and external trace data sets. There are several exit modules that are used to produce particular reports. This section includes information on three of these modules which produce reports of locking information.

## **Record Format and Print Module (DFSERA30)**

The Record Format and Print Module (DFSERA30) is one of the exits used with the File Select and Formatting Print Utility (DFSERA10). This module has the capability to produce a Deadlock Analysis report. The report is produced from the X'67FF' log record that is created when a U0777 abend occurs. The log record is created with both the PI lock manager and the IRLM. This log record contains information about resources and lock requestors involved in the deadlock. For each resource involved in the deadlock, the report includes the database name, the lock's resource ID and the root segment's key if it is available. For the requestor waiting on the resource, the report includes the following

- Job name or transaction code
- PSB name
- Region type (MPP, BMP, etc.)
- Type of call made (GET, ISRT, etc.)
- Lock request function (used to identify database record locks, dependent segment locks, etc.)
- Lock level (called state in the report)

For the requestor holding the lock on the resource, the report includes the following

- Job name or transaction code
- PSB name
- Region type (MPP, BMP, etc.)
- Lock level (called state in the report)

This information makes it easy to discover the programs and resources involved in deadlocks.

The report is usable with any log containing a U0777 abend. The X'67FF' log record from which the report is generated is created whether or not any traces are on.

The following is a sample deadlock report. In this report the deadlock has occurred because transaction TRLDCC1 in IMS2 is waiting on a block lock in database CMLDDCDB at RBA 7EB22000. This lock is currently held by transaction USMEED2 in IMS1. At the same time transaction USMEED2 is waiting on the

database record lock in the same database at RBA 7EB22B3E. This lock is held by transaction TRLDDC1. The database record locks are held and requested at IRLM level 6.

```
*****
DEADLOCK ANALYSIS REPORT - LOCK MANAGER IS IRLM
.....
RESOURCE DMB-NAME LOCK-LEN LOCK-NAME          - WAITER FOR THIS RESOURCE IS VICTIM
01 OF 02 CMLDDCDB      08      7EB22000843A01D7
KEY FOR RESOURCE IS FROM DELETE WORK AREA
KEY=(200414913326180)
      IMS-NAME TRAN/JOB PSB-NAME PCB--DBD  PST# RGN  CALL LOCK  LOCKFUNC STATE
WAITER   IMS2      TRLDDC1  CMLDDCDB CMLDDCDB 00003 MPP  DLET GBIDP 22400318 04-P
HOLDER  IMS1      USMEED2  CMLDDCDB ----- 00007 MPP  ----  ----- 04-P
.....
RESOURCE DMB-NAME LOCK-LEN LOCK-NAME
02 OF 02 CMLDDCDB      08      7EB22B3E843A01D7
KEY IS ROOT KEY OF DATA BASE RECORD ASSOCIATED WITH LOCK
KEY=(200414913326180)
      IMS-NAME TRAN/JOB PSB-NAME PCB--DBD  PST# RGN  CALL LOCK  LOCKFUNC STATE
WAITER   IMS1      USMEED2  CMLDDCDB CMLDDCDB 00007 MPP  GET  GRIDX 30400358 06-P
HOLDER  IMS2      TRLDDC1  CMLDDCDB ----- 00003 MPP  ----  ----- 06-P
.....
DEADLOCK ANALYSIS REPORT - END OF REPORT
*****
```

More detailed information about the report is included in the description of the Record Format and Print Module (DFSERA30) in the *IMS System Utilities* publication.

## PI Trace Record Format and Print Module (DFSERA40)

The PI Trace Record Format and Print Module (DFSERA40) is one of the exits used with the File Select and Formatting Print Utility (DFSERA10). It formats the trace records produced by the PI and lock traces.

When the PI lock manager is used, a typical sequence for a DLI call with two locks requests, the second of which must wait, would be the following.

1. X'CA' – Entry for a DLI call.
2. X'CA' – Lock request.
3. X'CC' – Exit from lock request handler.
4. X'CA' – Lock request.
5. X'CB' – Lock request. Request waited.
6. X'CC' – Exit from lock request handler

If the IRLM were used, the same call and lock requests would produce the following records.

1. X'CA' – Entry for DLI call.

2. X'C8' – Lock request entry into lock manager.
3. X'C9' – Lock request exit from lock manager.
4. X'CC' – Exit from lock request handler.
5. X'C8' – Lock request entry into lock manager.
6. X'C8' – Lock request entry suspended.
7. X'C8' – Lock request resumed.
8. X'C9' – Lock request exit from lock manager.
9. X'CC' – Exit from lock request handler.

The printing of X'CC' trace records by DFSERA40 indicates a lock request function in the column headed "ACT". These request functions include GRIDX, RRIDX, GSEGL, and similar five character designations. The following is a summary of these functions.

The format is 'abbbc' where:

- a – The first character is usually one of the following:

**G** Get – a lock

**R** Release – release a lock

**T** Test – test a lock

- bbb – The middle three characters are usually one of the following:

**RID** database record lock

**SEG** segment lock

**ZID** data set busy lock (open, close, extend)

**QCM** Q command code lock

**TLK** test a lock (wait if conflict, but do not lock)

**ALL** all locks (used to release all locks)

**BID** block or CI lock (used only by data sharing)

**XID** data set extension (used only by data sharing)

- c – The last character is usually one of the following:

<b>L</b>	local lock (used for local locking; with data sharing only used for Fast Path)
<b>X</b>	local lock and global lock (used for both local locking and data sharing)
<b>B</b>	global lock (used for data sharing, but not local locking)
<b>P</b>	global lock (used for data sharing, but not local locking)
<b>U</b>	This is a combination of two requests used with GRID. It indicates a request to get a new root lock (GRIDX) and then release the old root lock (RRIDX)
<b>W</b>	This is a special case of X. It is used with RRID to indicate that the lock and global alternate lock should be released. The alternate lock is associated with a previously locked database record, not the last one locked. It is held while other database records are locked and then examined in attempting to satisfy a call

Using this information, we can interpret the following common function requests.

<b>GRIDX</b>	get a lock (local lock and data sharing) on a database record
<b>RRIDX</b>	release a lock (local lock and data sharing) on a database record
<b>GRIDU</b>	get a lock (local lock and data sharing) on a database record and release the lock on the previous database record
<b>RRIDW</b>	release a lock (local lock and data sharing) on the alternate database record, that is, not the last one locked.
<b>GSEGL</b>	get a PI lock on a dependent segment
<b>RSEGL</b>	release a PI lock on a dependent segment
<b>GQCMX</b>	get a Q command code lock (local lock and data sharing) on a segment
<b>GFPLL</b>	get a Fast Path lock
<b>RFPLL</b>	release a Fast Path lock
<b>TTLKL</b>	test a PI lock
<b>GZIDL</b>	get a PI data set busy lock
<b>RZIDL</b>	release a PI data set busy lock
<b>RALLX</b>	release all locks (sync point processing)

More information on the lock request function may be found by assembling the following macro from IMS's MACLIB (SDFS MAC).

DFSLR FUNC = HELP

## IMS Trace Table Record Format and Print Module (DFSERA60)

The IMS Trace Table Record Format and Print Module (DFSERA60) is one of the exits with the File Select and Formatting Print Utility (DFSERA10). It writes all trace records that are on the log or external trace data set. It does minimal formatting of these records and produces a "dump-like" output. For the X'CC' records it decodes some of the fields in the record to list the lock request type, whether the lock request is conditional or unconditional, and the lock level. The lock request types, such as GRIDX, are the same as are documented for the DFSERA40 module in the *IMS System Utilities* publication. The lock levels used are the following:

- E – PI level 4
- U – PI level 3
- S – PI level 2
- R – PI level 1

Since DFSERA60 does minimal formatting of the trace records, descriptions of the records are usually needed when working with the output this module. They are described in the *IMS Diagnosis* publication under "DL/I Trace". DSECTs for these trace records may be generated by assembling the following macro from IMS's MACLIB.

IDLIVSAM TRACENT

## Trace and Report Matrix

The following table summarizes the programs that produce reports on locking activity.

Table 16. Locking Trace and Report Programs							
Report Program	Trace Required	Lock Mgr.		Information Reported			
		PI	IRLM	Lock Requests	Waits	Dead-Locks	FP
IMS Monitor	Monitor	Y	Y	N	Y	Y	Y
DFSPIRP0	PI or Lock	Y	N	N	Y	N	N
RMF II ILOCK	SMF	N	Y	N	Y	N	Y
DFSERA30	None	Y	Y	N	N	Y	Y
DFSERA40	PI or Lock	Y	Y	N	N	Y	Y
DFSERA60	PI or Lock	Y	Y	Y	Y	Y	Y
DFSKLTC0	Lock	N	Y	Y	Y	N	Y
DFSKTDL0	None	Y	Y	Y*	Y*	Y	Y

\* Only for locks which are participants in deadlocks.

## IMS Performance Analyzer for z/OS Reports

The IMS Performance Analyzer for z/OS is an IBM product which produces performance reports from IMS logs and IMS Monitor data sets. Several of these reports include locking information. Three sample reports are shown below. Information about all of the available reports and their contents is available in the *IMS Performance Analyzer for z/OS Report Reference* publication.

### Deadlock Summary

The IMS Performance Analyzer for z/OS Deadlock Summary report provides a summary of deadlocks by database. It may be used to identify databases with a high incidence of deadlocks. The following is an example of the Deadlock Summary report.

```
Start 21May2006 16.24.57.06          IMS Performance Analyzer          End 21May2006 16.30.22.96  Page
3

                        Deadlock Summary

***** Losing Program *****
DMB-name  IMS-name  Tran/Job  PSB-name  PCB--DBD  Deadlocks
-----
DBD01P    IMD3      MKR#LK1A  FUNPSB01  DBD01P      1

***** Winning Program *****
DMB-name  IMS-name  Tran/Job  PSB-name  PCB--DBD  # Waits
-----
DBD01P    IMD3      MKR#LK1B  FUNPSB01  DBD01P      1
DBD01P    IMD3      MKR#LK1C  FUNPSB01  DBD01P      1
DBD01P    IMD3      MKR#LK1D  FUNPSB01  DBD01P      1

DBD01P    IMD3      MKR#LK1D  FUNPSB01  DBD01P      1
DBD01P    IMD3      MKR#LK1B  FUNPSB01  DBD01P      1
DBD01P    IMD3      MKR#LK1C  FUNPSB01  DBD01P      1

Total number of Deadlocks =          2
```

### Deadlock List

The IMS Performance Analyzer for z/OS Deadlock List report is similar to the Deadlock Analysis report created by DFSERA30 as shown on page 62. The Deadlock List report enhances this report to include explanations of some of the data in the report. The following is an example of this report.



Start 21May2006 16.24.57.06 IMS Performance Analyzer Page 1

Pseudoabend record Abend No = U0777 Time 16:30:22.73 Date 21May2006 Recno = 00000000CDF1CEA3

**Deadlock List**

Deadlock Analysis Report - Lock Manager is IRLM

Resource DMB-name Lock-len Lock-name

01 of 02 VIB2201 08 00008C40815E01C6 (RBA = 008C4000, DMCB# = 815E, AN = 01, F-Lock)

Key for resource is not available

IMS-name	Tran/Job	PSB-name	PCB--DBD	PST#	RGN	Call	Lock	State	Lockfunc
Blocker IMSBXRF	VIT228	VIP228		00103	MPP			08 (Excl)	
Waiter IMSBXRF	VIT228	VIP228		00018	MPP	ISRT	GFPLL	08 (Excl)	904004F0 Func=Get FP Lock Mode=Uncond State=Excl Flag=Get, Fast Path, Local

Resource DMB-name Lock-len Lock-name \*\* Waiter for this resource is VICTIM \*\*

02 of 02 AWB02P 10 0242EAE8C809C01400002 (RBA = 0042EAE8C, DMB# = 809C, DCB = 01, Local, SSID = 0002)

Locking on HDAM anchor, key displayed is HDAM key requested...

000000 01310790 C6D4F0F0 F840 \*.....FM008 \*

IMS-name	Tran/Job	PSB-name	PCB--DBD	PST#	RGN	Call	Lock	State	Lockfunc
Blocker IMSBXRF	VIT228	VIP228		00018	MPP			06-P (Update, Pri)	
Waiter IMSBXRF	VIT228	VIP228		AWB02P	00018	MPP	GET	GRIDX 06-P (Update, Pri)	30400378 Func=Get Local and Global Root Locks Mode=Uncond State=Update Flag=Get, Root Lock

## Fast Path DEDB Resource Contention Summary

The IMS Performance Analyzer for z/OS includes a Fast Path DEDB Resource Contention Summary report. This report provides summary information about IWAITs on DEDB locks of various types, including CI, UOW, segment level, area, buffer overflow, MSDB and command locks.

The following is an example of this report.

Report from 09Jun2006 14.25.56.36			IMS 8.1.0		IMS Performance Analyzer 4.1			Report to 09Jun2006 14.30.06.71						
Fast Path DEDB Resource Contention Summary														
From 09Jun2006 14.26.11.74 To 09Jun2006 14.29.21.57			Elapsed=		0 Hrs		3 Mins		09.836.240 Secs					
**** CI Lock IWAIT ****			Sharing Types:											
Area Sharing		Elap/Count		Max IWAIT		Pct Tot		Pct Tot		A : Area / Non Level Share				
Name Type		Counts	Sc.Mil.Mic	StDev	Sc.Mil.Mic	Counts	Counts	IW Elp	B : 1 IRLM Block Level Share					
									C : 2 IRLM Block Level Share					
DB23AR0 A		3	3.313	0.466	5.498	9.09%		0.05%						
DB23AR1 A		4	2.222	0.551	3.386	12.12%		0.04%						
DB23AR3 A		1	4.871.974	0.000	4.871.974	3.03%		24.50%						
DB23AR4 A		1	0.257	0.000	0.257	3.03%		0.00%						
DB23AR5 A		11	1.358.286	1.620	4.981.761	33.33%		75.15%						
DD01AR0 A		13	3.880	0.499	6.863	39.39%		0.25%						
** Total		33	602.504	2.668	4.981.761	100.00%		100.00%						
**** Area Lock IWAIT ****			Sharing Types:											
Area Sharing		Elap/Count		Max IWAIT		Pct Tot		Pct Tot		A : Area / Non Level Share				
Name Type		Counts	Sc.Mil.Mic	StDev	Sc.Mil.Mic	Counts	Counts	IW Elp	B : 1 IRLM Block Level Share					
									C : 2 IRLM Block Level Share					
BANKC00 C		11	18.813	0.129	22.795	39.29%		15.18%						
BANKC01 C		17	68.036	2.828	837.022	60.71%		84.82%						
** Total		28	48.699	3.118	837.022	100.00%		100.00%						
**** CI Lock IWAIT ****														
		Average SD/Avg Max-Value		Average SD/Avg Max-Value		Average SD/Avg Max-Value		Average SD/Avg Max-Value						
		3.313 .471 5.498		2.222 .556 3.386		4.871.974 .005 4.871.974		0.257 .005 0.257						
Range		Count in	Areaname=DB23AR0		Count in	Areaname=DB23AR1		Count in	Areaname=DB23AR3		Count in	Areaname=DB23AR4		
Sc Mil Mic		Range	Share Type=A		Range	Share Type=A		Range	Share Type=A		Range	Share Type=A		
To Maximum		0			0			1	*****		0			
256.000		0			0			0			0			
128.000		0			0			0			0			
64.000		0			0			0			0			
32.000		0			0			0			0			
16.000		0			0			0			0			
8.000		1	*****		0			0			0			
4.000		2	*****		2	*****		0			0			
2.000		0			1	*****		0			0			
1.000		0			1	*****		0			1	*****		
Total=		3	10	20	30	40	50%	4	10	20	30	40	50%	
		Average SD/Avg Max-Value		Average SD/Avg Max-Value		Average SD/Avg Max-Value		Average SD/Avg Max-Value						
		1.358.286 1.624 4.981.761		3.880 .503 6.863		602.504 2.673 4.981.761								
Range		Count in	Areaname=DB23AR5		Count in	Areaname=DD01AR0		Count in	Areaname=** Total					
Sc Mil Mic		Range	Share Type=A		Range	Share Type=A		Range	Share Type=A					
To Maximum		3	*****		0			4	****					
256.000		0			0			0						
128.000		0			0			0						
64.000		1	****		0			1	*					
32.000		0			0			0						
16.000		2	*****		3	*****		5	*****					
8.000		1	****		2	*****		4	*****					
4.000		2	*****		3	*****		9	*****					
2.000		1	****		5	*****		7	*****					
1.000		1	****		0			3	****					
Total=		11	10	20	30	40	50%	13	10	20	30	40	50%	

# Glossary

**APPC.** Advanced program-to-program communications, a programming interface standard for communications using SNA LU 6.2.

**BLDS.** See *block level data sharing*

**Block level data sharing (BLDS).** A kind of data sharing that enables application programs in different IMS subsystems to update concurrently.

**Commit point.** The point at which an application program commits that a section of work is done and that the data it has modified or created is consistent and complete. Its output, which has been held up to that time, is sent to its destination(s); its input (if any) is removed from the message queues; and its database updates are confirmed and made available to other applications. A commit point occurs when a program terminates normally, when it issues a checkpoint call or command, or when it issues a commit verb. If a program processes messages, a commit point may also occur when it retrieves a new message. Commit points are also called synchronization points or sync points.

**CPIC driven application program.** An application program that uses CPI communications calls to receive an incoming message and to send a reply.

**Database Control (DBCTL).** An IMS facility that provides an IMS Database Manager subsystem without the IMS Transaction Manager. It may be used by CICS, ODBA threads, and BMPs for access to IMS databases.

**Database level data sharing.** A kind of data sharing that enables one IMS subsystem to update a database while other IMS subsystems read the database without integrity or allows multiple IMS subsystems to read a database with integrity.

**Database record.** In a database, a collection of segments that contains one occurrence of the root segment type and all of its dependents arranged in a hierarchic sequence. It may be smaller than, equal to, or larger than the access method logical record.

**Data entry database (DEDB).** A Fast Path database that consists of one or more areas, with each area containing both root segments and dependent segments.

**Data sharing.** The concurrent access of databases by two or more IMS subsystems. The IMS subsystems can be in one processor or in separate processors. They can share data at either the database level or the block level.

**DBCTL.** See *Database Control*

**DDIR.** Database directory control block. An IMS system contains one of these control blocks for each database defined to the system.

**DEDB.** See *data entry database*

**ECNT.** Extended Communications Name Table, a control block related to an IMS terminal and used as an index to terminal related MSDBs.

**EPS.** See *Extended pointer set*

**Extended pointer set (EPS).** In a HALDB, an expanded segment prefix that includes information that allows the use of indirect pointers. An EPS is created for logical child segments and secondary index segments.

**Fast Path databases.** Two types of IMS databases designed to provide high availability and fast processing for IMS applications. See also *main storage database* (MSDB) and *data entry database* (DEDB).

**Full function databases.** IMS databases that provide a wide range of capabilities, including logical relationships and secondary indexing. Full function databases include HDAM, HIDAM, PHDAM, PHIDAM, PSINDEX, HSAM, HISAM, SHSAM, SHISAM, and INDEX.

**HALDB.** See *High Availability Large Database*.

**HALDB Online Reorganization (OLR).** A function of IMS that allows non-disruptive, online reorganization of PHDAM and PHIDAM partitions.

**High Availability Large Database (HALDB).** A partitioned full function database. The supported database organizations are PHDAM, PHIDAM, and PSINDEX.

**Independent overflow (IOVF).** In a Data Entry Database (DEDB) the part of the area which contains roots and direct dependents which have overflowed from the UOWs containing RAP CIs and dependent overflow CIs.

**Internal Resource Lock Manager (IRLM).** An IMS component that provides lock management for use by IMS subsystems that share data at the block level. The IRLM also may be used to provide lock management for resources accessed in a single system.

**IOVF.** See *Independent overflow*

**IRLM.** See *Internal Resource Lock Manager*

**Logical relationship.** In a database, a path between two independent segments where the relationship is user defined.

**Logical child.** In a database, a pointer segment that establishes an access path between its physical parent and its logical parent. It is a physical child of its physical parent and a logical child of its logical parent.

**Logical parent** In a database, the segment to which a logical child points. It can also be a physical parent. Furthermore, it contains the common reference data. The pointer in the logical child to the logical parent can be symbolic or direct.

**Main storage database (MSDB).** A Fast Path root segment database which resides in main storage.

**Modified standard application program.** An IMS application program that uses CPI-C calls to allocate LU 6.2 conversations and sends and receives data.

**MSDB.** See *Main Storage Database*

**ODBA.** See *Open Database Access*

**OLR.** See *HALDB Online Reorganization*

**Open Database Access (ODBA).** A callable interface that can be used by a z/OS application program to issue DL/I calls to an IMS DB system. The application program must use Resource Recovery Services (RRS) of

z/OS as a sync point manager. ODBA is used by DB2 Stored Procedures, WebSphere Application Server, ODBM, and other address spaces to access IMS databases.

**Output Thread.** An asynchronous task which writes committed DEDB updates to disk, a VSO data space, or a Coupling Facility structure.

**PHDAM.** A partitioned Hierarchical Direct Access Method database organization, one type of High Availability Large Database (HALDB).

**PHIDAM.** A partitioned Hierarchical Indexed Direct Access Method database organization, one type of High Availability Large Database (HALDB).

**Physical child.** In a database, a segment type that is dependent on a segment type defined in the next higher level in the database hierarchy. All segment types in a database, except the root, are physical children since each is dependent on its parent.

**Physical parent.** In a database, a segment type that has a dependent segment type defined at the next lower level in the physical database hierarchy.

**PI.** See *program isolation*

**Pointer segment.** In a secondary index, the segment that contains the data and pointers used to index the target segments.

**Private attribute.** With the IRLM, the private attribute in a lock request prevents the lock from being held by requestors from different subsystems.

**Program isolation (PI).** An IMS facility that separates all activity of an application program from any other active application program until that application program indicates, by reaching a synchronization point, that the data it has modified or created is consistent and complete.

**Program isolation (PI) lock manager.** An IMS lock manager that supports only local locking. The PI lock manager is used in systems for which no IRLM has been defined.

**Protected conversation.** A protected conversation links separate pieces of a distributed application into a single transaction using RRS. All resource managers participating in the protected conversation either commit or back out together.

**PSINDEX.** A partitioned secondary index database organization, one type of High Availability Large Database (HALDB).

**RAP.** See *root anchor point*

**RBA.** See *relative byte address*

**Relative byte address (RBA).** Address in a database data set that is expressed as a number of bytes from the beginning of the data set.

**Relative record number (RRN).** Address in a database data set that is expressed as a number of logical records from the beginning of the data set.

**Resource Recovery Services (RRS).** A component of z/OS which provides a global syncpoint manager that any resource manager on z/OS can exploit.

**Root addressable part.** In a DEDB, the part of the area which contains CIs with root anchor points and the dependent overflow CIs.

**Root anchor point (RAP).** In a DEDB, HDAM, or PHDAM database, a pointer at the beginning of a physical block or CI that points to a root segment whose key randomizes to that RAP.

**RRN.** See *relative record number*.

**RRS.** See *resource recovery services*.

**SDEP.** See *sequential dependent segment*.

**Secondary index.** See *secondary index database*

**Secondary index database.** An index used to establish accessibility to a database by a path different from the one provided by the database definition. It contains pointer segments.

**Sequential dependent segment (SDEP).** A segment of a data entry database that is chained off the root segment and inserted (last-in first-out) into the last part of a DEDB area. After being inserted by an online program, the SDEP cannot be modified.

**Shared VSO.** The implementation of VSO in which an area is read into a coupling facility structure so that it may be shared by multiple IMS systems.

**Source segment.** A database segment containing the data used to construct the secondary index pointer segment.

**Sync interval.** See *unit of work*

**Sync point.** See *commit point*

**Synchronization point.** See *commit point*

**Target segment.** In a database, the segment pointed to by a secondary index entry, that is, from an index pointer segment.

**Unit of reorganization (UOR).** A set of database records which are reorganized by HALDB online reorganization in one unit of work.

**Unit of work (UOW).** (1) For a DEDB, a number of contiguous CIs in the root-addressable part of an area. (2) A set of updates which are committed by a program at the same time. This time is called a commit point or sync point. If a commit point is not reached, which would happen if the program abends, all of the updates in a unit of work are undone or backed out. A unit of work is sometimes called a sync interval.

**UOR.** See *unit of reorganization*

**UOW.** See *unit of work*

**Virtual Storage Option.** An option for DEDB areas that maps an area into a data space or a coupling facility structure when the area is opened. The share level of the database determines whether a data space or coupling

facility structure is used. Any VSO area CI that has been loaded into a data space or structure is subsequently read from the data space rather than from DASD.

**VSO.** See *Virtual Storage Option*





# Index

-911 ..... 41  
 -913 ..... 41  
 abend.....32, 33, 36, 38, 39, 44, 48, 61  
 ACCEPT STATUSGROUP ..... 37, 42  
 ADCD ..... 36, 38, 39  
 ADCI..... 42  
 ADLA ..... 44, 48  
 APPC ..... 32, 42, 69  
 ATRBACK ..... 41  
 BA..... 42  
 Batch Backout utility ..... 5  
 BB ..... 42  
 BC..... 36, 37  
 BKO=..... 5  
 BKO=Y ..... 48  
 BLDS ..... 1, 2, 3, 9, 11, 12, 13, 14, 29, 30, 42, 69  
 block ... 1, 11, 12, 13, 16, 24, 31, 37, 43, 48, 50, 52,  
     63, 69, 70, 72  
 block level data sharing ..... 1, 12, 13, 31, 48, 69  
 block lock..... 11, 37  
 BMP ..... 5, 31, 33, 34, 36, 37, 38, 39, 41, 44, 58, 61  
 busy lock..... 12, 63, 64  
 CCTL ..... 42  
 CHANGE..... 22, 24, 25  
 CHKP..... 5, 39  
 CI3, 11, 12, 13, 16, 20, 21, 22, 23, 43, 45, 47, 52,  
     53, 63, 72, 73  
 CICS ... 5, 33, 36, 37, 38, 39, 40, 41, 42, 44, 45, 48,  
     49, 58, 69  
 close ..... 13, 63  
 command5, 9, 11, 14, 15, 16, 22, 23, 25, 26, 27, 28,  
     34, 45, 54, 57, 63, 64, 69  
 command lock..... 25, 26  
 commit point..... 37, 69, 72  
 CORE= ..... 44  
 coupling facility ..... 42  
 CPIC driven application program ..... 32, 38, 42, 69  
 data set reference lock..... 14, 43  
 data sharing... 1, 2, 4, 12, 13, 14, 30, 31, 43, 48, 50,  
     63, 64, 69  
 database record ... 3, 8, 9, 10, 11, 15, 16, 17, 18, 19,  
     20, 23, 29, 45, 46, 47, 48, 50, 61, 63, 64  
 DB2..... 1, 37, 40, 41, 49, 71  
 DBB ..... 31, 33, 36, 37  
 DBCTL ..... 1, 2, 37, 39, 40, 44, 46, 48, 54, 55, 69  
 DBFLGSYN ..... 31  
 DCB#..... 50, 51, 52  
 DDIR..... 50, 51, 69  
 deadlock .. 2, 6, 7, 20, 34, 35, 36, 37, 38, 39, 40, 41,  
     49, 56, 58, 61

Deadlock Analysis report ..... 61, 66  
 Deadlock List report..... 66  
 Deadlock Summary report..... 66  
 DEADLOK..... 34  
 DEDB ..... 6, 20, 21, 23, 24, 39, 45, 69, 70, 71, 72  
 DEQ..... 15, 22, 23  
 DFS3304I ..... 42  
 DFSBBO00 ..... 5  
 DFSERA10..... 61, 62, 65  
 DFSERA30..... 61, 62, 65, 66  
 DFSERA40..... 3, 62, 63, 65  
 DFSERA60..... 65  
 DFSKLT A0 ..... 59  
 DFSKLT B0 ..... 59  
 DFSKLT C0 ..... 59, 65  
 DFSKTDL0 ..... 58, 65  
 DFSLOG41 ..... 31  
 DFSLR ..... 65  
 DFSPIRP0 ..... 56, 65  
 DFSUTR20..... 54, 55  
 DFSXFER ..... 31  
 DLET..... 24, 25  
 DLI ..... 15, 31, 33, 36, 37, 55, 62  
 DMB#..... 50, 52  
 DMCB# ..... 52  
 DTB..... 38, 44  
 DXR162I ..... 33  
 dynamic transaction backout ..... 38, 44  
 ECSA..... 44  
 EMH..... 5  
 EPS ..... 17, 18, 70  
 ESDS ..... 11, 16, 51  
 exclusive..... 29, 30, 37, 40, 48  
 EXEC DLI..... 15  
 extend lock ..... 13  
 extended pointer set..... 17, 18, 70  
 F irlmproc..... 33, 34, 57  
 Fast Path ... 1, 2, 3, 9, 20, 22, 23, 25, 26, 27, 28, 36,  
     37, 38, 39, 41, 44, 45, 47, 52, 53, 55, 64, 69, 70  
 Fast Path DEDB Resource Contention Summary  
     report..... 68  
 FD..... 36, 38, 39  
 File Select and Formatting Print Utility .. 61, 62, 65  
 FLD ..... 22, 23, 24, 25  
 full function .. 3, 8, 9, 15, 20, 24, 25, 26, 29, 39, 44,  
     45, 47, 50, 57  
 GET ..... 61  
 get hold..... 15, 23, 24  
 get lost ..... 48  
 GFPLL..... 64

global cycle..... 36  
 GQCMX ..... 64  
 GRIDU..... 64  
 GRIDX..... 63, 64, 65  
 GSEGL ..... 63, 64  
 GZIDL ..... 64  
 HALDB Online Reorganization ..... 11, 30  
 HDAM..8, 9, 10, 11, 12, 15, 17, 18, 29, 47, 50, 51,  
 70, 72  
 HIDAM.....8, 9, 10, 11, 12, 17, 18, 29, 50, 51  
 HISAM .....8, 9, 10, 11, 12, 18, 19, 50, 51, 70  
 hold .....3, 6, 11, 15, 23, 24, 37, 45, 46  
 HSSP ..... 21  
 IFP .....5, 31, 33, 37, 38, 39, 41, 44  
 ILOCK ..... 57, 58, 65  
 IMS Monitor .....54, 55, 65, 66  
 IMS Performance Analyzer for z/OS..... 66  
 independent overflow ..... 21  
 INDICES..... 19  
 INIT .....37, 38, 39, 42  
 insert .....12, 16, 20, 37  
 IOVF ..... 21, 70  
 IRLM .. 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
 20, 23, 26, 27, 28, 33, 34, 36, 37, 41, 42, 43, 44,  
 47, 50, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,  
 65, 70, 71  
 IRLM Lock Trace Analysis Utilities ..... 58  
 ISRT.....24, 25, 60, 61  
 JBP.....31, 33, 36, 39  
 JMP .....31, 32, 33, 38, 39, 41  
 KBLA ..... 58  
 Knowledge Based Log Analysis..... 58  
 KSDS .....11, 12, 13, 16, 37, 51  
 level... 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 18, 20,  
 21, 22, 23, 24, 26, 27, 28, 29, 35, 36, 37, 40, 41,  
 43, 45, 48, 56, 61, 65, 69, 70, 71, 72  
 local cycle ..... 36  
 lock reject..... 42  
 lock structure ..... 42  
 LOCKMAX ..... 31  
 LOCKTIME..... 33, 34, 57  
 LOG4ILKH..... 31  
 logical relationship..... 17, 29  
 modified standard application program .. 32, 38, 42  
 Modified standard application program..... 70  
 MPP .....5, 31, 32, 33, 36, 37, 38, 39, 41, 44, 61  
 MSDB .....6, 23, 24, 25, 39, 53, 54, 70  
 OBA ..... 27, 53  
 ODBA .....5, 33, 36, 37, 38, 39, 69, 70  
 OLR ..... 30, 70  
 open.....12, 13, 14, 16, 27, 28, 63  
 OSAM.....3, 11, 12, 16, 51  
 output thread ..... 23

PCB 8, 9, 18, 20, 21, 22, 23, 29, 37, 39, 45, 46, 47,  
 48, 55, 56  
 PHDAM 8, 9, 10, 11, 12, 15, 17, 29, 50, 51, 70, 71  
 PHIDAM ..... 8, 9, 10, 11, 12, 17, 29, 50, 70, 71  
 PI 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 15, 16, 20, 23,  
 32, 33, 36, 37, 41, 42, 44, 47, 49, 50, 52, 53, 54,  
 55, 56, 57, 61, 62, 64, 65, 71  
 PI Trace ..... 3  
 PI Trace Record Format and Print Module ..... 3, 62  
 pointer..... 12, 17, 18, 29, 69, 70, 72  
 private attribute.....4, 11, 12, 13, 71  
 PROCOPT... 8, 9, 20, 22, 23, 24, 25, 29, 30, 45, 48  
 PROCOPT=E ..... 29, 48  
 Program Isolation ..... 1, 57  
 protected conversation.....42  
 Protected Conversation..... 71  
 PSB ..... 8, 19, 29, 39, 45, 46, 58, 61  
 PSINDEX ..... 50, 70, 71  
 PST ..... 23, 58, 60  
 Q command code..... 15, 22  
 RALLX..... 64  
 RAP ..... 8, 9, 10, 15, 17, 18, 20, 51, 52, 71, 72  
 RBA8, 10, 17, 18, 21, 50, 51, 52, 53, 57, 58, 60, 71  
 RCT ..... 41  
 read..... 2, 23, 29, 43, 46, 58, 59, 69, 72, 73  
 Record Format and Print Module ..... 61, 65  
 release.... 6, 9, 15, 20, 23, 35, 39, 40, 45, 48, 63, 64  
 REPL ..... 24, 25  
 Resource Recovery Services ..... 72  
 Retained locks ..... 42  
 RFPLL..... 64  
 RLSE ..... 8, 9, 25, 30, 47, 48  
 RMF ..... 57, 58, 65  
 ROLB ..... 5, 6, 36, 39, 41  
 ROLL ..... 5  
 ROLS..... 5  
 root addressable part.....21, 72  
 RRIDW..... 64  
 RRIDX ..... 63, 64  
 RRN..... 10, 51, 57, 71, 72  
 RRS ..... 42, 72  
 RSEGL ..... 64  
 RZIDL ..... 64  
 secondary index..... 9, 18, 19, 29, 71, 72  
 segment lock.....23, 63  
 SENSEG..... 19  
 SETS..... 6  
 SETU ..... 6  
 share ..... 2, 8, 14, 26, 30, 37, 45, 48, 69, 70, 72  
 SHARELVL..... 2  
 space management.....56  
 SQL ..... 41  
 SRRBACK ..... 41

SSA .....	19, 48
SYNC .....	5, 39
sync point .....	5, 8, 9, 10, 11, 12, 15, 18, 21, 22, 23, 24, 30, 31, 37, 38, 39, 41, 44, 45, 46, 47, 64, 72
SYNCLKS .....	31
SYNCPOINT .....	45
TENQ .....	10, 15, 29
TERM .....	45
test enqueue .....	10, 15, 29
thread .....	5, 21, 23, 37, 38, 39, 47
TIMEOUT .....	33, 34, 57
trace .....	54, 55, 57, 58, 59, 62, 63, 65
TTLKL .....	64
U0123 .....	38
U0124 .....	32
U0125 .....	42
U0775 .....	44, 48
U0777 .....	32, 36, 38, 39, 41, 61
U0778 .....	6
U2478 .....	32, 42

U3300 .....	44, 48
U3301 .....	31
U3303 .....	42
unit of reorganization .....	9, 12, 30, 72
unit of work .....	21, 72
UOR .....	30, 72
UOW .....	21, 53, 72
update .....	2, 8, 9, 11, 12, 13, 15, 16, 21, 24, 36, 37, 43, 45, 46, 47, 69
VERIFY .....	22, 24, 25
virtual storage .....	44, 48, 54
VSAM .....	3, 11, 12, 40
VSO .....	21, 23, 27, 71, 72, 73
VUNLOAD .....	27, 28, 53
x'37' .....	31
x'41' .....	31
x'5937' .....	31
x'67FF' .....	58
XFERLHLD .....	31