IMS

An Introduction to IMS

Version 9

IMS

An Introduction to IMS

Version 9

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 345.

Quality Partnership Programm (QPP) Edition (April 2004) (Softcopy Only)

This QPP edition replaces or makes obsolete the previous edition, ZES1-2350-00.This edition is available in softcopy format only. The technical changes for this version aresummarized under “Changes to the Current Edition of this Book for IMS Version 9” on page xxi.

Contents

Figures

Tables

Foreward

Forward by Vern Watts will go here sometime before GA.

About This Book

This softcopy book is available only in PDF and BookManager formats and also as part of the DB2 Information Management Software Information Center for z/OS Solutions. To get the most current versions of the PDF and BookManager formats, go to the IMS Library page at www.ibm.com/software/data/ims/library.html. To view the information within the DB2 Information Mangement Software Information Center for z/OS Solutions, go to publib/boulder.ibm.com/infocenter/db2zhelp.

Note: This book is at an early draft level and is new for IMS Version 9. We welcome your comments about this information, especially regarding coverage, level of detail, accuracy, and clarity.

IBM Information Management System (IMS) is one of the world's premiere software products. Period.

The purpose of this book is twofold:

- To introduce IMS to those who have not heard about it and provide an basic education about this cornerstone product
- To re-introduce IMS to the computer science field in general

IMS is not in the news and is barely mentioned in today's computer science classes, but it has been and, for the foreseeable future, will continue to be a major, crucial component of the world's software infrastructure.

From its beginnings with NASA, IMS has provided the foundation that enables government agencies and businesses to manage, access, manipulate, and exploit their vast stores of data. As the Information Age evolves and matures, so does IMS.

Related Reading: For more information about IMS, see the IMS library (listed in the "Bibliography" on page 349) and visit the IMS Web site at www.ibm.com/ims.

Who Uses IMS

In spite of rumors that the mainframe and IMS died a long time ago, IMS and the mainframe are alive and well and you use both of them every day. Over 90 percent of the top world-wide companies in the following industries use IMS to run their daily operations:

- Manufacturing
- Finance
- Banking
- Retailing
- Aerospace
- Communications
- Government
- Insurance
- High technology
- Health Care

Here are some interesting facts about how IMS is used.

IMS manages most of the world's corporate data

- Over 95% of Fortune 1000 companies use IMS
- IMS manages over 15 million gigabytes of production data

- $2.5 trillion per day are transferred through IMS by one customer

IMS handles over 50 billion transactions per day

- IMS serves over 200 million users every day
- IMS processes over 100 million transactions per day for one customer
- IMS processed over 120 million transactions per day (7 million per hour) for another customer
- IMS has processed 14,000 transactions per second (over 1 billion per day) using IMS data sharing and shared queues
- A single IMS has handled over 6000 transactions per second over a TCP/IP connection

Gartner Group Quote

"A large and loyal IMS installed base. Rock-solid reputation of a transactional workhorse for very large workloads. Successfully proven in large, Web-based applications. IMS is still a viable, even unmatched, platform to implement very large OLTP systems and, in combination with Web Application Server technology, it can be a foundation for a new generation of Web-based high-workload applications."

Related Reading: For more examples of the industries and customers that use IMS, visit the IMS Web site (www.ibm.com/ims) and click on "Featured Customer", "IMS Newsletter", or "Overview".

Overview of This Book

This book is organized in the following manner:

1. Part 1 is a high-level overview the IMS product. Part 1 contains the following chapters:
   - Chapter 1, "Introduction to IMS," on page 3 discusses a brief history of IMS, an overview of the product, and some of its availability and recovery features
   - Chapter 3, "Setting Up and Running IMS," on page 27 discusses the installation and operation of IMS
   - Chapter 2, "IMS and z/OS," on page 11 discusses the relationships between IMS and the operating system
2. Part 2 is a more detailed look at the IMS Database Manager component of IMS. Part 2 contains the following chapters:
   - Chapter 4, "Overview of IMS DB," on page 37 discusses the functions of IMS DB, the types of IMS databases, and a brief discussion of updating data in IMS and DB2 databases
   - Chapter 5, "Overview of the IMS Hierarchical Database Model," on page 41 is a more detailed discussion of the IMS hierarchical database model
   - Chapter 6, "Implementing the IMS Database Model," on page 51 discusses:
     – The various IMS database types that use the hierarchical model
     – The relationships between the IMS databases and the operating system access methods
     – A brief discussion of functions that ensure data integrity
   - Chapter 7, "Choosing the Correct Database Type," on page 75 discusses some of the criteria for choosing the various IMS database types
   - Chapter 8, "Data Sharing," on page 83 is a brief discussion of data sharing between multiple IMSs

- Chapter 9, “The Database Reorganization Process,” on page 85 discusses the purpose for reorganizing databases, an overview of the process, and a brief introduction to some of the reorganization utility routines that come with IMS
- Chapter 10, “The Database Recovery Process,” on page 101 introduces the database recovery process, discusses some of the IMS backup and recovery utility routines that come with IMS, and briefly discusses backup and recovery procedures

3. Part 3 is a more detailed look at the IMS Transaction Manager component of IMS. Part 3 contains the following chapters:
   - Chapter 11, “Overview of IMS TM,” on page 113 discusses the functions of IMS TM and the relationships between IMS TM, the network, messages, and other subsystems
   - Chapter 12, “IMS TM Control Region,” on page 119 is a more detailed look at how messages are handled by IMS TM
   - Chapter 13, “How IMS TM Processes Input,” on page 123 discusses how IMS TM processes input from a variety of sources
   - Chapter 14, “Fast Path Transactions,” on page 135 introduces Fast Path transactions and discusses how IMS TM processes them
   - Chapter 15, “The Master Terminal,” on page 137 introduces and discuses the responsibilities and capabilities of the IMS Master Terminal
   - Chapter 16, “Application Program Processing for IMS TM,” on page 141 discusses how IMS TM processes application program requests
4. Part 4 is a detailed look at application programming as it relates to IMS. Part 4 contains the following chapters:
   - Chapter 17, “Application Programming Overview,” on page 149 discusses the components of an IMS application program, the setup needed before running the application program, and the IMS database application programming interface
   - Chapter 18, “Application Programming for the IMS Database Manager,” on page 165 discusses database processing that results from application program calls, application language considerations, and other topics related to application programming and IMS DB
   - Chapter 19, “Application Programming for the IMS Transaction Manager,” on page 197 discusses how IMS TM processes application programs and the design of IMS TM applications
   - Chapter 20, “The IMS Message Format Service,” on page 207 contains an overview of the Message Format Service (MFS) function of IMS
   - Chapter 21, “Application Programming in IMS Java,” on page 223 discusses the IMS Java environment and provides an overview of writing IMS applications in Java
5. Part 5 contains information related to administering IMS. Part 5 contains the following chapters:
   - Chapter 22, “The IMS System Definition Process,” on page 231 contains an overview of the types of IMS system definitions, the IMS macros used for system definition, and a discussion of the Extended Terminal Option (ETO) function of IMS
   - Chapter 23, “Customizing IMS,” on page 245 contains an introduction to how you can customize IMS by using exit routines

- Chapter 24, “IMS Security,” on page 253 contains a brief history of IMS security, a discussion of what resources can be protected, and a list of the facilities that can be used for security
- Chapter 25, “IMS Logging,” on page 257 contains discussions on how IMS records events, the data sets where these events are recorded and introduces the Database Recovery Control (DBRC) facility of IMS
- Chapter 26, “Database Recovery Control (DBRC),” on page 263 contains a detailed discussion of using DBRC, an introduction and discussion of the RECON data sets, the support built into DBRC for Remote Site Recovery (RSR), and some recommendations regarding the RECON data sets
- Chapter 27, “Controlling IMS,” on page 277 discusses various administrative tasks related to running IMS
- Chapter 29, “IBM IMS Tools,” on page 311 contains a brief introduction to the various IBM IMS Tools that are available

6. Part 6 discusses how IMS relates to the Parallel Sysplex environment of z/OS. Part 6 contains the following chapters:
   - Chapter 30, “Introduction to Parallel Sysplex,” on page 315 contains an overview of the Parallel Sysplex environment and how IMS takes advantage of this environment
   - Chapter 31, “IMSplexes,” on page 337 contains a discussion of how multiple IMSs can be managed as a single unit in the Parallel Sysplex environment

How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can do one of the following:

- Go to the IMS Library page at www.ibm.com/software/data/ims/library.html and click the Library Feedback link, where you can enter and submit comments.
- Send your comments by e-mail to imspubs@us.ibm.com. Be sure to include the title, the part number of the title, the version of IMS, and, if applicable, the specific location of the text you are commenting on (for example, a page number in the PDF or a heading in the Information Center).

Summary of Changes

This edition is a draft version of this new book intended for use during the Quality Partnership Program (QPP). Contents of thisbook are preliminary and under development.

Changes to the Current Edition of this Book for IMS Version 9

This edition of this book contains many updates, most notably in the graphics.

Library Changes for IMS Version 9

Changes to the IMS Library for IMS Version 9 include the addition of new titles, the change of one title, and a major terminology change. Changes are indicated by a vertical bar (|) to the left of the changed text.

New and Revised Titles

The following list details the major changes to the IMS Version 9 library:

- *IMS Version 9: HALDB Online Reorganization Guide and Reference*

  The library includes new information: *IMS Version 9: HALDB Online Reorganization Guide and Reference*. This information is available only in PDF and BookManager formats.
- *IMS Version 9: An Introduction to IMS*

  The library includes new information: *IMS Version 9: An Introduction to IMS*.
- The information formerly titled *IMS Version 8: IMS Java User's Guide* is now titled *IMS Version 9: IMS Java Guide and Reference*.

Terminology Changes

IMS Version 9 introduces new terminology for IMS commands:

type-1 command
: A command, generally preceded by a leading slash character, that can be entered from any valid IMS command source. In IMS Version 8, these commands were called *classic* commands.

type-2 command
: A command that is entered only through the OM API. Type-2 commands are more flexible and can have a broader scope than type-1 commands. In IMS Version 8, these commands were called *IMSplex* commands or *enhanced* commands.

Accessibility Enhancements

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products, including IMS, enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

User Assistive Technologies

Assistive technology products, such as screen readers, function with the IMS user interfaces. Consult the documentation of the assistive technology products for specific information when you use assistive technology to access these interfaces.

Accessible Information

Online information for IMS Version 9 is available in BookManager format, which is an accessible format. All BookManager functions can be accessed by using a keyboard or keyboard shortcut keys. BookManager also allows you to use screen readers and other assistive technologies. The BookManager READ/MVS product is included with the z/OS base product, and the BookManager Softcopy Reader (for workstations) is available on the IMS Licensed Product Kit (CD), which you can download from the Web at www.ibm.com.

Keyboard Navigation of the User Interface

Users can access IMS user interfaces using TSO/E or ISPF. Refer to the *z/OS V1R1.0 TSO/E Primer*, the *z/OS V1R1.0 TSO/E User's Guide*, and the *z/OS V1R1.0 ISPF User's Guide, Volume 1*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Part 1. Overview of IMS

Chapter 1. Introduction to IMS

This chapter contains an overview of the entire IMS™ product. It includes both the Transaction Manager and Database Manager components. The following sections are covered in this chapter:

History of IMS

As shown in the next few sections, IMS has been an important part of world-wide computing since its inception.

Beginnings at NASA

On May 25, 1961, United States President John F. Kennedy challenged American industry to send an American to the moon and have him return safely to earth. This feat was to be accomplished before the end of the decade. American Rockwell won the bid to build the spacecraft for the Apollo Program and, in 1965, they established a partnership with IBM to fulfill the requirement for an automated system to manage large bills of material for the construction of the spacecraft.

In 1966, 12 members of the IBM team, along with 10 members from North American Rockwell and 3 members from Caterpillar Tractor, started the design and development of the system that was called Information Control System (ICS) and Data Language/Interface (DL/I). During the design and development process, the IBM team was moved to Los Angeles and increased to 21 members. This team completed and shipped the first release of ICS.

In April, 1968, ICS was installed. The first "READY" message was displayed on an IBM 2740 typewriter terminal at the Rockwell Space Division in Downey California, on August 14, 1968.

ICS was renamed Information Management System/360™ (IMS/360) in 1969 and became available to the world.

Since 1968, IMS:

- Helped achieve President Kennedy's dream.
- Started the database management system revolution.
- Continues to evolve to meet and exceed the data processing requirements demanded by today's businesses and governments.

IMS as a Database Management System

The IMS database management system (DBMS) realized the concept of separating application code from the data. The point of separation was the Data Language/Interface (DL/I). IMS controlled the access and recovery of the data.

This separation established a new paradigm for application programming. The application code could now focus on the manipulation of the data and not have the overhead associated with the access and recovery of the data. This paradigm virtually eliminated the need for redundant copies of the data. Multiple applications could access and update a single instance of the data, thus providing current data for each application.

The DL/I Callable Interface

Application programs still access and navigate through the data by using the DL/I standard callable interface. Online access to the data became possible because the application code was separated from the data control.

IMS as a Transaction Manager

IBM developed the online component to ICS/DL/I to support data communication access to the databases. The DL/I callable interface was expanded to the online component of the product to enable data communication transparency to the application programs. A message queue function was created to maintain the integrity of data communication messages and to provide a queuing concept for scheduling application programs.

The online component to ICS/DL/I ultimately became the Data Communications (DC) function of IMS. IMS DC became the IMS Transaction Manager (IMS TM) in IMS Version 4.

Overview of the IMS Product

IMS delivers accurate, consistent, timely, and critical information to application programs, which deliver the information to many end users simultaneously.

IMS has been developed to provide an environment for applications that require very high levels of performance, throughput, and availability. IMS uses the maximum facilities that the operating system and hardware have to offer. Currently, IMS runs on z/OS® and on zSeries hardware.

IMS consists of three components, the Database Manager (IMS DB) component, the Transaction Manager (IMS TM) component, and a set of system services that provide common services to the other two components. Together, (known as IMS DB/DC) they create a complete online transaction processing environment providing continuous availability and data integrity. The individual functions provided by these components are described in more detail later in this book.

IMS DB is a DBMS that helps you organize business data with both program and device independence. With IMS DB:

- Database transactions (inserts, updates, and deletes) are performed as a single unit of work so that the entire transaction either occurs or does not occur.
- The data in each database is guaranteed to be consistent.
- Multiple database transactions can be performed concurrently with the results of each transaction kept isolated from the others.
- The data in each database is guaranteed to remain even when the DBMS is not running.

IMS TM is a message-based transaction processor. IMS TM provides services to:

- Process input messages received from a variety of sources (such as the terminal network, other IMSs, and the Web).
- Process output messages created by application programs.
- Provide an underlying queueing mechanism for handling these messages.
- Provide high-volume, high-performance, high-capacity, low-cost transaction processing for both IMS DB's hierarchical databases and DB2®'s relational databases.

IMS TM supports many terminal sessions at extremely high transaction volumes.

IMS TM and IMS DB can be ordered and paid for separately if the functions of the other component are not required. The appropriate system services are provided for the component ordered.

IMS has been developed so that each new release of IMS is upwardly compatible, so investment in existing applications is preserved. To accommodate the changing requirements of IT systems, many new features have been added. This has also resulted in a number of IMS features being wholly or partially superseded by newer features that provide better functionality.

Applications written to use IMS functions can be written in a number of programming languages. Programming languages currently supported are Assembler, C, COBOL, Java™, Pascal, PL/I and REXX. The IMS resources are accessed by the application by calling a number of standard IMS functions. Applications access these functions through a standard application programming interface (API) for both the Transaction Manager and Database Manager components. This interface is DL/I.

IMS Database Manager

At the heart of IMS DB are its databases and its data manipulation language (DL/I calls). IMS DB lets you:

- Maintain data integrity.
- Define the database structure and the relationships among the database elements.
- Query information in the database.
- Add new information to the database.
- Delete information from the database.
- Update information in the database.

Additionally, IMS DB lets you adapt IMS databases to the requirements of your many and varied applications. Application programs can access common and, therefore, consistent data, reducing the need to maintain the same data in multiple ways in separate files for different applications.

IMS DB provides:

- A central point of control and access for the IMS data that is processed by IMS applications.
- Facilities for securing (backup and recovery) and maintaining the databases. It allows multiple tasks (batch and/or online) to access and update the data while retaining the integrity of that data. It also provides facilities for tuning the databases by reorganizing and restructuring them.

IMS databases are hierarchical. Data within the database is arranged in a tree structure, with data at each level of the hierarchy related to, and in some way dependent upon, data at the higher level of the hierarchy (see Figure 1 on page 6). By following this model, a specific data item only needs to be stored within the database once. The data item is then available to any user who is authorized to use it. Users do not need to have personal copies of the data.

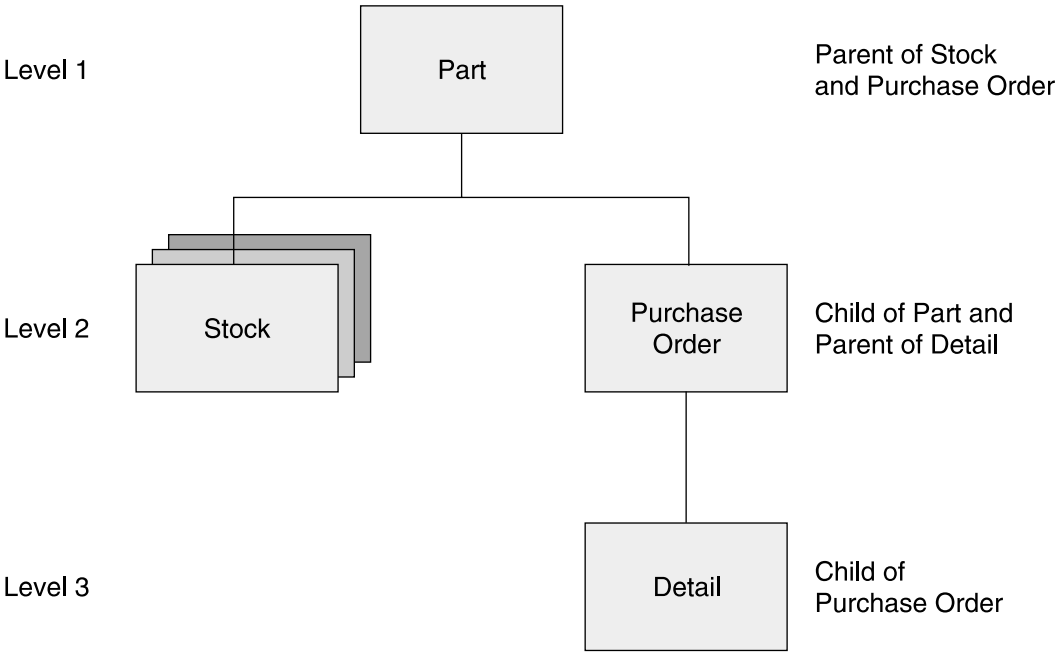

Figure 1. Example of a Hierarchical Data Model

IMS databases are accessed internally using a number of IMS's database organization access methods. The actual database data is stored on disk storage using normal z/OS access methods.

IMS DB provides access to these databases from applications running under the IMS Transaction Manager, CICS® Transaction Server for OS/390® and z/OS, z/OS batch jobs, WebSphere® Application Server for z/OS, and DB2 UDB for z/OS stored procedures.

IMS DB can be ordered separately from the base IMS product. This configuration is called DB control (DBCTL).

Related Reading: For more information about IMS DB, see Part 2, "IMS Database Manager," on page 35.

IMS Transaction Manager

IMS TM provides users of a network with access to applications running under IMS. The users can be people at terminals or workstations, or other application programs, either on the same z/OS system, on other z/OS systems, or on other non-z/OS platforms.

A transaction is a specific setup of input data that triggers the execution of a specific business application program. The message that is destined for an application program, and the return of any results, is considered one transaction.

When IMS TM is used with IMS DB, it extends the facilities of that database management system to the online, real-time environment. IMS TM enables terminals or other devices or subsystems to enter transactions that initiate application programs, which access IMS DB or DB2 databases and return results.

You can define a variety of online processing options. For example, you can define transactions for high-volume data-entry applications, others for interactive

applications, and still others to support predefined queries. IMS TM supports a wide variety of terminals and devices. It also enables you to develop a wide range of high-volume, rapid-response applications, and to geographically disperse your data processing locations, while keeping centralized control of your database.

IMS TM can be ordered separately from the base IMS product. This configuration is called DC control (DCCTL).

Related Reading: For more information about IMS TM, see Part 3, “IMS Transaction Manager,” on page 111.

IMS System Services

There are a number of functions that are common to both the Database Manager and Transaction Manager. These services:

- Recover data
- Restart and recover IMS following failures
- Provide security (controlling access to and modification of IMS resources)
- Manage the application programs (dispatching work, loading application programs, providing locking services)
- Provide diagnostic and performance information
- Provide facilities for operating IMS
- Provide interfaces to other z/OS subsystems that communicate with IMS applications

Another IMS system service is Database Recovery Control (DBRC). DBRC provides the recovery services part of the IMS system. DBRC:

- Controls the allocation and use of all IMS logs in an online environment
- Can provide access control for databases
- Can control database recovery
- Can work closely with the IMS recovery utilities

DBRC uses a set of control data sets, (collectively called the Recovery Control data sets or the RECON data sets) to store the control information that is required to fulfill these functions.

Related Reading: A more detailed description of DBRC is found in Chapter 26, “Database Recovery Control (DBRC),” on page 263.

Accessing IMS

Network access to IMS Transaction Manager was originally by IBM’s systems, which evolved into the System Network Architecture (SNA), as implemented in the VTAM® program product (now a component of z/OS). Now, there are multiple ways to access IMS resources by networks using Transmission Control Protocol/Internet Protocol (TCP/IP), as well as other methods (such as IMS’s database resource adapter (DRA) or through other products like Websphere MQ).

The interfaces to IMS are pictured in Figure 2 on page 8.

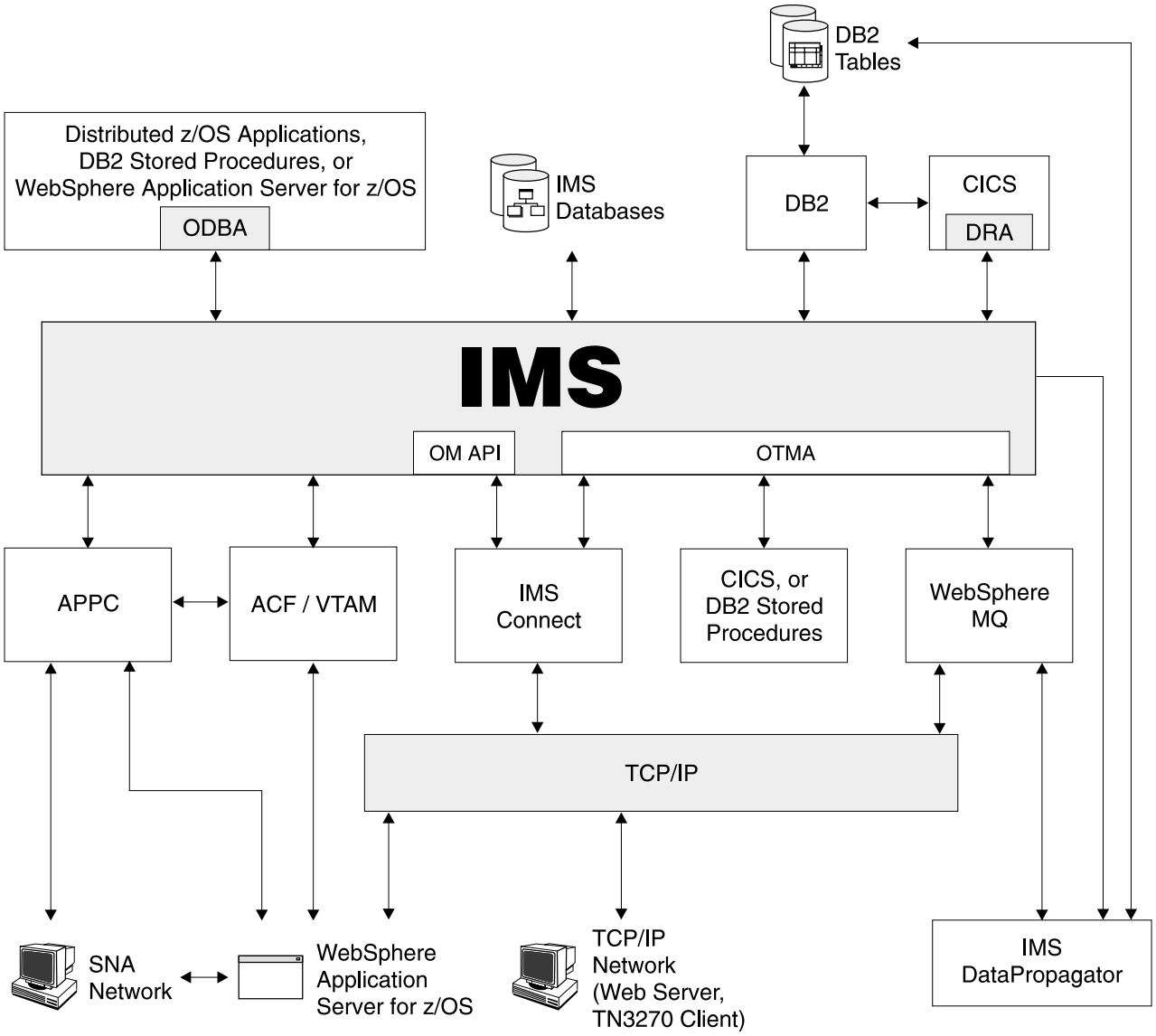

Figure 2. Interfaces to IMS

How IMS Relates to z/OS

IMS runs on IBM zSeries or compatible mainframes that run the z/OS operating system. In fact, there is a symbiotic relationship between IMS and z/OS. Both are tailored to provide the most efficient use of the hardware and software components.

IMS runs as a z/OS subsystem and uses several address spaces. There is one controlling address space (called a control region), several separate address spaces that provide IMS services, and several address spaces (called dependent regions) that run IMS application programs. The various components of an IMS system are explained in more detail in “Structure of IMS Subsystems” on page 11.

Related Reading: For more information about the relationships between IMS and z/OS, see Chapter 2, “IMS and z/OS,” on page 11. For full details on the compatibility of IMS releases with versions of the operating system and associated products, see the current release planning guides:

- *IMS Version 7: Release Planning Guide*
- *IMS Version 8: Release Planning Guide*
- *IMS Version 9: Release Planning Guide*

Parallel Sysplex

IMS exploits the z/OS Parallel Sysplex® environment to enable a more dynamic, available, manageable, scalable, and well performing environment for database, transaction, and systems management.

In a Parallel Sysplex environment, you can run multiple IMS subsystems that share message queues and databases. This sharing enables workload balancing and insulation from individual IMS outages. If one IMS in the sysplex fails, others continue to process the workload, so the enterprise is minimally affected.

Related Reading: For more information on this topic, see Part 6, “IMS in a Parallel Sysplex Environment,” on page 313.

Chapter 2. IMS and z/OS

This chapter describes how IMS subsystems are implemented on an z/OS system. It then gives an overview of IMS's use of z/OS facilities.

The following sections are covered in this chapter:

Structure of IMS Subsystems

This section describes the various types of z/OS address spaces and their relationship with each other. z/OS address spaces are sometimes called regions, as in the IMS control region. The term region is synonymous with a z/OS address space.

The core of an IMS subsystem is the control region, running in one z/OS address space. For each control region there are multiple separate address spaces that provide additional services to the control region or in which the IMS application programs run.

In addition to the control region, some applications and utilities used with IMS run in separate batch address spaces. These are separate to an IMS subsystem and its control region and have no connection with it.

IMS Control Region

The control region (CTL) is a z/OS address space that can be initiated through a z/OS start command, or by submitting JCL.

The IMS control region provides the central point for an IMS subsystem. The control region:

- Provides the interface to the SNA network for the Transaction Manager functions.
- Provides the Transaction Manager OTMA interface for access to non-SNA networks.
- Provides the interface to z/OS for the operation of the IMS subsystem.
- Controls and dispatches the application programs running in the dependent regions.

The control region also provides all logging, restart and recovery functions for the IMS subsystems. The terminals, message queues, and logs are all attached to this region, and the Fast Path database data sets are also allocated by the control region.

A type 2 supervisor call routine (SVC) is used for switching control information, message and database data between the control region, all other regions, and back.

There are three different types of IMS control regions, depending on whether the Database Manager or Transaction Manager components (or both) are being used. These three control region types are:

- DB/DC — This is a control region with both Transaction Manager and Database Manager components installed. It provides the combined functionality of both the other two types of control regions listed below. Note that when a DB/DC region is providing access to IMS databases for a CICS region, it is referred to in some documentation as providing DBCTL services, though it might, in fact, be a full DB/DC region and not just a DBCTL region. The “DC” in DB/DC is a left over from when the Transaction Manger was called the Data Communications function of IMS. As shown in Figure 3 on page 13, the DB/DC control region provides access to the:
  - IMS message queues for IMS applications running in the message processing program (MPP) or Java message processing regions.
  - IMS libraries.
  - IMS logs.
  - Fast Path databases.
  - DL/I separate address space.
  - Database Recovery Control (DBRC) region.
  - IMS Fast Path region (IFP).
  - Java message processing program (JMP) region.
  - Java batch processing program (JBP) region.
  - BMP address spaces.

  Related Reading: For more information about the separate address spaces, see “IMS Separate Address Spaces” on page 14. For more information about the various types of regions for application programs, see “Application Dependent Regions” on page 16.

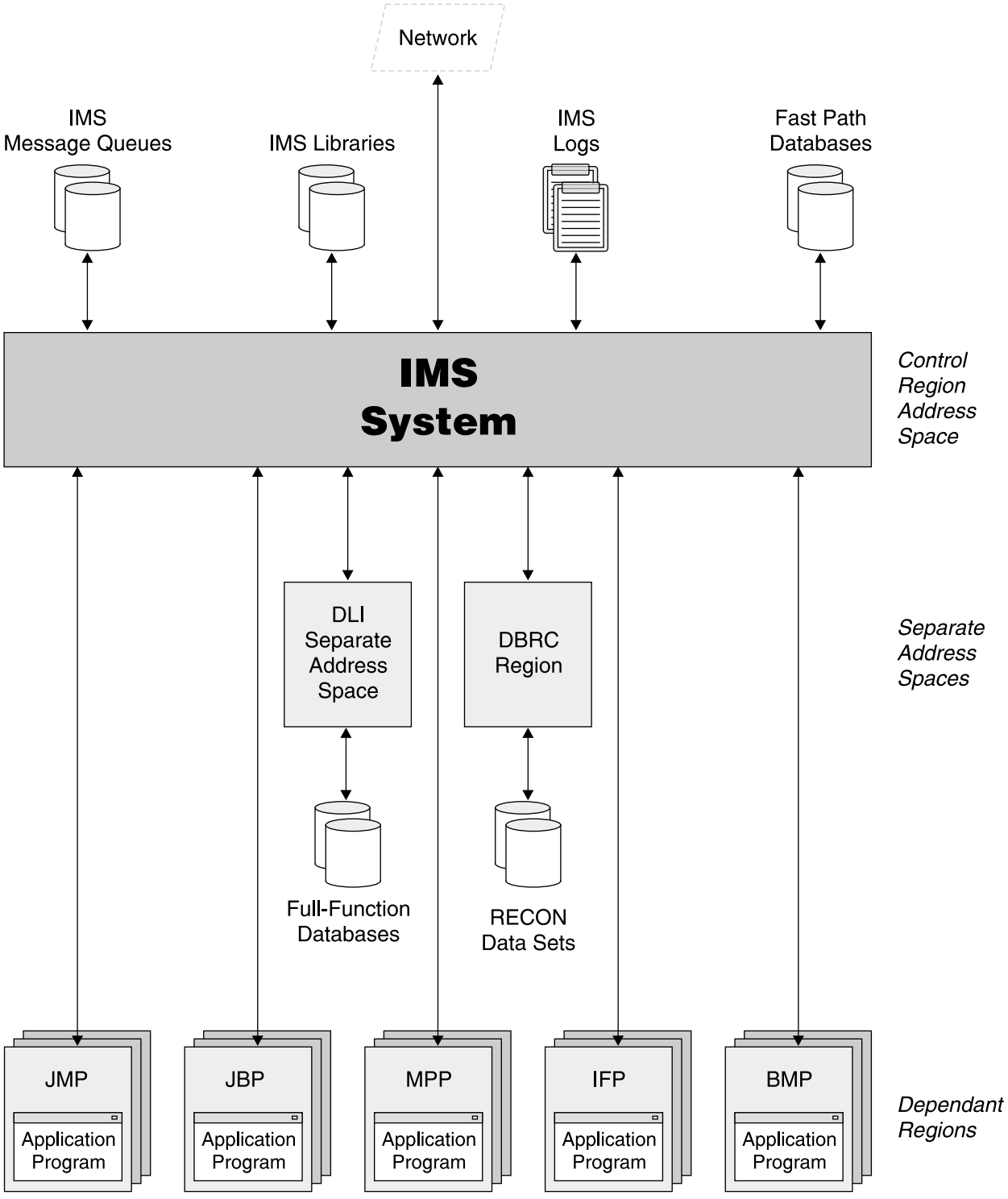

Figure 3. Structure of an IMS DB/DC Subsystem

- DBCTL — This is a control region with only the Database Manager component installed (pronounced DB Control). DBCTL can provide IMS database functions to batch message programs (BMP and JMP application programs) connected to the IMS control region, to application transactions running in CICS Transaction Manager regions, and to other z/OS address spaces (for example, DB2 UDB for z/OS stored procedures) by using the Open Database Access (ODBA) interface.

- DCCTL — This type of control region has only the Transaction Manager component installed (pronounced DC Control). DCCTL can also be used as the Transaction Manager front end for a DB2 UDB for z/OS.

In some of the IMS documentation, the terms DB/DC, DBCTL, and DCCTL are also used to see what sort of IMS system is being defined during an IMS system definition; that is, for what functions will be in the IMS libraries after the system definition process has completed.

IMS Separate Address Spaces

The control region has separate address spaces to provide some of the services of the IMS subsystem.

These regions are automatically started by the IMS control region as part of its initialization, and the control region will not complete initialization until these dependent regions have started and connected to the IMS control region. Every IMS control region has a DBRC region. The other two separate address spaces are optional, depending on the IMS features used. For DL/I, separate address space options can be specified at IMS initialization.

DBRC Region

The DBRC region processes all access to the DBRC recovery control (RECON) data sets. It also performs all generation of batch jobs for DBRC (for example, for archiving the online IMS log). All IMS control regions have a DBRC address space, as it is needed, at a minimum, for managing the IMS logs.

DL/I Separate Address Space (DLISAS)

This address space performs most data set access functions for the IMS Database Manager component (except for the Fast Path DEDB databases, described later). The full-function database data sets are allocated by this address space. It also contains some of the control blocks associated with database access and some database buffers.

This address space is not present with a DCCTL system because the Database Manager component is not present.

For a DBCTL control region, this address space is required and always present.

For a DB/DC control region, you have the option of having IMS database accesses performed by the control region or having the DB/DC region start a DL/I separate address space. For performance and capacity reasons, use a DL/I separate address space.

Common Queue Server (CQS) Address Space

Common Queue Server (CQS) is a generalized server that manages data objects on a z/OS coupling facility on behalf of multiple clients. One CQS is shipped with every IMS.

CQS uses the z/OS coupling facility as a repository for data objects. Storage in a coupling facility is divided into distinct objects called structures. Authorized programs use structures to implement data sharing and high-speed serialization. The coupling facility stores and arranges the data according to list structures. Queue structures contain collections of data objects that share the same name, known as queues. Resource structures contain data objects organized as uniquely named resources.

CQS receives, maintains, and distributes data objects from shared queues on behalf of multiple clients. Each client has its own CQS access the data objects on the coupling facility list structure. IMS is one example of a CQS client that uses CQS to manage both its shared queues and shared resources.

CQS runs in a separate address space that can be started by the client (IMS). The CQS client must run under the same z/OS operating system where the CQS address space is running.

CQS is used by IMS DCCTL and IMS DB/DC control regions if they are participating in sysplex sharing of IMS message queues or resource structures.

Clients communicate with CQS using CQS requests that are supported by CQS macro statements. Using these macros, CQS clients can communicate with CQS and manipulate client data on shared coupling facility structures. Figure 4 shows the communications and the relationship between clients, CQSs, and the coupling facility.

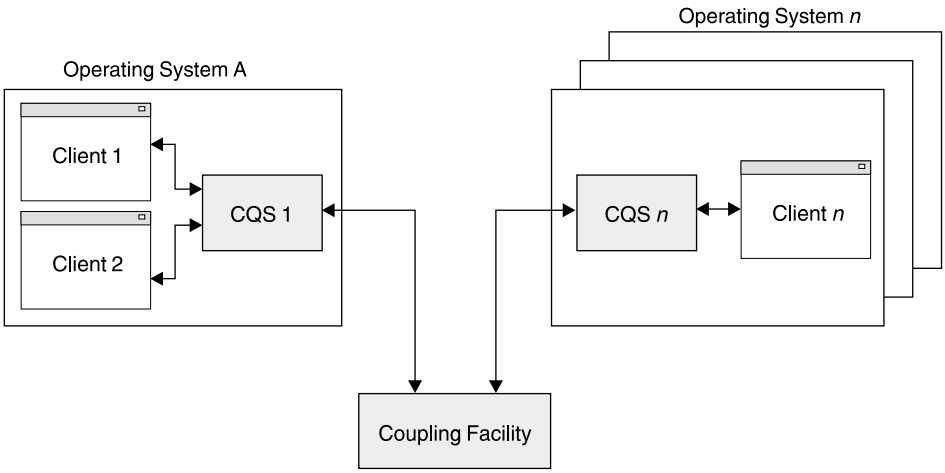

Figure 4. Client Systems, CQS, and a Coupling Facility

Related Reading: For complete information about CQS, see the *IMS Version 9: Common Queue Server Guide and Reference*.

Common Service Layer

The IMS Common Service Layer (CSL) is a collection of IMS manager address spaces that provide the infrastructure needed for systems management tasks. The CSL address spaces include Operations Manager (OM), Resource Manager (RM), and Structured Call Interface (SCI). They are briefly described in the following sections.

The IMS CSL reduces the complexity of managing multiple IMS systems by providing you with a single-image perspective in an IMSplex. An IMSplex is one or more IMS subsystems that can work together as a unit. Typically, but not always, these subsystems:

- Share either databases or resources or message queues (or any combination)
- Run in an z/OS sysplex environment
- Include an IMS CSL

Related Reading: For a further discussion of IMS in a sysplex environment, see:

- Chapter 31, “IMSplexes,” on page 337
- *IMS Version 9: Administration Guide: System*

For a detailed discussion of IMS in a sysplex environment, see:

- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*

Operations Manager Address Space: The Operations Manager (OM) controls the operations of an IMSplex. OM provides an application programming interface (the OM API) through which commands can be issued and responses received. With a single point of control (SPOC) interface, you can submit commands to OM. The SPOC interfaces include the TSO SPOC, the REXX SPOC API, and the IMS Control Center. You can also write your own application to submit commands.

Related Reading: For a further discussion of OM, see “Operations Manager” on page 339.

Resource Manager Address Space: The Resource Manager (RM) is an IMS address space that manages global resources and IMSplex-wide processes in a sysplex on behalf of its clients. IMS is one example of an RM client.

Related Reading: For a further discussion of RM, see “Resource Manager” on page 339.

Structured Call Interface Address Space: The Structured Call Interface (SCI) allows IMSplex members to communicate with one another. The communication between IMSplex members can happen within a single z/OS image or among multiple z/OS images. Individual IMS components do not need to know where the other components reside or what communication interface to use.

Related Reading: For a further discussion of SCI, see “Structured Call Interface” on page 339.

Application Dependent Regions

IMS provides dependent region address spaces for the execution of system and application programs that use IMS services.

The application dependent regions are started as the result of JCL submission to the operating system by the IMS control region, following an IMS command that had been entered.

After they are started, the application programs are scheduled and dispatched by the control region. In all cases, the z/OS address space executes an IMS region control program. The application program is then loaded and called by the IMS code.

There can be up to 999 application dependent regions connected to one IMS control region, made up of any combination of the following dependent region types:

- Message processing region (MPR)
- IMS Fast Path region (IFP), processing Fast Path applications or utilities
- Batch message processing (BMP) region, running with or without HSSP (High Speed Sequential Processing)
- Java message processing (JMP) region

- Java batch processing (JBP) region
- DBCTL thread (DBT)

The combination of what region type can be used in the various types of IMS control regions, can be found in Table 1.

Table 1. Support for Region Types by IMS Control Region Type

Application Address Space Type	**DCCTL**	**DBCTL**	**DB/DC**
MPR	Y	N	Y
IFP	Y	N	Y
BMP (transaction oriented)	Y (1)	N	Y
BMP (batch)	N	Y	Y
JMP	Y	N	Y
JBP	Y	Y	Y
Batch	N	N	N
DBT	N	Y	Y

1. BMP regions attached to a DCCTL control region can only access the IMS message queues and DB2 UDB for z/OS databases.

Message Processing Region

This type of address space is used to run applications to process messages input to the IMS Transaction Manager component (that is, online programs). The address space is started by IMS submitting the JCL as a result of an IMS command. The address space does not automatically load an application program but will wait until work becomes available.

There is a complex scheme for deciding which MPR to run the application program. We will give a brief description below. When the IMS dispatching function determines that an application is to run in a particular MPR, the application program is loaded into that region and receives control. It processes the message, and any further messages for that transaction waiting to be processed. Then, depending on options specified on the transaction definition, the application either waits for further input, or another application program will be loaded to process a different transaction.

Fast Path Region

This type of address spaces runs application programs to process messages for transactions that have been defined as Fast Path transactions.

Fast Path applications are very similar to the programs that run in an MPR. Like MPRs, the IFP regions are started by the IMS control region submitting the JCL as a result of an IMS command. The difference with IFP regions is in the way IMS loads and dispatches the application program and handles the transaction messages. To allow for this different processing, IMS imposes restrictions on the length of the application data that can be processed in an IFP region as a single message.

IMS employs a user-written exit routine, which you have to write, to determine whether a transaction message should be processed in an IFP region and which IFP region it should be processed in. The different dispatching of the transaction messages by the control region is called Expedited Message Handling (EMH). The

intention is to speed the processing of the messages by having the applications loaded and waiting for input messages, and, if the message is suitable, dispatching it directly in the IFP region, bypassing the IMS message queues. Fast Path was originally a separately priced function available with IMS, intended to provide faster response and allow higher volumes of processing. It is now part of the IMS base product.

Batch Message Processing Region

Unlike the other two types of application dependent regions, the BMP is not started by the IMS control region, but is started by submitting a batch job, for example by a user from TSO or by a job scheduler. The batch job then connects to an IMS control region defined in the execution parameters. There are two types of applications that can run in BMP address spaces:

- Message Driven BMPs (also called transaction-oriented BMPs) that read and process messages off the IMS message queue.
- Non-message BMPs (batch-oriented) that do not process IMS messages.

BMPs have access to the IMS full-function databases (not Fast Path), providing that the control region has the Database Manager component installed. BMPs can also read and write to z/OS sequential files, with integrity, using the IMS GSAM access method DBCTL Thread (DBT).

When a CICS system connects to IMS (either as DBCTL or as IMS DB/DC) using the Database Resource Adapter (DRA), each CICS system will have a pre-defined number of connections with IMS. Each of these connections is called a thread. See Figure 5 on page 19.

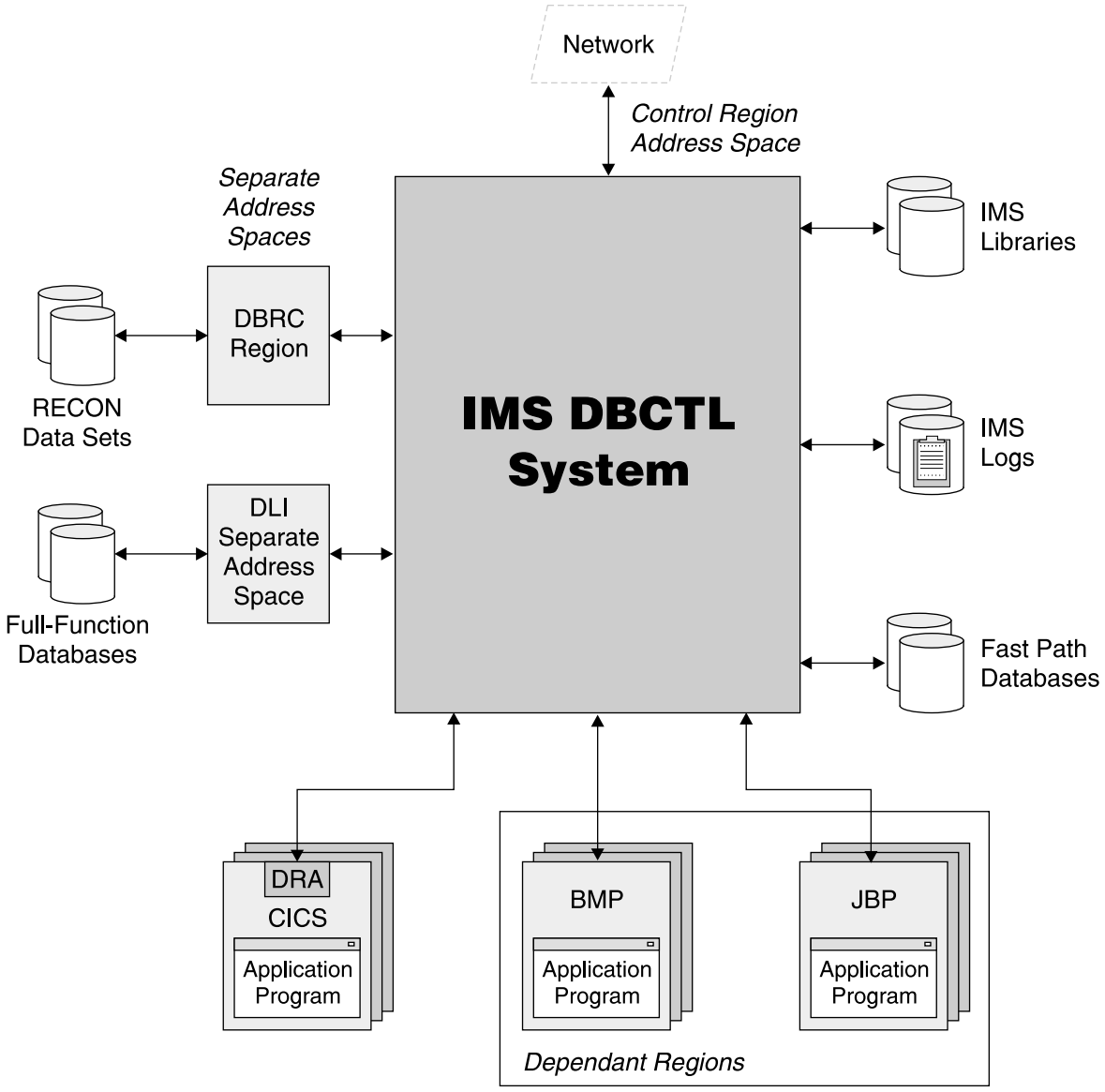

Figure 5. Structure of an IMS DBCTL System

Although these threads are not jobs in their own right, from IMS's perspective, each thread appears just like another dependent region and when CICS requires a DL/I call to IMS, the program will effectively be running in one of these DBT regions.

Java Dependent Regions

IMS Java application programs run in one of two IMS dependent regions that provide a Java Virtual Machine (JVM) environment for the Java application. The Java dependent region types are:

- Java Message Processing (JMP) for message-driven Java applications. JMP applications can process input messages from the message queue (similar to MPPs) and can access DB2 data (using RRSAF). JMP regions can run in DB/DC or DCCTL environments.
- Java Batch Processing (JBP) for non-message-driven Java applications. JBP applications run in an online batch mode and do not process input messages (similar to non-message-driven BMP applications), and can access DB2 data. JBP regions can run in DB/DC, DCCTL, or DBCTL environments.

Utility Regions

BMP and IFP regions can also be used for other types of work besides running application programs. BMPs can be used for HSSP processing, and IFPs can be used for Fast Path utility programs. For further discussion on these, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Batch Application Address Space

In addition to the dependent application address spaces discussed in “IMS Separate Address Spaces” on page 14 and “Application Dependent Regions” on page 16, IMS application programs that only use IMS Database Manager functions can be run in a separate z/OS address space, not connected to an IMS control region. This would normally be done for very long running applications that perform large numbers of database accesses or for applications that do not perform syncpoint processing. These batch applications can only access full-function databases.

This is similar to a BMP, in that the JCL is submitted through TSO or a job scheduler. However, all IMS code used by the application resides in the address space that the application is running in. The job executes an IMS batch region controller that then loads and calls the application. Figure 6 shows an IMS batch region.

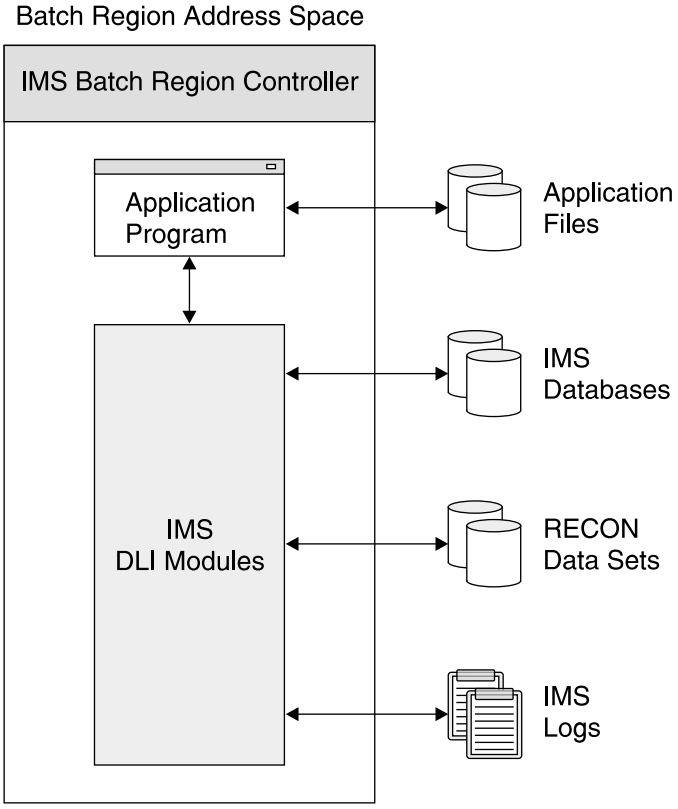

Figure 6. Structure of an IMS Batch Region

The batch address space opens and reads the IMS database data sets directly.

Attention: If there are requirements for other programs, either running under the control of an IMS control region or in other batch regions, to access the databases at the same time, then caution should be exercised to protect data integrity. See Chapter 8, “Data Sharing,” on page 83 for more information about how the data can be updated by multiple applications in a safe manner.

The batch address space writes its own separate IMS log. In the event of a program failure, it might be necessary to take manual action (for example, submit jobs to run IMS utilities) to recover the databases to a consistent point. With dependent application address spaces, this would be done automatically by the IMS control region. DBRC can be used to track the IMS logs and ensure that correct recovery action is taken in the event of a failure.

An application can be written so that it can run in both a batch and BMP address space without change. Some reasons you may want to change programs between batch and BMP address spaces include length of run time, need of other applications to access the data at the same time, and your procedures for recovering from application failures.

Internal Resource Lock Manager (IRLM)

The IRLM address space is only needed if you are going to use block-level or sysplex data sharing for the IMS databases. The IRLM address space is started before the IMS control region with the z/OS start command. The IMS control region, if the start-up parameters specify IRLM, connects to the IRLM specified on startup and will not complete initialization until connected.

There is one IRLM address space running on each z/OS system to service all IMS subsystems sharing the same set of databases. For more information on data sharing in sysplex environment, see:

- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*

IRLM is delivered as an integral part of the IMS program product, though as mentioned, you do not have to install or use it unless you need to perform block-level or sysplex data sharing.

IRLM is also the required the lock manager for DB2 UDB for z/OS.

Do **not** use the same IRLM address space for IMS and DB2 because the tuning requirements of IMS and DB2 are different and conflicting. The IRLM code is delivered with both the IMS and DB2 program products and interacts closely with both these products. Therefore, you might want to install the IRLM code for IMS and DB2 separately (that is, in separate SMP/E zones) so you can maintain release and maintenance levels independently. This can be helpful if you need to install prerequisite maintenance on IRLM for one database product, as it will not affect the use of IRLM by the other.

Running an IMS System

The procedures to run IMS address spaces are supplied by IBM. The procedures are in the IMS.PROCLIB data set. There are procedures for each type of region.

These procedures should be modified with the correct data set names for each IMS system. The following list contains the procedure member names (as found in IMS.PROCLIB) along with the type of region that each will generate:

Procedure Member Name	Region Name
IMS	DB/DC control region
DCC	DCCTL control region
DBC	DBCTL control region
DLISAS	DLI separate address space
DBRC	Database Recovery Control
DXRJPROC	Internal Resource Lock Manager (IRLM)
DFSMPR	Message processing region (MPR)
IMSBATCH	IMS batch processing region (BMP)
IMSFP	Fast Path region (IFP)
FPUTIL	Fast Path utility region
DLIBATCH	DLI batch region
DFSJBP	IMS Java batch processing (JBP) region
DFSJMP	IMS Java message processing (JMP) region
IMSRDR	IMS JCL reader region

Related Reading: For details of these and other procedures supplied in IMS.PROCLIB, see the “Procedures” chapter in the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Running Multiple IMS Systems

Multiple IMS systems can be run on a single z/OS image or on multiple z/OS images. One instance of an IMS system (control region and all associated dependent regions) is referred to as one IMS system. In many cases, these would be production and testing systems.

Running Multiple IMS Systems on One z/OS Image

The number of subsystems you can run on a single image of z/OS will depend on many factors. In most installations, you can run up to four IMS subsystems, although some installations run as many as eight small ones running concurrently. The number will vary depending on the size of each IMS system. The amount of z/OS common service area (CSA) required by each IMS is often one of the most limiting factors in the equation.

Each IMS subsystem should have unique VTAM ACB and IMSID names. The application dependent regions use the IMSID to connect to the corresponding IMS control region. If the dependent region starts and there is no control region running using that IMSID, the dependent region issues a message to the z/OS system console and then waits for a reply. Each IMS subsystem can have up to 999 dependent regions. However, there are other limiting factors, such as, storage limitations because of pool usage.

Running Multiple IMS Systems on Multiple z/OS Images

There are basically three ways to run multiple IMSs on multiple z/OS images. They are:

- Multiple Systems Coupling (MSC)

  MSC only supports IMS-to-IMS connections. For more information about MSC, see “Multiple Systems Coupling (MSC)” on page 116.
- Inter System Communications (ISC)

  ISC is another way to connect multiple subsystems. ISC is more flexible than MSC, in that ISC supports connections to IMS and other z/OS products, such as CICS. For more information about ISC, see “Intersystem Communications (ISC)” on page 117.
- Parallel Sysplex

  Running multiple IMSs in a Parallel Sysplex environment is a good way to balance workload, build scalability into your systems, and provide maximum availability. For more information on this topic, see “Parallel Sysplex” on page 25 and Chapter 31, “IMSplexes,” on page 337.

How IMS Uses z/OS Services

IMS is designed to make the best use of the features of the z/OS operating system. This includes:

- Running in multiple address spaces — IMS subsystems (except for IMS batch applications and utilities) normally consist of a control region address space, separate address spaces for system services, and dependent address spaces for application programs. Running in multiple address spaces gives the following advantages:
  - Maximizes use of CPUs when running on a multi-processor CPC. Address spaces can be dispatched in parallel on different CPUs.
  - Isolates the application programs from the IMS systems code. Reduces outages from application failures.
- Runs multiple tasks in each address space — IMS, particularly in the control region, creates multiple z/OS subtasks for the various functions to be performed. This allows other IMS subtasks to be dispatched by z/OS while one IMS subtask is waiting for system services
- IMS uses z/OS cross memory services to communicate between the various address spaces making up an IMS system. It also uses the z/OS CSA to store IMS control blocks that are frequently accessed by the address spaces making up the IMS system. This minimizes the overhead in running in multiple address spaces.
- IMS uses the z/OS subsystem feature — IMS dynamically registers itself as a z/OS subsystem. It uses this facility to detect when dependent address spaces fail, prevent cancellation of dependent address spaces.
- IMS can make use of an z/OS sysplex. Multiple IMS subsystems can run on the z/OS systems making up the sysplex and access the same IMS databases and the same message queue. This gives:
  - High availability — z/OS systems and IMS subsystems can be taken in and out of service without interrupting production.
  - High capacity — the multiple IMS subsystems can process far greater volumes than individual IMSs can.

Related Reading: For further details on sysplex data sharing and shared queues, see:

- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*

Transmission Control Protocol/Internet Protocol (TCP/IP)

IMS provides support for z/OS TCP/IP communications through a function called Open Transaction Manager Access (OTMA). Any TCP/IP application can have access to IMS by using OTMA. A related IBM product, IMS Connect for z/OS, uses the OTMA interface to connect IMS to Web servers.

Related Reading: For details on OTMA and IMS Connect for z/OS, see:

- *IMS Version 9: Open Transaction Manager Access Guide and Reference*
- *IMS Connect Guide and Reference*

Advanced Program-to-Program Communications (APPC)

IMS supports z/OS CPI communications interface, which defines the Logical Unit type 6.2 formats and protocols for program-to-program communication. IMS's support for APPC is called APPC/IMS.

APPC/IMS enables applications to be distributed throughout your entire network and to communicate with each other regardless of the underlying hardware.

Related Reading: For more information about IMS's support for APPC, see "Advanced Program-to-Program Communication (APPC)" on page 114.

Resource Access Control Facility (RACF)

IMS was developed prior to the introduction of RACF® (part of the Security Server for z/OS) and other security products. As a result, IMS initially incorporated its own security mechanisms to control user access to the various IMS resources, transactions, databases, and so forth. This security was controlled by a number of means. A number of security exits were provided. Also, a series of bitmaps defined users and their access to resources. This is referred to as a security matrix. These are load modules produced by the IMS Security Maintenance utility.

With the introduction of RACF, IMS was enhanced to use RACF (or equivalent product) for controlling access to IMS resources. It is now possible to use the original IMS security features, the RACF features, and combinations of these.

Recommendation: Use RACF because it provides more flexibility and the Security Maintenance utility will not be supported in future releases of IMS.

The normal features of RACF can also be used to protect IMS data sets, both system and database.

Related Reading: For further information about protecting IMS resources, see Chapter 24, "IMS Security," on page 253. For complete information regarding IMS and security, see the security chapter in the *IMS Version 9: Administration Guide: System*.

Resource Recovery Services (RRS)

With z/OS comes a system resource recovery platform. Resource Recovery Services (RRS) is the sync-point manager, coordinating the update and recovery of

multiple protected resources. RRS controls how and when protected resources are committed by coordinating with the resource managers, such as IMS, that have registered with RRS.

RRS provides a system resource recovery platform such that applications executing on z/OS (such as IMS) can have access to local and distributed resources and have system-coordinated recovery management of these resources. The support includes:

- A sync-point manager to coordinate the two-phase commit process
- Implementation of the SAA® Commit and Backout callable services for use by application programs
- A mechanism to associate resources with an application instance
- Services for resource manager registration and participation in the two-phase commit process with RRS
- Services to allow resource managers to express interest in an application instance and be informed of commit and backout requests
- Services to enable resource managers to obtain system data to restore their resources to consistent state
- A communications resource manager (called APPC/PC for APPC/Protected Conversations) so that distributed applications can coordinate their recovery with participating local resource managers

Related Reading: For more information about how IMS uses RRS, see the *IMS Version 9: Administration Guide: System*.

Parallel Sysplex

A Parallel Sysplex environment in z/OS is a combination of hardware and software components that enable sysplex data sharing. In this context, data sharing means the ability for sysplex member systems and subsystems to store data into, and retrieve data from a common area known as a coupling facility. In short, a Parallel Sysplex can have multiple CPCs and multiple applications (like IMS) that can directly share the workload.

IMS exploits the z/OS Parallel Sysplex environment to enable a more dynamic, available, manageable, scalable, and well performing environment for database, transaction, and systems management.

In a Parallel Sysplex environment, you can run multiple IMS subsystems that share message queues and databases. This sharing enables workload balancing and insulation from individual IMS outages. If one IMS in the sysplex fails, others continue to process the workload, so the enterprise is minimally affected.

Related Reading: For more information on this topic, see Chapter 30, “Introduction to Parallel Sysplex,” on page 315 and Chapter 31, “IMSplexes,” on page 337.

Chapter 3. Setting Up and Running IMS

This chapter contains general information about installing, defining, and operating IMS.

The following sections are covered in this chapter:

Installing IMS

The IMS installation task includes:

- The initial activity of installing IMS on your z/OS system using the SMP/E installation process.
- Verifying the installation using the IMS-supplied Installation Verification Program (IVP).
- A variety of other activities (such as, initially tailoring your IMS system, customizing your IMS system, defining resources to IMS, and so forth).

Most IMS installations involve migrating an existing version of IMS to a newer version rather than installing just a new instance of IMS. With this scenario, there are migration, coexistence, and maintenance steps and issues to consider as part of the installation process. The migration issues are usually version specific.

Related Reading: For the details of installing, verifying the installation, tailoring, and migrating IMS, see:

- *Program Directory for Information Management System Version 9*
- *IMS Version 9: Release Planning Guide*
- *IMS Version 9: Installation Volume 1: Installation Verification*
- *IMS Version 9: Installation Volume 2: System Definition and Tailoring*

For more information about customizing IMS, see Chapter 23, "Customizing IMS," on page 245 and the *IMS Version 9: Customization Guide*.

IMS Installation Verification Program (IVP)

The Installation Verification Program (IVP) facility, which comes with IMS, is an ISPF application that is used to verify the majority of IMS features and functions of a newly installed IMS. The IVP uses a sample IMS system to perform this verification.

The IVP provides guidance for performing a combination of the following jobs and tasks (depending on your environment):

- Allocating data sets

- Defining the characteristics of an IMS system through the process of system definition
- Establishing IMS interfaces to z/OS and VTAM
- Preparing the IMS system
- Performing an initial program load (IPL) of z/OS
- Preparing the IVP system and IMS applications
- Initializing the IVP system and running IMS applications

You must define the IMS system and you must establish the interface between your IMS system and z/OS before you can run IMS.

Related Reading: For complete information about the IVP, see *IMS Version 9: Installation Volume 1: Installation Verification*.

Defining an IMS System

Before you can use IMS TM or IMS DB, you must define the elements and functions that make up the IMS system. These include:

- Databases
- Application programs
- Terminals

IMS provides macros and procedures that enable you to define your system. IMS also provides exits (strategic places in IMS's logic flow) that enable you to customize what happens at that particular point in the processing.

All optional features of IMS, including what type of control region is required (DB/DC, DBCTL, DCCTL), must be defined to IMS prior to using it.

Almost all programs, databases, transactions, and terminals (unless the ETO feature is used) within IMS must also be predefined to IMS. The Extended Terminal Option (ETO) is a separately-priced feature that allows you to dynamically define terminals while IMS is running.

You can either customize the sample IMS system that was verified with the IVP (see "IMS Installation Verification Program (IVP)" on page 27) or copy the sample IMS system and customize the copy to satisfy your installation's needs.

Related Reading: For more information about the IMS definition process, see Chapter 22, "The IMS System Definition Process," on page 231 and the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*. For more information about the IMS exits, see Chapter 23, "Customizing IMS," on page 245.

Defining IMS Security

Setting up security for the IMS system is part of the system definition process. IMS itself has security functions built in and also has the ability of providing more extensive security by employing user-written exit routines, a security product (like RACF), or both.

Some of the things that can be secured are:

- Sign-on attempts
- Transactions

- Programs
- Commands
- Resources

Related Reading: For more information about IMS security, see Chapter 24, “IMS Security,” on page 253.

IMS Startup

The following two sections describe the types of IMS starts that can be performed for an IMS system and the methods for starting IMS-associated regions. These sections are:

- “Types of IMS System Starts”
- “Starting IMS-Associated Regions” on page 30

Types of IMS System Starts

This section describes the common types of IMS starts that can be performed. These IMS starting types are applicable to both IMS TM and IMS DB.

Cold start

An IMS control region cold start is done the first time you start the system. During cold start, IMS formats (initializes) the message queues, the dynamic log, and the restart data sets.

Automatic restart

With an automatic restart, IMS will startup, using either normal restart or emergency restart, depending on the previous shutdown status.

If the last IMS shutdown was successful, then a normal restart will be performed. If the last IMS shutdown was abnormal (from an abend), then IMS will automatically perform an emergency restart.

For most installations, automatic restart should be the default.

Normal restart

Normal restart or warm start is done from a previous normal IMS termination. The message queues are preserved in this way.

Emergency restart

In case of failure, IMS is restarted with the logs active at the time of failure. Restart processing will back-out the full-function database changes of incomplete transactions. The output messages inserted into the message queues by these incomplete transactions will be deleted.

After back-out, the input messages are re-enqueued, and the pending output messages are (re)-transmitted. Application programs must be restarted manually. If a BMP or JBP application was active at the time of failure, it must be resubmitted by using z/OS job management. If the BMP uses the XRST/CHKP calls, it must be restarted from its last successful checkpoint. In this way, missing or inconsistent output is avoided.

Other restarts

There are numerous other types of manual starts possible with IMS, each with unique requirements. For detailed information about these other types of restarts, see the *IMS Version 9: Operations Guide* and the *IMS Version 9: Command Reference*.

Starting IMS-Associated Regions

The following sections discuss how the various IMS-associated regions are started.

Address Spaces

All the address spaces can either run as a started task or as a job. In most cases the IMS control region and the separate address spaces will run as started tasks. The application dependent regions are run as either jobs or started tasks.

When a control region is started, it will issue a z/OS START command to start the DLISAS and DBRC regions, as shown in the following example:

```
/START xxxxxxxx,PARM=(DLS,imsid)
/START xxxxxxxx,PARM=(DRC,imsid)
```

The xxxxxxx fields are the procedure names. These commands will start the DLISAS and DBRC regions respectively.

Starting Application Dependent Regions

IMS will not automatically start application dependent regions. There are several ways start these regions.

- The Time Control Option (TCO) of IMS can issue `/START REGION` commands. TCO is a time-initiated IMS facility that can generate any valid operator IMS input.
- Some forms of automation programs can issue either IMS or z/OS start commands.
- A job scheduling system can submit jobs based on time or the notification of IMS being started. The notification can be in the form of automated messages.

Message Processing Regions

IMS MPR regions are normally started by an IMS start region command as shown below:

```
/START REGION xxxxxxxx
```

The xxxxxx fields are the member names in a library. The members contain the jobs for the MPR regions. The IMSRDR procedure is used if the MPRs are jobs. The IMSRDR procedure is customized to point to the correct library to find the job JCL. If you are running multiple IMS subsystems on a single z/OS system, they normally use a different version of the IMSRDR procedure each pointing at different libraries. The procedure name is specified on the IMSCTF macro in the system definition.

Related Reading: For the details of the IMSRDR procedure or the IMSCTF macro, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*, for more information.

Fast Path Application Regions

Fast Path application (IFP) regions are normally started in a similar fashion as MPR regions and follow the same rules and procedures.

Batch Message Processing Regions

These regions are almost always started outside of IMS. Most BMPs are scheduled at appropriate times to meet application requirements. As long as the IMS control region is available, the BMPs can be run. BMPs can execute even though there are no MPRs running at the time.

Java Non-Message Driven Application Processing Region

This region, which is called a Java batch processing (JBP) region, is similar to a BMP region. The JBP region is started in the same manner as a BMP region. The default job name is IMSJBP.

Java Message-Driven Application Processing Region

This region, which is called a Java message processing (JMP) region, is similar to an MPP region. The JMP region is started in the same manner as an MPP region. The default job name is IMSJMP.

IMS Logging

While IMS is running, IMS records information about everything necessary to restart the system if a hardware or software failure is encountered. The event information is recorded on a online log data set (OLDS).

When an OLDS is filled, or some other event switches the OLDS, it is archived to the system log data set (SLDS). There are a finite number of OLDS data sets (although this number can be dynamically changed), that are pre-allocated and redefined to the IMS Control Region. The OLDS are cycled through during the duration of the control region. There can be an infinite number of SLDS, which are created and allocated as needed.

Related Reading: For more information about the IMS logging function, see Chapter 25, “IMS Logging,” on page 257.

IMS Utilities

To help run, fine tune, and monitor IMS, there are a lot of utility programs that come with the product. These utilities help you:

- Generate and maintain IMS system control blocks
- Make online changes to the IMS system
- Allocate, monitor, and recover the IMS log data sets
- Analyze system performance
- Generate and maintain the Message Format Service (MFS)
- Maintain multiple IMS systems
- Maintain time-controlled operations
- Define and maintain databases
- Reorganize databases
- Make backup copies of databases
- Recovery databases

Some of these utilities are discussed in the following sections:

- “Overview of the Reorganization Process” on page 88
- “Generating MFS Control Blocks” on page 220
- Chapter 24, “IMS Security,” on page 253
- “Archiving” on page 259
- “Using IMS System Log Utilities” on page 277
- “Running Recovery-Related Utilities” on page 288

Related Reading: For information about all of the IMS utilities, see:

- The *IMS Version 9: Utilities Reference: Database and Transaction Manager*
- The *IMS Version 9: Utilities Reference: System*

IMS Recovery

There are also a number of tools and features available with IMS to help in recovery scenarios:

Extended Recovery Facility (XRF)
With XRF, you can have an alternate IMS standby system ready to take over within the same site. For more information about XRF, see “Overview of Extended Recovery Facility (XRF)” on page 308.

Remote Site Recovery (RSR)
With RSR, you can recover the complete IMS system (or systems) very quickly at another site when complete site disasters occur. For more information about RSR, see “Overview of Remote Site Recovery (RSR)” on page 308.

Fast Database Recovery (FDBR)
The FDBR provides a solution to sysplex customers who need quick access to shared database resources that might otherwise be locked by a failed IMS until the failed system is restarted.

In a sysplex data sharing environment, multiple IMS subsystems can access a single, shared database resource. If one of the IMS subsystems fails while it has a lock on the database, the other IMS subsystems must wait until the failed IMS is restarted and the locks on the resource are released. Because an emergency restart can take a significant amount of time, waiting for a full restart is unacceptable in situations that require continuous availability of database resources.

FDBR creates a separate IMS control region (the Fast DB Recovery region) that monitors an IMS subsystem, detects failure, and recovers any database resources that are locked by the failed IMS, making them available for other IMS subsystems.

Related Reading: For more information about FDBR, see “Fast Database Recovery” on page 319.

IMS Database Recovery Facility (DRF)
DRF allows you to recover multiple database data sets and Fast Path areas in an IMS DBCTL or DB/DC environment simultaneously. It simplifies the database recovery process by eliminating the need to run separate recovery jobs for each database data set that requires recovery. Recovery using DRF reduces the time that broken databases and areas are unavailable by processing input data in parallel and recovering multiple database data sets and areas simultaneously.

DRF is one of the IMS Tools offered by IBM.

Related Reading: For more information about the this and other IMS tools, see Chapter 29, “IBM IMS Tools,” on page 311.

IMS Shutdown

There are also several different ways of shutting down IMS, depending on what type of control region you are running (DB/BC, DBCTL, or DCCTL), and whether or not the IMS message queues are required following the next startup.

A common sequence for shutting down the entire online IMS system is:

1. For an IMS DB/DC or DCCTL environment, stop the transactions. For an IMS DBCTL environment, disconnect from the Coordinated Controller (CCTL).
2. Stop the dependent regions.
3. Stop the control region.
4. For an IMS DB/DC or DBCTL environment, stop the Internal Resource Lock Manager (IRLM).
5. For a shared-queues environment, shut down the Common Queue Server (CQS), if it has not been shut down already.
6. For an IMSplex environment (see Chapter 31, “IMSplexes,” on page 337), shut down the IMS components that participate in the IMSplex and then shut down the Common Service Layer.

Related Reading: For more information about:

- The process for shutting down an IMSplex, see “Operating an IMSplex” on page 340.
- The process of shutting down an IMS, see the *IMS Version 9: Operations Guide*.
- The commands involved in shutting down an IMS, see the *IMS Version 9: Command Reference*.

Part 2. IMS Database Manager

Chapter 4. Overview of IMS DB

The IMS Database Manager can be ordered and installed with or without the IMS Transaction Manager.

The following sections are covered in this chapter:

Functions of the IMS Database Manager

A database management system (DBMS) provides facilities for business application transaction or process to access stored information. The role of a DBMS is to provide the following functions:

- Allow access to the data for multiple users from a single instance of the data.
- Control concurrent access to the data so as to maintain integrity for all updates.
- Minimize hardware device and operating systems access method dependencies.
- Reduce data redundancy by maintaining only one instance of the data.
- Interface with the operating system and manage the physical location of the data. Application programs that access and manipulate the data do not need to know where the data actually resides.

The IMS Database Manager provides a central point for the control and access to application data. IMS provides a full set of utility programs to provide all these functions within the IMS product.

Implementation of IMS Databases

IMS DB supports multiple forms of enterprise databases, so that varied application requirements can be met by exploiting whichever database technology best suits the users' requirements.

The types of databases are:

IMS Full-Function Databases
: Used to be known as DL/I databases.

IMS DEDBs
: Data Entry databases, often referred to as Fast Path databases.

IMS MSDBs
: Main storage databases, another type of Fast Path databases. MSDB functionality has been superseded by the Virtual Storage Option (VSO) of the DEDB, so MSDBs are not described in this book, and you are advised not to use them.

IMS uses a hierarchical model for its database, described in more detail in Chapter 5, "Overview of the IMS Hierarchical Database Model," on page 41. The data stored in the IMS databases is organized internally using a number of internal IMS access methods. Each of these access methods suits certain types of access

to the database. The choice of the appropriate access method is discussed in detail Chapter 6, “Implementing the IMS Database Model,” on page 51.

No single technology is the best option for all applications — even though industry trends might suggest that an organization standardize on only one database type. To do this, for example, to say that you wish to use only relational database technology (DB2), would preclude consideration of other technologies that, for suitable applications, would make massive savings in processing or application development costs — far in excess of the small additional cost of introducing DEDBs to your organization.

Each of the database implementations supported by IMS has different characteristics:

Full-function databases
: Full-function databases provide a hierarchically structured database, that can be accessed by record or sequentially, and by other sequences that were planned and provided for when the database was designed. Full-function databases are limited in size to 4GB or 8GB per data set unless a portioning database product is used

DEDBs
: DEDBs are particularly suited for use where large databases, or very low processing costs are required, or when particularly high data availability or very high performance is required. DEDBs were originally part of a separately priced, optional feature. This results in the documentation and code being separate from that for the full-function databases.

Note: DB2 UDB for z/OS, as compared to IMS DB, provides well for unstructured or unplanned access to data and so provides flexibility in the support of future application requirements. However, DB2 usually has a significantly higher processing cost than any IMS database.

The IMS access methods are underpinned by the use of operating system access methods to store data on disk storage. The software access methods which IMS makes use of are:

- VSAM (Virtual Storage Access Method) - VSAM is a z/OS access method.
- OSAM (Overflow Sequential Access Method) - OSAM is an IMS data management access method that combines selected characteristics of z/OS BSAM (Basic Sequential Access Method) and BDAM (Basic Direct Access Method).

Full-Function Databases

Full-function databases are designed to support most types of database requirements. These can be used in a wide variety of applications. Most IMS applications make use of full-function databases unless there are specific requirements for one of the other types of databases. The major characteristics of full-function databases are:

- Small or large databases.
- Access to records through unique or non-unique keys.
- Many types of segments (up to 15 levels allowed).
- Records can be stored in key sequence, but it is not a requirement.

One function associated with full-function databases is called *multiple data set groups*. With multiple data set groups, you can put some segments in a database record in data sets other than the primary data set. This can be done without destroying the hierarchic sequence of segments in a database record. One reason to use multiple data set groups is to accommodate the differing needs of your applications. By using multiple data set groups, you can give an application program fast access to the segments in which it is interested. The application program simply bypasses the data sets containing unnecessary segments. Up to 10 data set groups can be defined for a single full-function database.

Prior to IMS Version 7, full-function databases are limited in size: the maximum data set size is limited to 4 GB for VSAM and 8 GB for OSAM. IMS Version 7 introduced High Availability Databases (HALDBs) to address this size limit.

HALDB allows full-function databases to grow much larger. A HALDB is a partitioned full-function database. Partitioning a database allows the use of smaller data sets that are easier to manage. Multiple partitions decrease the amount of unavailable data if a partition fails or is taken offline.

HALDB allows the grouping of full-function database records into sets of partitions that are treated as a single database while permitting functions to be performed independently for each partition. Each HALDB partition has the same capacity limit as a non-HALDB database. Like a non-HALDB database, each partition can consist of up to 10 data sets, however the number of data sets selected will then apply to all the partitions in that HALDB. This allows a large amount of data to be contained in a single partition.

Related Reading: For more information about HALDBs, see “PHDAM and PHIDAM” on page 60.

Fast Path Databases

The Data Entry Database (DEDB) was designed to support particularly intensive IMS database requirements, initially in the banking industry, for:

- Large databases containing millions of records, extending well beyond the original 4 GB database limits of full-function databases
- Access to each database record by a key field
- Lower processing costs for each database record and update than are required for full-function databases
- The capability to support higher transaction workloads than full-function databases can sustain, while maintaining per-transaction cost advantages
- Improved availability, with reduced requirements for database outage, especially for database maintenance activities such as database reorganizations
- Lower processing costs for particular types of processing, where data is inserted online and retrieved in batches for further processing, and eventually deleted in batches
- The possibility of eliminating transaction-related I/O from database processing

All the above requirements were satisfied, while maintaining the conventional DL/I application interface so that application programming for DEDBs is little different from that for full-function databases.

Data in IMS and DB2

Some business applications require that the data be kept in both IMS database and DB2 databases. One such scenario is a high-performance production application that works with the data in a hierarchical IMS database and a business decision support application that works with the same data in a relational DB2 database. Production applications running in IMS TM can update data stored in a DB2 database as well as data stored in an IMS database, but the coordinating of these updates can be complex to ensure all updates are consistently applied.

IBM IMS DataPropagator™ can automatically duplicate data from IMS databases to DB2 UDB for z/OS tables.

Related Reading: For more information about the IBM IMS DataPropagator, see Chapter 29, “IBM IMS Tools,” on page 311 or go to www.ibm.com/software/data/dpropnr.

Chapter 5. Overview of the IMS Hierarchical Database Model

IMS uses a hierarchical model as the basic method of storing data. Unlike the relational model used by DB2, which was the result of theoretical work, the hierarchical model was arrived at as a pragmatic way of storing and retrieving data quickly while using as few computer resources as possible.

In this model, the individual entity types are implemented as segments in a hierarchical structure. An entity is something that can be uniquely defined and something you could collect substantial information about.

The hierarchical structure is based on the relationship between the entities and the access paths required by the applications.

IMS uses the term database slightly differently to its use in other DBMSs. In IMS, a database is commonly used to describe the implementation of one hierarchy, so that an application would normally access a large number of IMS databases. Compared to the relational model, an IMS database is approximately equivalent to a table.

A database segment definition defines the fields for a set of segment instances similar to the way a relational table defines columns for a set of rows in a table. In this regard, segments relate to tables, and fields in a segment relate to columns in a table, as illustrated by comparing Figure 7 to Figure 8 on page 42. Similarly, an instance of a segment in a database corresponds to a row (or tuple) in a table. Note that if a segment does not have a unique key, the corresponding relational table should be viewed as having a generated unique key added to its column (field) list. Also note that the tables are implicitly joined.

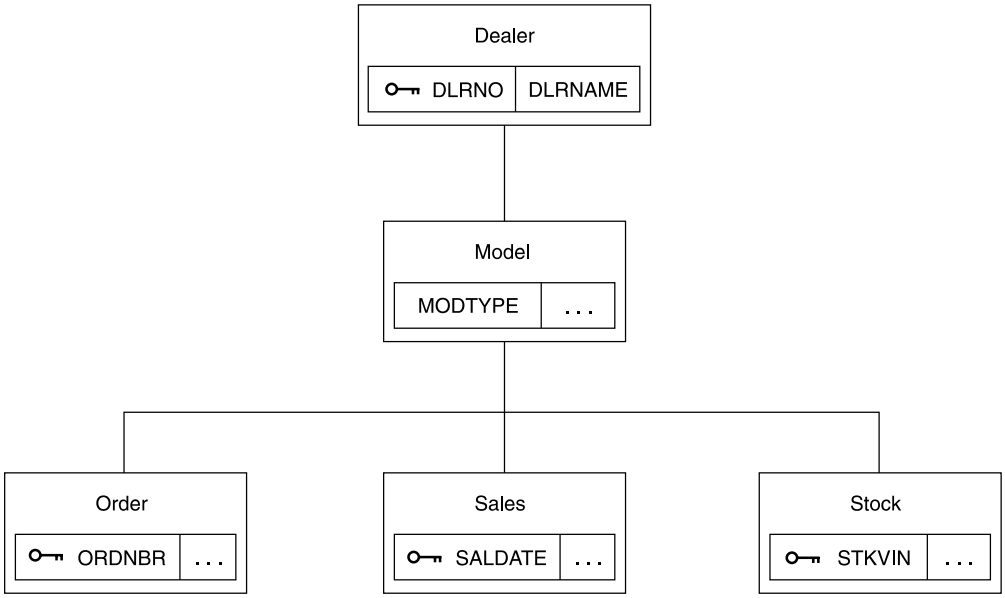

Figure 7. Example of a Hierarchical Dealership Database

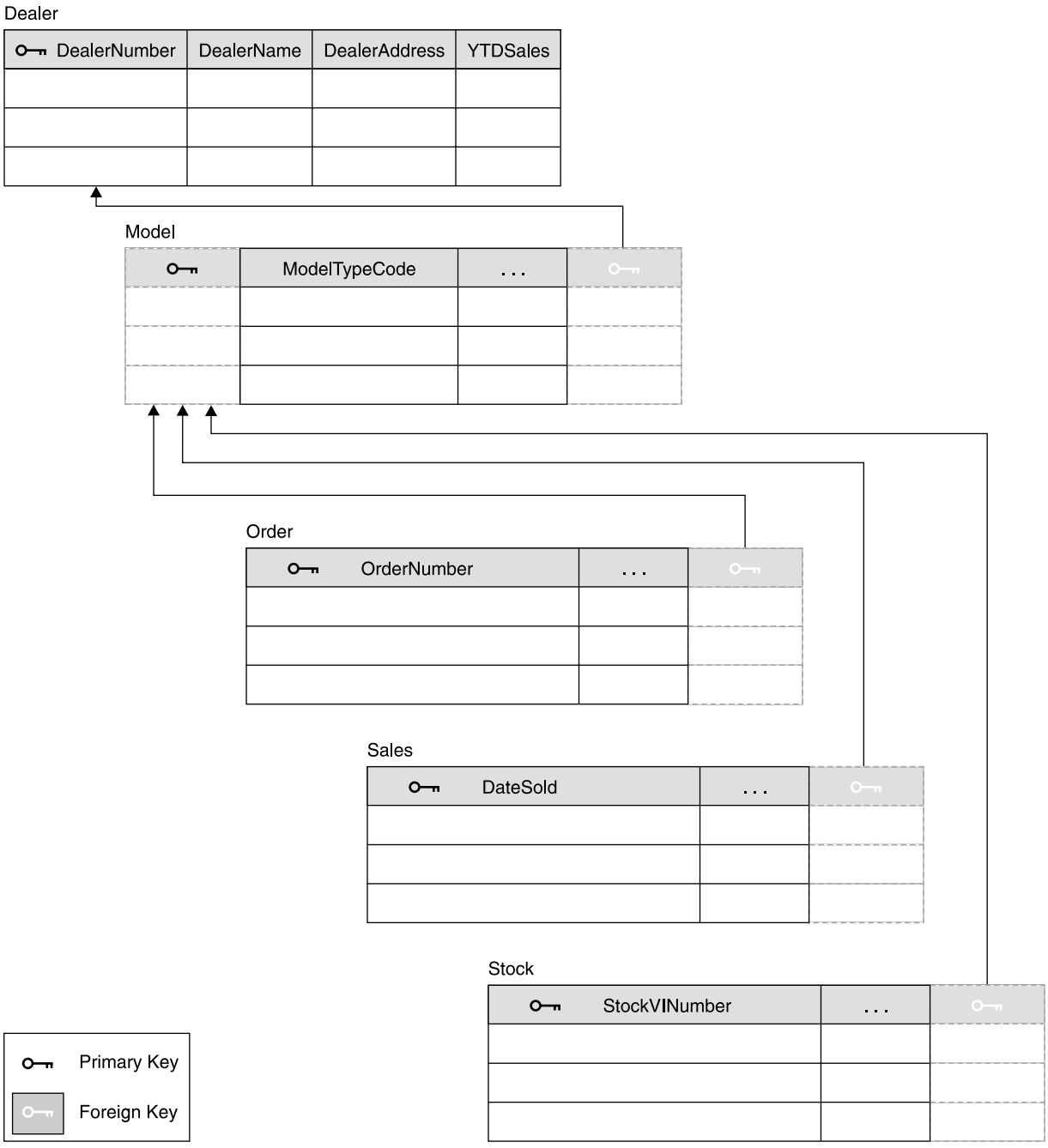

Figure 8. Relational Representation of the Dealership Database

The hierarchical data structure in Figure 9 on page 43 describes the data as seen by the application program. It does not represent the physical storage of the data. The physical storage is of no concern to the application program.

The basic building element of a hierarchical data structure is the parent/child relationship between segments of data, also illustrated in Figure 9 on page 43.

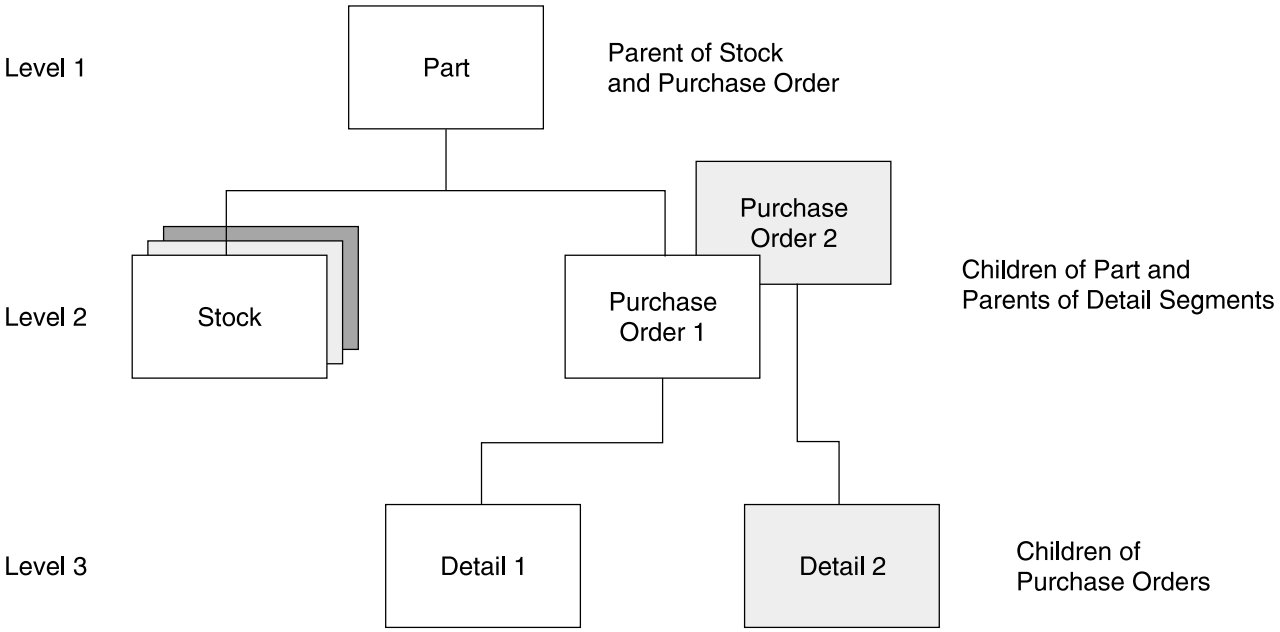

Figure 9. Hierarchical Data Structure

Each occurrence (or instance) of a parent segment is associated with 0 or more occurrences of a child segment. Each child segment occurrence is associated with one, and only one, occurrence of a parent segment.

Sometimes it is necessary to distinguish between a segment type, that is, the kind of segment; and the segment occurrence, that is, the particular instance of its contents and location.

As shown in Figure 9, a parent can have several child segment types. Also, a child segment can, at the same time, be a parent segment; that is, it can have children below it. The segment with no parent segment (the one at the top) is called the root segment.

All the parent and child occurrences for a given root segment are grouped together in a database record. The collection of all of the database records with the same root and hierarchical structure (in Figure 9, each PART segment with its dependent STOCK, ORDER, and DETAIL segments) is an IMS database (the PARTS database).

Only one segment can appear at the first level in the hierarchy, but multiple segments can appear at lower levels in the hierarchy. For example, multiple STOCK and ORDER segments can exist for one PART segment. Since each dependent segment in the hierarchy has only one parent, or immediate superior segment, the hierarchical data structure is sometimes called a tree structure. Each branch of the tree is called a hierarchical path. A hierarchical path to a segment contains all consecutive segments from the top of the structure down to that segment.

Through the concept of program sensitivity, IMS allows a program to be restricted to “seeing” only those segments of information that are relevant to the processing being performed. For example, an inventory program could be written to see only the PART and STOCK segments of the database record shown in Figure 9. The program need not be aware of the existence of the ORDER segment.

IMS allows a wide variety of data structures. The maximum number of different segment types is 255 in a single database. A maximum of 15 segment levels can be defined in a hierarchical data structure. There is no restriction on the number of occurrences of each segment type, except as imposed by physical access method limits.

Basic Segment Types

The following list contains a detailed description of the various segment types and their interrelations within a hierarchical data structure. See Figure 9 on page 43 and Figure 10 while reading these description.

- The segment on top of the structure is the root segment. Each root segment normally has a key field that serves as the unique identifier of that root segment, and as such, of that particular database record (for example, the part number).
- A dependent segment relies on the segments above it in the hierarchy for its full meaning and identification.
- A parent/child relationship exists between a segment and its immediate dependents.
- Different occurrences of a particular segment type under the same parent segment are twin segments.
- Segment occurrences of different types under the same parent are sibling segments.

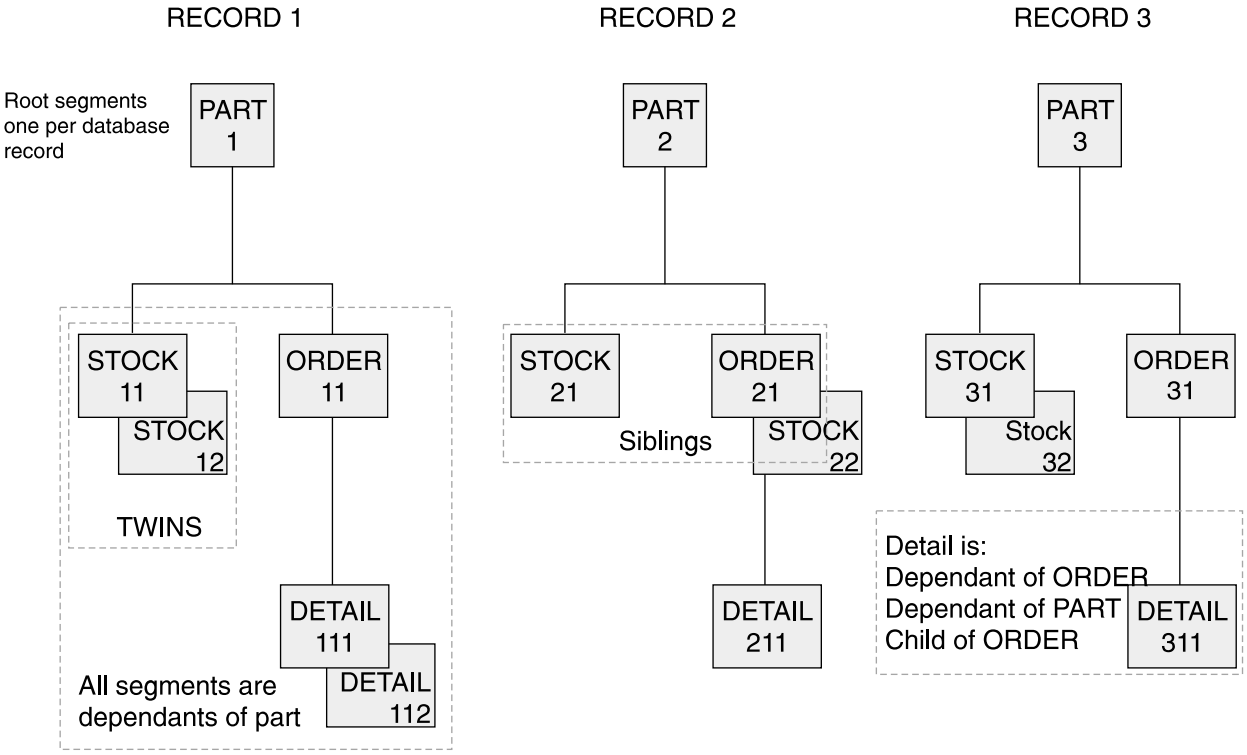

Figure 10. Segment Types and Their Relationships

Sequence Fields and Access Paths

To identify and to provide access to a particular database record and its segments, IMS uses sequence fields. Each segment normally has one field denoted as the sequence field. The sequence fields should be unique in value for each occurrence of a segment type below its parent occurrence. However, not every segment type need have a sequence field defined. Particularly important is the sequence field for the root segment, because it serves as the identification for the database record. Normally, IMS provides a fast, direct access path to the root segment of the database record based on this sequence field. This direct access is extended to lower level segments if the sequence fields of the segments along the hierarchical path are specified, too.

Note: The sequence field is often referred to as the key field, or simply the key.

In Figure 10 on page 44, an example of an access path would be the PART, ORDER and DETAIL segments. It must always start with the root segment. This is the access path as used by IMS. The application program, however, can directly request a particular DETAIL segment of a given ORDER of a given PART in one single DL/I call by specifying a sequence field value for each of the three segment levels.

In addition to the basic hierarchical data structure discussed so far, IMS provides two additional methods of defining access paths to a database segment. These are:

Logical relationships
Logical relationships allow a logical view to be defined of one or more physical databases.To the application this will look like a single database.

Secondary indexes
Secondary indexes give an alternate access path for full-function databases, by using a root or dependent segment to the database record in one physical database.

Both provide a method for an application to have a different access path to the physical databases. They are defined to IMS in addition to the basic hierarchical structure already defined. The logical relationships and secondary indexes are automatically maintained by IMS, transparent to the application.

You should only use these extra facilities if there are strong application and/or performance reasons for doing so. Both involve additional overheads. The following two sections (“Logical Relationships,” and “Secondary Indexing” on page 48) describe these facilities in more detail and indicate where you might wish to use them.

Logical Relationships

Through logical relationships, IMS provides a facility to interrelate segments from different hierarchies. In doing so, new hierarchical structures are defined that provide additional access capabilities to the segments involved. These segments can belong to the same database or to different databases. A new database can be defined called a logical database. This logical database allows presentation of a new hierarchical structure to the application program. Notice that although the connected physical databases could constitute a network data structure, the application data structure still consists of one or more hierarchical data structures.

For example, given the entities and relationships illustrated in Figure 11, it may have been decided that, based on the applications most common access paths, the data should be implemented as two physical databases, with hierarchies as shown in Figure 12 on page 47. However, there are some reasons why other applications need to use the relationship between the PART and order DETAIL (reasons for wanting to do this are discussed in the following figures). So a logical relationship is to be built between PART and DETAIL.

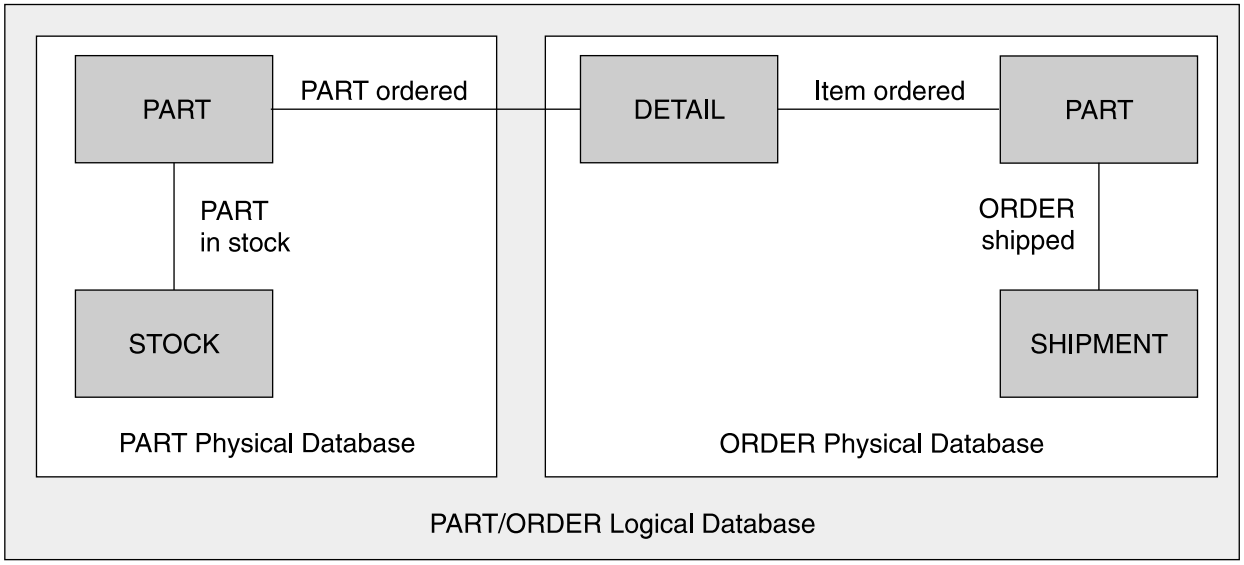

Figure 11. Example of Logical and Physical Databases

The basic mechanism used to build a logical relationship is to specify a dependent segment as a logical child, by relating it to a second parent, the logical parent.

In Figure 12 on page 47, the logical child segment DETAIL exists only once, yet participates in two hierarchical structures. It has a physical parent, ORDER, and logical parent, PART. The data in the logical child segment and in its dependents, if any, is called intersection data.

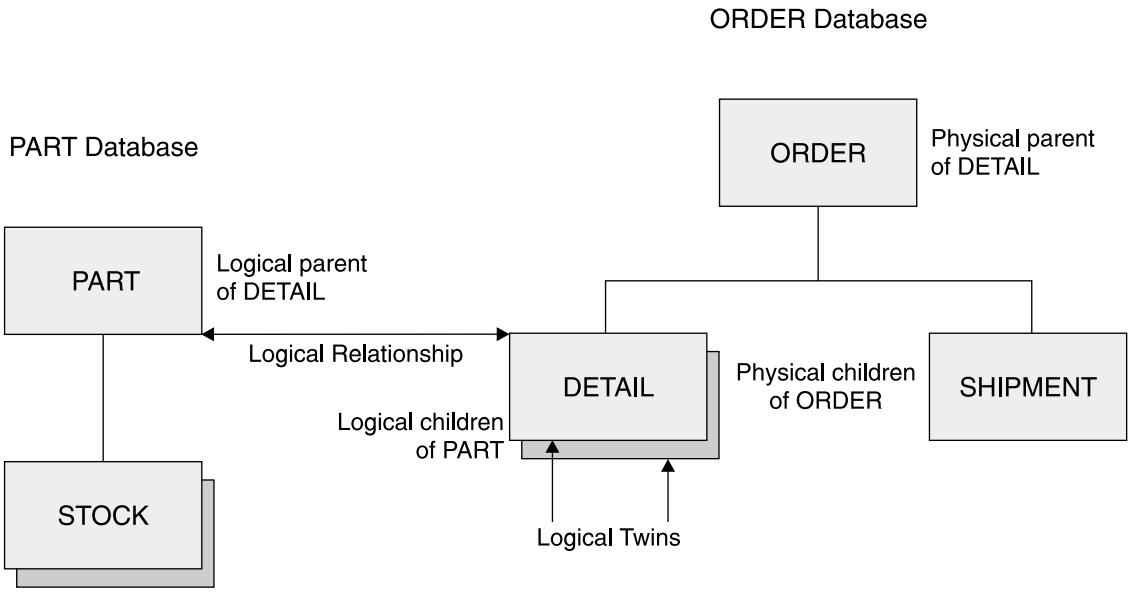

Figure 12. Two Logically Related Physical Databases: PART and ORDER

By defining two additional logical databases, two new logical data structures shown in Figure 13 can be made available for application program processing, even within one single program.

The DETAIL/PART segment in Figure 13 is a concatenated segment. It consists of the logical child segment plus the logical parent segment. The DETAIL/ORDER segment in Figure 13 is also a concatenated segment, but it consists of the logical child segment plus the physical parent segment. Logical children with the same logical parent are called logical twins, for example, all DETAIL segments for a given PART segment. As can be seen in Figure 12, the logical child has two access paths. One via its physical parent, the physical access path, and one via its logical parent, the logical access path. Both access paths are maintained by IMS and can be concurrently available to one program.

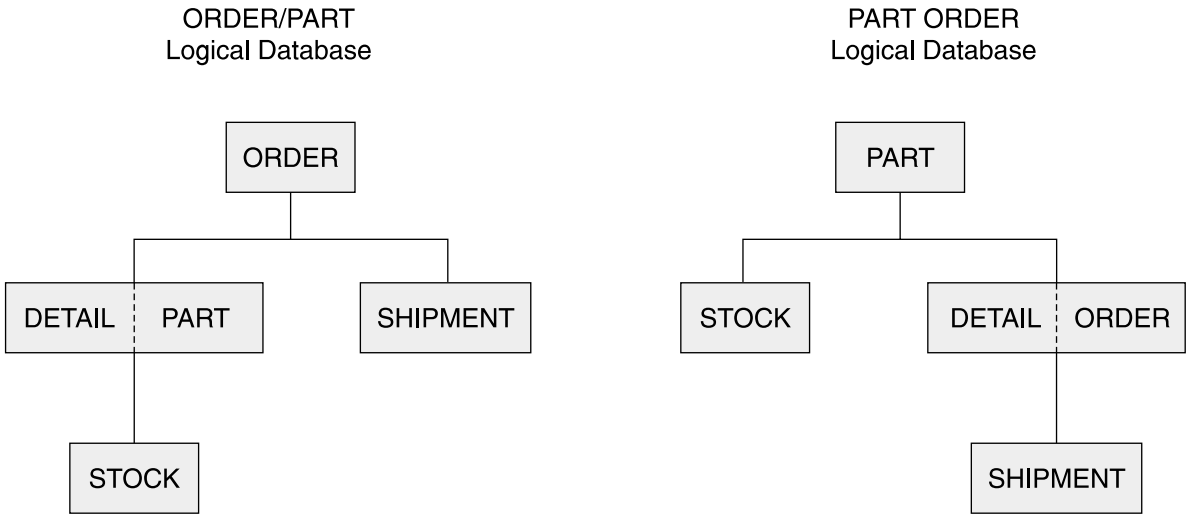

Figure 13. Two Logical Databases After Relating the PARTS and ORDER Databases

Some reasons you may want to use logical relationships are:

- They provide an alternate access path for the application. For example, they allow (depending on pointer choice) an application to have direct access from a segment in one physical database to a lower level segment in another physical database, without the application having to access the second physical database directly and read down through the hierarchy.
- They provide an alternate hierarchical database structure for an application so that different applications, or parts of applications, can have a view of the physical databases that most closely matches that application's view of the data.
- They can make IMS enforce a relationship between two segments in two physically separate databases (that is, it will preserve referential integrity). You can define the relationship such that a logical parent cannot be deleted if it still has logical children, and a logical child cannot be added it there is no logical parent. For example, referring to Figure 13 on page 47, you could define the relationship such that no order DETAIL could be inserted if there were no corresponding PART, and no PART could be deleted if there were still order DETAILs for that part. Any application attempting to do this would have the database call rejected by IMS.

Potential disadvantages in using logical relationships are:

- There are performance overheads in maintaining the pointers used in the logical relationships. Every time a segment participating in a logical relationship is updated, the other segment (in another physical database) that participates in the relationship may need to be updated. This can result in an appreciable increase in physical I/Os to auxiliary storage.
- When a database needs to be reorganized, except with some very limited pointer choices, all other databases that are logically related must be reorganized at the same time, as the pointers used to maintain the logical relationships rely on the physical position of the segments in that database, which can be altered by the reorganization.

Before choosing to use logical relationships, you need to carefully weigh up the performance and administrative overheads against the advantages of using logical relationships.

Related Reading: For further details on implementing logical relationships see *IMS Version 9: Administration Guide: Database Manager*.

Secondary Indexing

IMS provides additional access flexibility with secondary index databases. Each secondary index represents a different access path to the database record other than via the root key. The additional access paths can result in faster retrieval of data. For example, the PART and ORDER segments in Figure 14 on page 49 could be retrieved based on the order number in the ORDER segment, if an index were defined for that field. Once an index is defined, IMS will automatically maintain the index if the data on which the index relies changes, even if the program causing that change is not aware of the index.

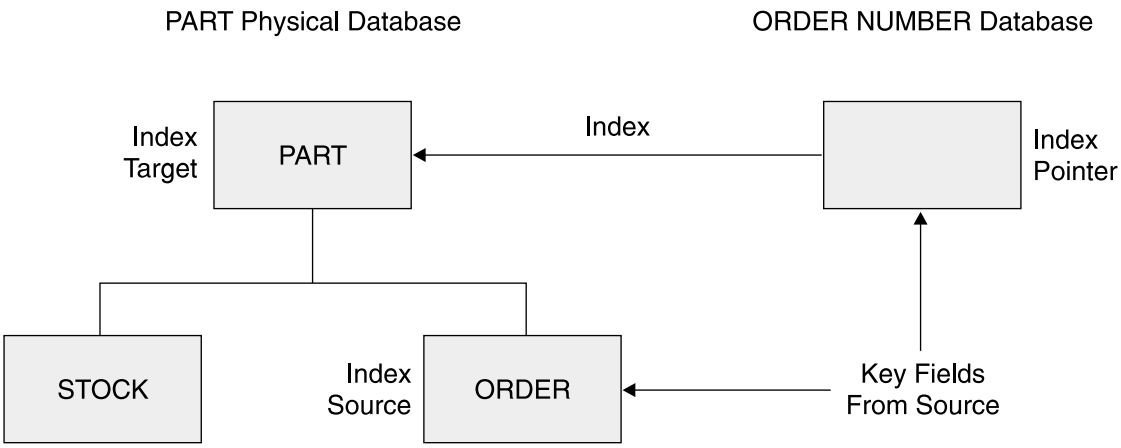

Figure 14. A Database and Its Secondary Index Database

The secondary index is implemented by defining a secondary Index database to IMS. This contains segments that point to the segment in the main physical database that contains the key values the constitute the secondary index key. As this Index database is itself a physical database, it can be accessed independently by applications.

The segments involved in a secondary index are depicted in Figure 14.

The index source segment contains the source fields on which the index is constructed. For example, for the ORDER# segment:

- The index pointer segment is the segment in the index database that points to the index target segment. The index pointer segments are ordered and accessed based on the field contents of the index source segment (for example, the order number). This is the secondary processing sequence of the indexed PARTS database. There is, in general, one index pointer segment for each index source segment, but multiple index pointer segments can point to the same index target segment.
- The index target segment is the segment that becomes initially accessible from the secondary index. It is in the same hierarchical record as the index source segment and is pointed to by the index pointer segment in the index database. Quite often, but not necessarily, it is the root segment.
- The index source and index target segment may be the same, or the index source segment may be a dependent of the index target segment as shown in Figure 14.

The secondary index key (search field) is made up of one to five fields from the index source segment. The search field does not have to be a unique value, but IBM strongly recommends you make it a unique value to avoid the overhead in storing and searching duplicates. There are a number of fields that can be concatenated to the end of the secondary index search field to make it unique:

- A subsequence field, consisting of one to five more fields from the index source segment. This is maintained by IMS but, unlike the search field, cannot be used by an application for a search argument when using the secondary index.
- A system defined field that uniquely defines the index source segment: the /SX variable.
- A system defined field that defines the concatenated key (the concatenation of the key values of all of the segment occurrences in the hierarchical path leading to that segment) of the index source segment: the /CX variable.

Another technique that can be used with secondary indexes is sparse indexing. Normally IMS will maintain index entries for all occurrences of the secondary index source segment. However, it is possible to cause IMS to suppress index entries for some of the occurrences of the index source segment. You might wish to do this if you were only interested in processing segments that had a non-null value in the field. In the example in Figure 14 on page 49, say that the ORDER had a field set in it to indicate the order could not be fulfilled immediately, but needed to be back ordered. You could define a secondary index including this field, but suppress all entries that did not have this field set, giving rapid access to all back orders. As a general rule, only consider this technique if you expect 20% or less of the index source segments to be created. The suppression can be done either by specifying that all bytes in the field should be a specific character (NULLVAL parameter) or by selection with the Secondary Index Maintenance exit routine.

Some reasons for using secondary indexes are:

- Quick access, particularly random access by online transactions, by a key other than the primary key of the database.
- Access to the index target segment without having to negotiate the full database hierarchy (particularly useful if the index target segment is not the root segment). This is similar to using logical relationships, but provides a single alternate access path into a single physical database. If this is all that is required, then a secondary index is the better technique to use.
- Ability to process the index database separately. For example, a batch process might need to process only the search fields.
- A quick method of accessing a small subset of the database records by using a sparse index.

Potential disadvantages in using secondary indexes are:

- The performance overheads in updating the secondary index database every time any of the fields making up the search field in the index source segment is updated or when the index source segment is inserted or deleted.
- The administrative overheads in setting up, monitoring, backing up, and tuning the secondary index database.
- When the database containing the index source segment is reorganized, the secondary index must also be re-built because the pointers used to maintain the connection between the source segment and the secondary index database rely on the physical position of the source segment in the database, which can be altered by the reorganization.

As with logical relationships, consider carefully whether the benefits of using a secondary index outweigh the performance and administrative overheads.

RELATED READING: For details on implementing secondary indexes, see *IMS Version 9: Administration Guide: Database Manager*.

Chapter 6. Implementing the IMS Database Model

Chapter 5, “Overview of the IMS Hierarchical Database Model,” on page 41 described the logical model for IMS databases. This chapter looks at how this model is physically implemented using the IMS Database Manager and z/OS services.

Application programs interface with IMS through functions provided by the IMS DL/I application programming interface (API). This is true for both IMS DB and IMS TM (see Figure 15). The following sections only address the functions relevant to IMS DB.

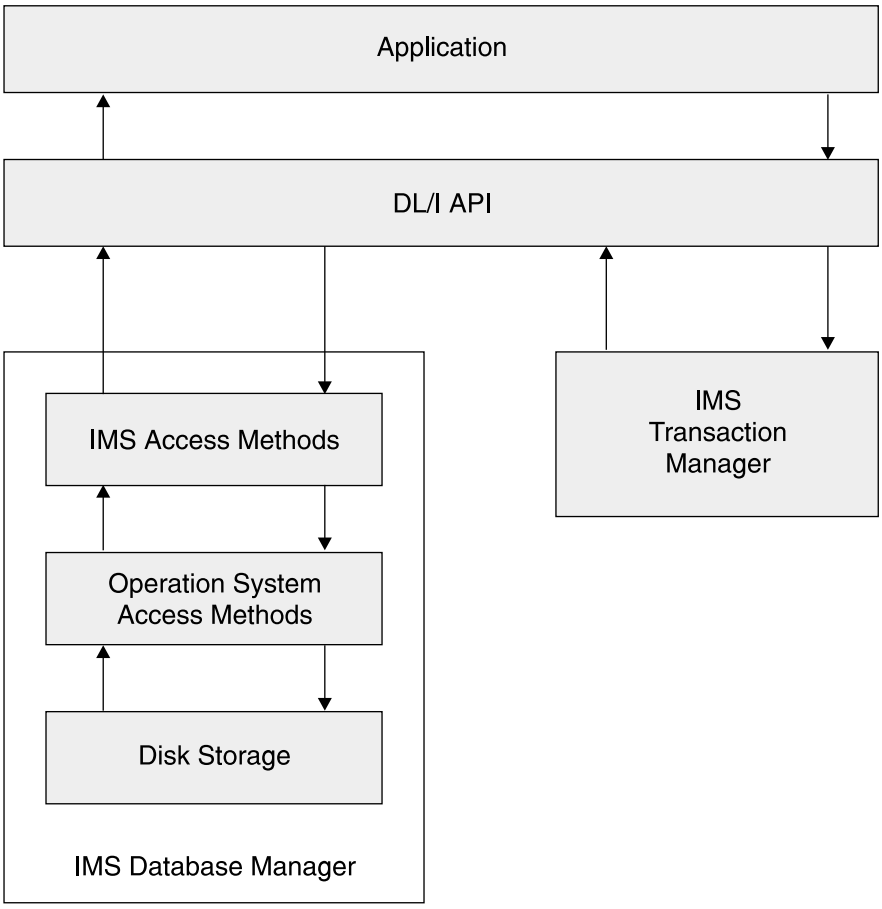

Figure 15. Elements of the Physical Implementation

The individual elements that make up the database, segments, and database records are organized using different IMS access methods. The choice of access method can influence the functionality available to your application, the order in which data is returned to the application, the functionality available to the application, and the performance the application receives from IMS DB.

Underlying the IMS access methods, IMS uses VSAM or OSAM to store the data on DASD and move the data between the DASD and the buffers in the IMS address space, where the data is manipulated.

The structure of the IMS databases, and a program's access to them, is defined by a set of IMS control blocks:

- The database description block (DBD)
- The program specification block (PSB)
- The application control block (ACB)

These are coded as sets of source statements that then have to be generated into control blocks for use by IMS DB and the application.

Segments, Records, and Pointers

As described in Chapter 5, "Overview of the IMS Hierarchical Database Model," on page 41, a segment is used to represent one entity, or grouping of related fields. In IMS, unlike DB2 or many other DBMSs, it is not mandatory to define all the fields to IMS. It is only necessary to define the segment as being long enough to contain all the application data to be stored. The only fields you must define to IMS are those you need to use for identifying and searching for segments. Specifying non-search fields (field-level sensitivity) is optional.

In addition to the application data, each segment will also contain control information used by IMS. The control information is placed at the beginning of the segment in a segment prefix. Figure 16 shows the layout of a segment with the prefix and application data portions. The prefix is automatically maintained by IMS and is not accessible to the application. The control information in the prefix consists of various flags, descriptive fields (segment type code and delete byte), and pointers to implement the hierarchical structure and access paths. The contents of the prefix will vary, depending on the IMS access method and options chosen when the database is defined.

Figure 16. Example of a Typical Segment Layout

Prefix					**Data**
Segment Type Code	Delete Byte	RBA Pointer	RBA Pointer	RBA Pointer	Application Data

These pointers consist of the relative offset (number of bytes) of the segment being pointed at, from the start of the data set being used to store the data. This is the relative byte address (RBA). For example, a root segment would contain pointer fields in the prefix for, at a minimum all of the dependent segment types under the root. IMS will automatically define the minimum set of pointers to maintain the hierarchical structure. The database designer has the option to specify additional pre-defined types of pointers above those necessary for the minimum hierarchical structure. This pointer selection can influence the performance of applications using the databases. Figure 17 on page 53 shows database segments with their pointers.

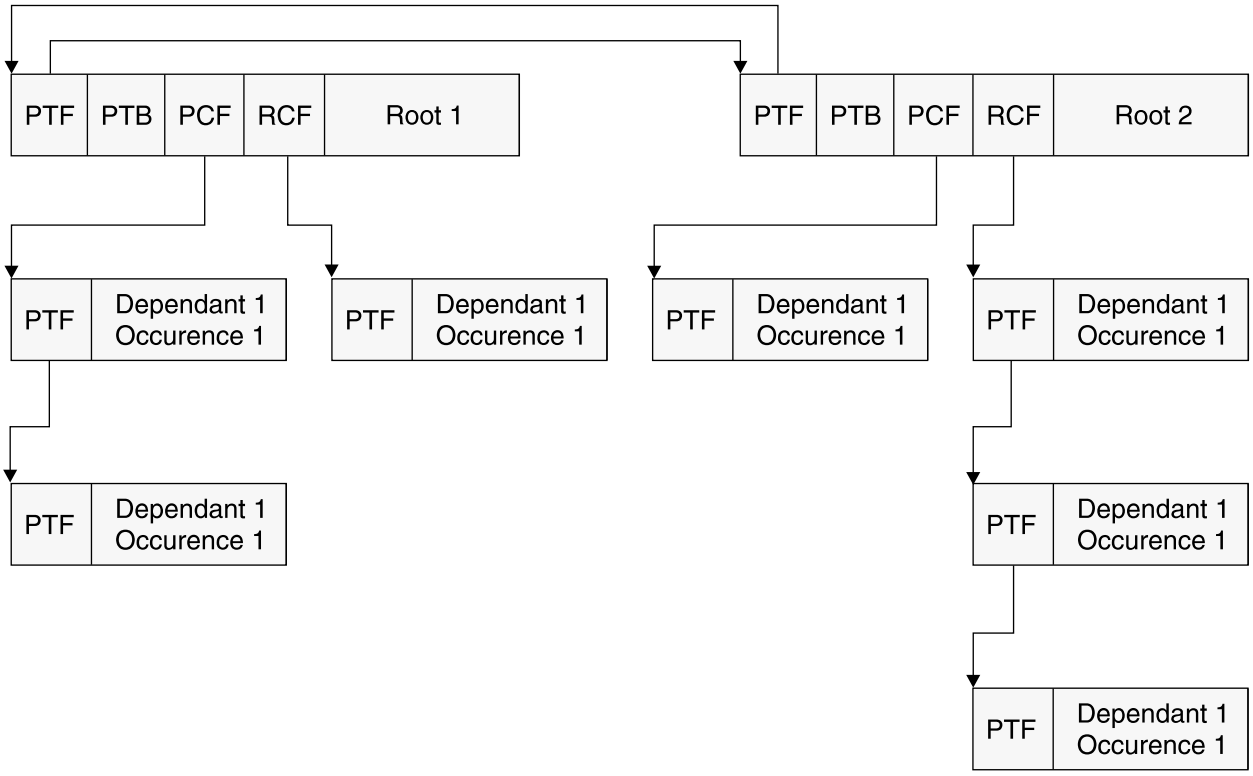

Figure 17. Database Record and Pointers

IMS Hierarchic Access Methods

There are different IMS access methods you can use to organize and store the data segments and records. The choice of which access method to use should be made after a careful analysis of the access requirements of the applications (for example, the functionality available to the application, the order in which segments are returned to the application, database performance considerations).

Access methods (VSAM or OSAM) can, in general, be changed during reorganization without affecting application programs. Choose the access method carefully because the access method is one of the most critical performance factors. Database types (HIDAM, HDAM, HISAM) cannot be changed during reorganization without affecting the application.

The following list describes the most commonly used database organizations.

Database Type	Organization
HDAM	Hierarchical Direct Access Method
PHDAM	Partitioned Hierarchical Direct Access Method
HIDAM	Hierarchical Index Direct Access Method
PHIDAM	Partitioned Hierarchical Index Direct Access Method
SHISAM	Simple Hierarchical Index Sequential Access Method
HISAM	Hierarchical Index Sequential Access Method

GSAM Generalized Sequential Access Method

DEDB Data Entry Database

The z/OS access methods, VSAM and OSAM, that underlay the IMS access methods, are mentioned in this section, but are discussed in more detail in the following sections.

The three major IMS access methods are:

- Hierarchical Direct — Consisting of the Hierarchical Direct Access Method (HDAM) and the Hierarchical Indexed Direct Access Method (HIDAM). Both of these methods are described in “HDAM” on page 55, and “HIDAM” on page 58.

  HDAM and HIDAM databases, which have many similarities, are referred to as HD databases. These HD databases can be partitioned using either the HALDB Partition Definition utility (%DFSHALDB) or DBRC commands. After you partition an HDAM database, it becomes a partitioned hierarchical direct access method (PHDAM) database. After you partition a HIDAM database, it becomes a partitioned hierarchical indexed direct access method (PHIDAM) database. PHDAM and PHIDAM databases are generically referred to as High Availability Large Databases (HALDBs).

  For information about HALDBs, see “PHDAM and PHIDAM” on page 60.
- Hierarchical Sequential — Consisting of the Hierarchical Sequential Access Method (HSAM) and the Hierarchical Indexed Sequential Access Method (HISAM). These are less used today, as the HD access methods have a number of advantages. A short description of them, together with their limitations, is given in “HSAM and HISAM” on page 66.

  There are also simple variations of HSAM and HISAM, namely SHSAM and SHISAM. These are also briefly described in “HSAM and HISAM” on page 66.
- Data Entry DataBase (DEDB) — DEDB has characteristics that make it suitable for high performance and high availability applications. However, the application must be specifically designed and written to make use of these characteristics. It is described in detail in “DEDB” on page 61.

The Hierarchical Direct (HD) and Hierarchical Sequential (HS) databases are full-function databases, and DEDB databases are referred to as Fast Path databases. Because of its original development as a separately orderable feature, Fast Path functions are normally described in separate sections or chapters in the IMS manuals.

In addition, there are two more IMS access methods that provide additional functionality:

- Index Databases — These are used to physically implement secondary indexes and primary indexes for HIDAM and PHIDAM databases. For more information, see “Index Databases” on page 61.
- Generalized Sequential Access Method (GSAM) — This is used to extend the restart/recovery facilities of IMS Database Manager to non-IMS sequential files being processed by IMS batch programs and BMPs. These files can also be accessed directly by using z/OS access methods. For more information, see “GSAM” on page 65.

Exception: Most types of application regions and access a majority of the database organization types. The exceptions are:

GSAM GSAM databases cannot be accessed from MPP, JMP, JBP, or CICS regions.

DEDB DEDB databases cannot be accessed from BMP regions.

HDAM

See Figure 18 on page 56 for the following discussion. An HDAM database normally consists of one VSAM ESDS or OSAM data set. To access the data in an HDAM database, IMS uses a randomizing module. The randomizing module is used by IMS to compute the address for the root segment in the database. This address consists of the relative number of a VSAM control interval (CI) or OSAM block within the data set and the number of an anchor point within that block. Anchor points are located at the beginning of each CI or block. They are used for the chaining of root segments that randomize to that CI or block. All chaining of segments is done using a 4-byte address. This address, the relative-byte address (RBA), is the byte that the segment starts at relative to the start of the data set.

A general randomizing module, DFSHDC40, is supplied with IMS. This is suitable for most applications. The *IMS Version 9: Customization Guide* describes this module. It also gives details about modifying this module or developing your own randomizing routines.

The VSAM ESDS or OSAM data set is divided into two areas:

- The root addressable area. This is the first number of CIs or blocks in the data set. You define it in your database definition (DBD).
- The overflow area is the remaining portion of the data set. The overflow area is not explicitly defined, but is the remaining space in the data set after space is allocated for the root addressable area.

The root addressable area (RAA) is used as the primary storage area for segments in each database record. IMS will always attempt to put new and updated segments in the RAA. The overflow area is used when IMS is unable to find enough space for a segment being inserted in the RAA.

IMS uses a number of techniques to distribute free space within the RAA to allow future segments to be inserted in the most desirable block. Because database records will vary in length, a the *bytes* parameter in the RMNAME= keyword (in the DBD) is used to control the amount of space used for each database record in the root addressable area. Note that this limitation only applies if the segments in the record are inserted at the same time. The *bytes* parameter limits the number of segments of a database record that can be consecutively inserted into the root addressable area. When consecutively inserting a root and its dependents, each segment is stored in the root addressable area until the next segment to be stored will cause the total space used to exceed the specified number of bytes.

The total space used for a segment is the combined lengths of the prefix and data portions of the segment. When exceeded, that segment and all remaining segments in the database record are stored in the overflow area. It should be noted that the value of the bytes parameter only controls segments consecutively inserted in one database record. Consecutive inserts are inserts to one database record without an intervening call to process a segment in a different database record.

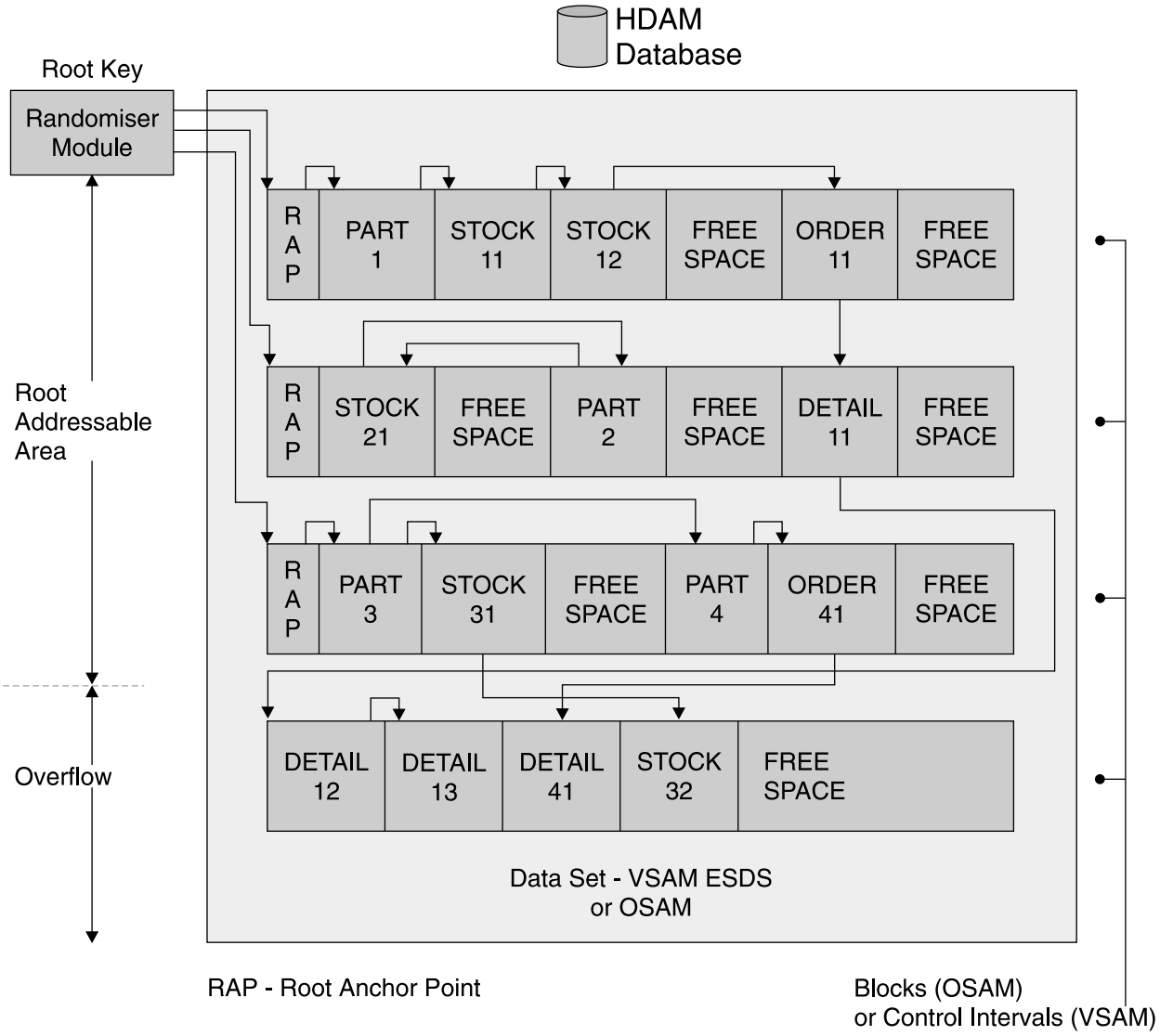

Figure 18. HDAM Database in Physical Storage

When you initially load HDAM databases, you can specify that a percentage of the space in each block should be left for subsequent segments to be inserted. This freespace will allow subsequent segments to be inserted close to the database record they belong to. This freespace percentage is specified on the DBD. You can also specify in the DBD that a percentage of blocks in the data set are left empty, but you should not do this with HDAM databases, as this will result in IMS randomizing segments to a free block, then placing them in another block. This would result in additional I/O (the block they randomize to, plus the block they are in) each time the segment is retrieved. You should analyze the potential growth of the database to enable you to arrive at a figure for this free space.

When IMS is inserting segments, it uses the HD space search algorithm to determine which control interval (CI) block to put the segment in. This attempts to minimize physical I/Os while accessing segments in a database record by placing the segment in a CI/block as physically close as possible to other segments in the database record. The HD space search algorithm is described in the chapter, “Designing Full-Function Databases”, in the *IMS Version 9: Administration Guide: Database Manager*.

In addition to organizing the application data segments in an HDAM database, IMS also manages the freespace in the data set. As segments are inserted and deleted, areas in the CI/blocks become free (in addition to the freespace defined when the database is initially loaded). IMS space management allows this free space to be re-used for subsequent segment insertion. To enable IMS to quickly determine which CI/blocks have space available, IMS maintains a table (bit map) that indicates which CI/blocks have a large enough area of contiguous free space to contain the largest segment type in the database. Note that if a database has segment types with widely varying segment sizes, even if the CI/block has room for the smaller segment types, it would be marked as having no free space if it cannot contain the largest segment type. The bit map consists of one bit for each CI/block, set on (1) if space is available in the CI/block, set off (0) if space is not available in the CI/block. The bit map is in the first (OSAM) or second (VSAM) CI/block of the data set and occupies the whole of that CI/block. Figure 19 illustrates the free space management.

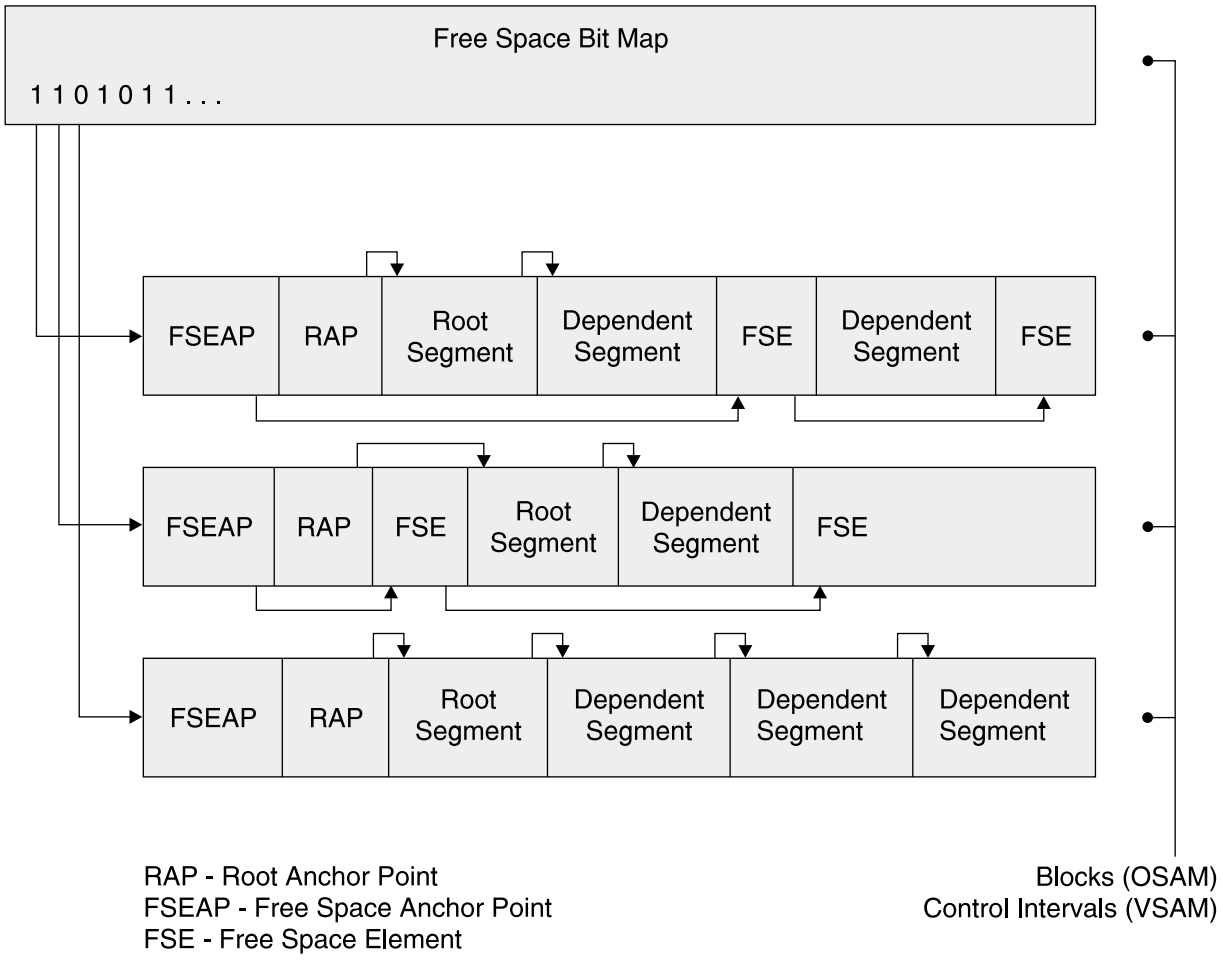

Figure 19. HDAM Database Free Space Management

Within the CI/block itself, IMS maintains a chain of pointers to the areas of freespace. These are anchored off a Free Space Element Anchor Point (FSEAP). This contains the offset, in bytes from the start of the CI/Bock, to the first Free Space Element (FSE), if freespace exists. Each area of freespace greater than 8 bytes contains a FSE containing the length of the freespace, together with the offset from start of CI/block to the next FSE.

All management of free space and application segments in the data sets is done automatically by IMS and is transparent to the application. You only need to be aware of these because of the performance and space usage implications.

Related Reading: A full description of the HDAM internal organization is given in the chapter on Choosing a Database Type in *IMS Version 9: Administration Guide: Database Manager*.

The principle features of the HDAM access method are:

- Fast random access to the root segments, via the randomizer
- Quick access to segments in a database record, as IMS attempts to store them in the same, or physically near, CI/block
- Automatic re-use of space after segment deletions
- Can have non-unique root segment keys

The principle weaknesses of the HDAM access method are:

- It is not possible to access the root segments sequentially, unless you create a randomizing module that randomizes into key sequence, or incur the overhead of creating and maintaining a secondary index
- It is slower to load than HIDAM, unless you sort the segments into randomizer sequence (for example by writing user exits for the sort utility that call the randomizing module)
- It is possible to get poor performance if too many keys randomize to the same anchor point

HIDAM

A HIDAM database in DASD is actually comprised of two physical databases that are normally referred to collectively as a HIDAM database, see Figure 20 on page 59. When defining each through the DBD, one is defined as the HIDAM primary index database and the other is defined as the main HIDAM database. In the following discussion the term “HIDAM database” refers to the main HIDAM database defined through DBD.

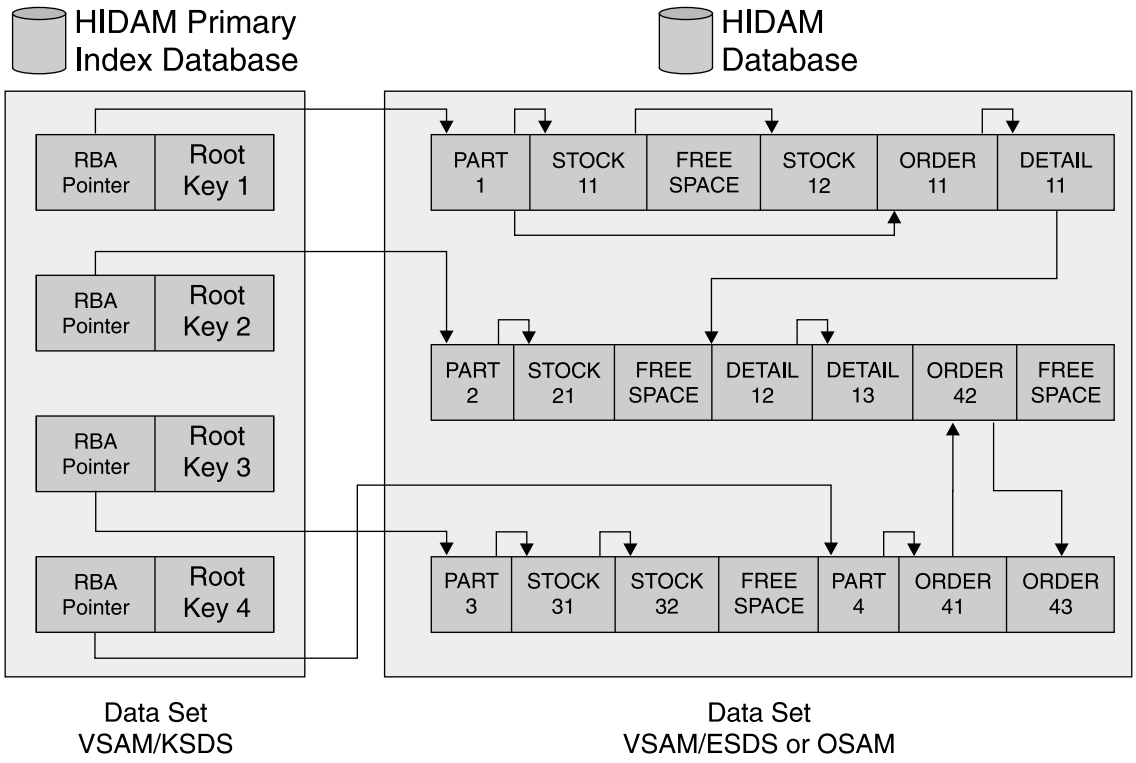

Figure 20. HIDAM Database in Physical Storage

The main HIDAM database is similar to an HDAM database. The main difference is in the way root segments are accessed. In HIDAM, there is no randomizing module, and normally no RAPs. Instead, the HIDAM primary index database takes the place of the randomizer in providing access to the root segments. The HIDAM primary index is an indexed sequential file (VSAM KSDS) that contains one record for each root segment, keyed on the root key. This record also contains the pointer (RBA) to the root segment in the main HIDAM database.

The HIDAM primary index database is used to locate the database records stored in a HIDAM database. When a HIDAM database is defined through the DBD, a unique sequence field must be defined for the root segment type. The value of this sequence field is used by IMS to create an index segment for each root segment (record in the KSDS). This segment in the HIDAM primary index database contains, in its prefix, a pointer to the root segment in the main HIDAM database.

When the HIDAM database is initially loaded, the database records are loaded into the data set in root key sequence. Providing root anchor points are not specified, reading the database in root key sequence will also read through the database in the physical sequence the records are stored in on the DASD. If you are processing the databases in key sequence like this, and regularly inserting segments and new database records, you should specify sufficient freespace when the database is initially loaded so that the new segments/records can be physically inserted adjacent to other records in the key sequence.

Related Reading: For a full description of the HIDAM internal organization, see the chapter on “Choosing a Database Type” in *IMS Version 9: Administration Guide: Database Manager*.

Free space in the main HIDAM database is managed in the same way as in an HDAM database. IMS keeps track of the free space using Free space elements anchor points. When segments are inserted, the HD free space search algorithm is used to locate space for the segment. The HIDAM primary index database id processed as a normal VSAM KSDS, and, consequently, a high level of insert/delete activity will result in CI/CS splits, which may require the index to be reorganized.

When the HIDAM database is initially loaded, free space can be specified as a percentage of the CI/blocks to be left free, and as a percentage of each CI/block to be left free. This is specified on the DBD.

The principle advantages of the HIDAM access method are:

- Ability to process the root segments and database records in root key sequence
- Quick access to segments in a database record, as IMS attempts to store them in the same, or physically near, CI/block
- Automatic re-use of space after segment deletions
- Ability to reorganize the HIDAM primary index database in isolation from the main HIDAM database (but NOT the other way round)

The principle weaknesses of the HIDAM access method are:

- Longer access path, compared to HDAM, when reading root segments randomly by key. There will be at least one additional I/O to get the HIDAM primary index record, before reading the block containing the root segment (excluding any buffering considerations)
- Extra DASD space for the HIDAM primary index
- If there is frequent segment insert/delete activity, the HIDAM primary database will require periodic reorganization to get all database records back to there root key sequence in physical storage

PHDAM and PHIDAM

PHDAM databases are partitioned HDAM databases and PHIDAM databases are partitioned PHDAM databases. Figure 21 illustrates a logical view of an HDAM and a PHDAM database.

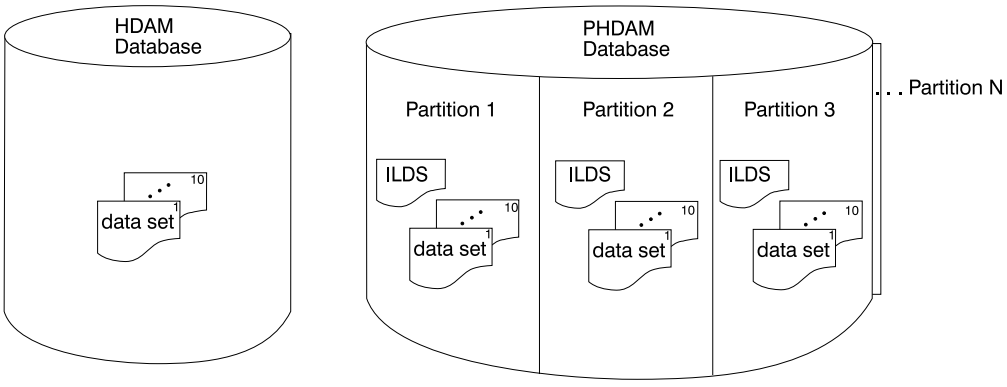

Figure 21. A Logical View of an HDAM and a PHDAM Database

HDAM and HIDAM databases are limited in size because segments of the same type must be in the same data set with the maximum data set size limited to 4 GB for VSAM and 8 GB for OSAM. HALDBs allows IMS databases to grow much

larger. Partitioning a database allows the use of smaller elements that are easier to manage. Multiple partitions decrease the amount of unavailable data if a partition fails or is taken offline.

HALDB allows the grouping of IMS database records into sets of partitions that are treated as a single database while permitting functions to be performed independently for each partition. Each HALDB partition has the same capacity limit as an IMS non-HALDB database. Like an IMS non-HALDB database, each partition can consist of up to 10 data sets; however the number of data sets selected will then apply to all the partitions in that HALDB. This allows a large amount of data to be contained in a single partition. HALDBs can contain up to 1001 partitions.

Each partition must have an indirect list data set (ILDS). The ILDS is a VSAM KSDS data set. It is the repository for indirect pointers. These pointers eliminate the need for updating logical relationship or secondary index pointers after a reorganization. An ILDS contains indirect list entries (ILEs), which are composed of keys and data. The data parts of ILEs contain direct pointers to the target segments.

Index Databases

Index databases are used to implement secondary indexes, and the primary index of HIDAM and PHIDAM databases. The index database is always associated with another HD database. It cannot have an existence by itself. It can, however, be processed separately by an application program.

The Index database consists of a single VSAM KSDS (Key Sequenced Data Set). Unlike the VSAM ESDSs used by IMS, which are processed at block (Control Interval) level, the index database is processed as a normal indexed file. IMS uses the normal VSAM access method macros to access it.

The records in the KSDS contain the fields that make up the key, and a pointer to the target segment. For a secondary index, the pointer can be direct (relative byte address of the target segment) or symbolic (the complete, concatenated key of the target segment). For a HIDAM primary index, it is always direct.

As the indexes are a normal VSAM KSDS (and no relative address are used for data in the index database) they can be processed using the normal VSAM Access Method Services (IDCAMS). For example you could use the REPRO function to copy the database and remove CI/CA splits or resize the data set, without having to perform any other IMS reorganization.

DEDB

The DEDB implementation of the IMS hierarchical database model is broadly the same as the IMS HDAM access method. However, there are important differences:

- The implementation of the IMS access method onto the operating system access method data sets is different (and appreciably more complicated) than with HDAM. This is done to provide the additional features offered by DEDBs.
- There are various restrictions on facilities available with this access method, again a trade-off for the additional features provided.

The hierarchical structure of a database record within a DEDB is the same as HDAM, except for an additional dependent segment type. There is one root segment in each database record and from 0 to 126 dependent segment types. One of these segment types can, optionally, be a sequential dependent segment

type. As with HDAM, a randomizing module is used to provide initial access to the database data sets containing the DEDB database.

The highest level in the structure used to implement a DEDB is the area. A DEDB can consist of from 1 to 2048 areas. Each area is implemented as one VSAM ESDS data set.

Each DEDB area data set is divided into:

- A root addressable part — This contains VSAM CIs that are addressable by the randomizing middle.
- An independent overflow part.
- A sequential dependent part — This is optional, and is only defined if the DEDB has a sequential dependent segment defined in the hierarchical structure.

The root addressable part is further subdivided into units of work (UOWs). These should not be confused with the unit of work that encompasses an application's minimum set of updates to maintain application consistency. The DEDB UOW is similar, however, as it is the smallest unit that some Fast Path utilities (for example, reorganization) work with, and lock, preventing other transactions accessing them. Each unit of work consists of from 2 to 32767 CIs, divided into a base section of 1 or more CIs and a dependent overflow section, consisting of the remaining CIs.

Figure 22 on page 63 shows segments stored in a DEDB area data set.

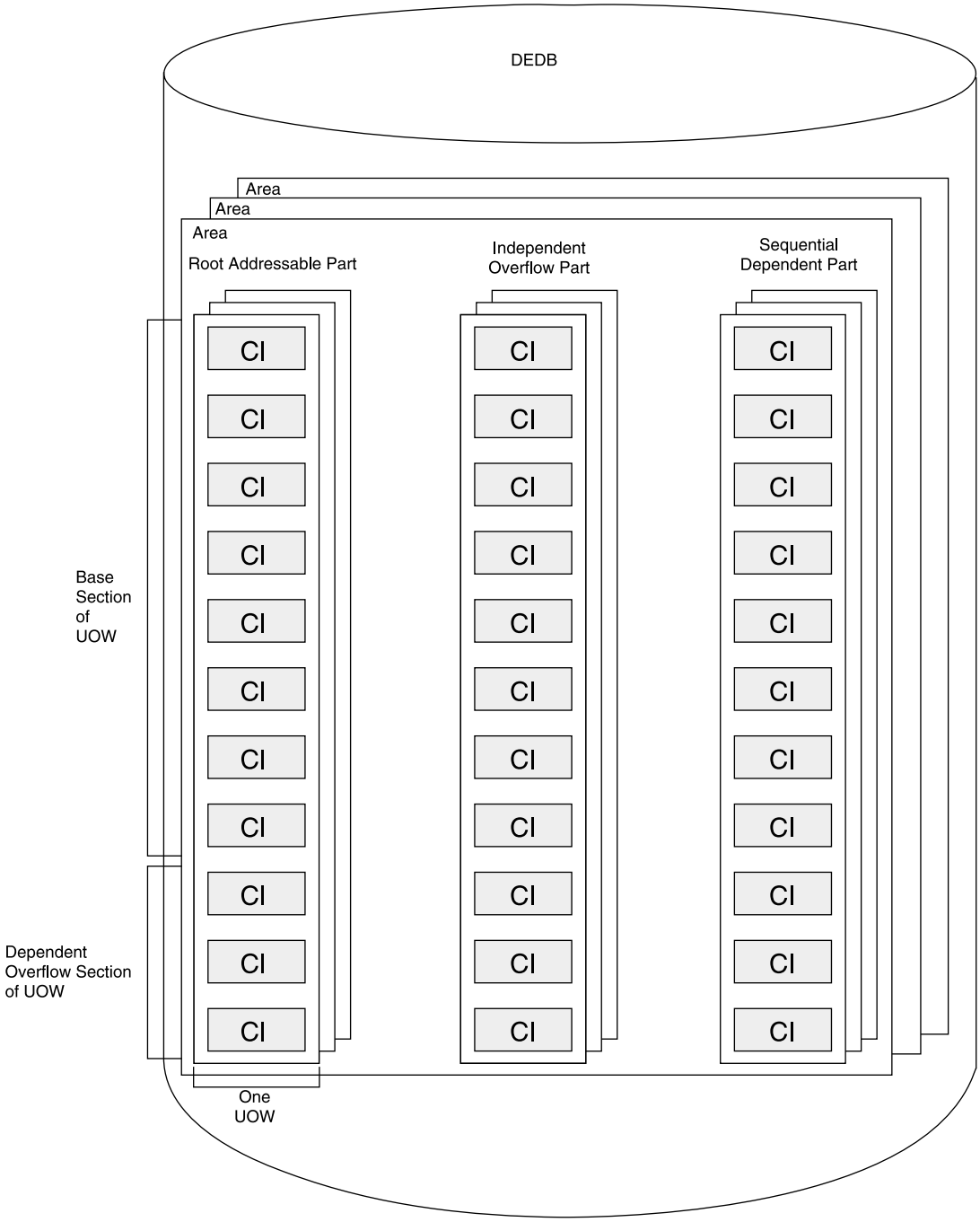

Figure 22. Overall Structure of a Fast Path DEDB

The randomizing module works in a similar way to an HDAM database. It takes the key value of the root segment and performs calculations on it to arrive at a value for a root anchor point. However, for a DEDB this is the root anchor point within the Area data set. The randomizer must also provide the value of the area data set that contains the RAP. Again, there is a sample randomizer provided with IMS, although due to the unique characteristics of DEDBs, you should look closely at whether you need to code your own.

The randomizer will produce the value of a root anchor point in the base section of a unit of work. IMS will attempt to store all dependent segments (except sequential

dependents) of the root in the same UOW as the root. If more than one root randomizes to the same RAP, then they are chained off the UOW in key sequence. If there is insufficient space in the base section, then root and non-sequential dependent segments are placed in the dependent overflow section of that UOW. If there is no space in the dependent overflow section in the UOW, a CI in the independent overflow part of the DEDB Area is allocated to that UOW and the segment is stored there. This CI in the independent overflow part is then used exclusively by that UOW, and is processed with that UOW by the DEDB reorganization utility.

The free space between the data segments in the CIs in the root addressable part and Independent overflow part of a DEDB area data set are managed in the same way as in an HDAM data set. with a free space element anchor point at the start of the CI pointing to a chain of free space elements. As with HDAM, space from deleted segments is automatically re-used, and the UOW can be reorganized to consolidate fragmented free space (without making the database unavailable). Unlike an HDAM database, there is no free space map. The segments for a database record can only be allocated in the same UOW (or attached segments in dependent overflow) as the root segment. An out of space condition results if insufficient free space is available in the UOW or Independent overflow.

The following, optional, features can also be used with a DEDB:

Virtual Storage Option (VSO)
This stores the CIs of a DEDB in z/OS data spaces and coupling facility cache structures, eliminating I/O to the DASD system. The data can either be loaded (partially or completely) when the database is opened, or loaded into the dataspace as it is referenced.

Shared VSO
You can share VSO DEDB areas, which allows multiple IMSs to concurrently read and update the same VSO DEDB area. The three main participants are the coupling facility hardware, the coupling facility policy software, and the XES and z/OS services.

Multiple Area Data Sets
You can define DEDB areas so that IMS will automatically maintain up to seven copies of each area. This can be used to provide a backup if I/O errors occur, allow data sets to be re-defined on a different device without taking the database offline, or to provide parallelism in I/O access for very busy applications.

High Speed Sequential Processing
This is a function provided by Fast Path to enhance the performance of programs that are processing segments sequentially in a database. IMS issues a single I/O request that reads one UOW at a time. This causes a reduction in the overhead of multiple I/O requests and stores the CIs in a separate buffer pool. HSSP also issues the read request in advance of the program asking for the data, to provide parallel processing. In this way, the segments in the database are available to the program without any delays to wait for I/O processing. The overall runtime can be significantly reduced, as long as the database is being read sequentially.

Sequential Dependent Segments
A DEDB database can have one sequential dependent segment type defined in the database record. This is processed completely separately to the other dependent segments. Normal application programs can only Insert new sequential dependent segments or read existing sequential dependent

segments. All other processing of these sequential dependents is performed by IBM supplied utility programs. The sequential dependents are stored in the Sequential dependent part of the area data set in chronological sequence, and processed by the IMS utilities, to read or delete them, in the same sequence.

The main situations where you might consider using Fast Path DEDBs are:

- Where you have very high volumes of data to store. The DEDB can be spread over up to 2048 VSAM ESDS data sets, each with a maximum capacity of 4GB. However not all this space is available for application data as some minimal space is needed for IMS and VSAM overhead and free space.
- Where you have a small to medium database that needs extremely fast access. you could use the DEDB VSO option and have the data held in an z/OS dataspace, making a major reduction in the physical I/O associated with the database.
- If you needed a database with very high availability. The use of multiple area data sets, the ability to reorganize online and the DEDBs tolerance to I/O errors mean the database can be kept available for extended periods.
- Where an application needs to record large amounts of data very quickly (for example journaling details of online financial transactions) but does not require to update this data, except at specified times (for example, an overnight process), then a DEDB with a sequential dependent segment could provide the solution.

The principal disadvantages of DEDBs are:

- This method is more complicated than other IMS access methods. Consequently, it requires a higher degree of support both for initial setup and running.
- The person designing the application must understand the restrictions and special features of DEDBs and design the application accordingly
- The DEDBs are only available for applications running against an IMS control region (MPP, IFP, BMP and CICS applications). There is no batch support except indirectly via the IMS supplied utilities to extract the data.
- Fast Path DEDBs do not support logical relationships or secondary indexes, so these functions must be implemented in the application

Related Reading:

- For more details on using DEDBs, together with samples of their use, see the ITSO publication *IMS Fast Path Solutions Guide*.
- The features of DEDBs are described in detail in chapters on designing a Fast Path database in *IMS Version 9: Administration Guide: Database Manager*.
- The utilities used with DEDB are described in *IMS Version 9: Utilities Reference: Database and Transaction Manager* and the randomizer and other Fast Path exits are in *IMS Version 9: Customization Guide*.

GSAM

A z/OS sequential file being used as an interface to or from an IMS application can be defined to IMS as a GSAM database. However, the normal concepts of hierarchical structures do not apply to GSAM, as it just contains the normal data records, with no IMS information.

These files can be z/OS sequential files, or VSAM ESDSs. Before or after the IMS application processes them, other applications can process them using the normal BSAM, QSAM and VSAM access methods.

When using GSAM for sequential input and output files, IMS will control the physical access and position of those files. This is necessary for the repositioning of such files in case of program restart. When using GSAM, IMS will, at restart time, reposition the GSAM files in synchronization with the database contents in your application program's working storage. To control this, the application program should use the restart (XRST) and checkpoint (CHKP) calls. These calls will be discussed in "Using Batch Checkpoint/Restart" on page 192. Note that IMS can not re-position VSAM ESDS files on restart. There are also some other restrictions on restarting, detailed in the chapter, "Designing Full-Function Databases" in the *IMS Version 9: Administration Guide: Database Manager*.

Whenever you want your program to be restartable, you should use GSAM for its sequential input and output files. There are two reasons why you should want to do this. The first is to save time if a program rerun is required in case of program system failure. This is normally only done for long-running update programs (one or more hours). The other reason stems from a planned online usage of the databases.

HSAM and HISAM

The two Hierarchical Sequential (HS) databases, HSAM and HISAM, use the sequential method of accessing data. All database records and all segments within each database record are physically adjacent in storage. Unlike HSAM, however, each HISAM database record is indexed, allowing direct access to a database record.

HSAM and HISAM have now been superseded by the HD access methods. The HD access methods have a number of features that would almost always make them a better choice.

The HSAM access method will not allow updates to a database after it was initially loaded and the database can only be read sequentially. HSAM was used in the past to process operating system sequential files, but GSAM is now a better choice.

The HISAM access method offers similar functionality to HIDAM, but has poorer internal space management than the HD access methods that would normally result in more I/O to retrieve data, and the need to reorganize the databases much more frequently.

A simple HSAM (SHSAM) database is an HSAM database containing only one type of segment, a root segment. Similarly, a simple HISAM (SHISAM) database is a HISAM database containing only one type of segment, a root segment.

Related Reading: For additional details about the HS access methods, see the *IMS Version 9: Administration Guide: Database Manager*.

Physical Segment Design

After you decide what access method (or database type) you want to use for a particular application, you need to design the segments that will be in that database. When designing segments, the physical parameters are important. The following sections discuss these details.

Segment Length

IMS will use the segment length as defined in the DBD to store each segment. If you have left free space at the end of the segment for future use, that space will be

physically hold space on DASD unless you have compressed the segment. If the application is likely to have additional requirements later, it can be easier to make use of this free space than to increase the segment length later. You have to balance the cost of making the change to the databases and programs against the cost of wasted DASD space.

Number of Occurrences Per Segment Per Parent

Try to avoid long twin chains, that is, many occurrences of a particular segment type under one parent. Chain lengths should be estimated in terms of blocks needed to store each such segment.

Location of Segments in the Hierarchy

Try to locate the segments most often used together with the root segment into one control interval/block. The segments are initially physically stored in hierarchical sequence, so the most frequently used segments should be on the left of the structure (low segment codes).

Average Database Record Size

The average database record is calculated by the total bytes of all segments under the root segment. Small segments with more twins than larger segments with fewer twins, although having almost the same number of bytes, results in better performance and space usage.

Operating System Access Methods

To underpin the IMS access methods, IMS uses two operating system access methods to store the data on disk storage, and move the data between the disk storage and the buffers in the IMS address space. These are:

Virtual Sequential Access Method (VSAM)
Two of the available VSAM access methods are used, Key Sequenced Data Sets (KSDS) for Index databases, and Entry Sequenced Data Sets (ESDS) form the primary data sets for HDAM, HIDAM, etc. The data sets are defined using the VSAM Access Method Services (AMS) utility program.

Overflow Sequential Access Method (OSAM)
This access method is unique to IMS and is delivered as part of the IMS product. It consists of a series of channel programs that IMS executes to use the standard operating system channel I/O interface. The data sets are defined using JCL statements. As far as the operating system is concerned, an OSAM data set is described as a physical sequential data set (DSORG=PS).

For more information about these operating system access methods, see “VSAM or OSAM” on page 68.

There are two types of data sets defined using these access methods:

Indexed sequential data sets
These are all defined and accessed as VSAM KSDSs, and are used to implement primary and secondary index databases. These databases are processed using the standard record level instructions of VSAM. A catalogue listing (VSAM LISTCAT) will show all current details of the files. They are susceptible to the normal performance degradation of VSAM KSDSs from CI/CS splits caused by insert/delete activity. They can, if necessary, be processed using AMS utilities such as REPRO.

Sequential data sets

These are defined and accessed either as VSAM ESDSs or using OSAM. It is important to note that, while these data sets appear as sequential data sets to the operating system, IMS accesses them randomly. The data sets do not contain records as such. IMS always processes them at the CI (VSAM) or block (OSAM) level. The internal structure within each CI/block is arranged as described in “IMS Hierarchic Access Methods” on page 53. Interpreting catalogue listings of these files as if they were sequential files can, at times, be misleading.

In addition to using VSAM or OSAM, IMS data sets can be managed by Data Facility Storage Management Subsystem (DFSMS). For more information, see “IMS and System Managed Storage” on page 69.

VSAM or OSAM

While most physical databases are implemented over a single VSAM ESDS or OSAM data set, IMS provides facilities to spread an HDAM or HIDAM physical database over up to nine additional data sets (multiple data set groups). The is facility is restricted as, with the current release of IMS, the 1st, primary data set group, that is always defined, must contain the root segments, and can contain any dependent segment type. The other (secondary) data set groups can each contain any dependent (non-root) segment type. However, each dependent segment type can only be defined in one data set group. This is, aside from performance implications, transparent to applications. If the database needs to be reorganized, then all data sets that make up the physical database have to be reorganized at the same time.

The reasons why you may wish to use secondary data set groups are:

- To separate little used segments from the main data set, to leave more space for frequently used segments. This will increase the chance the all regularly accessed segments are in the same block with the root, and enhance performance. For example, you might have a segment type that has a new occurrence inserted each month, say month end audit totals. This is only rarely accessed after insertion. Placing in this segment type in a secondary data set group, while imposing an overhead on the program that inserted it, could improve performance of all other programs as there is an increased chance segments they access are in the same block as the root, and more database records can be packed into one CI/block.
- If you have a database with one very large segment type, and a number of other small segment types than, as described above, this can result in unusable space as IMS space management only regards a CI/block within a data set as having freespace if it can accommodate the largest segment type stored in that data set. Putting this large segment type in a secondary data set group means that the other data set groups will now only be regarded as full if they could not contain the second largest segment type.
- You can specify different freespace parameters on the different data set groups, so you could place non-volatile segment types in a data set group with little free space, to increase packing in a CI/block, and consequently the chances of having several segments a program is retrieving in the same block. Volatile segment types (that is, frequent insert/delete) could be placed in a data set group with a large freespace specification, allowing segments to be inserted near related segments.
- For very large databases, you may be approaching the structural limit of the data set access method (4 GB of data). If you have one or two segment types that

occur very frequently, the each of these segment types could be placed in one or more secondary data set groups to provide more space. But in this case, see also the additional features of OSAM below, and also look closely at DEDBs, which can be spread over many more data sets.

When performing space calculations, you need to be aware that, in addition to the overhead for IMS control information (pointers, etc.), VSAM data sets will also contain a suffix area at the end of the CI that contains VSAM control information. This makes the space available in the CI for IMS data slightly less than the VSAM CI size.

The choice between OSAM and VSAM ESDS for the primary database data sets will depend, to some extent, on whether your site already uses VSAM and whether you need to make use of the additional features described below. The choice between VSAM ESDS and OSAM is not final, as a database can be changed from one access method to the other by unloading the database, changing and regenerating the DBD, then re-loading the database.

As the OSAM access method is specific to IMS, it has been optimized for use by IMS. Reasons you may want to use OSAM are:

- Sequential Buffering (SB). With this feature, IMS will detect when an application is processing data sequentially and pre-fetch blocks it expects the application to request from DASD, so they will already be in the buffers in the IMS address space when the application requests segments in the block. This is manually activated for specific IMS databases/programs. It can appreciably decrease run times for applications processing databases sequentially. It is similar to the sequential prefetch available with some DASD controllers, but has the advantage that the data if fetched into the address space buffer in main memory, rather than the DASD controller cache at the other end of the channel

  Related Reading: See the “Full-Function DB Design Considerations” chapter in *IMS Version 9: Administration Guide: Database Manager* for details on sequential buffering.
- The structural limit on the amount of data that IMS can store in a VSAM ESDS is 4GB of data. OSAM can process a data set up to 8GB in size.
- Overall, OSAM is regarded as more efficient as it is more efficient, buffering, shorter instruction path.

IMS and System Managed Storage

Most of the IMS data sets can be managed by System Managed Storage (SMS). The only concern would be the online data sets (OLDS). If they should get migrated (not very likely in most installations), they might be recalled with different attributes.

Related Reading: For more information about the data sets that IMS uses for logging events, see Chapter 25, “IMS Logging,” on page 257.

OLDS data sets must be allocated in contiguous space. It could also be possible for both the primary and secondary OLDS data sets to be on the same volume. This is a major problem if that volume becomes unreadable. You should use management classes to avoid this.

Write ahead data sets (WADS) have very high write rate and are very sensitive to slow response. These data sets should be placed with some care. SMS may not provide a good place to allocate them.

If any OLDS, recovery log data sets (RLDS) or system log data sets (SLDS) or image copy data sets are SMS managed, the CATDS parameter must be set for the RECON. This will tell DBRC to use the system catalog to find data sets and not be concerned if they are not on the same volumes which they were originally allocated.

IMS Checkpoints

A database management system, such as IMS, provides facilities to keep all the application data stored in the databases in a consistent state. This discussion is principally concerned with keeping the application data consistent, from an applications point of view. It relies on the application using the facilities provided by IMS. However, the facilities to consistently update the database also ensure that all internal IMS information (pointers, free space elements, etc.) are kept consistent, though this is transparent to the application program.

An application program might make updates to several IMS databases. If a problem is encountered part of the way through these updates, either the program fails, or application logic dictates it cannot continue with the processing, then it will need to restore the data in the databases to the state when it started updating them. For example, a program adds a detail to the order, in the order database, and then needs to update the parts database to reduce the quantity of the part available for ordering. If the program updates the order database, but then fails before updating the parts database, the order is recorded, but the quantity of the part is still shown as available for ordering on the parts database. The update to the order database and the update to the parts database make up a single unit of work (UOW). For the application data to be consistent, either all the updates in a unit of work must be written to the database successfully (committed) or none of the updates in the UOW must be committed.

To maintain database consistency, IMS uses the concept of the *application checkpoint*. You should not confuse the application checkpoint, which applies to the single execution of an application program, with the system checkpoints IMS subsystems take. System checkpoints are taken to allow the IMS subsystem to recover from a failure of the complete IMS subsystem. The application checkpoint indicates to IMS the end of the applications unit of work and causes IMS to commit all updates made in that UOW.

An application's UOW commences when the application program starts running. By default, IMS takes an application checkpoint, and commits all updates when the application terminates normally. You can also explicitly request a checkpoint, using the CHKP function of the DL/I API. The CHKP call is also taken as starting another UOW. If an application program terminates abnormally, then all database changes are backed out to the last commit point (start of program if application checkpoints are not being used or last CHKP call if they are). The application can also explicitly back out all updates within the current UOW by using the ROLB, ROLL or ROLS functions of the DL/I API (the difference between the calls relate to action taken by the Transaction Manager component, if applicable, and whether the application regains control after the call).

Related Reading: See the *IMS Version 9: Application Programming: Database Manager* (under "maintaining database integrity") for complete descriptions of the functions mentioned in the previous paragraph.

For long running batch and BMP application programs, you should issue explicit checkpoint calls at regular intervals. As the programs read database records, details of these database records (internal IMS addresses) are stored by the IMS

subsystem until the application reaches a commit point (issues a CHKP or terminates). This is done to prevent other application programs updating these database records while the application is working with them. These details are stored in an internal buffer in the IMS address space. Failure to issues regular checkpoints can cause the following problems:

- The IMS address space has insufficient storage to contain all the buffers needed to contain these details, resulting in the application program being terminated
- If the application fails, or issues a ROLL, ROLB or ROLS call, IMS will have to back out all the updates performed by the application. If it has been running for a long time without checkpointing, it may well take the same time to back out all the updates as it took to apply them. Equally, if you then correct the problem and re-start the program, it will take the same time again to re-process the updates.
- For BMPs, other applications processing the databases by the same IMS control region might be prevented from accessing these database records. This can cause severe response-time problems if the other applications are being used by online users. For Batch jobs, you can encounter similar problems if block level data sharing is being used. Also, the IMS ENQ/DEQ block supply might become exhausted, which results in a U0775 abend of *all* of the application programs that are running at the time of the abend.

Long running programs should issue checkpoints based on the number of database calls made. As a rule of thumb, initially issue batch checkpoints at about every 500 database calls. You do not want to checkpoint too frequently, as there is an overhead in writing out all updates and your application re-positioning itself in all the IMS databases after the CHKP call. IMS loses the position in the databases after a CHKP call, so such a call must be followed up with a GU call to the last record retrieved in databases where such positioning is important to the application logic.

Obviously you cannot CHKP more frequently than the number of calls in one UOW. As you might need to tune the checkpoint frequency, IBM recommends that you code the program so it can be easily changed. It is best to code it in the program as a variable, possibly input as a parameter at run time.

The functions described in the previous paragraphs are referred to as *basic checkpoint*. For applications running in Batch and BMP address spaces, there is also extended checkpoint functionality available. This is referred to as *symbolic checkpointing*. Symbolic checkpointing provides the following additional facilities that enable application programs running in batch or BMP address spaces to be re-started:

- The XRST function call is made at the start of the program and indicates to IMS that the application is using symbolic checkpointing
- The CHKP function is extended to allow the application to pass up to seven areas of program storage to IMS. These areas are saved by IMS and returned to the program if it is restarted. This can be used for any variables, (for example, accumulated totals, parameters) that the application would need to resume processing
- Each CHKP call is identified by an ID that is generated by the application program. This ID is displayed in an IMS message output to the operating system log when the checkpoint is successfully complete. While a good programming practice would be to ensure this ID is unique, nothing in IMS enforces this practice.
- If the program fails, after correcting the problem, it can be restarted from either the last or any previous successful checkpoint in that run. IMS will re-position the databases (including non-VSAM sequential files accessed as GSAM) to the

position they were at when the checkpoint was taken. When the XRST call is made on re-start, the program will receive the ID of the checkpoint it is re-starting from, together with any user areas passed to IMS when that CHKP call was issued

Related Reading: Full details of symbolic checkpointing, along with various restrictions on what can be done, are in the chapter on maintaining database integrity in *IMS Version 9: Application Programming: Database Manager*.

Locking

The other main facility a Database Management System (as distinct from the use of a database) provides, is the ability for more than one application to simultaneously access the database for update, while preserving database integrity.

This prevents situations such as in the following example: Application A reads a record. While it is processing it (waiting for a user to respond at a terminal), application B reads the same record. While application B is processing the record, application A writes back the updated record. The user of application B now responds, and application B writes back the updated record, overwriting the update to the record made by application A.

The mechanism used to prevent this is to lock (enqueue) the database segments/records until the application has finished processing them successfully, that is reached the end of a unit of work. While this discussion is mainly concerned with ensuring application data is updated consistently, the mechanisms used by IMS also ensure that IMS's internal information in the databases (pointers, and so forth) remains consistent.

One problem that can occur from this enqueueing of database segments, is a deadlock between two application programs. For example, application A reads database record 1. While A is doing other processing, application B reads database record 2, then tries to read database record 1, and is suspended waiting for it, as it is enqueued by application A. Application A now attempts to read database record 2, and is suspended, as it is enqueued by application B. Both applications are now suspended waiting for a record enqueued by the other — a deadlock. IMS detects this, and will abnormally terminate (abend) the application it assesses has done the least work, backing out its updates to the last commit point. The mechanism IMS uses to detect the deadlock depends on what method of data sharing is being used (see below). This is either direct detection of the deadlock from the details enqueued, or by timeout; that is, terminating a task after a (parameter specified) period of waiting for a database record.

If the application is accessing DB2 tables, DB2 also detects deadlocks by timeouts and will instruct IMS to abend the program. The abend code issued is the same as for an IMS database deadlock. What IMS cannot detect is a deadlock between two applications where the two different resources the applications are trying to get are being managed by two separate resource managers. This is most common with CICS applications using IMS/DB databases. For example, CICS task A reads, and enqueues a database record. CICS task B then issues a CICS ENQ for a resource, for example to serialize on the use of a TDQ. CICS task B then attempts to read the database record held by task A, and is suspended, waiting for it. CICS task A then attempts to serialize on the resource held by task B and is suspended. We now have a deadlock between task A and B. But neither IMS or CICS is aware of the problem, as both can only see the half of the deadlock they are managing. Unless IMS was using one of the data sharing techniques that timed out application

that wait for the database, or CICS was set up to abend tasks after a very short time suspended, this deadlock would have to be resolved manually.

The person designing an application that uses IMS databases needs to be aware of the potential problems with database deadlocks, and design the application to avoid them. If the application also locks resources managed by another product, they also need to be aware of the potential for a deadlock developing between the IMS database records and the resources managed by the other product. Unfortunately, deadlocks often only occur when the application processes very large volumes, as they often require very precise timing to occur. This means that they are often not detected during testing with small volumes.

IMS supports three methods of sharing data between a number of application tasks:

Program Isolation (PI)
This can be used where all applications are accessing the IMS databases via a single IMS control region. IMS maintains tables of all database records enqueued by the tasks in buffers in the control region address space. This provides the lowest level of granularity for the locking, and the minimum chance of a deadlock occurring. Deadlocks are resolved by IMS checking the tables of database records enqueued to ensure there is not a deadlock situation, and abending one of the tasks if there is.

Block level data sharing
This allows any IMS control region or batch address space running on an OS/390 system to share access to the same databases. It uses a separate feature, the Internal Resource Lock Manager (IRLM). This is delivered as part of the IMS product, but needs to be separately installed. It runs in its own address space in the OS/390 system and maintains tables of the locks in this address space. With block level data sharing IMS locks the databases for the application at the block level. This locking is at a higher level than with program isolation (that is, all database records in a block are locked). Because of this coarser level of locking, there is an increased risk of deadlocks and contention between tasks for database records. Deadlocks are resolved by a timeout limit specified to the IRLM. If the disk storage the databases are on is shared between two OS/390 systems, it is also possible to share the databases between IMS applications running on the two OS/390 images, by running an IRLM address space on each of the two OS/390 images. The IRLMs communicate using VTAM but maintain lock tables in each IRLM address space. IRLM is also used as the lock manager for DB2 but, because of the different tuning requirements, you should use separate IRLM address spaces for DB2 and IMS.

Sysplex data sharing
Where a number of OS/390 systems are connected together in a sysplex, with databases on DASD shared by the sysplex, it is possible for IMS control regions and batch jobs to run on any of these OS/390 images and share access to the databases. To do this, an IRLM address space, running version 2 of IRLM, must be running on each OS/390 image the IMS address spaces are running on. The IRLMs perform the locking at block level, as in the previous case. However, instead of holding details of the locks in the IRLM address space, the lock tables are stored in shared structures in the sysplex coupling facility. Deadlocks are resolved by a timeout limit specified to IRLM.

Related Reading: For further details on data sharing, see:

• Chapter 8, “Data Sharing,” on page 83

- Chapter 30, “Introduction to Parallel Sysplex,” on page 315
- The chapter on administering a data sharing environment in *IMS Version 9: Administration Guide: System*

Chapter 7. Choosing the Correct Database Type

Some database types will provide a better solution to your business needs than others. This chpater discuss how to choose the best type of database for your applications.

The following sections are covered in this chapter:

Applications Suitable for Full-Function Databases

The following sections discuss the advantages and disadvantages of the different types of full-function databases.

When to Choose HDAM

HDAM is recognized, in practice, to be the most efficient storage organization of an IMS database. It should be considered first. After looking at all the required access to the database, if there are not requirements to process a large section of the database in key sequence, then HDAM should be chosen. If sequential access of the root keys is required, the process can retrieve the data in physical sequence and sort the output.

The primary advantages of HDAM are:

- Fast direct access (no index accesses) with few I/O operations
- Single data associated control blocks
- Small working set in main storage for IMS
- Good physical sequential access

Some considerations of using HDAM are:

- You need a randomizing module.
- Sequential access in root key order is not possible if the physical sequence of database records in storage is not the same as the root key sequence. This is dependent on the randomizing module and root key characteristics.
- If the database exceeds the space in the RAA (root addressable area), it will extend into overflow. After it is in overflow, the performance of the access to these segments can increase drastically.
- To increase the space of the database, a DBDGEN is required to increase the number of blocks in the RAA. This will also require an ACBGEN to rebuild the online ACBs for use in the online system. This will require that you take the database offline (making it unavailable) to complete and coordinate the change. For more information about DBDGEN and ACBGEN, see “Generating IMS Control Blocks” on page 158.

In many cases, the disadvantages for HDAM do not apply or can be circumvented. The effort needed to circumvent should be weighed against the savings in terms of main storage and CPU usage. There is no doubt, however, that an application with only HDAM databases is the most compact one. Some possible solutions for the previously mentioned HDAM disadvantages are:

- The IMS provides a general randomizing module, DFSHDC40, which can be used for any key range
- If heavy sequential processing is required and a randomizing module which maintains key sequence cannot be designed, then sort techniques can be used:
  - If the program is non-input driven, as is the case with many report programs, simple get-next processing presents all the database records in physical sequential order. The output could then be sorted in the desired order. Also, in many instances, only certain selected segments are required. So the output file of the extract can be a fairly small file
  - If there are input transactions which would normally be sorted in root key sequence. This can be readily done with an E61 sort exit routine which passes each root key to the randomizing module for address calculation and then sorts on the generated addresses plus the root key instead of the root key itself
- A secondary index could be built with the root key as index search argument. The cost of this should be weighed against the cost of sorting as in 2 above. The secondary index provides full generic key search capability, however. A secondary index on the root segment should never be used to process the whole database, as this will cost a lot more I/Os than to process the database in physical sequence.

When to Choose HIDAM

Overall, only choose HIDAM if there are requirements to regularly process the database in root segment key sequence. If there are also requirements for fast random access to roots (from online systems), look at alternatives for the sequential access, such as unload/sort or secondary indexes.

HIDAM is the most common type of database organization. It has the advantages of space usage like HDAM but also keeps the root keys available in sequence. These days, with the speed of DASD the extra read of the primary index database can be incurred without much overhead. The most effective way to do this is to specify specific buffer pools for use by the primary index database, thus reducing the actual IO to use the index pointer segments.

HIDAM does not need to be monitored as closely as HDAM.

When to Choose PHDAM or PHIDAM

The reasons for choosing PHDAM or PHIDAM are the same as described in “When to Choose HDAM” on page 75 and “When to Choose HIDAM.” The differences are the size of the data store and some administrative considerations.

You might not need to change any of your application programs when you migrate HDAM or PHDAM databases to HALDBs, but there might be exceptions. Exceptions include the initial loading of logically-related databases and the processing of secondary indexes as databases.

You might also want to change applications to take advantage of some HALDB capabilities. These capabilities include processing partitions in parallel, processing individual partitions, and handling unavailable partitions.

Related Reading: For more complete information about HALDBs, see the following publications:

- *IMS Version 9: Administration Guide: Database Manager*

- *The Complete IMS HALDB Guide All You Need to Know to Manage HALDBs*

Applications Suitable for HSAM and HISAM

HISAM is not a very efficient database organization. All HISAM databases can easily be converted to HIDAM. The application should receive significant performance improvements as a result. The only situation where HISAM might be desirable over a HIDAM database is when it is a root-segment-only database.

Even so, segments are not deleted and free space reclaimed after a segment is deleted until the next database reorganization.

Applications Suitable for Fast Path Databases

The application area that the DEDB was originally designed—the management of customer accounts in a retail bank—is an ideal candidate for that database implementation, but it is far from the only one, and some of functions of the DEDB, notably the virtual storage option (VSO), extend the application areas that you should consider using a DEDB.

Many users have not realized the dramatic operational and performance benefits available with DEDBs and have, for various reasons, not familiarized themselves with that database implementation. In one example, a customer who preferred to use only DB2 for new databases was convinced to use a DEDB with a saving of some 65% in the processor requirements for that very large application.

Initially, it might seem daunting to introduce a DEDB to an organization where the users are unfamiliar with that technology, but practical experience has shown that user education is really a small, easily contained issue, and the benefits of the DEDB for well-suited applications, greatly outweigh the additional effort for the introduction of this type of database.

The examples in the following list are drawn from many industries and show that, especially with VSO, the DEDB is very effective.

Account database: retail bank
: This application exploits the characteristic effectiveness of the sequential dependent to collect transactions for reprocessing (posting) at the end of the business day. The low cost of deletion of the sequential dependent reduces the overheads for very large numbers of transactions. The DEDB also allows near-continuous operation and portioning of the data to ensure manageability of the large databases involved.

  Access to the account by account number requires only one I/O and almost all processing can be done, with one read I/O and one write I/O because you can practically ignore the I/Os for the sequential dependents.

  One disadvantage is that the DEDB requires all access to the account be through the account number (because Fast Path does not support secondary indexes), so a second database is necessary to access the account record from another key. This would be the credit card database mentioned below, and so access via the credit card would require one I/O to the credit card database and one to the account database.

Credit card database: retail bank
: To provide access to an account DEDB from a credit card number, a cross-reference database is required and must be maintained manually (unlike a secondary index, which is maintained by IMS). This is usually a

root-only database with little data in each segment: primarily the relevant account number to which the credit card transactions are to be posted, and the status of the card itself.

Teller control database: retail bank

Teller transaction journals can be readily kept as sequential dependents, provided they are not usually required for online access. If online access is necessary, then a direct dependent segment would be more suitable.

Account database: utility company

A utility company (telephone, electricity, gas, or water) requires very similar processing to that for a retail bank and a similar data structure is very suitable. All the remarks above for the banking application are relevant, though there is one interesting difference. In the case of a telephone utility, there is a need to see a meter database that is similar to a credit card database for the banking environment.

Meter database: utility company

This database provides a cross reference from a meter identifier to the account that is to be billed for usage recorded by that meter. Because meter readings are input and processed in batches that tend to be quite predictable, then it could be very effective to exploit this predictability: to put the meter data into areas that correspond to the batches of data that are processed in one BMP execution and to switch an area into VSO prior to each batch run and out of VSO afterwards.

Audit and history database

This database illustrates a good use of the sequential dependent segment type to provide a historical journal of activities. For a non-shared DEDB, the sequential dependents will be in absolute time sequence, and if the DEDB is shared, then the segments can be readily sorted into time sequence.

Status report database

This database was designed to hold a few tens of lines of report data for each of several thousand destinations. Each report was generated daily, and access was required to each report for typically three days before it could be purged. Access to the reports was occasional and a high percentage of the data was accessed either once or not at all.

By using the sequential dependent to store each report, the detail lines of each report were kept in the same or adjacent CIs (as they were inserted within one unit of work), so that online access to them was quite efficient.

As each report was generated, a summary was placed in the report summary segment so that it could be accessed at the same time as the root segment.

Bet status database: gambling system

This database is designed to support an online totalizer system where the total of all bets placed on a given horse is required to be kept up to date with very high concurrency, and sometimes there are high transaction rates against a few records for a relatively short period. Here, the judicious use of VSO can allow records that are currently active to be held in VSO, while less active records will stay on DASD. Note that there is an option to restrict this database to a root-only design, obeying the constraints of the old main storage database, and thus allowing the use of FLD calls that can reduce the scope and duration of data locking. This should substantially increase the level of maximum concurrency that can be achieved.

The following sections discuss the different requirements that should be considered while evaluating whether a DEDB is likely to be the correct database type for a particular application.

Very Large Databases

The structure of the DEDB was designed to facilitate handling of very large databases by implementing each database as 1-240 areas (pre-IMS Version 8), each of which can be as large as 4 GB. This provides an effective mechanism for processing and managing large databases as multiple units.

As of IMS Version 8, DEDBs can have up to 2048 areas.

The areas are relatively independent of each other — and for batch-style processing, multiple areas can easily be processed in parallel, which dramatically reduces run times for such things as overnight update and report runs (executing as bumps), image copy jobs, and similar tasks that involve processing entire databases. If the area breakup can be in processing units, then individual areas can be processed independently. For example, if an area is dedicated to one subsidiary within a conglomerate business, then the processing for that subsidiary can be optimized and performed independently of other subsidiaries.

The algorithm by which data records are assigned to area is entirely under the user control, so data and application requirements can readily exploit the area structures by using separate areas for groupings of data that have different characteristics (and so require different space definitions for optimal performance) or are processed on different schedules. For example, separate areas could be used for records representing different business units, or different regions for which processing is done on different cycles.

The high performance characteristics of the DEDB, discussed below, are particularly important for large databases, as in many instances, the sheer size of a database may impose a requirement for high performance, particularly in batch or “whole of database” processing.

High Availability Requirements

The requirement for extended outages for planned maintenance is dramatically reduced because the implementation of a DEDB is designed so that almost all maintenance, such as image copying or database reorganization, can be performed while the database is online. During a database reorganization, only a small part of the data, one unit of work (UOW), which might typically be a few tens of control Intervals, is locked at any particular time. Thus, online processing can generally proceed with minimal impact during a reorganization.

Additionally, the scheduling of a PSB to access the DEDB does not depend on the availability of all areas, so even when one area is not available for access, say a database recovery is in progress, then all other areas are accessible and transaction and BMP scheduling can occur. In one customer’s retail banking DEDB, 20 areas located on 20 separate DASD device were used, so that even if a single area or the DASD device on which it was located were unavailable, 95% of the data should still be accessible.

The DL/I programming interface to the DEDB provides for an application that attempts to accesses data in an area that is currently not available to be given the same DL/I status code as for an I/O error, which now generalizes the meaning of that status code to be: “The data you requested is temporarily not available”. This

can be meaningfully handled by most existing programs. The net result of this is that, when one area of database is unavailable, processing for other areas can proceed normally, which is in contrast with an IMS full-function database, where unavailability of any part of the database precludes all scheduling.

Another availability feature for DEDBs is the ability to have multiple copies of the VSAM data sets that contain the data for one area. These data sets are called area data sets (ADS). Installations can create as many as seven copies (multiple area data sets, MADS) of each ADS, making the data more available to application programs. Each copy of an ADS contains exactly the same user data. Fast Path maintains data integrity by keeping identical data in the copies during application processing.

Highly Active Databases

If the Virtual Storage Option (VSO) of a non-shared DEDB (local to one IMS) is exploited for one or more areas, then all records in those areas are held in virtual storage during database processing. Updates are logged for recoverability and written to DASD periodically in an asychronous process. If the DEDB is participating in Parallel Sysplex data sharing, then all database updates are written to structure in the coupling facility to be shared with other IMSs. These mechanisms avoid I/O for most database accesses.

Limited Data Lifetime

A user can define that one segment within a DEDB is stored in a form called the sequential dependent segment. This is managed by IMS in a very different way from other data segments in the DEDB (where the storage mechanisms are rather similar to those used in a full-function database). The data entry segment type is designed to optimize the interim storage and retrieval of data (as the name suggests) for which only a short lifetime is normal before the data is reprocessed by some form of batch processing. The sequential dependent data storage mechanism is therefore ideally suited to data entry style applications where data may be inserted progressively over a period, is not accessed heavily by online transactions, and is extracted for reprocessing in bulk at intervals, and deleted in bulk at some time after that. This suits such applications as the maintenance of an audit trail or the collection of transactions for batch reprocessing, sometimes involving very high rates of data insertion into the database.

Extreme Performance Levels

There are several different aspects of the DEDB that are designed to minimize the number of I/Os necessary for data access and update, to minimize the path length of instructions used for a DEDB activity, and to ensure parallelism between multiple nearly simultaneous applications. These improve the performance of online and BMP processing, thus allowing either higher workloads on any given processor, or reduced processing costs for a given application workload.

This capacity to handle extreme workloads has been amply demonstrated by various Fast Path benchmarks showing the capability to exceed 11,000 transactions per second. More recent work has far exceeded even that performance level. Note that these benchmarks were achieved on processors that are quite small by today's standards.

Reduced I/O Usage

The space search and usage algorithms for the root and direct dependent segment data in a DEDB are markedly simpler than other database implementations, while

usually providing good locality of data, thus reducing the number of I/Os required for a given process compared to say a full-function database implementation.

If the sequential dependent segment type is used, the total number of I/Os required for insertion of data and deletion is substantially less than for other segment types for the typical insert-retrieve-delete sequence of processing.

When DEDB database control intervals are written as the result of add/update/delete calls, the I/Os are asynchronous to the transaction or BMP unit of work. The I/Os are done after the sync point is complete — which results in improved transaction response times, and improved BMP elapsed times.

If the high speed sequential processing (HSSP) functions of the DEDB are employed, many of the Read I/Os to access data are also done asychronously, which can again greatly reduce BMP elapsed times.

DEDB updates are logged in a slightly different manner from full-function database updates. During each Sync interval (an online transaction or BMP Checkpoint), changed data is written to the database only after the sync point processing has committed the changes. There is no requirement for “before” image data to be logged as would happen for full-function database updates, thus substantially reducing the volumes of log data and thus reducing the total I/O workload.

CPU Utilization

Since the DEDB implementation uses simpler algorithms for most functions than does full-function implementation, the CPU utilization for similar processing workloads is typically approximately one half that of full-function. It is also notable that almost all processing for an online transaction, or for a BMP, takes place under the TCB of the region processing that transaction or BMP, thus allowing a very high degree of transaction parallelism.

All the mechanisms mentioned above to reduce I/Os have a secondary effect that the CPU utilization to perform those I/Os is similarly reduced.

Summary of When to Choose DEDB

The art of knowing when to use a DEDB depends on understanding the differences between DEDBs and other database types. The following list describes some reasons and considerations for choosing DEDBs.

Advantages of areas

Most Fast Path commands and utilities operate on an area level, so they do not affect the whole database at once (unlike a full-function database). For example, you can recover one area of a DEDB while the rest of it is in use.

Another reason you might want to use areas is to spread the I/O load across several devices (and hopefully several physical paths in the system I/O configuration).

When to use VSO

Use VSO for your most frequently used databases or those for which fast access is crucial. It is also good for data you update frequently, even if several applications want to update the same field at the same time. These considerations also apply to shared VSO.

When to use MADS

Use MADS to ensure that I/O errors do not affect a database. Normally two copies of each area is sufficient, but you can have up to seven copies if you need to.

Using MADS is costly because you have several copies of the data. There is also a cost at execution time because IMS has to update several copies of the database simultaneously. The transactions using the DEDB do not notice the extra I/O because the output threads handle it asynchronously. You should use MADS only when you can justify the extra DASD cost.

When to use HSSP

Use HSSP for only those programs that conform to its restrictions because you get better performance.

Consider using the option to let HSSP take an image copy while it is running. This will save you time if you would normally take an image copy after your program finishes. HSSP knows not to log updates for a database it is copying.

When to use SDEPs

You would typically use SDEPs when you want to insert data quickly, but do not need to read it again until later. For example you might want to use SDEPs to hold audit records describing sensitive actions the user takes. You would not use SDEPs to hold data for a long time.

Chapter 8. Data Sharing

An IMS system includes a set of databases that are potentially available to all the declared application programs. Access to an individual database is a characteristic defined in a program's PSB. Data sharing support makes it possible for application programs in separate IMSs to have concurrent access to the same set of databases. To ensure that database changes at the segment level originating from one program are fully committed before other programs can access that segment's data, IMSs use lock management.

IMS systems can share data in a sysplex environment and in a nonsysplex environment.

- Sysplex data sharing is data sharing between IMS systems on different operating systems. A coupling facility is used by IRLM to control access to databases.

  Related Reading For more information about IMS and running in a sysplex environment, see Chapter 30, "Introduction to Parallel Sysplex," on page 315.
- Nonsysplex data sharing is data sharing between IMS systems on a single operating system image. A coupling facility can be used, but is not required.

With data sharing, two levels of control are possible:

- With database-level sharing, an entire database is locked while an application program is making updates. Locking prevents concurrent database access and scheduling of application programs that might jeopardize database integrity.
- With block-level sharing, you can use a global block-locking scheme to maintain database integrity during concurrent access of a database. The blocks are locked instead of the entire database. Multiple application programs can update a database at the same time if they are updating different blocks.

Some differences exist in support for data sharing configurations. Generally, a complete database is regarded as a data resource. When invoked within an IMS online system, or as a batch IMS system, the data resource must be available for an individual application program to process. The resource is not available if, for example, a data resource is used exclusively by one IMS, is flagged as needing recovery, or backup procedures are in process.

For DEDBs, the data resource is further divided; each individual area is considered a unit of data resource. When this chapter refers to "database", it is equivalent to a DEDB area unless otherwise noted.

Here are some of the restrictions that apply to data sharing:

- Batch IMS support excludes use of MSDBs and DEDBs.
- Only IMS online systems that use Fast Path can share DEDBs.
- Data sharing support excludes MSDBs and GSAM databases.

Related Reading For more information about the concepts of IMS data sharing, see *IMS Version 9: Administration Guide: System*. For information about operating an IMS data sharing environment, see *IMS Version 9: Operations Guide*.

The following sections are covered in this chapter:

DBRC and Data Sharing

Concurrent access to databases by systems in one or more operating systems is controlled with a common (shared) Database Recovery Control (DBRC) RECON data set. IMSs perform an automatic sign-on to DBRC, and this action ensures that DBRC knows which IMSs and utilities are currently participating in shared access. Subsequently, a system's eligibility to be authorized for access to a database depends on the declared degree of sharing permitted and other status indicators in the RECON data set.

To maintain data integrity, status indicators in the RECON data set control concurrent access and recovery actions for the databases. This common RECON data set is required in a data sharing IMSplex because a given database must have a DMB number that uniquely identifies it to all the sharing subsystems. The DMB number that DBRC records in its RECON data set is related to the order in which databases are registered to DBRC. Using multiple RECON data sets can result in the same DMB number existing in each RECON data set for different databases. This condition can result in damage to databases.

Databases that are to take part in data sharing must be registered in RECON. Each registered database has a current status that reflects whether it can take part in sharing and the scope of the sharing. The concept of scope combines several ideas:

- The type of access--read or update
- Whether more than one access can occur within the database simultaneously
- Whether an IMS needing access is in the same or a different operating system

Related Reading For more information about DBRC, see Chapter 26, "Database Recovery Control (DBRC)," on page 263.

How Applications Share Data

To understand data sharing, you must understand how applications and IMSs share data.

The processing options for an application program are declared in the PSB and express the intent of the program regarding data access and alteration. They are specified with the PROCOPT keyword as part of the group of statements that make up the PCB for a particular database access. The PCB declaration implies a processing intent.

If the application program is to insert, delete, replace, or perform a combination of these actions, the application program is said to have update access. An online program having exclusive access, specified as PROCOPT=E, is interpreted as having update access.

Programs that need access to a database but do not update the data can do so in two ways. They can access the data with the assurance that any pending changes have been committed by the program that instigated the change; this is termed read access (PROCOPT=G). Alternatively, they can read uncommitted data, if the program does not specify protection of data status. This is termed read-only access (PROCOPT=GO).

Related Reading For more information about PROCOPT values, see *IMS Version 9: Utilities Reference: System*.

Chapter 9. The Database Reorganization Process

In this chapter, we provide an overview of the database reorganization tasks that will need to be performed by the IMS database administrator function. We start with general background information regarding IMS database reorganization, then look in more detail at reorganizing HD databases.

As of IMS Version 9, you can reorganize HALDB databases without taking them offline. For more information, see “Online Reorganization” on page 97.

Specifically, this chapter:

- Introduces the function of database reorganization in an IMS environment. It is a first-time general introduction into the requirements for, and the process of, IMS database reorganization.
- Gives a formal description of the available IMS utilities for reorganizing HD databases.
- Introduces the use of the utilities for particular situations. It describes what needs to be run to reorganize an HD database with and without logical relationships or secondary indexes. It also looks at partial reorganization of HD databases.
- Finally, there is a short discussion on initial loading of databases with logical relationships and secondary indexes, because this also requires the reorganization utilities to build the logical relationships and secondary indexes

The following sections are covered in this chapter:

Purpose of Reorganization

Reorganization is the process of changing the physical storage and/or structure of a database to better achieve the application’s performance requirements. We distinguish between the following two types: physical reorganization, to optimize the physical storage of the database; and restructuring, to alter the database structure.

The most common reasons a database will need reorganizing are:

- To reclaim and consolidate free space that has become fragmented due to repeated insertion and deletion of segments
- To optimize the physical storage of the database segments for maximum performance (get dependent segments that are in distant blocks, increasing physical I/O, back in the same block as the parent and/or root). This situation is normally the result of high update activity on the database
- To alter the structure of the database, change the size of the database data sets, alter the HDAM root addressable area, add or delete segment types

The first two reasons would be described as reorganization, the last one as restructuring. The need for reorganization is always due to change, either setting up a new database, amending the structure of the database as application requirements change, or as a result of update activity against the database. If you do not update a database, then once you have gotten it to an optimum state for performance, there is no further need to reorganize it.

Reorganizing and restructuring the databases is only part of the process of tuning and monitoring access to IMS databases. There are also many things that can be done to tune the database manager component in the IMS subsystem and the applications accessing of the databases. This is covered in detail in chapters 11 and 12 of the ITSO publication *IMS Version 5 Performance Guide*.

When to Reorganize

There are no fixed rules about when to reorganize. There are two approaches to deciding when to reorganize, reactive and proactive. You will probably do a mixture of both. When you initially install the application and set up the databases, a lot of the reorganization will be done reactively, as performance and space problems manifest themselves (while you can reduce this by careful analysis of the databases and application access to them, there will normally be things that only come to light after implementation). As you develop a history of the behavior of the application and the databases, the scheduling of reorganization should become more proactive.

Reactive scheduling of reorganization will normally be a result of perceived problems with the performance of the application, or problems with shortage of freespace in the database.

Where there are perceived application performance problems, you need to monitor closely what the application is doing. The initial thing to look at is, what the average and maximum online response times and batch run times are. Are they excessive for the amount of work the application is doing? The ITSO publication IMS Version 5 Performance Guide, SG24-4637 covers in great detail monitoring and investigating performance of IMS application and subsystems. If there are performance problems, then go through the process described in the document to monitor the performance and identify where the problems are.

Only once you have gone through the procedures detailed in this document and identified potential problems with the databases should you start to look at reorganizing the database. Do not look only at the total time that the application program takes for database processing, but also look at the amount of database calls it is processing. For example, if an online application is taking 10 seconds for database processing, but is reading 3-4000 database segments, then there may be little room for database tuning. However, you may want to look more closely at why (and whether) the application really needs to read all these segments. The solution to performance problems is normally an interactive process involving the database administrator, application support function, and the operating system support function, as all three control areas that affect performance.

When you encounter problems due to shortage of space in database data sets, there is little you can do but schedule a database reorganization to increase the database size. However, you should then pursue the growth rate with the application support function (this is where it is useful to have a history of the volume of the application data stored in the database over time). Questions to ask are whether growth will continue at the current rate, or at a different rate, and whether this data all needs to be online. Remember there are finite architectural limits to the size of the databases which vary depending on the IMS and operating system access methods.

The proactive approach to scheduling database reorganization relies on regular monitoring of the databases. Some products for monitoring the databases are covered in more detail in “Monitoring the Database” on page 88. In addition, you should maintain a history of the monitoring information you collect, so you can

analyze this for trends and schedule database reorganization and restructuring before any problems occur. When you decide to make a change to a database, only change one thing at a time, if possible, and then monitor application performance before and after the change so you can see what effect this one change had.

The main things you will be doing when you look at the monitoring data will be to try to minimize the physical I/O for each database access, and optimize the free space available in the database so it is not excessive, but sufficient for normal update access on the databases.

The physical I/O from the disk storage into the buffers in the IMS subsystem is the major component of the elapsed time for database access. You will want to minimize this by:

- Making the best use of buffers in the IMS subsystem; the more requests for database access you satisfy from the buffers, the fewer physical I/Os are necessary. This is covered in the IMS Version 5 Performance Guide, SG24-4637
- Minimizing the number of physical I/Os when a segment does have to be retrieved from disk. For example, trying to place as many dependents as possible in the same block/CI as its parent, ensuring HDAM root segments are in the same block/CI as the RAP. This is where database reorganization and restructuring is used

While there are no fixed guidelines for when to reorganize an IMS database, the following guidelines were used successfully with a medium-sized commercial application using IMS HD databases stored in VSAM files. You may wish to use them as a starting point for scheduling database reorganization and, when you have monitored the effects of the reorganization, adjust these parameters accordingly.

HD databases (HDAM and HIDAM) in general

- Less than 50% of database records have all segments making up the record (root and dependents) in the same block/CI
- Limit your freespace to less than 20%. You way want to increase this limit if you have volatile data or infrequent windows for reorganization

HDAM databases only

- Put less than 75% of root segments in the root addressable area (RAA). Recalculate the RAA (as described in Chapter 6, “Implementing the IMS Database Model,” on page 51). Reorganize the database if calculation of RAA showed it needed to be larger, then restructure at same time.
- Less than 50% of root anchor points (RAPs) point to root segments in the same block/CI. That is, the RAP points to a root that has been placed in another block/CI because there is not room in this block/CI. This causes two I/Os, one to the RAP block, and one to the block that the root is in, instead of one I/O.

VSAM or OSAM file

Put the file in secondary extents. You may wish to resize the file, if this is caused by growth.

VSAM KSDS

- When your VSAM KSDS (index) has CA splits or more than 15 CI splits.
- When your VSAM KSDS (index) has less than 20% free space (as IMS manages freespace in VSAM ESDS, this only applies to a KSDS)

For DEDB databases, reorganize when there are lots of database segments in the independent overflow (IOVF) portion of the DEDB area.

Monitoring the Database

Monitor your databases in order to determine when they might need reorganizing.

The database monitoring divides in to two categories. Monitoring program and subsystem access to the databases, and monitoring the structure, space usage and pointer chains in the actual database data sets.

The principle tools provided by IMS that are used to monitor database access are:

- The IMS monitor, to gather details of buffer usage and database calls over a specified time period in an IMS subsystem.
- The //DFSSTAT DD statement, used in batch JCL to provide a summary of buffer usage and database calls. As there is very little overhead in including this statement (the details printed to the DD at region termination are accumulated by the IMS region controller whether they are output or not), it is normally worthwhile putting this in all batch jobs.
- Running the DB monitor on a batch job, to collect similar details to the IMS monitor in an online system. As there is an overhead on running this, it would normally only be turned on when specific problems are being investigated

Related Reading: There are a number of products available to let you monitor the databases and the data sets in which they are stored. For more information about these products, click on the “IMS Tools” link on the IMS Web site at www.ibm.com/ims. For information about monitoring program access to the database, see *IMS Performance Guide*.

Overview of the Reorganization Process

The database reorganization process can vary from very simple to very complex, depending on the databases involved. If the databases involved do not have IMS logical relationships or secondary indexes, then the process is very simple. When logical relationships and secondary indexes are involved the process becomes more involved.

There are three types of reorganization:

- “Offline Reorganization”
- “Online Reorganization” on page 97
- “Fast Path Reorganization” on page 97

Offline Reorganization

The offline process, in its simplest form, is to unload the database, delete and redefine the physical data set, and then reload it. If the database is not involved in any logical relationships and does not have any secondary indexes, then that is the complete process. Database reorganization of HD databases would normally take the following steps if both logical relationships and secondary indexes are involved:

1. Back up the databases (both the data and, if you are changing them, the appropriate control blocks, for example, DBDSs) so you have a fallback point if there are any problems during the reorganization. See Chapter 10, “The Database Recovery Process,” on page 101 for more information.
2. Unload the existing database data sets to sequential files using the IMS utilities. The process in discussed in “Database Unload Process” on page 89.
3. Delete the database data sets. If you are making any changes to the definitions of the database data sets, make them now, remembering to save the old definitions as a fallback.

4. Redefine the database data sets.
5. This step is only necessary if you are making any changes to the database structure by altering the DBD. Make the changes to the DBD and reassemble it by running the DBDGEN utility. Then run the ACBGEN utility with DBD= parameter to ensure all appropriate control blocks are regenerated. It cannot be overemphasized that you must make sure all programs/utilities use the new versions of the control blocks if you change the DBD; otherwise, database corruption will result.
6. Run the IMS utilities to reload the database. If you have altered the DBD, the utility, and any subsequent programs/utilities, should use the new DBD.
7. If the database has secondary indexes, or participates in logical relationships, then you will need to run additional utilities to rebuild these connections. These connections (unless using symbolic pointers) rely on the database segments relative position in the database, which has been altered by the reorganization. The utilities will determine the new positions and amend the direct pointers in the indexes and logically related databases.
8. If your databases are registered with DBRC (and they **should** be registered), then you will need to take an image copy of the reorganized databases. This is for the same reason as above. IMS database forward recovery, using changes recorded in IMS logs, relies on the position of the segments relative to the start of the data set, which is altered by the reorganization. You need to take the image copies to establish a new base from which the databases can be rolled forward.

Database Unload Process

The unload processing for HD databases is very simple. The HD unload utility will unload the main database and the primary index data set if the database is HIDAM. The output of the utility is a sequential data set which is input to the HD reload utility. If the database is a HIDAM database, then the primary index database must also be present. The utility can only unload a single database at a time. If there are logically related databases which are to be reorganized at the same time then the step should be executed once for each database. Figure 23 shows a diagram of the utility.

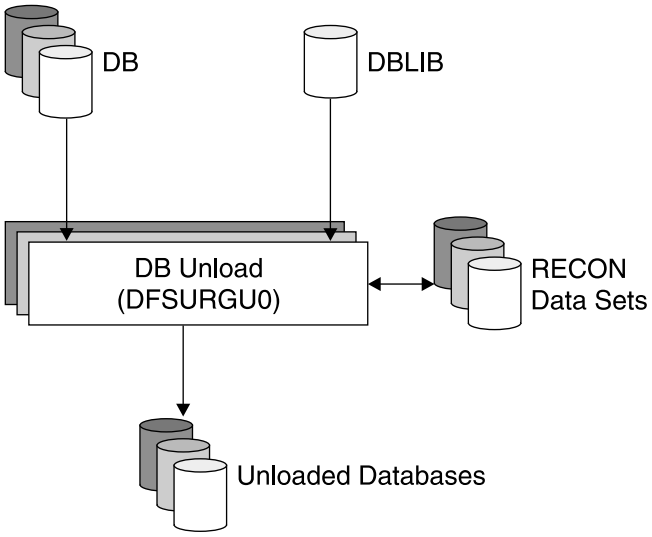

Figure 23. Database Unload Processing

There are some considerations to be kept in mind when planning for the unload process:

- IBM highly recommends that you make an image copy of the database before attempting to reorganize it.
- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used; the database DD card(s) must be present.
- If a HIDAM database is being unloaded, the primary index database DD statement must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request RD access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the old DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one output data set.
- The reload utility can only unload one database per job step. To unload multiple databases, you must use multiple job steps.

Defining Databases

If the access method used for the database is VSAM, then an IDCAMS job step is required to delete and redefine the VSAM cluster. The reload utility will fail if the data sets are not empty. If OSAM is used, the a DISP=OLD can be used to overwrite the data set. However, if the database is on more than a single DASD volume, IBM highly recommends that you delete the data set and redefine it (IEFBR14) to ensure that the correct end-of-file marker is placed.

Database Reload Process

The reload processing can be more complex then the unload processing. If the database is does not have any secondary indexes and is not involved in a logical relationship, then the database can simply be reloaded.

The reloading of the database itself is the same. However, there are additional utility programs that need to be run before and after the database is reorganized to rebuild logical relationships and secondary indexes so they reflect the new physical positions of the segments in the reorganized database. Until all this processing is complete, the logical relationships and secondary indexes are not usable. If you attempt to use them before completing this process, the applications will fail with IMS abend codes indicating that there are invalid IMS pointers.

The following sections discuss each combination of reload processing required. They are:

- “Reload Only”
- “Reload With Secondary Indexes” on page 91
- “Reload With Logical Relationships” on page 93
- “Reload With Logical Relationships and Secondary indexes” on page 94

Reload Only: The reload processing for a HD database without any logical relationships or secondary indexes is shown in Figure 24 on page 91.

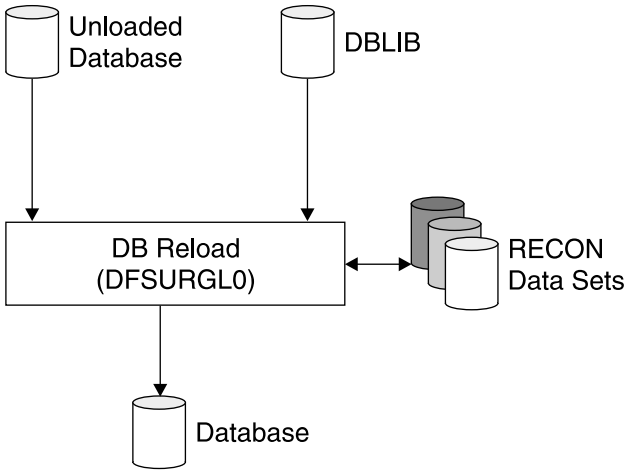

Figure 24. Overview of Database Reload Only Process

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used; the database DD statement(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one input data set.
- The reload utility can only reload one database per job step. To reload multiple databases you must use multiple job steps.
- The DFSURWF1 DD statement can be specified as DUMMY.

Reload With Secondary Indexes: The reload processing for an HD database but with secondary indexes requires the use of the prereorganization utility. It is used to define which databases are involved in the secondary index relationship. A control file is created with this information and passed to the subsequent utilities.

The HISAM unload utility will read the DFSURIDX data set which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The HISAM reload utility can reload all the secondary index database unloaded by the HISAM unload utility in one JOB step. Figure 25 on page 92 illustrates the reload process when secondary indexes are involved.

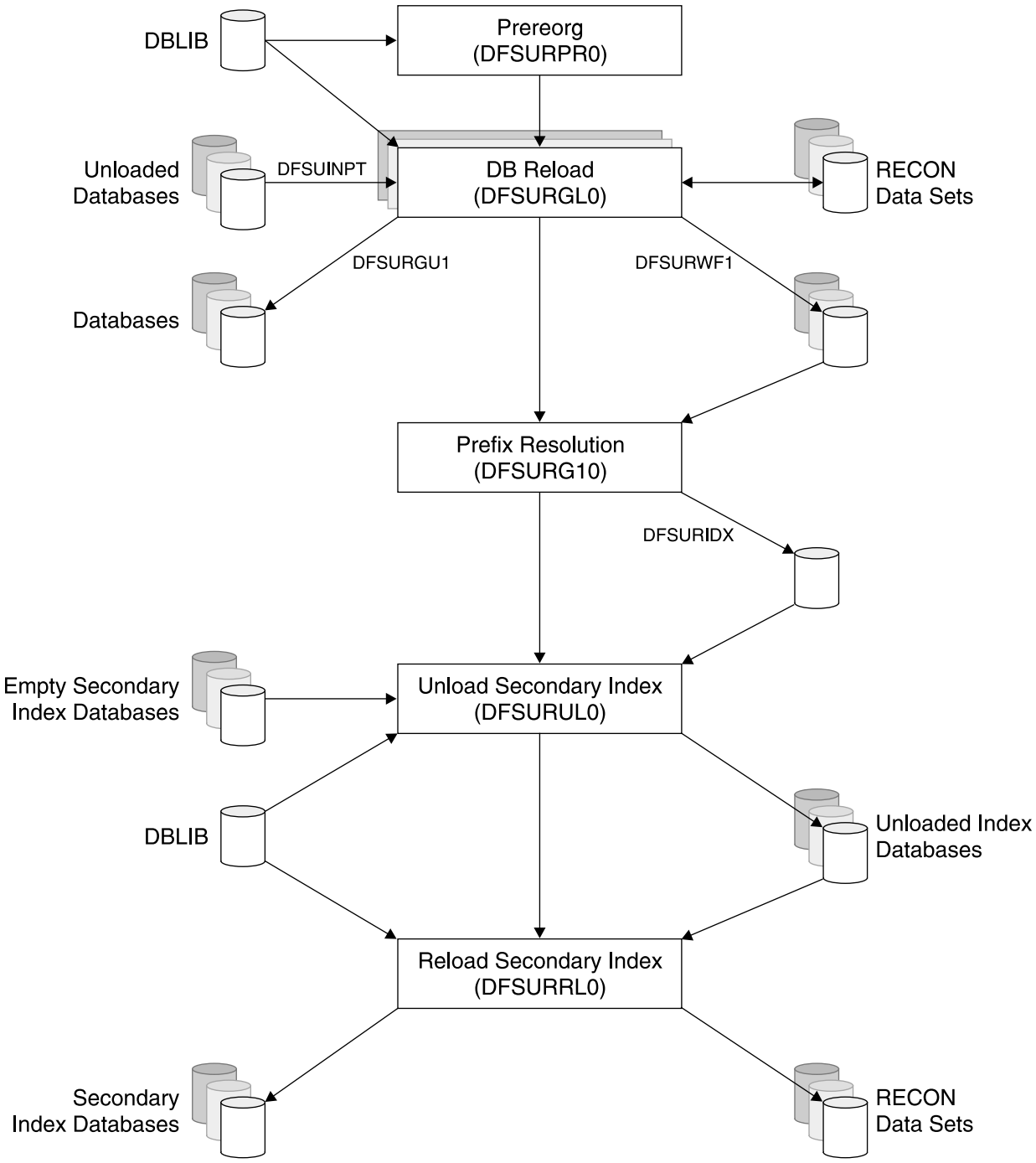

Figure 25. Overview of Reload Processing With Secondary indexes

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD statement(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.

- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one input data set.
- The reload utility can only reload one database per job step. To reload multiple databases, you must use multiple job steps.
- The DFSURWF1 DD statement can be specified as DUMMY, but it must be present.

Reload With Logical Relationships: The reload processing for a HD database but with logical relationships requires the use of the prereorganization utility. It is used to define which databases are involved in the logical relationship. A control file is created with this information and passed to the subsequent utilities. If all the databases logically related to each other are being reloaded then the DBIL option on the control card should be used. These will reset all the pointers and logical parent counters. If not then the DBR option should be used.

All databases involved in the logical relationships should normally be reloaded. The DFSURWF1 work files from all steps should be passed to the prefix update utility as illustrated in Figure 26 on page 94. The HISAM unload utility will read the DFSURIDX data set, which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The prefix resolution utility will extract the RBAs from the required segments and sort them. This file will be passed the prefix update utility to update the database segment prefixes.

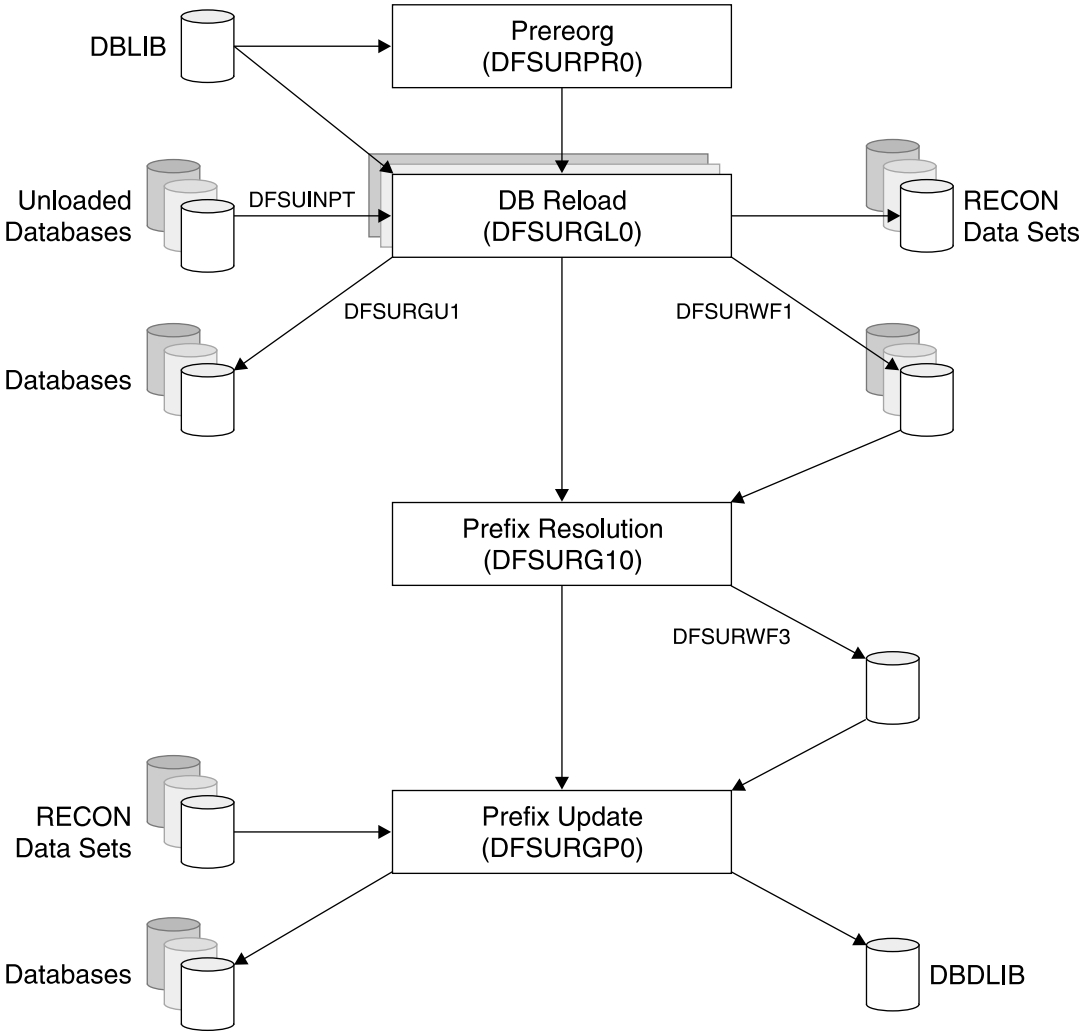

Figure 26. Overview of Database Reload Process When Logical Relationships Exist

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD statement(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- The reload utility can only reload one database per job step. To reload multiple databases, you must use multiple job steps.
- The DFSURWF1 DD statement must be present.
- The prefix update utility will acquire EX access to the databases being updated.
- The IMAGE COPY NEEDED flag will be set on by the reload utility.

Reload With Logical Relationships and Secondary indexes: The reload processing for both secondary indexes and logical relationships is a combination of

both the individual reload processes described in “Reload Only” on page 90, “Reload With Secondary Indexes” on page 91, and “Reload With Logical Relationships” on page 93.

The reload processing for a HD database but with secondary indexes and logical relationships requires the use of the prereorganization utility. It is used to define which databases are involved in the relationships. A control file is created with this information and passed to the subsequent utilities.

The prefix resolution utility will extract the RBAs from the required segments and sort them. This file will be passed the prefix update utility to update the database segment prefixes. It will also create a file with the secondary index information to be passed the HISAM unload utility.

The HISAM unload utility will read the DFSURIDX data set which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The HISAM reload utility can reload all the secondary index database unloaded by the HISAM unload utility in one JOB step. Figure 27 on page 96 illustrates the reload process.

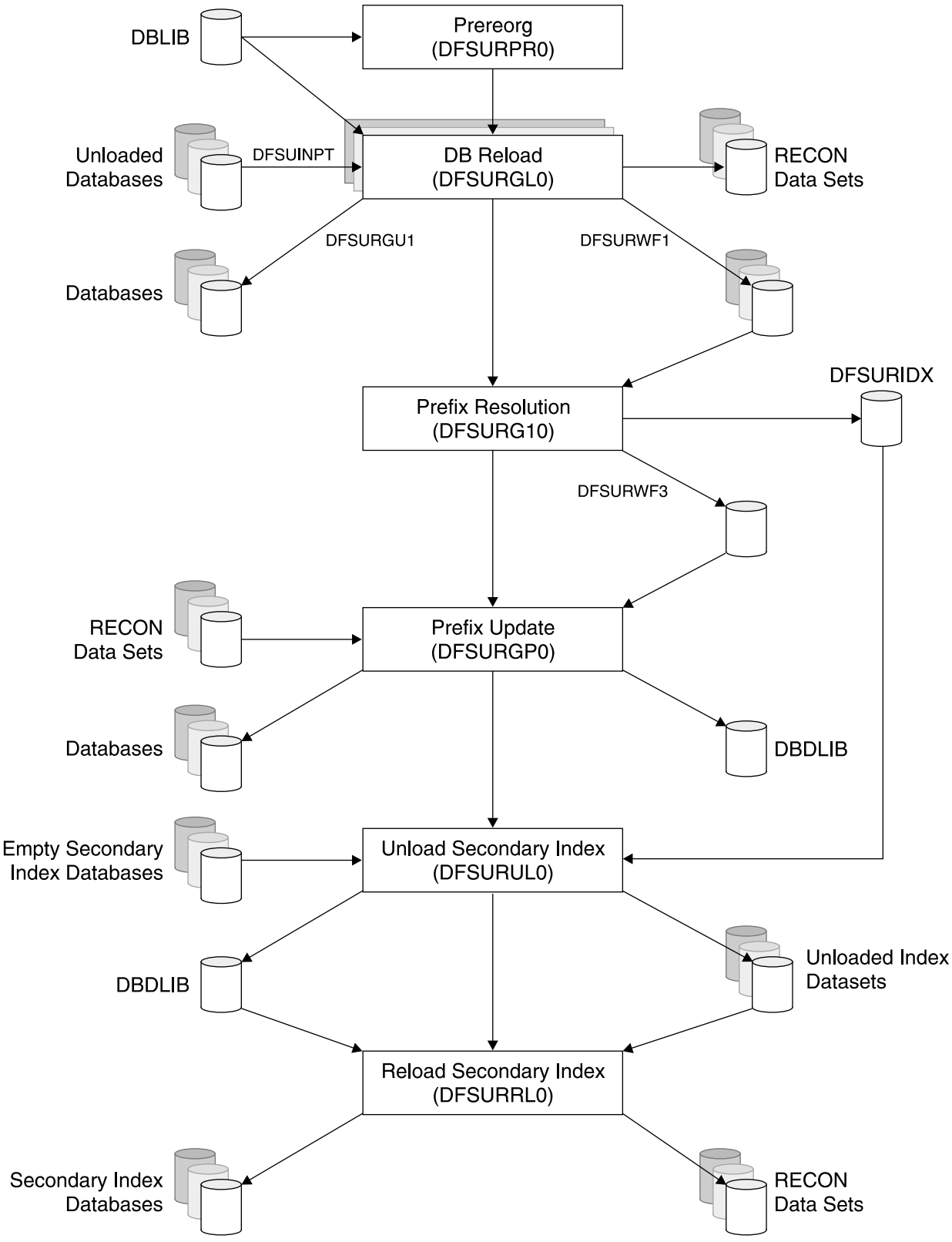

Figure 27. Overview of the Database Reload Process When Secondary Indexes and Logical Relationships Exist

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD statement(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used
- The reload utility can only reload one database per job step. To reload multiple databases you must use multiple job steps.
- The DFSURWF1 DD statement must be present.
- The prefix update utility will acquire EX access to the databases being updated.
- The IMAGE COPY NEEDED flag will be set on by the reload utility.

Fast Path Reorganization

The process for reorganizing a Fast Path DEDB can be appreciably different.

If you are only reorganizing to reclaim fragmented free space and/or get the best placement of the segments for performance (that is, DBD/data set definitions not being changed), then you can run the high speed DEDB direct reorganization utility DBFUHDR0. This can be run without making the database unavailable (that is, no service outage). See *IMS Version 9: Utilities Reference: Database and Transaction Manager* for further details.

If you are reorganizing a DEDB to alter the structure, then you need to have your own user-written programs to unload and reload the database data set at the appropriate points, or use the DEDB unload/reload utility programs from the separately priced IMS Database Tools (DBT) V2, 5685-093. You also need to run the DEDB initialization utility, provided with the IMS base product, immediately prior to reloading the database. However, as the DEDB does not support secondary indexes and logical relationships, you do not have to worry about running further utilities after the database is reloaded.

More information about the database reorganization process, and what steps you have to take to alter specific attributes of the structure of the database are in the chapter on monitoring and tuning the databases in *IMS Version 9: Administration Guide: Database Manager*.

Online Reorganization

With offline reorganization, the database is unavailable during the reorganization process. With online reorganization, which is available only for HALDB databases, most of the database remains available for updates during the reorganization process.

The HALDB Online Reorganization function provides non-disruptive reorganization of HALDB PHDAM and PHIDAM partitions. Online Reorganization reduces the planned data outage time, which is the largest amount of time that data is generally unavailable.

The online reorganization of a HALDB PHDAM or PHIDAM partition, upon command initiation, will run in the DLISAS address space. The dual data set,

cursor-based, reorganization function is performed non-disruptively. That is, concurrent IMS updates are allowed while small amounts of data are moved and reorganized.

Online reorganization has extended the data definition and data set naming convention established for HALDB. Multiple data set groups in a HALDB database use the characters A-through-J in the DDNAMEs and data set names of the supported ten data set groups, and the primary index for a PHIDAM database uses the character X in these names. This has been expanded by implementing the characters M-through-V and Y for an alternate (or paired) set of data sets.

Before online reorganization starts, there is a single active set of data sets for the HALDB partition: either the A-through-J and X set, or the M-through-V and Y set. The data sets in the other (inactive) set contain no useful information and one or more of these data sets need not even exist before the reorganization is started.

Ownership of the online reorganization is established during initialization and is recorded in the partition database record in the RECON data set. After ownership of an online reorganization is established, no other IMS subsystems are allowed to obtain ownership. Ownership can be released by this IMS prior to the end of initialization or prior to the completion of the online reorganization; then another IMS can obtain ownership and finish the online reorganization.

When the entire initialization process (including the validation or possible automatic creation of the output data sets) is complete, the active set of data sets is treated as the input set, and the inactive set becomes the output set. At the end of this initialization process, the online reorganization of the HALDB partition is recorded in the RECON data set with line (ONLINE REORG ACTIVE=YES) that shows a cursor-active status.

When the cursor-active status is recorded, and until this reorganization completes or until a batch reorganization reload is done, the HALDB partition is comprised of both the A-through-J and X set of data sets and the M-through-V and Y set of data sets. During this time, the HALDB partition cannot be accessed unless both sets of data sets are physically available. Database records are then copied from the input to the output data sets in multiple units of reorganization. During the reorganization, IMS application programs can make database changes to the parts of the input data sets that have not yet been copied to the output data sets and to parts of the output data sets to which data have already been copied.

Figure 28 on page 99 illustrates the conceptual relationship between the database records in the input and output data sets at a point during the reorganization.

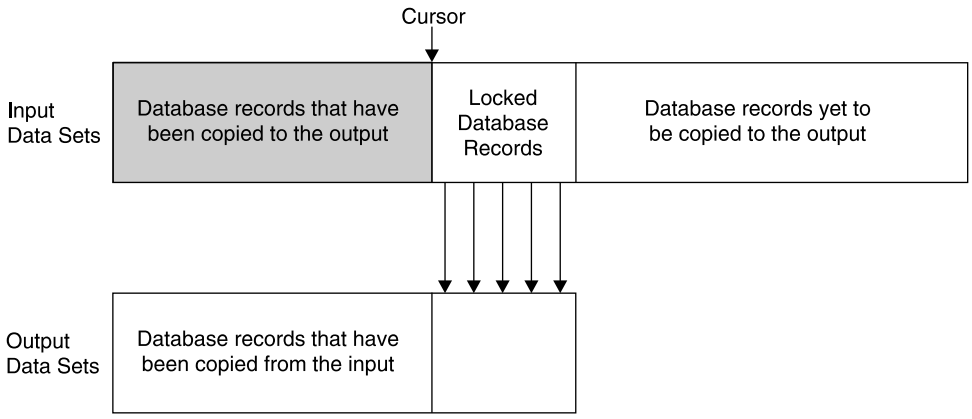

Figure 28. Relationship Between DB Records in the Input and Output Data Sets at a Point During Reorganization

Related Reading: For complete information about HALDB Online Reorganization, see the *IMS Version 9: HALDB Online Reorganization Guide and Reference*.

Reorganization Utilities

The IMS utilities available for database reorganization are described in *IMS Version 9: Administration Guide: Database Manager* and *IMS Version 9: Utilities Reference: Database and Transaction Manager*. The following sections briefly describe these utilities.

The reorganization utilities can be classified into three groups, based on the type of reorganization you plan to perform:

- "Partial Reorganization"
- "Reorganization Using the Utility Control Facility" on page 100
- "Reorganization Without the Utility Control Facility" on page 100

Partial Reorganization

If you are reorganizing an HD database, you can reorganize parts of it rather than the whole database. You would need to reorganize parts, rather than all of it, for two reasons:

- Only parts of it need to be reorganized.
- By reorganizing only parts of it, you can break the amount of time it takes to do a total reorganization into smaller pieces.

The utilities that perform a partial reorganization are:

- The Database Surveyor utility, which helps you determine which parts of your database to reorganize.
- The Partial Database Reorganization utility, which does the actual reorganization.

Note: The Partial Reorganization, Prefix Resolution and Prefix Update utilities do not apply to HALDBs.

Reorganization Using the Utility Control Facility

Reorganization can be done using a single program, called the Utility Control Facility (UCF), or by using various combinations of utilities. When UCF is used, it acts as a controller, determining which of the various reorganization utilities needs to be run and then running them. UCF:

- Reduces the number of JCL statements you must create.
- Eliminates the need to sequence the running of the various utilities.
- Allows you to stop and then later restart a job.
- Reduces the number of decisions operations people must make.

Note: The only reorganization utilities that cannot be run under the control of UCF are the Database Surveyor utility and the Partial Database Reorganization utility. Also, UCF does not support HALDBs.

Reorganization Without the Utility Control Facility

When you do not use UCF, reorganization of the database is done using a combination of utilities. Which utilities you need to use, and how many, depends on the type of database and whether it uses logical relationships or secondary indexes.

If your database does not use logical relationships or secondary indexes, you simply run the appropriate unload and reload utilities, which are as follows:

- For HISAM databases, the HISAM Reorganization Unload utility and the HISAM Reorganization Reload utility
- For HIDAM index databases (if reorganized separately from the HIDAM database), the HISAM Reorganization Unload utility and the HISAM Reorganization Reload utility
- For SHISAM, HDAM, and HIDAM databases, the HD Reorganization Unload utility and the HD Reorganization Reload utility

If your database does use logical relationships or secondary indexes, you need to run the HD Reorganization Unload and Reload utilities (even if it is a HISAM database). In addition, you must run a variety of other utilities to collect, sort, and restore pointer information from a segment's prefix. Remember, when a database is reorganized, the location of segments changes. If logical relationships or secondary indexes are used, update prefixes to reflect new segment locations. The various utilities involved in updating segment prefixes are:

- Database Prereorganization utility
- Database Scan utility
- Database Prefix Resolution utility
- Database Prefix Update utility

Chapter 10. The Database Recovery Process

The following sections provide an overview of the backup and recovery tasks that are part of administering IMS databases. They give a general background on IMS database backup and recovery concepts and then discuss additional details of the processes involved.

This chapter discusses:

- "When Recovery is Needed"
- "Overview of the Database Recovery Process"
- "IMS Backup and Recovery Utilities" on page 102

When Recovery is Needed

Database recovery is normally on done when there has been a failure of some sort. Most of the time it is done as a result of a system, hardware, or application failure. However, it can be used to return a database to a point-in-time to recover out of application logic failures.

In general, a database may need to be recovered under the following circumstances:

- A DLI batch update job fails after making at least one database update.
- A failure has occurred on a physical DASD device.
- A failure has occurred in a database recovery utility.
- A failure of dynamic backout or batch backout utility has occurred.
- An IMS online system failure and emergency restart has not been completed.

Overview of the Database Recovery Process

Database recovery, in its simplest form, is the restoration of a database after its (partial) destruction due to some failure. In order to facilitate this process, some forward planning needs to be done.

Periodically, a copy of the data in the database is saved. This copy is normally referred to as a backup or image copy. These image copies can reside on DASD or cartridges. Though this process can be done anytime, it is normally done when there is no other database activity at the same time. This creates a complete backup. There are other strategies for taking a database backup, but they will not be discussed in this book.

In addition to taking an image copy of the database(s), all changes made to the data in the database can be logged and saved, at least until the next image copy. These changes are contained in data sets called log data sets. This provides a complete recovery environment so that no data is lost in the event of a system or application failure.

There is an IMS facility called database recovery control (DBRC) that provides database integrity and can be used to help ensure that there is always a recovery process available. Using DBRC to control database backup and recovery is not mandatory, but is *highly recommended*.

Related Reading: For more information about DBRC, see Chapter 26, “Database Recovery Control (DBRC),” on page 263.

The following sections discuss other aspects of the recovery process:

- “Online Programs and Recovery”
- “DL/I Batch Programs and Recovery”

Online Programs and Recovery

IMS online transactions use dynamic backout to “undo” updates done in any incomplete unit of work. Abending online programs are automatically backed out by the online system using the log records. In addition, if the system should fail while an application program is active, any updates made by that program will be automatically backed out when the system is restarted.

If the program was a BMP, the updates are automatically backed out to its most recent checkpoint. Because of this automatic backout, the recovery of individual databases will not be needed.

At IMS restart time, if the emergency restart cannot complete the backout for any individual transactions, then the databases affect by those updates are stopped, and DBRC is requested to set the recovery needed flag to ensure that a correct recovery is completed before the database is opened for more updates. In the case of dynamic backout failure, a batch backout or database recovery needs to be performed, depending on the reason for the backout failure.

DL/I Batch Programs and Recovery

DLI Batch update programs can make use of dynamic backout like BMP, provided the following JCL changes are done:

- The BKO=Y parameter is set in the EXEC statement
- A DASD log data set is provided in the IEFRDER DD statement
- A ROLB Call is issued in the program code for non-system abends

The dynamic backout will then back out the updates to the last checkpoint found on the log data set.

IMS Backup and Recovery Utilities

IMS provides utilities for recovering a database. They are:

“Database Image Copy Utility” on page 104
The Database Image Copy utility is used to create image copies of databases.

“Database Image Copy 2 Utility” on page 105
The Database Image Copy 2 utility is used to take image copies of IMS databases by using the concurrent copy function of the Data Facility Storage Management Subsystem (DFSMS).

Online Database Image Copy Utility
The Online Database Image Copy utility is used to create an as-is image copy of the database while it is being updated by the online system.

“Database Change Accumulation Utility” on page 106
The Database change accumulation utility is used to accumulate database changes from DL/I log tapes since the last complete image copy.

“Database Recovery Utility” on page 107
The Database recovery utility is used to restore the database, using a prior database image copy and the accumulated changes from DL/I log tapes.

“Database Batch Backout Utility” on page 108
The Database backout utility is used to remove changes made to databases by a specific application program.

Another utility program, the system log recovery utility (DFSULTRO), is used to close a log data set in the event of an operating system or hardware failure, thus enabling use of the log by the four principal programs of the recovery system.

For those databases which consist of multiple data sets, recovery is done by individual data set. To recover a complete database composed of multiple data sets, database recovery must be performed for each of its component data sets.

Figure 29 illustrates the relationship between the backup and recovery utilities.

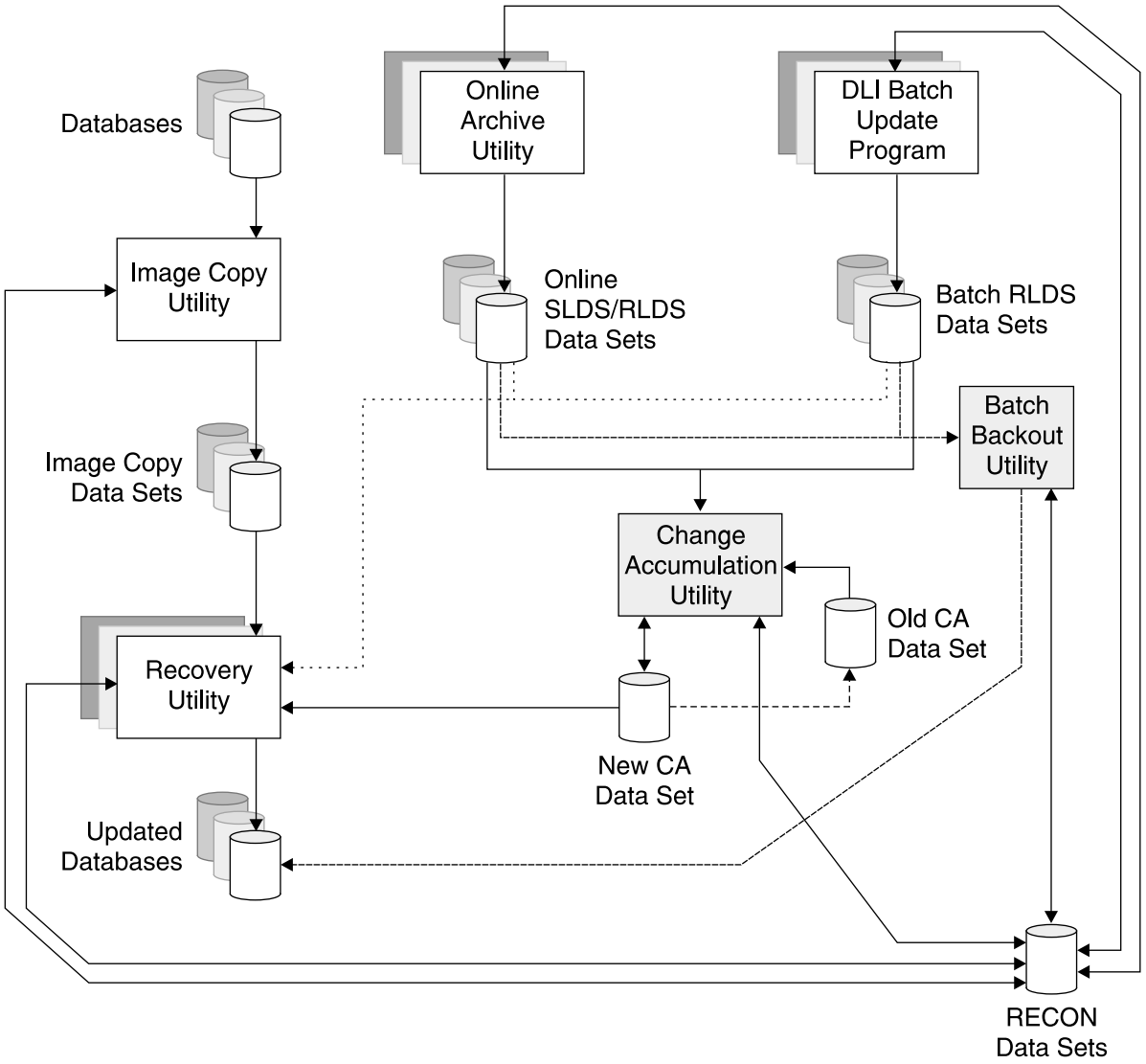

Figure 29. Overview of the Recovery Utilities

Database Image Copy Utility

The database image copy utility (DFSUDMP0) creates a copy of the data sets within the databases. the output data sets is called an IMAGE COPY. It is a sequential data set and can only be used as input to the database Recovery utility. The IMAGE copy utility does not use DLI to process the database. Track I/O is used. There is no internal checking to determine if all the IMS internal pointer are correct. There are tools available to run as part of the image copy utility to do this checking. IBM recommends that at least periodic checking of these internal pointers is done.

There can be no changes to the DBD when this databases is recovered using the IMS recovery utility. In order to make changes to the DBD, a database reorganization is needed to implement those changes.

Multiple databases and data sets can be copied with one execution of the image copy utility. All data sets of a database should be copied at the same time. In our subset, we presume that all database data sets are dumped at the same time, that is, no intervening database processing.

The Database Recovery Control (DBRC) function of IMS can be used to generate the JCL to run this utility if required.

Related Reading: For more information about DBRC, see Chapter 26, “Database Recovery Control (DBRC),” on page 263.

A flow diagram of the database image copy utility is shown in Figure 30.

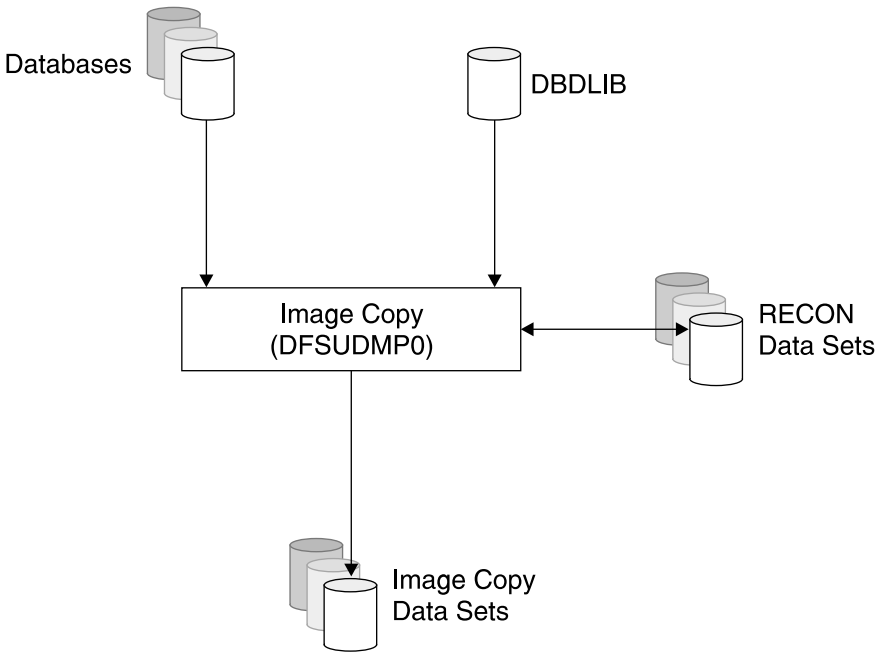

Figure 30. Inputs and Outputs for the Image Copy Utility

Database Image Copy 2 Utility

The Database Image Copy 2 utility (DFSUDMT0) is very similar to the Database Image Copy utility (DFSUDMP0). DFSUDMT0 has several advantages, however, in that it can take image copies with databases being unavailable for a very short time. The Database Image Copy 2 can also take fuzzy KSDS copies, something that Batch Image Copy cannot do.

The Database Image Copy 2 utility takes image copies of IMS databases by using the concurrent copy function of the Data Facility Storage Management Subsystem (DFSMS).

The concurrent copy function of DFSMS is a hardware and software solution that allows you to back up a database or any collection of data at a point in time and with minimum down time for the database. The database is unavailable only long enough for DFSMS to initialize a concurrent copy session for the data, which is a very small fraction of the time that the complete backup will take.

Related Reading: For more information on DFSMS, see *DFSMS V1R5 DFSMSdss Storage Administration Guide*, or *DFSMS V1R5 DFSMSdss Storage Administration Reference*.

The functional differences between the two image copy utilities are:

- The data sets to be copied must reside on a subsystem that supports DFSMS concurrent copy. DBRC is required for this utility. For fuzzy KSDS copies, the database define cluster must specify BWO(TYPIMS) and the KSDS data sets must be managed by SMS.
- An Image copy created by the utility is in DFSMS dump format, rather than standard batch image copy format. The copy is registered with DBRC as an SMSNOCIC or SMSCIC image copy, depending on the parameters specified when the image copy was taken.
- Up to four copies of a data set can be created. Only the primary and secondary (first and second) copies are recorded in the RECON data set.

Related Reading: For more information about the Database Image Copy 2 utility, see *IMS Version 9: Utilities Reference: Database and Transaction Manager*.

A flow diagram of the Database Image Copy 2 utility is shown in Figure 31 on page 106.

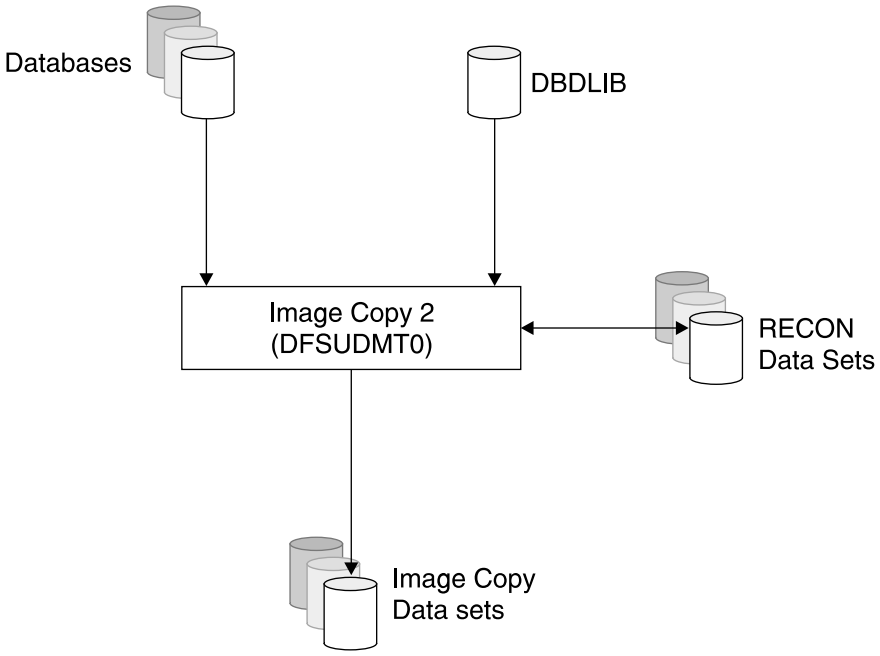

Figure 31. Inputs and Outputs for the Database Image Copy 2 Utility

Database Change Accumulation Utility

The function of the Database Change Accumulation utility (DFSUCUM0) is to create a sequential data set that contains only that database log records from all the log data sets which are necessary for recovery. This accumulation log data set is to be used by the database recovery utility. This accumulation is done by sorting only the required log records in physical record within data set sequence. This provides efficient database recovery whenever needed. The number of log data sets which need to be kept will be significantly reduced.

The change accumulation utility can be run independently of DL/I application programs. The new output database recovery utility.

IBM highly recommends that you use DBRC to create the JCL for each execution of this utility. DBRC will ensure that a complete set of log data sets is used to create the change accumulation data set. The logs records must be supplied to in the correct sequence.

A flow diagram of the change accumulation utility is shown in Figure 32 on page 107.

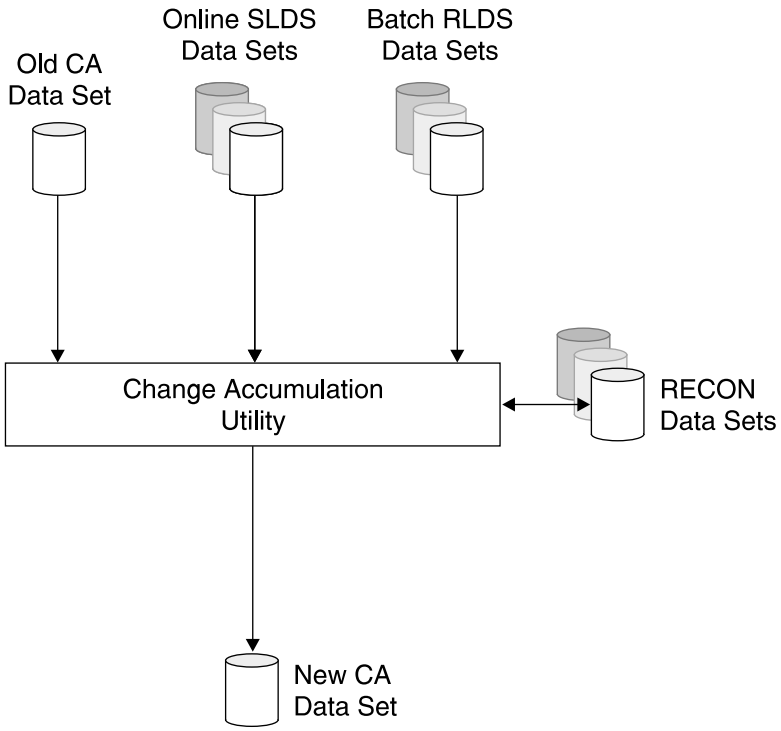

Figure 32. Inputs and Outputs for the Change Accumulation Utility

The input to the database change accumulation utility consists of:

- All log data sets created since either the last image copy utility execution or the last execution of this utility.
- The previous change accumulation data set. This would be the output from the last execution of this utility. The first change accumulation run after a new image copy must not include any old change accumulation data set, that is, those created during the previous period.
- An optional control statement (ID).

Output from the database change accumulation utility consists of a new change accumulation data set. This is a sequential data set containing the combined database records for all database data sets.

Database Recovery Utility

The database recovery utility (DFSURDB0) will restore a database data set. This utility does not provide a means of recovery from application logic errors: it is the user's responsibility to ensure the logical integrity of the data in the database.

Unlike the image copy utility, the recovery utility recovers one database data set per job step. Thus to recover multiple data sets for a database the utility must be run once for each data set.

It is **highly recommended** that DBRC be used to create each execution of this utility. DBRC will ensure that all the correct inputs are supplied.

The recovery utility can be run in a number of ways depending on what input is required. Generally the main input to the recovery utility is the image copy data set. Other input can consist of any log data sets or change accumulation data sets

which might be needed. The utility can be run with only the log information as input, in this case the database already existing would be used.

A flow diagram is shown in Figure 33.

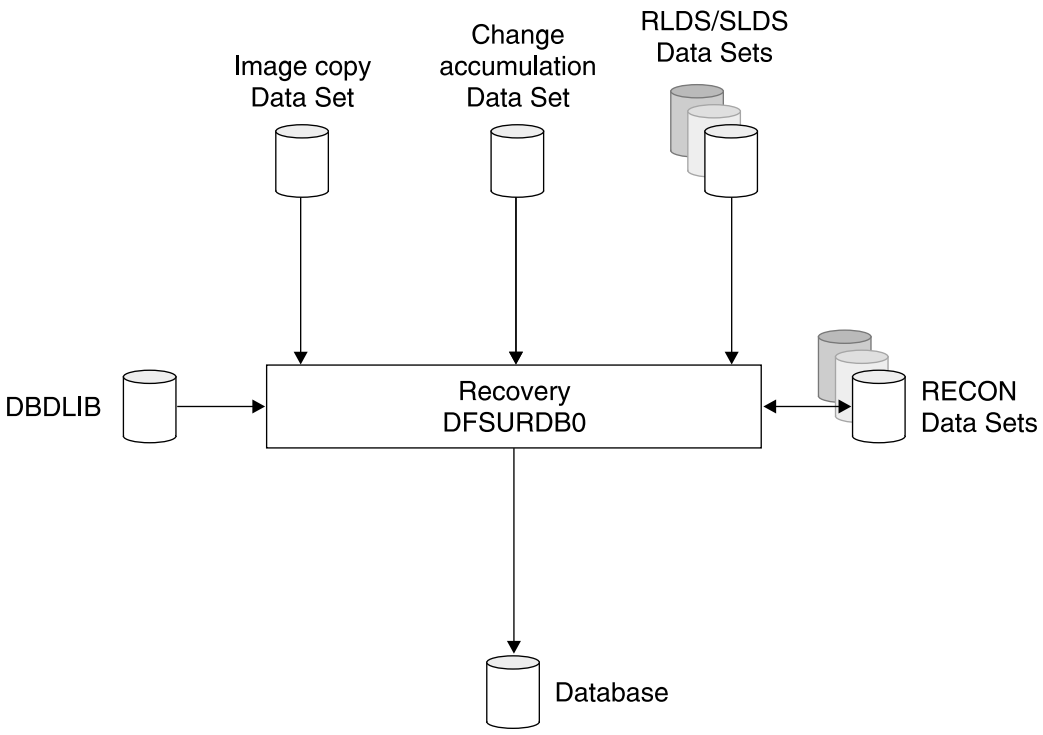

Figure 33. Inputs and Outputs of the Database Recovery Utility

The input to the recovery utility consists of an image copy data set and, optionally, an accumulated change data set and any log data sets not included in the change accumulation data set.

The database recovery utility program is executed in a DL/I batch region. It will allocate the database in exclusive mode so that there can be no other database activity at the time.

Database Batch Backout Utility

Batch backout, in it simplest form, is the reading of log data set (or sets) to back out all database updates. This is done by using the “before image data” in the log record to re-update the database segments. It has the effect of undoing the previous updates.

Note: The Database Batch Backout utility only supports full-function databases.

The database backout utility removes changes in the database which were made by a specific failing program. The following limitations apply:

- The log data set of the failing program must be on DASD.
- No other update programs should have been executed against the same database (s) between the time of the failure and the backout.

The program operates as a normal DL/I batch job. It uses the PSB used by the program whose effects are to be backed out. All databases updated by the program must be available to the backout utility.

Figure 34 illustrates the inputs and outputs for the Batch Backout utility.

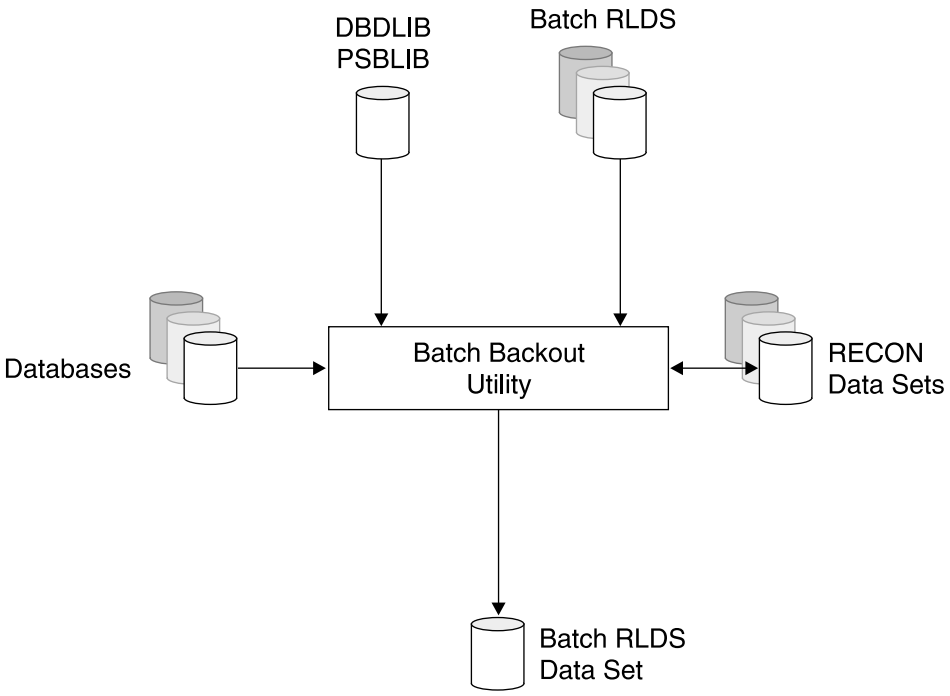

Figure 34. Inputs and Outputs for the Batch Backout Utility

A log data set is created during the backout process. This data set, preceded by the log data set produced for the failing job, must be included in the next change accumulation run, as any other log data set. This data set must not be used as input to any subsequent backout attempt.

Usage Notes for the Batch Backout Utility

Keep the following items in mind when using the Batch Backout utility:

- If checkpoint/restart is not used, then backout always backs out all the database changes of the program.
- If checkpoint/ restart is used (program uses XRST and CHKP-ID calls), then backout will only do backout if the specified CHKP-ID is found on the log data set during read forward. If no CHKP-ID is specified, then the last one on the log data set is used (the first one encountered during read backward).
- If, when using checkpoint/restart, you want to be able to completely back out a job (steps), you must issue a CHKP call immediately after the XRST call, that is, before any real database activity. The CHKP-ID of this call can then be used for a full backout operation.
- To run batch backout for a DLI batch which had completed successfully, the DBRC=“C” parameter must be added to the EXEC PARM keyword.

Part 3. IMS Transaction Manager

Chapter 11. Overview of IMS TM

IMS TM provides a high-performance transaction processing environment for database management systems, such as IMS DB and DB2 UDB for z/OS.

IMS TM can be ordered and installed with or without IMS DB.

The following sections are covered in this chapter:

Functions of IMS TM

IMS TM provides solutions for cooperative processing, distributed database processing, and continuous operation. IMS TM:

- Enhances system management.
- Simplifies network administration.
- Manages and secures the IMS TM terminal network.
- Routes messages from terminal to terminal, from application to application, and between application programs and terminals.
- Queues input and output messages, and schedules messages by associating programs with the transactions they are to process.
- Participates in distributive processing scenarios where other programs (such as WebSphere Application Studio) have a need to access IMS.

IMS TM and the Network

IMS TM interacts with:

- IBM Systems Network Architecture (SNA) network, as currently implemented by the Communication Server for z/OS, which includes the functions of VTAM. IMS TM interacts directly with the Communication Server for z/OS.
- Applications that use the z/OS Advanced Program-to-Program Communication (APPC) protocol.

  Related Reading: For more information about IMS's support for APPC, see "Advanced Program-to-Program Communication (APPC)" on page 114.
- Networks that use Transmission Control Protocol/ Internet Protocol (TCP/IP). Access by using TCP/IP is achieved by way of a separate z/OS address space. This address space uses IMS's Open Transaction Manager Access (OTMA) protocol. The other address space can be another program product such as IBM's Websphere MQ or IMS Connect.

  Related Reading: For more information about OTMA, see "Open Transaction Manager Access (OTMA)" on page 114. For further details on the options available for accessing IMS by using TCP/IP, see:
  - Chapter 30, "Introduction to Parallel Sysplex," on page 315
  - *IMS Version 9: Open Transaction Manager Access Guide and Reference*
  - *IMS Connect Guide and Reference*

Advanced Program-to-Program Communication (APPC)

As mentioned in “Advanced Program-to-Program Communications (APPC)” on page 24, APPC/IMS support for Logical Unit type 6.2 supports the formats and protocols for program-to-program communication.

APPC/VTAM is part of the Communication Server for z/OS. It facilitates the implementation of APPC/IMS support. In addition, APPC/MVS works with APPC/VTAM to implement APPC/IMS functions and communication services in a z/OS environment. APPC/IMS takes advantage of this structure and uses APPC/MVS to communicate with LU 6.2 devices. Therefore, all VTAM LU 6.2 devices supported by APPC/MVS can access IMS using LU 6.2 protocols to initiate IMS application programs, or conversely be initiated by IMS.

APPC/IMS provides compatibility with non-LU 6.2 device types by providing a device-independent API. This allows an application program to work with all device types (LU 6.2 and non-LU 6.2) without any new or changed application programs.

IMS supports APPC conversations in two scenarios:

Implicit
: In this case, IMS supports only a subset of the APPC functions, but enables an APPC incoming message to trigger any standard IMS application that is already defined in the normal manner to IMS, and uses the standard IMS message queue facilities, to schedule the transaction into any appropriate dependent region.

Explicit
: In this case, the full set of CPI Communications (CPI-C) command verbs can be used and the IMS application is written specifically to cater only for APPC triggered transactions. The standard IMS message queues are not used in this case, and the IMS control region only helps create the APPC conversation directly between the APPC client and the IMS dependent region to service this request. The IMS control region takes no further part, regardless of how much time the conversation might use while active.

Open Transaction Manager Access (OTMA)

OTMA provides an open interface to IMS TM customers. With OTMA, a z/OS or TCP/IP application program can send a transaction or command to IMS without using SNA or VTAM. Many programs can connect to IMS TM using OTMA: middleware software, gateway programs, database, and applications written by IMS customers. Each of the programs or applications that communicate with IMS using OTMA are considered OTMA clients.

The OTMA interface itself is very flexible. An OTMA client, an application program of the client, or both, can use OTMA in many different ways. The execution of some transactions can involve complex “handshaking” between IMS and the client program; some transactions can simply use the basic protocol.

The following list illustrates the ways that OTMA can be used to process an IMS transaction:

Commit-then-send
: For commit-then-send (CM0), IMS processes the transaction and commits the data before sending a response to the OTMA client. Input and output messages are recoverable.

Send-then-commit

For send-then-commit (CM1), IMS processes the transaction and sends the response to the OTMA client before committing the data. Input and output messages are non-recoverable.

If the application program uses send-then-commit, you must also decide which synchronization level, or “synclevel” to use. There are three choices:

- None - Output is sent and no response from the client is requested. Data is committed if send is successful. Data is backed out if the send fails.
- Confirm - Output is sent and response from the client is requested. The OTMA client must respond with an ACK or NACK. Data is committed if ACK is received. Data is backed out if NACK is received.
- Syncpt - Output is sent, and response from the client is requested. Use synclevel=syncpt to coordinate commit processing through RRS. The OTMA client must respond with an ACK or NACK. Data is committed if ACK is received and RRS commit is received. Data is backed out if NACK is received or RRS abort is received.

An application can decide, for example, that inquiry transactions should use synclevel=none because there are no database updates and that update transactions should use synclevel=confirm.

The OTMA resynchronization interface ensures that there are no duplicate CM0 input and output messages by using a unique recoverable sequence number in every CM0 message. The client can optionally initiate this during connection time. WebSphere MQ is the primary program that exploits this OTMA interface extensively. A WebSphere MQ application program can send a persistent message to IMS to take advantage of the resynchronization benefit. However, sending a WebSphere MQ non-persistent CM0 message to IMS bypasses the resynchronization service.

Table 2 can be used to help you decide which method is appropriate for your application.

Table 2. OTMA Processing Options

Type of Processing	Commit-then-send (CM0)	Send-then-commit (CM1)
Conversational transactions	Not supported	Supported
Fast Path transactions	Not supported	Supported
Remote MCS transactions	Supported	Supported
Shared queues	Supported in IMS V7 and above	Supported in IMS V8 and above
Recoverable output	Supported	Not supported
Synchronized Tpipes	Supported	Not supported
Program-to-program switch	Supported	Supported. However, if more than one program-to-program switch is performed, only one program processes as send-then-commit. The other program processes as commit-then-send.

IMS TM Messages

The network inputs and outputs to IMS Transaction Manager take the form of messages that are input or output, to or from IMS and the physical terminals (or application programs) on the network (referred to as destinations).

These messages are processed asynchronously (that is, IMS will not always send a reply immediately, or ever, when it receives a message, and unsolicited messages might also be sent from IMS). The messages can be of four types:

- Transactions. The data in these messages is passed to IMS application programs for processing
- Messages to go to another logical destination (for example, network terminals)
- Commands for IMS to process.
- Messages for APPC/IMS to process. Because IMS uses an asynchronous protocol for messages and APPC uses synchronous protocols (that is, it always expects a reply when a message is sent), the IMS TM interface for APPC has to perform special processing to accommodate this.

If IMS is not able to process an input message immediately, or cannot send an output message immediately, then the message is stored on a message queue external to the IMS system. IMS will not normally delete the message from the message queue until it has received confirmation that an application has processed the message or that the message has reached its destination.

Connections to Other IMS and CICS Subsystems

IMS has special protocols for connecting to other IMS systems, such as Multiple Systems Coupling (MSC), and to other CICS and IMS systems, such as Intersystem Communication (ISC), that allows work to be routed to and from the other systems for processing.

The MSC connections can be through the network to other IMS systems on the same or other z/OS systems, by using channel-to-channel connections to the same or another channel attached z/OS system or by using cross memory services to another IMS subsystem on the same z/OS system.

The ISC links to other CICS or IMS systems is provided over the network by using VTAM's LU 6.1 protocol.

Multiple Systems Coupling (MSC)

MSC is a part of the IMS Transaction Manager that provides the ability to connect geographically dispersed IMSs. MSC enables programs and operators of one IMS to access programs and operators of the connected IMSs. Communication can occur between two or more (up to 2036) IMSs running on any supported combination of operating systems.

MSC permits you to distribute processing loads and databases. Transactions entered in one IMS system can be passed to another IMS system for processing and the results returned to the initiating terminal. Terminal operators are unaware of these activities; their view of the processing is the same as that presented by interaction with a single system.

MSC only supports connecting one IMS to one other IMS. MSC supports transaction routing between the participating IMSs by options specified in the IMS system definition process.

The IMS system where the transaction is entered by the terminal user is referred to as the front-end system. The IMS system where the transaction is processed is referred to as the back-end system.

The transaction is entered in the front-end system, and based on the definitions in the IMS stage 1 definition, the transaction is sent to the back-end system. When the transaction reaches the back-end system, all standard IMS scheduling techniques apply. After processing, the results are sent back to the front-end system, which then returns the results to the terminal user, who was unaware that any of this occurred.

Intersystem Communications (ISC)

ISC is also part of the IMS Transaction Manager and is another way to connect multiple subsystems. ISC is more flexible than MSC, in that ISC supports the following connections:

- IMS-to-IMS
- IMS-to-CICS
- IMS-to-user-written VTAM program

The transaction routing specification for ISC is contained in the application program, instead of in the IMS system definition as in MSC.

ISC links between IMS and CICS use the standard LU 6.1 protocol available within the network. They can use standard VTAM connections or direct connections.

As defined under SNA, ISC is an LU 6.1 session that:

- Connects different subsystems to communicate at the application level.
- Provides distributed transaction processing permitting a terminal user or application in one subsystem to communicate with a terminal or application in a different subsystem and, optionally, to receive a reply. In some cases, the application is user written; in other cases, the subsystem itself acts as an application.
- Provides distributed services. Therefore, an application in one subsystem can use a service (such as a message queue or database) in a different subsystem.

SNA makes communication possible between unlike subsystems and includes SNA-defined session control protocols, data flow control protocols, and routing parameters.

MSC Versus ISC

As mentioned in “Multiple Systems Coupling (MSC)” on page 116 and “Intersystem Communications (ISC),” both MSC and ISC enable a user to:

- Route transactions
- Distribute transaction processing
- Grow beyond the capacity of one IMS system

Both ISC and MSC take advantage of the parallel session support VTAM provides. Some key differences exist, however. Table 3 on page 118 shows the major functions of MSC and ISC and shows the differences in support.

Table 3. Comparing MSC and ISC Functions

MSC Functions	ISC Functions
MSC connects multiple IMS systems only to each other. These IMS systems can all reside in one processor, or they can reside in different processors.	ISC can connect either like or unlike subsystems, as long as the connected subsystems both implement ISC. Thus, a user can couple an IMS subsystem to: • Another IMS subsystem • A CICS subsystem • A user-written subsystem
Communication in the MSC environment is subsystem-to-subsystem.	Communication is between application programs in the two subsystems. The subsystems themselves are session partners, supporting logical flows between the applications.
Processing is transparent to the user. That is, to the user, MSC processing appears as if it is occurring in a single system.	Message routing requires involvement by the terminal user or the application to determine the message destination because ISC supports coupling of unlike subsystems. Specified routing parameters can be overridden, modified, or deleted by Message Format Service (MFS).
When not using the MSC-directed routing capability, the terminal operator or application program does not need to know routing information. Routing is automatic based on system definition parameters.	ISC provides a unique message-switching capability that permits message routing to occur without involvement of a user application.
MSC supports the steps of a conversation to be distributed over multiple IMS subsystems, transparent to both the source terminal operator and to each conversational step (application).	ISC supports the use of MFS in an IMS subsystem to assist in the routing and formatting of messages between subsystems.
MSC does not support the use of the Fast Path Expedited Message Handler (EMH).	ISC supports the use of Fast Path Expedited Message Handler (EMH) between IMS subsystems.

Chapter 12. IMS TM Control Region

The IMS TM control region is a z/OS address space that can be initiated through an z/OS START command or by submitting JCL. The terminals, databases, message queues, and logs are all attached to this region. A type 2 supervisor call routine is used for switching control information, messages, and database data to the dependent regions and back.

The control region normally runs as a system task and uses z/OS access methods for terminal and database access.

The following sections are covered in this chapter:

IMS Messages

The goal of IMS TM is to perform online transaction processing. This consists of:

1. Receiving a request for work to be done. The request is entered at a remote terminal. It is usually made up of a transaction code, which identifies to IMS the kind of work to be done and some data that is to be used in doing the work.
2. Initiating and controlling a specific program that will use the data in the request to do the work the remote operator asked to be done, and to prepare some data for the remote operator in response to the request for work (for example, acknowledgment of work done or answer a query).
3. Transmission of the data prepared by the program back to the terminal originally requesting the work.

The above sequence is the simplest form of a transaction. It involves two messages: an input message from the remote operator requesting that work be done, and an output message to the remote operator containing results or acknowledgment of the work done.

Multiple and Single Segment Messages

A message, in the most general sense, is a sequence of transmitted symbols. In the context of IMS, this is called a transmission. A transmission may have one or more messages, and a message may have one or more segments. A segment is defined by an end-of-segment (EOS) symbol, a message is defined by an end-of-message (EOM) symbol and a transmission is defined by an end-of-data (EOD) symbol. The valid combinations of the conditions represented by EOS, EOM, and EOD can be found in Table 4.

Table 4. Valid Combinations of the EOS, EOM, and EOD Symbols

Condition	Represents
EOS	End of segment
EOM	End of segment / end of message
EOD	End of segment / end of message / end of data

The relationship between transmission, message and segment is shown in Figure 35 on page 120.

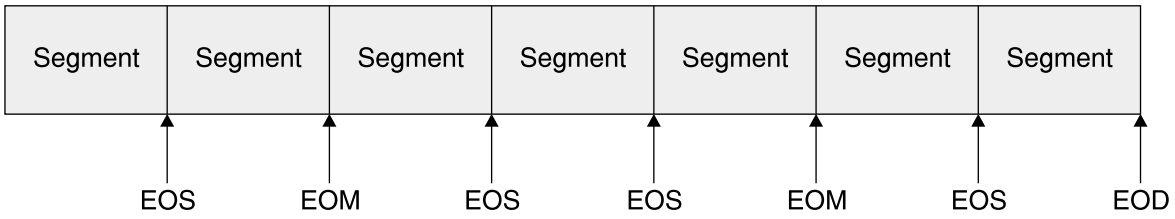

Figure 35. Transmission, Message, and Segment Relationships

The character values or conditions that represent the end of segment and the end of the message (or both) depend on the terminal type.

For 3270 terminals, the physical terminal input will always be a single segment message and transmission. The EOS, EOM, and EOD condition will all be set after the enter or program function key is pressed and the data is transmitted.

On the output side, a message can be divided into multiple segments. Also an application program can send different messages to different terminals, that is, a message to a printer terminal and a message to the input display terminal. Each segment requires a separate insert call by the application program.

The format of a message segment as presented to or received from an application program is shown in Figure 36, where:

LL Total length of the segment in bytes, including the LL and ZZ fields.

ZZ IMS system field

DATA Application data, including the transaction code

Figure 36. Format of a Message Segment

LL	ZZ	Data
2 bytes	2 bytes	n bytes

IMS Transaction Flow

Once the control region is started, it will start the system dependent regions (DLISAS and DBRC). The MPR and BMP regions can be started by:

- IMS jobs
- JOB submission
- Automated operations commands

The general flow of a message from a message processing program (MPP) is shown in Figure 37 on page 121. The intent of this figure is to give a general flow of the message through the system and not a complete detailed description.

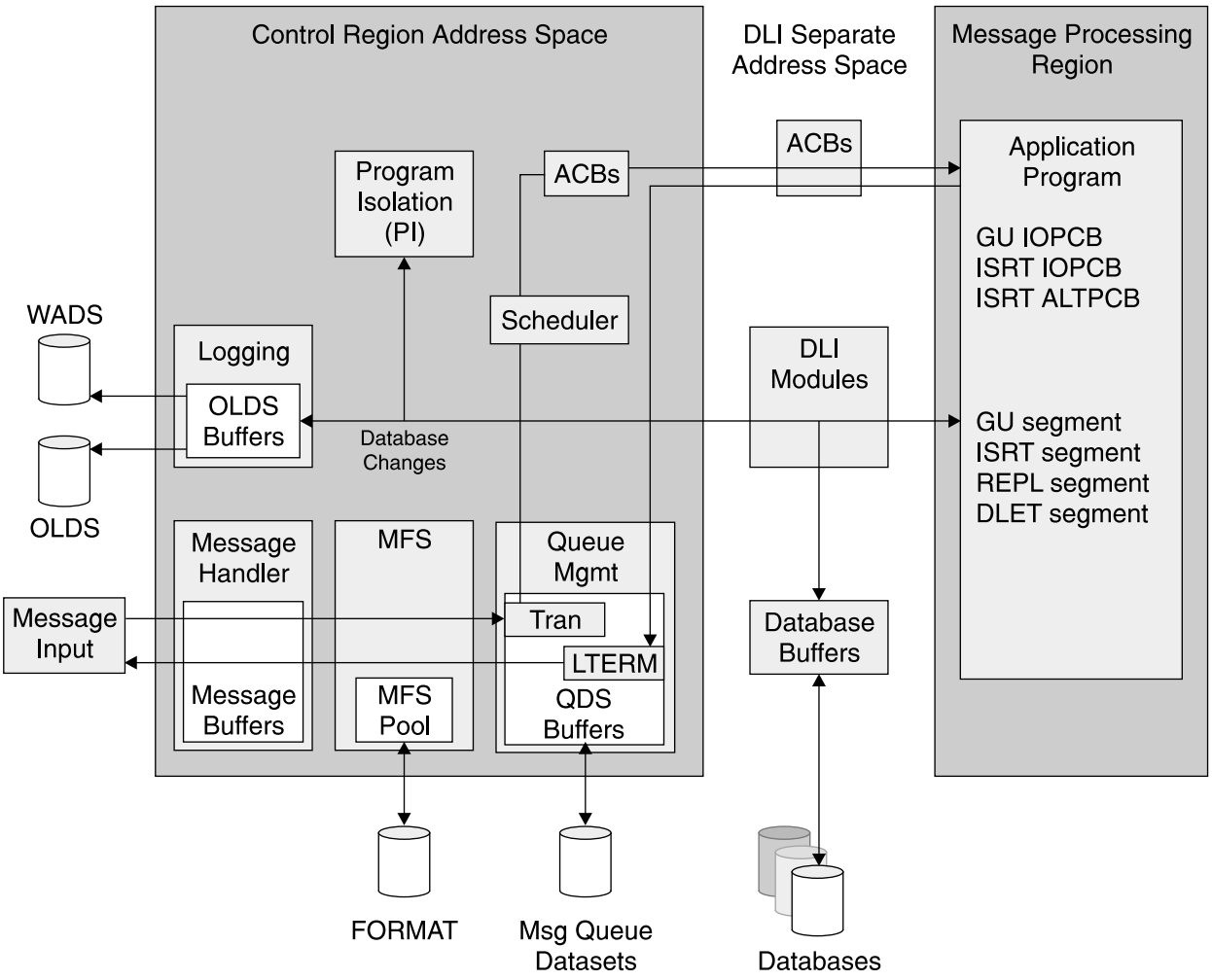

Figure 37. The IMS Control Region, Its Control, and Data (Message) Flow

A further description of Figure 37 follows:

1. The input data from the terminal is read by the data communication modules. After editing by message format service (MFS), and verifying that the user is allowed to execute this transaction, this input data is put in the IMS Message Queues. These are sequenced by destination, which could be either transaction code (TRAN) or logical terminal (LTERM). In the case of input messages, this would be the TRAN.
2. The scheduling modules will determine which MPP is available to process this transaction, based on a number of system and user specified considerations, and will then retrieve the message from the IMS message queues, and start the processing of a transaction in the MPP.
3. Upon request from an MPP or BMP, the DL/I modules pass a message or database segment to or from the MPP/BMP.

   Note: In z/OS, the DL/I modules, control blocks, and pools reside in the common storage area (CSA or ECSA) and the control region is not needed for most DB processing (the exception being Fast Path).
4. Once the MPP has finished processing, the message output from the MPP is also put into the IMS Message Queues, in this case, queued against the logical terminal (LTERM).

5. The communication modules retrieve the message from the message queues, and send it to the output terminal. MFS is used to edit the screen and printer output.
6. All changes to the message queues and the databases are recorded on the logs. In addition, checkpoints for system (emergency) restart and statistical information are logged.

   Notes:

   a. The physical logging modules run as a separate task and use z/OS ESTAE for maximum integrity.
   b. The checkpoint identification and log information are recorded in the restart and RECON data sets.
7. Program Isolation locking assures database integrity when two or more MPPs or BMPs update the same database. It also backs out database changes made by failing programs. This is done by maintaining a short-term, dynamic log of the old database element images. IRLM is an optional replacement for PI locking. IRLM is required, however, if IMS is participating in data sharing.

Chapter 13. How IMS TM Processes Input

IMS can accept input messages from a variety of sources. Originally, all input was from 3270 type terminals.

The following sections are covered in this chapter:

See Figure 38 while reading the sections listed above.

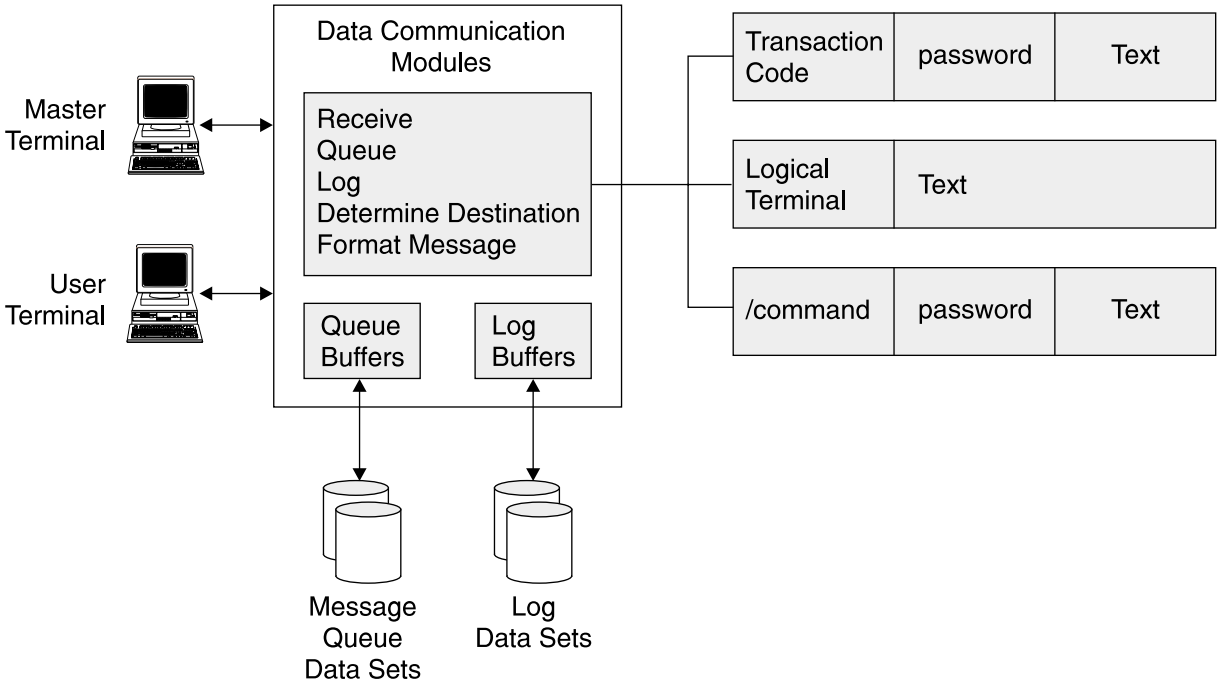

Figure 38. Input Message Processing

Input Message Types

When IMS reads data from a terminal that has come from the telecommunication access method, IMS first checks the type of input data.

Input from terminals can consist of three types of messages:

An input transaction message
This message is routed to an application program for processing with the first 1-to-8 bytes of the message identifying the transaction code.

A message switch
This message is routed to another terminal, with the first 1-to-8 bytes used

as the name of the destination logical terminal (LTERM). The LTERM can be a USERID if the Extended Terminal Option (ETO) is used.

A command
A command is processed by IMS itself.

Terminal Types

There are two basic types of terminals that can connect to IMS. They are:

Static The terminal is specifically defined in the IMS system definition, and this determines what physical terminal name (NODE NAME), and logical terminal name (LTERM) is available for use.

Dynamic
The terminal is not statically defined in the IMS system definition. IMS can create a dynamic terminal definition. This requires either the IMS Extended Terminal Option (ETO), a separately ordered feature of IMS or other third-party vendor products. Dynamic terminals have not been previously defined to IMS — their definitions are generated by ETO when the user logs on/ signs on.

If a terminal user attempts to connect to IMS using a terminal that is defined to IMS as static, then the user will use the defined NODE NAME / LTERM name combination.

If a terminal user attempts to connect to IMS using a terminal that is not defined to IMS as static, and dynamic terminal support is available, then the dynamic terminal product (such as ETO) will be used to determine what the LTERM name is; and whether it is based on the user's USERID, the NODE NAME, or some other value.

If a terminal user attempts to connect to IMS using a terminal that is not defined to IMS as static, and dynamic terminal support is not enabled, then the user will be unable to logon to IMS.

Input Message Origin

IMS maintains the name of the terminal or user from which an input message is received. When a message is passed to an application program, this is also made available to that program, via its program communication block (PCB).

This origin is the logical terminal name (LTERM). The LTERM name may be specific to the user, or may be specific to the physical location, depending on how the IMS system is defined. See "Terminal Types."

Terminal Input Destination

The destination of the terminal input is dependent upon the type of input.

An input command goes directly to the IMS command processor modules, while a message switch or a transaction are stored on the message queue. When a 3270-based message is received by IMS, the message input is first processed by message format service (MFS), except when input is from previously cleared or unformatted screen. MFS provides an extensive format service for both input and output messages. It is discussed in detail in Chapter 20, "The IMS Message Format Service," on page 207.

When the input message is enqueued to its destination in the message queue, the input processing is completed. If more that one LTERM is defined or assigned to a physical terminal, they are maintained in a historical chain: the oldest defined or assigned first. Any input from the physical terminal is considered to have originated at the first logical terminal of the chain. If, for some reason (such as security or a stopped LTERM), the first logical terminal is not allowed to enter the message, all logical terminals on the input chain are interrogated in a chain sequence for their ability to enter the message. The first appropriate LTERM found is used as message origin. If no LTERM can be used, the message is rejected with an error message.

Message Queueing

All full-function input and output messages in IMS are queued in message queues. See Figure 39 on page 126. For Fast Path transactions, see “Fast Path Transactions and Message Queues” on page 128.

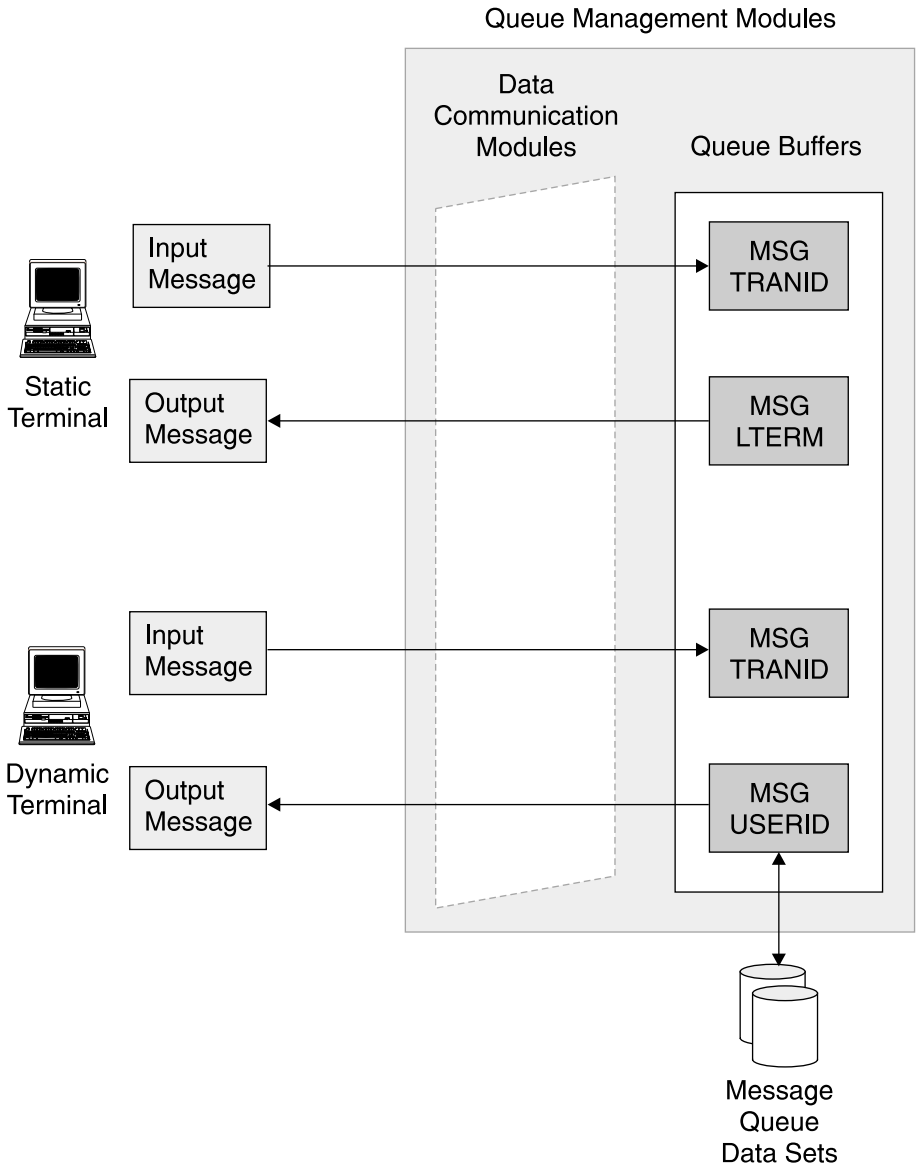

Figure 39. Overview of the Message Queuing Process

The use of message queues allows input processing, output processing, command processing, and application program processing to be performed asychronously, to a large extent. This means, for example, that the input processing of message A can be done in parallel with the database processing for message B and the output processing for message C. Messages A, B, and C can be different occurrences of the same or different message types and/or transaction codes.

Messages in the IMS message queues are stored by destination, priority, and the time of arrival in IMS. A destination can be:

- A message processing program (MPP), which is for transaction input. Ordering is by transaction code.
- A logical terminal (LTERM), which is for a message switch, command responses, and output generated by application programs.

The message queue buffers are maintained in main storage (defined by the MSQUEUE macro) unless shared queues are used. If the memory-based message

queue buffers become full, messages are then stored on the message queue data sets on DASD. The queue blocks in main storage and on direct access storage are reusable. As far as possible messages are stored in the message queue buffers, to minimize the number of I/O operations required during processing.

Queue Size and Performance Considerations

Messages in the IMS message queue are primarily held in buffers in main storage. However, when messages are added to the queues faster than IMS can process these messages, the message queue buffers can fill. In this situation, any new messages are written to the message queue data sets. The performance of these data sets then becomes very important. The data sets should be on a DASD volume with fast response times, and the data sets should be appropriately sized to ensure that there is always space available.

Multiple Message Queues

The IMS Queue Manager supports concurrent I/O operations to its message queue data sets, allowing the IMS message queue to be distributed across multiple physical queue data sets. This enhancement supports the long and short message queue data sets.

This function is activated when more than one DD statement per message queue data set is provided. You can supply up to ten DD statements for each queue data set. These DD statements can be allocated on different device types, but LRECL and BLKSIZE must be the same for all the data sets of a single queue.

IBM strongly recommends that multiple queue data sets be used, so that in an emergency situation, the IMS systems performance will not degrade while trying to handle a large volume of messages going to and from the message queue data sets.

Related Reading: See the *IMS Version 9: Installation Volume 1: Installation Verification* and *IMS Version 9: Installation Volume 2: System Definition and Tailoring* for further detailed guidelines for selecting message queue parameters such as block sizes, QPOOL size, queue data set allocation and so forth.

Shared Queues

IMS provides the facility for multiple IMS systems in a sysplex to share a single set of message queues. This function is known as IMS shared queues and the messages are held in structures in a coupling facility. All the IMS subsystems in the sysplex share a common set of queues for all non-command messages (that is, input, output, message switch, and Fast Path messages). A message that is placed on a shared queue can be processed by any of several IMS subsystems in the share queues group as long as the IMS has the resources to process the message.

The shared-queues function is optional and you can continue to process with the non-sysplex message queue buffers and message queue data sets.

The benefits in using shared queues enables automatic workload balancing across all IMS subsystems in a Sysplex. New IMS subsystems can be dynamically added to the Sysplex, and share the queues as workload increases, allowing in incremental growth in capacity The use of shared queues can also provide increased reliability and failure isolation: if one IMS subsystem in the Sysplex fails, any of the remaining IMS subsystems can process the work that is waiting in the shared queues.

Related Reading: For more information about IMS and shared queues in a sysplex environment, see Chapter 30, “Introduction to Parallel Sysplex,” on page 315.

Fast Path Transactions and Message Queues

Fast Path transactions do not use the standard IMS message queues. Fast Path transactions are scheduled by a separate function within the IMS transaction manager, called the Expedited Message Handler (EMH). For further scheduling information, see Chapter 14, “Fast Path Transactions,” on page 135.

APPC Driven Transactions and Message Queues

There are two types of APPC transactions, implicit and explicit. With implicit APPC transactions, IMS receives a transaction request via APPC. This transaction is placed onto the IMS message queues in the same way as a 3270-generated transaction. The message is passed to an MPP for processing, and the response is routed back to the originating APPC partner. The MPP program uses the DL/I interface to receive the message from the message queue, and put the response back onto the message queue.

With explicit APPC transactions, IMS schedules a program into an MPR (message processing region). This program uses APPC verbs to communicate with the APPC partner program to process the transaction. The standard IMS messages queues are not used for explicit APPC transactions.

OTMA Driven Transactions and Message Queues

OTMA allows IMS to receive a message through a different communications protocol (for example, TCP/IP sockets, MQ, remote procedure calls, IMS Connect, and so forth). The message is received by IMS, and it placed into the IMS message queue for processing in the usual manner. The response message is passed back to the originator through OTMA.

Message Scheduling

Scheduling is the loading of the appropriate program into a message processing region. IMS can then pass messages stored on the IMS message queue to this program when it issues the Get Unique (GU) IOPCB call. For more information about application calls, see Chapter 17, “Application Programming Overview,” on page 149.

Once an input message is available in the message queue, it is eligible for scheduling. Scheduling is the routing of a message in the input queue to its corresponding application program in the message processing region. See Figure 40 on page 129.

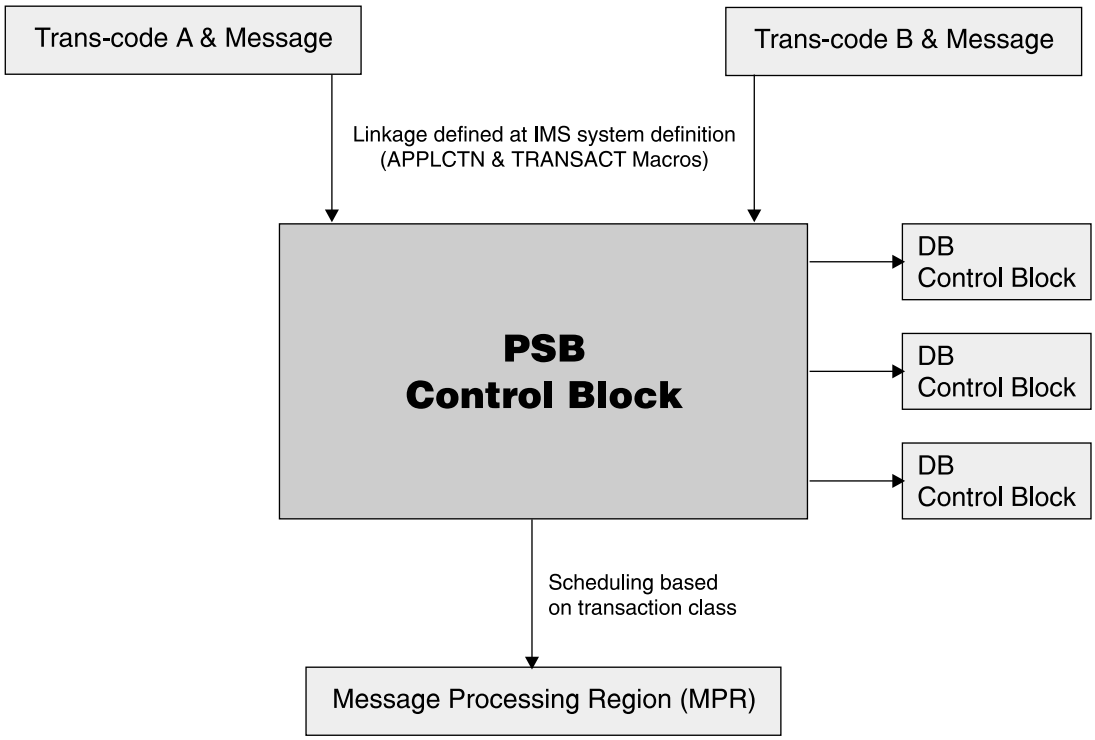

Figure 40. Message Scheduling

The linkage between an input message (defined by its transaction code) and an application program (defined by its name) is established at system definition time. Multiple transaction codes can be linked to a single application program, but only one application program can be linked to a given transaction code.

The class in which a transaction code with run is defined in two ways:

- On the APPLCTN macro
- On the MSGTYPE parameter of the TRANSACT macro

If the class is specified on the APPLCTN macro, it need not be defined on the TRANSACT macro. If it is specified on both, then the class on the TRANSACT macro will override the APPLCTN macro specification. Figure 41 illustrates the definition of a transaction.

```
APPLCTN PSB=DFSIVP1,PGMTYPE=TP
    TRANSACT CODE=IVTNO,MODE=SNGL,                          X
        MSGTYPE=(SNGLSEG,NONRESPONSE,1)
APPLCTN PSB=DFSIVP2,PGMTYPE=(TP,1)
    TRANSACT CODE=IVTNO2,MODE=SNGL,                         X
        MSGTYPE=(SNGLSEG,NONRESPONSE)
```

Figure 41. Sample APPLCTN Macro Transaction Definition in IMS Stage 1 Input

Notice the following about these transaction definitions:

- Transaction DFSIVP1 has the class defined as the third parameter on the MSGTYPE parameter on the TRANSACT macro.
- Transaction DFSIVP2 has the class defined on the APPLCTN macro.
- Both transactions are assigned to class 1.

Transaction Scheduling

The transaction scheduling algorithm can be a very sophisticated algorithm, as it needs to make use of all the IMS and system resources in the most efficient manner possible. However, most users do not need to use the power of the scheduling algorithms, as the resources available in IMS today (such as the number of message processing regions) are much greater than when these algorithms were designed several decades ago.

There are a few parameters on the transaction definition which will affect the scheduling options. These are:

PROCLIM

PARMLIM

MAXRGN

PRTY

Scheduling Conditions

The following conditions must be met for a successful scheduling:

- An MPR region must be available. Actually, the termination of a prior transaction running in an MPR region triggers the scheduling process.
- There must be a transaction input message in the queue.
- The transaction and its program are not in a stopped state.
- Enough buffer pool storage is available to load the program specification block (PSB) and the referenced database control blocks if not already in main storage.
- The database processing intent does not conflict with an already active application program (a BMP for instance). Processing intent is discussed in more detail in “Database Processing Intent” on page 133.

If the first transaction code with a ready input message does not meet all the above conditions, the next available input transaction is interrogated, and so forth. If no message can be scheduled, the scheduling process is stopped until another input message is enqueued. If the scheduling is successful, the IMS routines in the dependent region load the corresponding MPP and pass control to it.

Scheduling in a Dependent Region

The IMS scheduler will assign the application transaction processing to a dependent MPR. The number of MPRs available to an IMS system is 999 dependent regions.

The transactions are assigned to classes. The maximum number of transactions classes is set at system generation time by the MAXCLAS parameter of the IMSCTRL macro.

Class Processing

Each dependent MPR can run up to four transaction classes. The order in which they are specified is a priority sequence. That means that the transaction class named first is the highest and the one named last is the lowest. Each MPR can have a different sequence of the same or different transaction combinations. The classes are named on the PROC statement of the JCL running the MPR. Figure 42 on page 131 shows an example of the MPR JCL. The MPR can be run as a JOB or a started task.

```
//IVP6TM11 EXEC PROC=DFSMPR,TIME=(1440),
//         AGN=BMP01,              AGN NAME
//         NBA=6,
//         OBA=5,
//         SOUT='*',               SYSOUT CLASS
//         CL1=001,                TRANSACTION CLASS 1
//         CL2=006,                TRANSACTION CLASS 2
//         CL3=013,                TRANSACTION CLASS 3
//         CL4=000,                TRANSACTION CLASS 4
//         TLIM=10,                MPR TERMINATION LIMIT
//         SOD=,                   SPIN-OFF DUMP CLASS
//         IMSID=IMSY,       IMSID OF IMS CONTROL REGION
//         PREINIT=DC,             PROCLIB DFSINTXX MEMBER
//         PWFI=N                  PSEUDO=WFI
//*
```

Figure 42. Example of MPR PROC Statement

The classes which the MPR runs can be changed while the MPR is running. This is done through and /ASSIGN command. When the /ASSIGN command is executed, only those classes specified on the command will be available to that MPR. The changes will be maintained until the MPR is restarted, at which time the values on the PROC statement will be used again. Figure 43 illustrates an example of an /ASSIGN command. Again the order of classes on the command is the priority sequence of those classes.

```
/ASSIGN CLASS 1 4 6 9 TO REGION 1
```

Figure 43. Example of /ASSIGN CLASS Command

To list the classes assigned to an MPR the /DISPLAY ALL command can be used. Figure 44 shows the /DISPLAY ACTIVE command and the output.

```
/DIS ACTIVE
     REGID JOBNAME   TYPE TRAN/STEP PROGRAM STATUS          CLASS          IMSY
         1 SJIMSYM1  TP                     WAITING        1, 4, 6, 9    IMSY
         2 SJIMSYM2  TP                     WAITING        2, 3, 5, 1    IMSY
           BATCHREG  BMP   NONE   IMSY
           FPRGN     FP    NONE   IMSY
           DBTRGN    DBT   NONE   IMSY
           SJIMSYDB  DBRC                     IMSY
           SJIMSYDL  DLS     IMSY
     VTAM ACB OPEN          -LOGONS DISABLED   IMSY
     IMSLU=N/A.N/A            APPC STATUS=DISABLED    IMSY
     OTMA GROUP=IMSCGRP   STATUS=ACTIVE       IMSY
     APPLID=SCSIM6YA  GRSNAME=         STATUS=DISABLED   IMSY
     LINE ACTIVE-IN -    1 ACTIV-OUT -    0   IMSY
     NODE ACTIVE-IN -    0 ACTIV-OUT -    0   IMSY
     *99298/155826*     IMSY
```

Figure 44. Example of /DISPLAY ACTIVE Command

Note the following from the information from Figure 44:

- There are two MPRs.
- The MPR named SJIMSYM1 run classes 1, 4, 6, and 9.
- The MPR named SJIMSYM2 runs classes 2, 3, 5, 1.
- Class 1 has the highest priority in MPR SJIMSYM1 and the lowest in MPR SJIMSYM2.

When an MPR is looking to find the a transaction to schedule, it will use the following criteria:

1. The highest priority transaction ready in the highest priority class
2. Any other transaction in the highest priority class
3. The highest priority transaction ready in the second highest priority class
4. Any other transaction in the second priority class

This sequence of priorities will be used for all the available classes for this MPR.

Note: If a transaction has a class for which there are no MPRs currently allowed to run that class, the transaction will not be scheduled and will remain on the input queue.

PROCLIM Processing

IMS also tries to increase throughput of the MPR by processing more than one message for the same transaction. This is to make use of the fact that the program has already been loaded into the MPR's storage, and the PSB and DBD control blocks also have been loaded. This will increase the throughput of the number of messages processed by this MPR, as it will avoid some of the overhead with reloading the program and control blocks.

At the completion of the transaction, IMS with check the PROCLIM value on the TRANSACT macro for this transaction. The MPR will process the number of messages allowed in the first value of this keyword before looking to see what other transactions are available to be scheduled. This means the MPR can process more transactions without having to go through the scheduling logic for each transaction.

Parallel Scheduling

A transaction will only process in one MPR at a time unless parallel processing is specified. To allow more than one MPR to schedule a transaction type at a time, code the SCHDTYP parameter on the APPLCTN macro. For example:

```
APPLCTN PSB=DFSIVP1,PGMTYPE=(TP,1),SCHDTYP=PARALLEL
```

Unless there are application restrictions on processing the message in strict first-in, first-out sequence, parallel scheduling should be applied to all transactions. This will allow IMS to make the best use of IMS resources while providing the best possible response time to individual transactions.

The PARMLIM parameter on the TRANSACT macro will determine when a transaction will be scheduled in another region. When the number of messages on the queue for this transaction exceeds the value on the PARLIM, another region will be used.

The MAXRGN parameter is used to restrict the number of MPRs which can process a transaction at any one time. This is done to avoid the situation of all the MPRs being tied up by a single transaction type.

Priority

The PRTY parameter on the TRANSACT macro sets the priority of a transaction. This is how to differentiate one transaction from another if they run in the same transaction class. A transaction of a higher priority will be scheduled before a lower priority one. However an MPR will process a transaction in a higher class (for this MPR, see "Scheduling in a Dependent Region" on page 130 for more details) before a transaction in a lower class regardless of the priority. A transaction priority

will increase once the number of transactions on the message queue exceed the value set on the third value of the PRTY keyword. It will increase to the value set on the second parameter of the PRTY keyword. This has the effect of trying to avoid a long queue on any single transaction code by giving it a higher priority.

Another factor in transaction scheduling is the PROCLIM value. This is discussed in “PROCLIM Processing” on page 132.

Database Processing Intent

A factor that significantly influences the scheduling process is the intent of an application program toward the databases it uses. Intent is determined by examining the intent last associated with the PSB to be scheduled. At initial selection, this process involves bringing the intent list into the control region. The location of the intent list is maintained in the PSB directory. If the analysis of the intent list indicates a conflict in database usage with a currently active program in MPP or BMP region, the scheduling process will select another transaction and try again.

The database intent of a program as scheduling time is determined via the PROCOPT= parameters in the PCB.

An conflicting situation during scheduling will only occur if a segment type is declared exclusive use (PROCOPT=E) by the program being scheduled and a already active program references the segment in its PSB (any PROCOPT), or vice versa.

Scheduling a BMP

A BMP is initiated in a standard z/OS address space via any regular job submission facility. This could be from either:

- TSO and SUBMITing the job
- Some job scheduling system

However, during its initialization the IMS scheduler in the control region is invoked to assure the availability of the database resources for the BMP.

Shared Queues

Scheduling of transactions in a shared-queues environment is similar to those in a non-shared queues environment. All the checks, however, are across all the IMS systems in the shared-queues environment, and obviously, there are extra considerations as well.

Related Reading: For further information on scheduling shared queues, see:

- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*

Chapter 14. Fast Path Transactions

Apart from standard IMS transactions, there are two types of Fast Path online transactions. They are:

- “Fast Path Exclusive Transactions”
- “Fast Path Potential Transactions”

Fast Path Exclusive Transactions

Fast Path schedules input messages by associating them with a load balancing group. A load balancing group (BALG) is a group of Fast Path input messages that are ready for balanced processing by one or more copies of a Fast Path program. One LBG exists for each unique Fast Path message-driven application program.

The messages for each LBG are processed by the same Fast Path program. The EMH controls Fast Path messages by:

- Managing the complete execution of a message on a first-in-first-out basis.
- Retaining the messages that are received in the control program’s storage without using auxiliary storage or I/O operations.
- Supporting multiple copies of programs for parallel scheduling.
- Requiring that programs operate in a wait-for-input mode.

Fast Path Potential Transactions

Fast Path potential transactions are a mixture of standard IMS full-function and Fast Path exclusive transactions.

The same transaction code can be used to trigger either a full-function, or a Fast Path transaction, with an exit used to determine whether this instance of the transaction will be full-function, or Fast Path.

Chapter 15. The Master Terminal

The mission of the Master Terminal Operator (MTO) is to monitor and manage an individual IMS. As IMSs are joined together into sharing groups (sharing databases, resources, or message queues), system management becomes more complex. Prior to IMS Version 8, the IMS systems in sharing groups had to be managed individually.

IMS Version 8 introduced system management enhancements so that a single IMS or multiple IMS systems could be monitored and managed from a single point of control. You can issue commands and receive responses from one, many, or all of the IMSs in the group from this single point of control. For more information about these enhancements, see Chapter 31, “IMSplexes,” on page 337.

The master terminal operator (MTO) has the following responsibilities:

- Responsibility for running IMS

  The MTO starts and shuts down dependent regions and manages the system log.
- Knowledge of the ongoing status of the IMS subsystem

  The MTO continuously monitors processing and detects any error situations.
- Control over contents of the system and network

  The MTO can control the network, connect other IMS systems, and perform other prearranged tasks.
- Privileged commands

  In addition to routine work, the MTO responds to error conditions, changes the scheduling algorithm, alters passwords, and reconfigures the system as necessary.

Table 5 shows the actions usually performed by the MTO and the commands usually reserved for the MTO’s use.

Table 5. Master Terminal Operator Actions and Associated Commands

Activity	**IMS Command**
Activate IMS (cold start)	`/ERESTART COLDSYS`
Start a message region	`/START REGION IMSMSG1`
Start communications lines	`/START LINE ALL`
Display message queues	`/DISPLAY`
Start another message region	`/START REGION IMSMSG3`
Prepare for VTAM communication	`/START DC`
Initiate static VTAM sessions	`/OPNDST NODE ALL`
Initiate dynamic VTAM sessions	`/OPNDST NODE` *`nodename`*
Send a message to terminals	`/BROADCAST`
Shut down VTAM terminals and IMS	`/CHECKPOINT FREEZE QUIESCE`
Restart IMS (warm start)	`/NRESTART`

When the IMS system is generated, the IMS master terminal MUST be included, and consists of two components:

- Primary master

- Secondary master

All messages are routed to both the primary and secondary master terminals. Special MFS support is used for the master terminal.

The following sections of this chapter discuss the tasks of monitoring and managing an individual IMS using the MTO. The sections are:

The Primary Master

Traditionally, the primary master was a 3270 display terminal of 1920 characters (24 lines by 80 columns). A sample traditional IMS master terminal is shown in Figure 45.

```
   03/04/01  14:49:48                                          IMSC
  DFS249 14:43:46 NO INPUT MESSAGE CREATED
 DFS994I COLD START COMPLETED
 DFS0653I PROCECTED CONVERSATION PROCESSING WITH RRS/MVS ENABLED
 DFS2360I 14:29:28 XCF GROUP JOINED SUCCESSFULLY.



-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -




                                                          PASSWORD:
  -
```

Figure 45. Master Terminal Screen

The display screen of the master terminal is divided into four areas. They are the:

Message area
The message area is for IMS command output (except /DISPLAY and /RDISPLAY), message switch output that uses a message output descriptor name beginning with DFSMO (see MFS), and IMS system messages.

Display area
The display area is for /DISPLAY and /RDISPLAY command output.

Warning message area
The warning message area is for the following warning messages:

- MASTER LINES WAITING
- MASTER WAITING
- DISPLAY LINES WAITING
- USER MESSAGE WAITING

To display these messages or lines, press PA1. An IMS password can also be entered in this area after the "PASSWORD" literal.

User input area
The user input area is for operator input.

Program function key 11 or PA2 requests the next output message and program function key 12 requests the Copy function if it is a remote terminal.

The Secondary Master

Traditionally, the secondary master was a 3270 printer terminal.

This usage has also been phased out in many sites, who now have the secondary master defined as spooled devices to IMS, in effect writing the messages to physical data sets.

In this way, the secondary master can be used as an online log of events within IMS. To accomplish this, the definitions in Figure 46 needs to be put into the IMS Stage 1 system definition. These definitions need to follow the COMM macro and before any VTAM terminal definitions.

```
*
        LINEGRP DDNAME=(SPL1,SPL2),UNITYPE=SPOOL
             LINE BUFSIZE=1420
             TERMINAL FEAT=AUTOSCH
          NAME   (SEC,SECONDARY)
```

Figure 46. Sample JCL for the Secondary Master Spool

To complete the definitions, code SPL1 and SPL2 DD statements in the IMS control region JCL. The data sets should be allocated with the following DCB information:

```
DCB=(RECFM=VB,LRECL=1404,BLKSIZE=1414)
```

Using the z/OS Console as the Master Terminal

IMS always has a communications path with the z/OS system console. The write-to-operator (WTO) and write-to-operator-with-reply (WTOR) facilities are used for this. Whenever the IMS control region is active, there is an outstanding message requesting reply on the z/OS system console. This can be used to enter commands for the control region. All functions available to the IMS master terminal are available to the system console. The system console and master terminal can be used concurrently, to control the system. Usually, however, the system console's primary purpose is as a backup to the master terminal. The system console is defined as IMS line number one by default.

Extended MCS/EMCS Console Support

IMS can be also communicated with using the MCS/EMCS console support.

Any z/OS console can issue a command directly to IMS, using either a command recognition character (CRC) as defined at IMS startup, or using the 4-character IMS ID to be able to issue commands.

This interface has the option of using RACF or exit routines for command security. For further details, see Chapter 24, "IMS Security," on page 253.

Chapter 16. Application Program Processing for IMS TM

Once an application program is scheduled in a dependent region, it is loaded into that region by IMS.

The following sections are covered in this chapter:

Flow of Message Processing Programs (MPPs)

The scheduled program in the MPR is given control after it is loaded. The normal processing steps of an MPP are described in the list that follows Figure 47 on page 142.

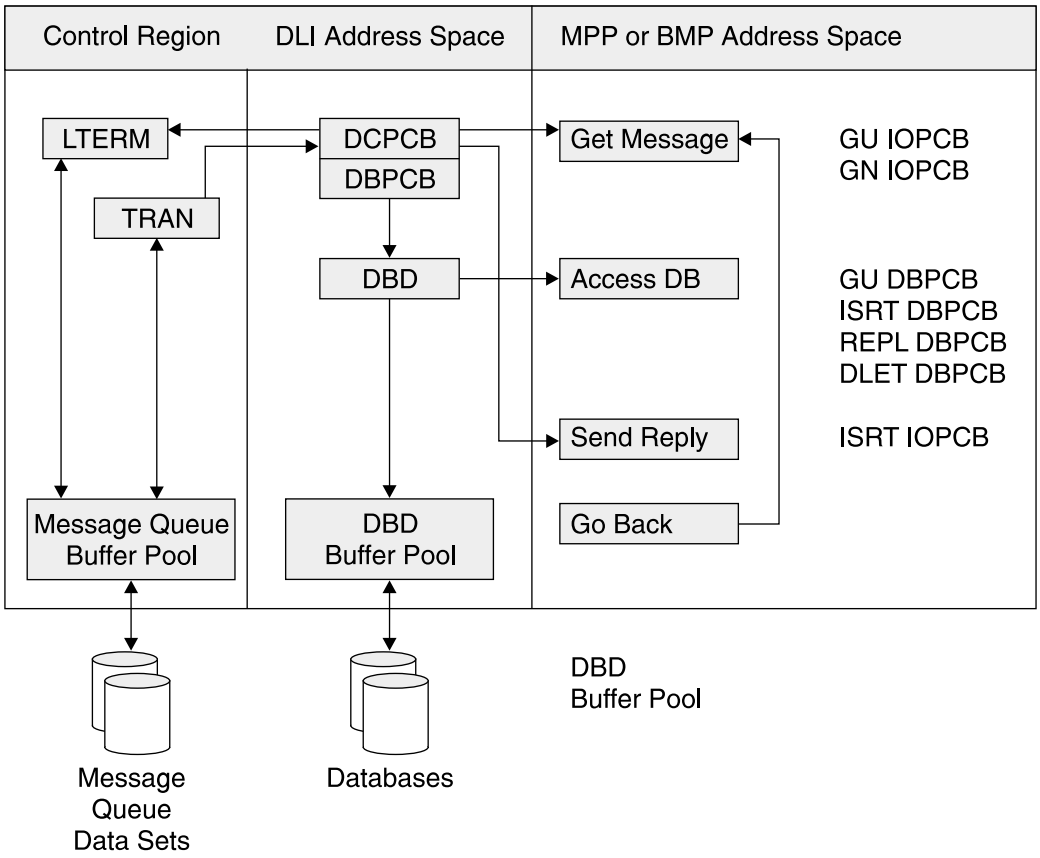

Figure 47. Overview of Basic Flow Through a MPP or BMP Address Space

1. Retrieve the input message by using a DL/I message call.
2. Check the input message for syntax errors.
3. Process the input message, requesting necessary IMS database accesses.
4. Send output to the originating and/or other (for example, printer) logical terminals by using DL/I message calls.
5. Retrieve the next input message or terminate.

Role of the PSB

The program specification block (PSB) for an MPP or a BMP contains, one or more PCBs for logical terminal linkage, in addition to database PCBs. The very first PCB always identifies the originating logical terminal (IOPCB). This PCB must be referenced in the get unique (GU) and get next (GN) message calls. It must also be used when inserting output messages to that LTERM. In addition, one or more alternate output PCBs can be defined. Their LTERM destinations can be defined in the PCBs or set dynamically with change destination calls.

DL/I Message Calls

The same DL/I language interface that is used for the access of databases is used to access the message queues.

The principal DL/I message call function codes are:

GU (get unique)
This call must be used to retrieve the first, or only, segment of the input message.

GN (get next)
This call must be used to retrieve second and subsequent message segments.

ISRT (insert)
This call must be used to insert an output message segment into the output message queue. Note: These output message(s) will not be sent until the MPP terminates or requests another input message by using a get unique call.

CHNG (change destination)
This call can be used to set the output destination for subsequent insert calls.

Program Isolation and Dynamic Logging

When processing DL/I database calls, the IMS program isolation function will ensure database integrity.

With program isolation, all activity (database modifications and message creation) of an application program is isolated from any other application programs running in the system until an application program commits, by reaching a synchronization point, the data it has modified or created. This ensures that only committed data can be used by concurrent application programs. A synchronization point is established with a get unique call for a new input message (single mode) and/or a checkpoint call (BMP only), or program normal termination (GOBACK or RETURN).

Program isolation allows two or more application programs to concurrently execute with common data segment types even when processing intent is segment update, add, or delete. This is done by a dynamic enqueue/dequeue routine which enqueues the affected database elements (segments, pointers, free space elements, etc.) between synchronization points.

At the same time, the dynamic log modules log the prior database record images between those synchronization points. This makes it possible to dynamically back out the effects of an application program that terminates abnormally, without affecting the integrity of the databases controlled by IMS. It does not affect the activity of other application program(s) running concurrently in the system.

With program isolation and dynamic backout, it is possible to provide database segment occurrence level control to application programs. A means is provided for resolving possible deadlock situations in a manner transparent to the application program.

One example of a deadlock occurs in the following sequence of events:

1. Program A updates database element X.
2. Program B updates database element Y.
3. Program A requests Y and must wait for the synchronization point of program B.
4. Program B in turn requests X and must wait for the synchronization point of program A.

A deadlock has now occurred: both programs are waiting for each other's synchronization point. The dynamic enqueue/dequeue routines of IMS intercept possible deadlocks during enqueue processing (in the above example, during enqueue processing of event 4).

When a deadlock situation is detected, IMS abnormally terminates (pseudo abends) one of the application programs involved in the deadlock. The activity of the terminated program is dynamically backed out to a previous synchronization point. Its held resources are freed. This allows the other program to process to completion. The transaction that was being processed by the abnormal terminated program is saved. The application program is an MPP, it is rescheduled. For a BMP region, the job must be restarted. This process is transparent to application programs and terminal operators.

There are two situations where the enqueue/dequeue routines of program isolation are not used in processing a database call:

- If PROCOPT=GO (read only) is specified for the referenced segment (s) of the call.
- If PROCOPT=E (exclusive) is specified for the referenced segment (s) in the call.

Notice that possible conflicts with exclusive extent are resolved during scheduling time and, as such, cannot occur at call time.

Notes:

1. With the GO option, a program can retrieve data which has been altered or modified by another program still active in another region, and database changes made by that program are subject to being backed out.
2. Exclusive intent may be required for long-running BMP programs that do not issue checkpoint calls. Otherwise, an excessively large enqueue/dequeue table in main storage may result.
3. Even when PROCOPT=E is specified, dynamic logging will be done for database changes. The ultimate way to limit the length of the dynamic log chain in a BMP is by using the XRST/CHKP calls. The chain is deleted at each CHKP call because it constitutes a synchronization point.
4. If one MPP and one BMP are involved in a deadlock situation, the MPP will be subject to the abnormal termination, backout, and reschedule process.

Internal Resource Lock Manager (IRLM)

When IMS is involved in a data-sharing environment with other IMS systems, IRLM is used instead of program isolation for lock management. See "Internal Resource Lock Manager (IRLM)" on page 21 for further details.

Abnormal Application Program Termination

When a message or batch-message processing application program is abnormally terminated for other reasons than deadlock resolution, internal commands are issued to prevent rescheduling. These commands are the equivalent of a /STOP command. They prevent continued use of the program and the transaction code in process at the time of abnormal termination. The master terminal operator can restart either or both stopped resources.

At the time abnormal termination occurs, a message is issued to the master terminal and to the input terminal that identifies the application program, transaction code, and input terminal. It also contains the system and user completion codes. In

addition, the first segment of the input transaction, in process by the application at abnormal termination, is displayed on the master terminal. The database changes of a failing program are dynamically backed-out. Also, any of its output messages that were inserted in the message queue since the last synchronization point are cancelled.

Conversational Processing

A transaction code can be defined as belonging to a conversational transaction during IMS system definition. If so, an application program that processes that transaction, can interrelate messages from a given terminal. The vehicle to accomplish this is the scratch pad area (SPA). A unique scratch pad area is created for each physical terminal which starts a conversational transaction. Each time an input message is entered from a physical terminal in conversational mode, its SPA is presented to the application program as the first message segment (the actual input being the second segment).

Before terminating or retrieving another message (from another terminal), the program must return the SPA to the control region with a message ISRT call. The first time a SPA is presented to the application program when a conversational transaction is started from a terminal, IMS will format the SPA with binary zero's (X'00'). If the program wants to terminate the conversation, it can indicate this by inserting a blank transaction code into the SPA.

Output Message Processing

As soon as an application reaches a synchronization point, its output messages in the message queue become eligible for output processing. A synchronization point is reached whenever the application program terminates or requests a new message/SPA from the input queue via a GU call.

In general, output messages are processed by the Message Format Service (MFS) before they are transmitted via the telecommunications access method.

Different output queues can exist for a given LTERM, depending on the message origin. They are, in transmission priority:

1. Response messages, that is, messages generated as a direct response (same PCB) to an input message from this terminal.
2. Command responses.
3. Alternate output messages, messages generated via an alternate PCB.

Logging, Checkpointing, and Restarting

To ensure the integrity of its databases and message processing, IMS uses logging and checkpoint/restart processing. In case of system failure, either software or hardware, IMS can be restarted. This restart includes the repositioning of users' terminals, transactions, and databases.

Related Reading: For further information on IMS logging facilities, see Chapter 25, "IMS Logging," on page 257.

At regular intervals during IMS execution, checkpoints are written to the logs. This limits the amount of reprocessing required in the case of an emergency restart. A

checkpoint is taken after a specified number of log records are written to the log tape after a checkpoint command. A special checkpoint command is available to stop IMS in an orderly manner.

A special disk restart data set is used to record the checkpoint identification and log tape volume serial numbers. This restart data set (IMS.RDS) is used during restart for the selection of the correct restart checkpoint and restart logs.

Message Switching

A message switch is when a user wishes to send a message to another user. The basic format of a message switch is the destination LTERM name followed by a blank and the message text.

A program-to-program switch or program-to-program message switch is a program that is already executing that requests a new transaction be put on the IMS message queues for standard scheduling and execution.

This second transaction can:

- Continue the processing of the first transaction (which, in this case, has probably terminated), and respond (if required) to the originating terminal, which is probably still waiting for a response.
- Be a second transaction, purely an offshoot from the first, without any relationship or communications with the originating terminal. In this case, the original transaction must respond to the terminal, if required.

Part 4. IMS Application Development

Chapter 17. Application Programming Overview

This chapter explains the basics for any programming running in an IMS environment.

IMS programs (online and batch) have a different structure than non-IMS programs (see “Program Structure”). An IMS program is always called as a subroutine of the IMS region controller. It also has to have a program specification block (PSB) associated with it. The PSB provides and interface from the program to IMS services which the program needs to make use of. These services can be:

- Sending or receiving messages from online user terminals
- Accessing database records
- Issuing IMS commands
- Issuing IMS service (checkpoint or sync) calls

The IMS services available to any program are determined by the IMS environment in which the application is running.

The following sections are covered in this chapter:

Java Programs

IMS Java application support (hereafter called IMS Java) allows you to write Java application programs that access IMS databases from IMS, IBM WebSphere Application Server for z/OS and OS/390, IBM CICS Transaction Server for z/OS, or IBM DB2 Universal Database™ for z/OS stored procedures.

Related Reading: For more information about IMS Java application programs, see Chapter 21, “Application Programming in IMS Java,” on page 223.

Program Structure

During initialization, both the application program and its associated PSB are loaded from their respective libraries by the IMS system. The IMS modules interpret and execute database CALL requests issued by the program. These modules may reside in the same or different z/OS address spaces depending on the environment in which the application program is executing.

Application programs executing in an online transaction environment are executed in a dependent region called the message processing region (MPR) or Fast Path region (IFP). The programs are often called message processing programs (MPP). The IMS modules that execute online services will run in the control region while the full-function database services will run in the DLI separate address space (DLISAS). The association of the application program and the PSB is defined at IMS system generation time via the APPLTN and TRANSACTION macros.

Batch application programs can execute in two different types of regions.

- Application programs which need to make use of message processing services or databases being used by online systems are executed in a batch message processing region (BMP).
- Application programs which can execute without messages services execute in a DLI batch region.

For both these types of batch application programs, the association of the application program to the PSB is done on the PARM keyword on the EXEC statement.

The application program interfaces with IMS by using the following program elements:

- An ENTRY statement specifying the PCBs utilized by the program (see “Entry to the Application Program” on page 151)
- A PCB-mask which corresponds to the information maintained in the pre-constructed PCB and which receives return information from IMS (see “PCB Mask” on page 151)
- An I/O area for passing data segments to and from the databases
- Calls to DL/I specifying processing functions (see “Calls to IMS” on page 155)
- Status code processing (see “Status Code Processing” on page 156)
- A termination statement (see “Termination of the Application” on page 156)

The PCB mask(s) and I/O areas are described in the program’s data declaration portion. Program entry, calls to IMS processing, and program termination are described in the program’s procedural portion. Calls to IMS, processing statements, and program termination can reference PCB mask(s) and/or I/O areas. In addition, IMS can reference these data areas. Figure 48 on page 151 illustrates how these elements are functionally structured in a program and how they relate to IMS.

The individual program elements mentioned in the previous list, are discussed in the sections that follow Figure 48 on page 151.

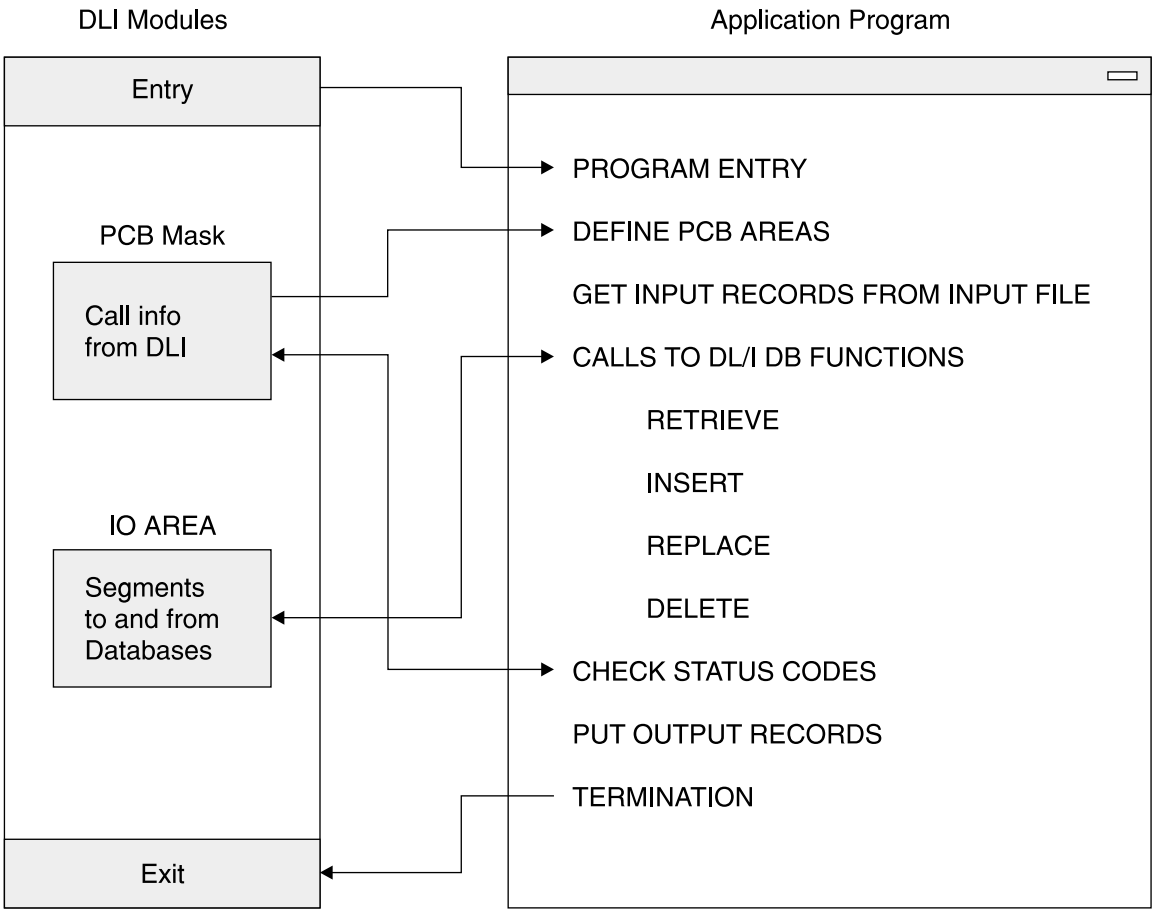

Figure 48. Structure of an IMS Application Program

Entry to the Application Program

Referring to Figure 48, when the operating system gives control to the IMS control facility, the IMS control program eventually passes control to the application program (through the entry point as defined below). At entry, all the PCB-names used by the application program are specified. The order of the PCB-names in the entry statement must be the same as in the PSB for this application program. The sequence of PCBs in the linkage section or declaration portion of the application program need not be the same as in the entry statement.

Notes:

1. Batch DL/I programs cannot be passed parameter information using the PARM field from the EXEC statement.
2. TP PCBs must proceed database PCBs in the PSB.

PCB Mask

A mask or skeleton database PCB structure is used by the application program to access data from a TP or database PCB. One PCB is required for each view of a database or online service. The program views a hierarchical data structure by using this mask.

One PCB is required for each data structure. An example of a database PCB mask is shown in Figure 50 on page 153 and explained in the text that follows the figure. An example of an TP PCB mask is shown in Figure 52 on page 155.

As the PCB does not actually reside in the application program, care must be taken to define the PCB mask as an assembler dsect, a COBOL linkage section entry, or a PL/I based variable.

The PCB provides specific areas used by IMS to inform the application program of the results of its calls. At execution time, all PCB entries are controlled by IMS. Access to the PCB entries by the application program is for read-only purposes. The PCB masks for an TP PCB and a database PCB are different. An example of both are shown in Figure 49.

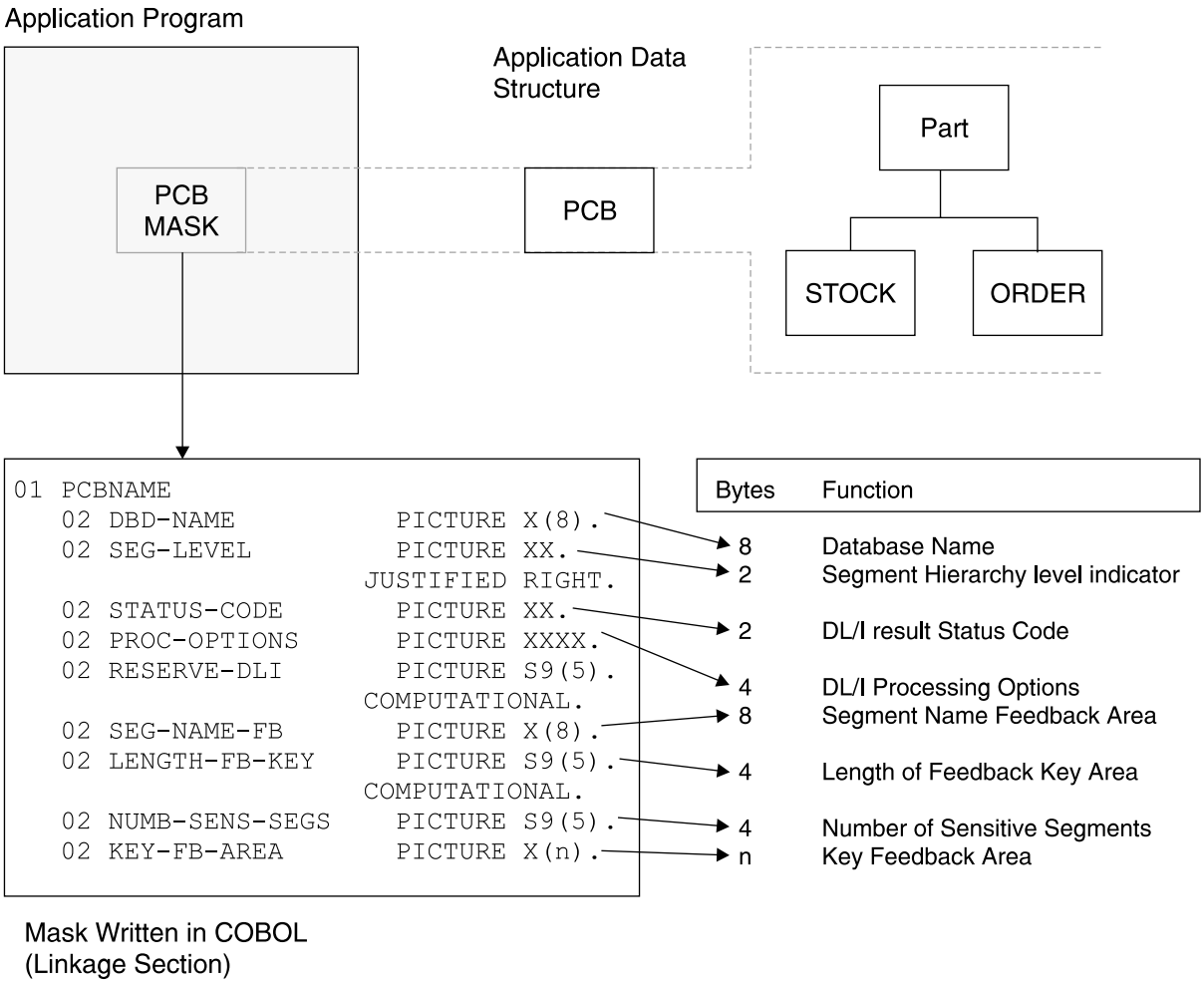

Figure 49. Application PCB Structure

Database PCB Mask

Figure 50 on page 153 shows an example of a DLI program's PCB mask, which defines the PCB area used by IMS to return the results of the call.

```
01 PCBNAME.
   02 DBD-NAME         PICTURE X(8).
   02 SEG-LEVEL        PICTURE XX.
   02 STATUS-CODE      PICTURE XX.
   02 PROC-OPTIONS     PICTURE XXXX.
   02 RESERVED-DLI     PICTURE S9(5).
   02 SEG-NAME         PICTURE X(8).
   02 LENGTH-FB-KEY    PICTURE S9(5).
   02 NUMB-SENS-SEGS   PICTURE S9(5).
   02 KEY-FB-AREA      PICTURE X(n).
```

Figure 50. Example of a Database Application PCB Mask

The following items comprise a PCB for a hierarchical data structure from a database:

Name of the PCB
This is the name of the area which refers to the entire structure of PCB fields. It is used in program statements. This name is not a field in the PCB. It is the 01 level name in the COBOL mask inFigure 50.

Name of the database
This is the first field in the PCB and provides the DBD name from the library of database descriptions associated with a particular database. It contains character data and is eight bytes long.

Segment hierarchy level indicator
IMS uses this area to identify the level number of the last segment encountered which satisfied a level of the call. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level number returned is that of the last segment that satisfied the search criteria along the path from the root (the root segment level being '01') to the desired segment. If the call is completely unsatisfied, the level returned is '00'. This field contains character data: it is two bytes long and is a right-justified numeric value.

DL/I status code
A status code indicating the results of the DL/I call is placed in this field and remains here until another DL/I call uses this PCB. This field contains two bytes of character data. When a successful call is executed, DL/I sets this field to blanks or to an informative status indication. A complete list of DL/I status codes can be found in the *IMS Version 9: Messages and Codes, Volume 1*.

DL/I processing options
This area contains a character code which tells DL/I the "processing intent" of the program against this database (that is, the kinds of calls that may be used by the program for processing data in this database). This field is four bytes long. It is left-justified. It does not change from call to call. It gives the default value coded in the PCB PROCOPT parameter, although this value may be different for each segment. DL/I will not allow the application to change this field, nor any other field in the PCB.

Reserved area for IMS
IMS uses this area for its own internal linkage related to an application program. This field is one fullword (4 bytes), binary.

Segment name feedback area
IMS fills this area with the name of the last segment encountered which satisfied a level of the call. When a retrieve call is successful, the name of the retrieved segment is placed here. If a retrieve is unsuccessful, the name

returned is that of the last segment, along the path to the desired segment, that satisfied the search criteria. This field contains eight bytes of character data. This field may be useful in GN calls. If the status code is 'AI' (data management open error), the DD name of the related data set is returned in this area.

Length of key feedback area
This entry specifies the current active length of the key feedback area described below. This field is one fullword (4 bytes), binary.

Number of sensitive segments
This entry specifies the number of segment types in the database to which the application program is sensitive. This would represent a count of the number of segments in the logical data structure viewed through this PCB. This field is one fullword (4 bytes), binary.

Key feedback area
IMS places in this area the concatenated key of the last segment encountered which satisfied a level of the call. When a retrieve is successful, the key of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the hierarchical path. When a retrieve is unsuccessful, the keys of all segments along the path to the requested segment, for which the search was successful, are placed in this area. Segments without sequence fields are not represented in this area.

Note: This area is never cleared, so it should not be used after a completely unsuccessful call. It will contain information from a previous call. See Figure 51 for an illustration of concatenated keys.

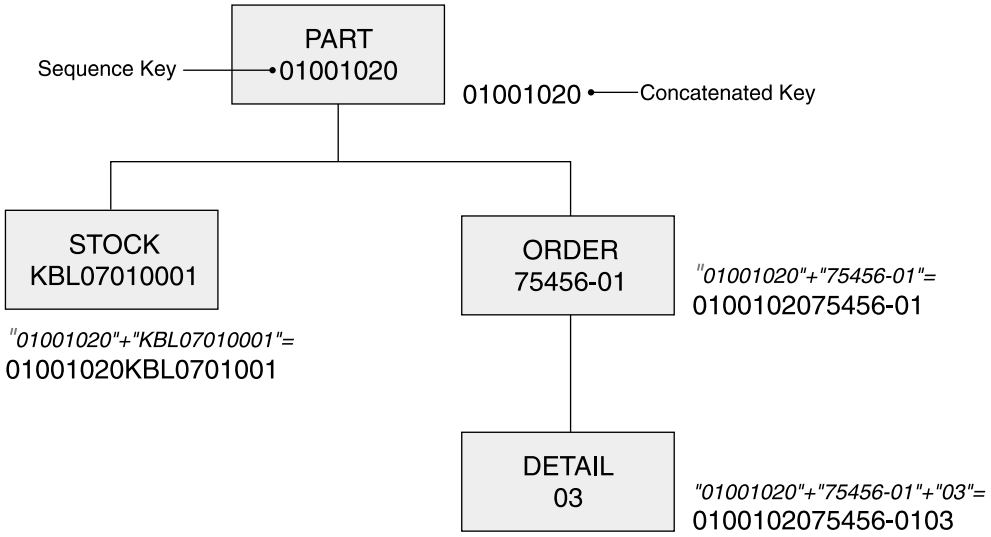

Figure 51. Examples of Concatenated Keys

TP PCB Mask

Figure 52 on page 155 shows an example of an online program's PCB mask, which defines the PCB area used by IMS to return the results of the call.

```
01 PCBNAME.
   02 DBD-NAME         PICTURE X(8).
   02 SEG-LEVEL        PICTURE XX.
   02 STATUS-CODE      PICTURE XX.
   02 PROC-OPTIONS     PICTURE XXXX.
   02 RESERVED-DLI     PICTURE S9(5).
   02 SEG-NAME         PICTURE X(8).
   02 LENGTH-FB-KEY    PICTURE S9(5).
   02 NUMB-SENS-SEGS   PICTURE S9(5).
   02 KEY-FB-AREA      PICTURE X(n).
```

Figure 52. Example of an Online Application PCB Mask

Calls to IMS

Actual processing of IMS messages, commands, databases and services are accomplished using a set of input/output functional call requests. A call request is composed of a CALL statement with an argument list. The argument list will vary depending on the type of call to be made.The argument list will consists of the following parameters:

- Function call
- PCB name
- I/O area
- Segment search argument (SAA) (database calls only)

Table 6 shows a brief explanation of the argument list items. The argument list items for database processing are discussed in more detail in Chapter 18, “Application Programming for the IMS Database Manager,” on page 165. The online services and commands argument list items are discussed in more detail in Chapter 19, “Application Programming for the IMS Transaction Manager,” on page 197.

Table 6. IMS Call Argument List

Application Component	**Description**
Function	Identifies the DL/I function to be performed. This argument is the name of the four character field which describes I/O operation. The DL/I functions are described in the individual chapters
PCB name	The name of the database program communication block (PCB). It is the name of the PCB within the PSB that identifies which specific data structure the application program wishes to process. The PCB is defined in more detail in “PCB Mask” on page 151
I/O area	The name of a I/O work area. This is an area of the application program into which DL/I puts a requested segment, or from which DL/I takes a designed segment. If this a common area is used to process multiple calls it must be long enough to hold the longest path of segments to be processed
SSA1...SSAn	The names of the Segment Search Arguments (SSAs). These are optional depending on the type of call issued. Used only used for database calls. The SSA provides information to define the segment to be retrieved or written.

Status Code Processing

After each IMS call, a two-byte status code is returned in the PCB which is used for that call. There are three categories of status codes:

- The blank status code, indicating a successful call
- Exceptional conditions and warning status codes from an application point of view
- Error status codes, specifying an error condition in the application program and/or IMS

The grouping of status codes in the above categories is somewhat installation dependent. This book, however, will give a basic recommendation after each specific call function discussion. It is also recommended that you use a standard procedure for status code checking and the handling of error status code. The first two categories should be handled by the application program after each single call. Figure 53 gives an example using COBOL.

```
CALL 'CBLTDLI' USING ....
IF PCB-STATUS EQ 'GE' PERFORM PRINT-NOT-FOUND.
IF PCB STATUS NE 'bb' PERFORM STATUS-ERROR.
everything okay, proceed...
```

Figure 53. Example of a COBOL Application Program Testing Status Codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving the real error situations to central status code error routine.

Termination of the Application

At the end of the processing of the application program, control must be returned to the IMS control program. The following list shows examples of the termination statements.

Language	Return Statement
Java	return;
COBOL	GOBACK.
PL/I	RETURN;
ASSEMBLER	RETURN(14,12),RC=0

Warning: Returning to IMS causes storage that was occupied by your program to be released because IMS links to your application program. Therefore you should close all non-DL/I data sets for COBOL and Assembler before return, to prevent abnormal termination during close processing by z/OS. PL/I automatically causes all files to be closed upon return.

IMS Setup for Applications

Before you can run an application program under IMS, control blocks must be defined and generated. The following sections cover this topic.

- “IMS Control Blocks” on page 157
- “Generating IMS Control Blocks” on page 158

IMS Control Blocks

A program specification block generation (PSBGEN) must be performed to create the program specification block (PSB) for the application program before the program can be run. The PSB contains one PCB for each DL/I database (logical or physical) the application program will access. The PCBs specify which segments the program will use and the kind of access (retrieve, update, insert, delete) the program is authorized to. The PSBs are maintained in one or more IMS system libraries called a PSBLIB library.

All IMS databases require a database descriptor block (DBD) created to have access to any IMS databases. The details of these control blocks are describe in “Generating IMS Control Blocks” on page 158. The database DBD is assembled into a system library called a DBDLIB.

The IMS system needs to combine and expand the PSB and DBD control blocks into an internal format called access control blocks (ACBs). The Application Control Blocks Maintenance Utility is used to create the ACBs.

In a batch DLI environment, the ACB blocks are either built dynamically at step initialization time (as specified in the DLIBATCH procedure) or the ACB blocks are built by running the ACB maintenance utility (as specified in the DBBBATCH procedure). In an online environment, the ACB blocks need to be created before an application can be scheduled and run. The ACB utility is run offline and the resulting control blocks are placed in an ACB library.

The IMS system needs to access these control blocks (DBDs and PSBs) in order to define the applications use of the varies IMS resources required. Depending on which environment the application program is executed in will determine how IMS accesses those control blocks. See Figure 54 on page 159 to see a overview of the processing.

The Transaction Processing (TP) PCB

Besides the default TP PCB, that does not require PCB statement, additional PCBs can be coded. These PCBs are used to insert output messages to:

- LTERMs other than the LTERM which originated the input message. A typical use of an alternate PCB is to send output to a 3270 printer terminal.
- A non-conversational transaction.
- Another USERID.

The destination of the output LTERM can be set in two ways:

- During PSBGEN by specifying the LTERM/TRANNAME in a alternate PCB.
- Dynamically by the MPP during execution, by using a change call against a modifiable alternate PCB.

The method used depends on the PCB statement.

The PCB Statement: This is the only statement required to generate an alternate PCB (multiple occurrences are allowed). Its format is:

```
PCB TYPE=TP,LTERM=name,MODIFY=YES
```

The following list describes the possible parameters.

Keyword	Description
TYPE=TP	Required for all alternate PCBs.

LTERM=*name*
Specifies this PCB is pointing at a known LTERM defined in the IMS system. The name is optional.

MODIFY=YES
If the modify is specified then the LTERM name may be changed by a CHANGE call within the application program.

Note: If MODIFY=YES is specified, the MPP must specify a valid alternate output LTERM with a change call before inserting any message via this PCB.

The Database PCB

The DB PCB for an MPP or BPP can be simple or complex. As compared to the TP PCB, two additional processing intent options can be specified with the PROCOPT= keyword of the PCB and/or SENSEG statement.

Here's an example of a simple database PCB:

```
PCB TYPE=DB,
DBDNAME=EXCEPTA,
PROCOPT=A,
KEYLEN=24
SENSEG NAME=QB01,
PARENT=0
```

In the previous example:

TYPE=DB
Required for all DB PCBs

DBDNAME=*name*
Specifies the database that this PCB is pointing to

PROCOPT=
Processing options

KEYLENGTH=
The length of the concatenated keys for this database

SENSEG
the SENSEG statement with the database PCB statement to define a hierarchically related set of data segments

Related Reading: For more information about generating these control blocks, see the *IMS Version 9: Utilities Reference: System*.

Generating IMS Control Blocks

In addition to database PCBs, a PSB for MPPs or BMPs contains one or more data communication PCBs.

The order of the PCBs in the PSB must be:

1. Data communication PCBs
2. Database PCBs
3. GSAM PCBs (not allowed for MPPs)

One data communication PCB is always automatically included by IMS at the beginning of each PSB of an MPP or BMP. This default data communication PSB is used to insert output messages back to the originating LTERM or USERID.

Note: One data communication PCB is always automatically included by IMS at the beginning of each PSB of an MPP or BMP. This default data communication PSB is used to insert output messages back to the originating LTERM or USERID.

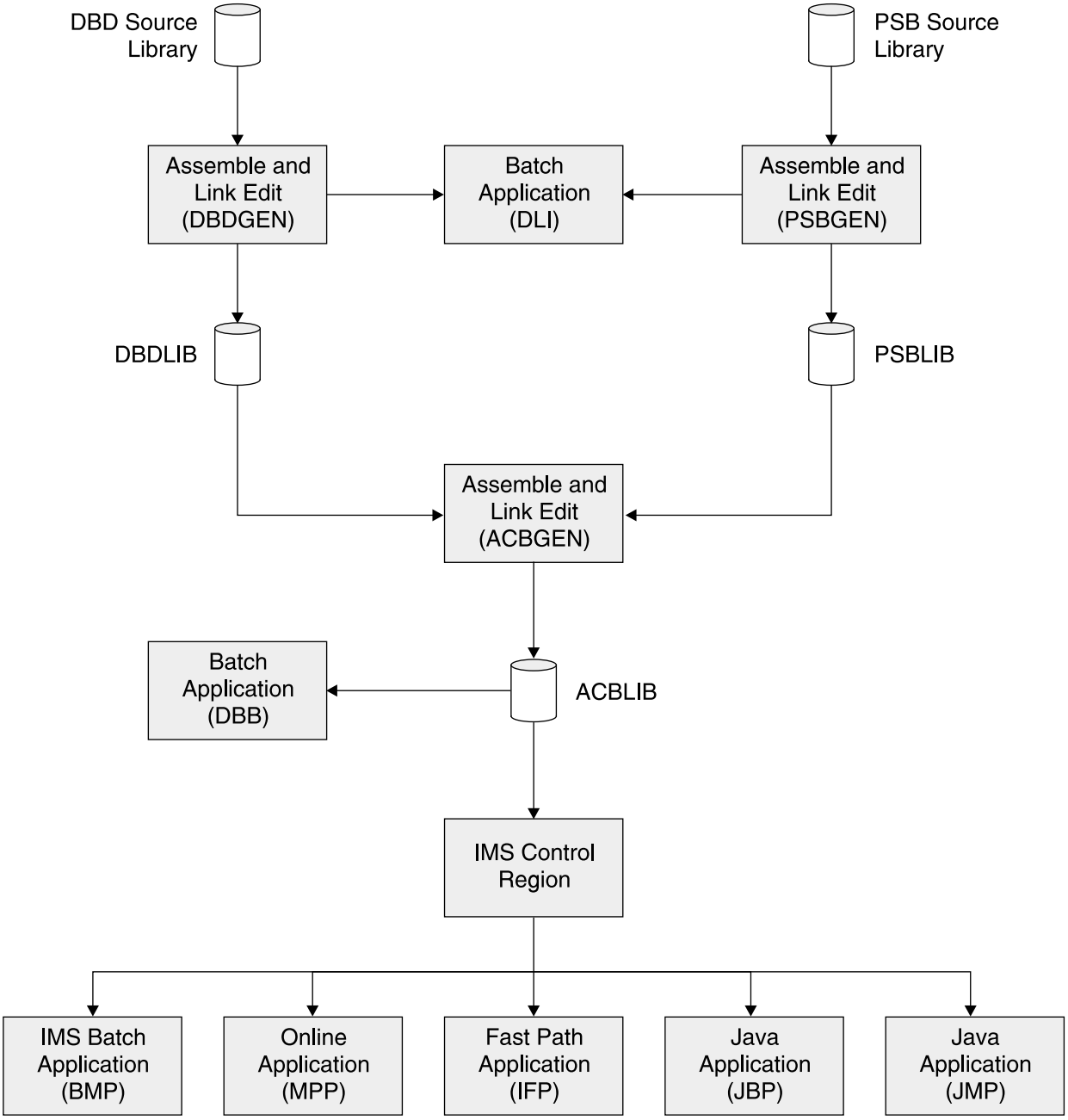

Figure 54. IMS Control Block Generation and Usage

Note: Multiple BUILD statements can be coded for both DBDs and PSBs, but the ones for DBDs must be first.

Generating PSBs

The PSBGEN statement is basically the same as for a database PCB. The IOEROPN= parameter must be omitted, the COMPAT=YES parameter is ignored.

Generating ACBs

Before PSBs and DBDs can be used by the control region, they must be expanded to an internal control block format. This expansion is done by the application control block generation (ACBGEN) utility. The expended control blocks are maintained in the IMS. ACBLIB. This is a standard z/OS partitioned data set. JCL Requirements.

An ACBGEN procedure is placed in IMS.PROCLIB during IMS system definition.

Note: Multiple BUILD statements can be coded for both DBDs and PSBs, but the ones for DBDs must be first.

Additional Application Processing Intent Options

The PROCOPT= keyword is extended with two additional processing intent options, “O” AND “E”. Their meanings are:

O Read only: no dynamic enqueue is done by program isolation for calls against this database. Can be specified with only the G intent option, as GO or GOP. This option is only valid for the PCB statement.

CAUTION:
If the ‘O’ option (read-only) is used for a PCB, IMS does not check the ownership of the segments returned. This means that the read-only user might get a segment that had been updated by another user. If the updating user should then abnormal terminate, and he backed out, the read-only user would have a segment that did not (and never did) exist in the database. Therefore, the ‘O’ option user should not perform updates based on data read with that option. An ABEND can occur with PROCOPT=GO if another program updates pointers when this program is following the pointers. Pointers are updated during insert, delete and backout functions.

E Forces exclusive use of this database or segment by the MPP/BMP. No other program which references this database/segment will be scheduled in parallel. No dynamic enqueue by program isolation is done, but dynamic logging of database updates will be done. E can be specified with G, I, D, B, and A.

IMS Database Application Programming Interface

IMS provides a standard set of functions to allow applications to access and manipulate data managed by the IMS Database Manager. These functions also allow applications to access and process messages managed by the IMS Transaction Manager and to perform certain system functions.

Calls to these functions can be made in a number of ways:

- A language specific call interface. There is one for each programming language that IMS applications can be written in.
- A language independent call interface for applications written in any language that supports IBM’s language environment product.
- The application interface block (AIB) call interface.
- For CICS applications that access IMS databases, the application can use the CICS command level interface to provide IMS DB support.
- REXX EXECs can invoke IMS functions by using the IMS adaptor for REXX

IMS Application Calls

The following list describes the calls that IMS applications can use.

Get Unique (GU)
The GU (get unique) call is used to retrieve a specific segment or path of segments from a database. At the same time it establishes a position in a database from which additional segments can be processed in a forward direction.

Get Next (GN)
The GN (get next) call is used to retrieve the next or path of segments from the database. The get next call normally moves forward in the hierarchy of a database from the current position. It can be modified to start at an earlier position than current position in the database through a command code, but its normal function is to move forward from a given segment to the next desired segment in a database.

Hold Form of Get Calls
GHU (get hold unique), or GHN (get hold next), indicates the intent of the user to issue a subsequent delete or replace call. A get hold call must be issued to retrieve the segment before issuing a delete or replace call.

Insert (ISRT)
The ISRT (insert) call is used to insert a segment or a path of segments into a database. It is used to initially load segments in databases, and to add segments in existing databases.

To control where occurrences of a segment type are inserted into a database, the user normally defines a unique sequence field in each segment. When a unique sequence field is defined in a root segment type, the sequence field of each occurrence of the root segment type must contain a unique value. When defined for a dependent segment type, the sequence field of each occurrence under a given physical parent must contain a unique value. If no sequence field is defined, a new occurrence is inserted after the last existing one.

Delete (DLET)
The DLET (delete) call is used to delete a segment from a database. When a segment is deleted from a DL/I database, its physical dependents, if any are also deleted.

Replace (REPL)
The REPL (replace) call is used to replace the data in the data portion of a segment or path of segments in a database. Sequence fields cannot be changed with a replace call.

System Service Calls
In addition to the functions above, used to manipulate the data, there are a number of system service calls provided to allow the application to make use of other facilities provided by IMS. These system service calls are described in Table 7 on page 162 and Table 8 on page 163.

IMS/DB2 Resource Translate Table

When an IMS transaction accesses DB2, the plan name used is, by default, the same as the PSB/APPLCTN name.

It is, however, possible to set up a translation table, the RTT, that translates an APPLCTN to a different DB2 plan name.

This is described in the DB2 (not IMS) documentation for attaching DB2 to IMS. See Defining DB2 Plans for IMS Applications in *DB2 for z/OS Installation Guide*. It is simply a table of macros, associating APPLCTN macros with DB2 plan names. This is assembled in a CSECT (with the name the same as the label of the 1st macro in the table). This must then be placed in an APF authorized library in the IMS.SDFSRESL concatenation of the IMS control region. The RTT is pointed to in the PROCLIB member that defines the DB2 attachment. If the RTT parameter is null, the RTT is not used.

The re-assembled table will be picked up the next time IMS is stopped/started or when a stop (/STO SUBSYS xxxx) and restart (/STA SUBSYS xxxx) of the DB2 connection.

IMS System Service Calls

Table 7 and Table 8 on page 163 contain summaries of the IMS system service calls that application programs can use in the DB and TM environments.

Related Reading: For complete information about the IMS system service calls, see:

- *IMS Version 9: Application Programming: Database Manager*
- *IMS Version 9: Application Programming: Transaction Manager*

Table 7. Summary of IMS DB System Service Calls

Function Code	Meaning and Use	Options	Valid for
CHKP (Basic)	Basic checkpoint; prepares for recovery	None	DB batch, TM batch, BMP, MPP, IFP
CHKP (Symbolic)	Symbolic checkpoint; prepares for recovery	Specifies up to seven program areas to be saved	DB batch, TM batch, BMP
GMSG	Retrieves a message from the AO exit routine	Waits for an AOI message when none is available	DB/DC and DCCTL (BMP, MPP, IFP), DB/DC and DBCTL (DRA thread), DBCTL (BMP non-message driven), ODBA
GSCD[1 on page 163]	Gets address of system contents directory	None	DB Batch, TM Batch
ICMD	Issues an IMS command and retrieves the first command response segment	None	DB/DC and DCCTL (BMP, MPP, IFP), DB/DC and DBCTL (DRA thread), DBCTL (BMP non-message driven), ODBA
INIT	Initialize; application receives data availability and deadlock occurrence status codes	Checks each PCB database for data availability	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
INQY	Inquiry; returns information and status codes about I/O or alternate PCB destination type, location, and session status	Checks each PCB database for data availability; returns information and status codes about the current execution environment	DB batch, TM batch, BMP, MPP, IFP, ODBA
LOGb[4 on page 163]	Log; writes a message to the system log	None	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
PCBb[4 on page 163]	Specifies and schedules another PSB	None	CICS (DBCTL or DB/DC)

Table 7. Summary of IMS DB System Service Calls (continued)

Function Code	Meaning and Use	Options	Valid for
RCMD	Retrieves the second and subsequent command response segments resulting from an ICMD call	None	DB/DC and DCCTL (BMP, MPP, IFP), DB/DC and DBCTL (DRA thread), DBCTL (BMP non-message driven), ODBA
ROLB	Roll back; eliminates database updates	Returns last message to i/o area	DB batch, TM batch, BMP, MPP, IFP
ROLL	Roll; eliminates database updates; abend	None	DB batch, TM batch, BMP, MPP, IFP
ROLS	Roll back to SETS; backs out database changes to SETS points	Issues call using name of DB PCB or i/o PCB	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
SETS/SETU	Set a backout point; establishes as many as nine intermediate backout points	Cancels all existing backout points	DB batch, TM batch, BMP, MPP, IFP, DBCTL, ODBA
SNAP[2]	Collects diagnostic information	Choose SNAP options	DB batch, BMP, MPP, IFP, CICS (DCCTL), ODBA
STAT[3]	Statistics; retrieves IMS system statistics	Choose type and format	DB batch, BMP, MPP, IFP, DBCTL, ODBA
SYNC	Synchronization; releases locked resources	Requests commit-point processing	BMP
TERM	Terminate; releases a PSB so another can be scheduled; commit database changes	None	CICS (DBCTL or DB/DC)
XRST	Extended restart; works with symbolic checkpoint to restart application program	Specifies up to seven areas to be saved	DB batch, TM batch, BMP

Note:

1. `GSCD` is a Product-sensitive programming interface.
2. `SNAP` is a Product-sensitive programming interface.
3. `STAT` is a Product-sensitive programming interface.
4. `b` indicates a blank. All calls must be four characters.

Table 8. Summary of IMS TM System Service Calls

Function Code	Meaning and Use	Options	Valid Usage
APSB	Allocate PSB. Allocates a PSB for use in CPI-C driven application programs.	None	MPP
CHKP (Basic)	Basic checkpoint. For recovery purposes.	None	batch, BMP, MPP
CHKP (Symbolic)	Symbolic checkpoint. For recovery purposes.	Can specify seven program areas to be saved.	batch, BMP
DPSB	Deallocate PSB. Frees a PSB in use by a CPI-C driven application program.	None	MPP

Table 8. Summary of IMS TM System Service Calls (continued)

Function Code	Meaning and Use	Options	Valid Usage
GMSG	Retrieve a message from the AO exit routine.	Can wait for an AOI message when none is available.	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)
GSCD [1]	Get the address of the system contents directory.	None	batch
ICMD	Issue an IMS command and retrieve the first command response segment.	None	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)
INIT	Application receives data availability status codes.	Checks each PCB for data availability.	batch, BMP, MPP, IFP
INQY	Inquiry. Retrieves information about output destinations, session status, execution environment, and the PCB address.	None	batch, BMP, MPP, IFP
LOGb [2]	Log. Write a message to the system log.	None	batch, BMP, MPP, IFP
RCMD	Retrieve the second and subsequent command response segments resulting from an ICMD call.	None	DB/DC and DCCTL(BMP, MPP, IFP), DB/DC and DBCTL(DRA thread), DBCTL(BMP non-message driven)
ROLB	Rollback. Backs out messages sent by the application program.	Call returns last message to i/o area.	batch, BMP, MPP, IFP
ROLL	Roll. Backs out output messages and terminates the conversation.	None	batch, BMP, MPP
ROLS	Returns message queue positions to sync points set by the SETS or SETU call.	Issues call with i/o PCB or aib	batch, BMP, MPP, IFP
SETS	Sets intermediate sync (backout) points.	Cancels all existing backout points. Can establish up to 9 backout points.	batch, BMP, MPP, IFP
SETU	Sets intermediate sync (backout) points.	Cancels all existing backout points. Can establish up to 9 backout points.	batch, BMP, MPP, IFP
SYNC	Synchronization	Request commit point processing.	BMP
XRST	Restart. Works with symbolic CHKP to restart application program failure.	Can specify up to 7 areas to be saved.	batch, BMP

Note:

1. GSCD is a Product-sensitive programming interface.
2. b indicates a blank. All calls must be four characters.

Chapter 18. Application Programming for the IMS Database Manager

There are two ways that application programs can interact with IMS DB:

- Traditional applications can use the DL/I database call interface.
- Java applications can use IMS Java's implementation of JDBC or the IMS Java hierarchical interface, which is a set of classes that you can use in Java that are similar to DL/I calls.

This chapter discusses the DL/I database call interface. See Chapter 21, "Application Programming in IMS Java," on page 223 for information about how Java applications call IMS.

The following sections are covered in this chapter:

Introduction to Database Processing

In general, database processing is transaction oriented. An application program accesses one or more database records for each transaction it processes. There are two basic types of DL/I application programs:

- The direct access program
- The sequential access program

A direct access program accesses, for every input transaction, some segments in one or more database records. These accesses are based on database record and segment identification. This identification is essentially derived from the transaction input. Normally it is the root-key value an additional (key) field values of dependent segments. For more complex transactions, segments could be accessed in several DL/I databases concurrently.

A sequential application program accesses sequentially selected segments of all of a consecutive subset of a particular database. The sequence is usually determined by the key of the root-segment. A sequential program can also access other databases, but those accesses are direct, unless the root-keys of both databases are the same.

A DL/I application program normally processes only particular segments of the DL/I databases. The portion that a given program processes is called an application data structure. This application data structure is defined in the program specification block (PSB). There is one PSB defined for each application program type. An application data structure always consists of one or more hierarchical data structures, each of which is derived from a DL/I physical or logical database.

Application Programming Interfaces to IMS

During initialization, both the application program and its associated PSB are loaded from their respective libraries by the IMS batch system The DL/I modules, which reside together with the application program in one region, interpret and execute database CALL requests issued by the program.

Calls to DL/I

A call request is composed of a CALL statement with an argument list. The argument list specifies the processing function to be performed, the hierarchic path to the segment to be accessed, and the segment occurrence of that segment. One segment may be operated upon with a single DL/I call. However, a single call never will return more than one occurrence of one segment type.

The arguments contained within any DL/I call request have been defined in “Calls to IMS” on page 155. The following is a sample for a basic CALL statement for COBAL:

```
CALL "CBLTDLI" USING function,PCB-name,I/O Area,  SSA1,...SSAn.
```

Table 9 describes some of the components of the CALL statement. Here you will find the basic DL/I call functions to request DL/I database services.

Table 9. DL/I Function Descriptions

RSF (request service function?)	DL/I Call Function
GET UNIQUE	’GUbb’
GET NEXT	’GNbb’
GET HOLD UNIQUE	’GHUb’
GET HOLD NEXT	’GHNb’
INSERT	’ISRT’
DELETE	’DLET’
REPLACE	’REPL’

Note: b stands for blank. Each CALL function is always 4 characters.

Table 10 constitutes the various categories of segment access types.

Table 10. Segment Access

Segment Access	DL/I Call Function
Retrieve a segment	GUbb, GNbb, GHUb, GHNb
Replace (update) a segment	REPL
Delete a segment	DLET
Insert (add) a segment	ISRT

In addition to the above database calls, there are the system service calls. These are used for requesting systems services such as checkpoint/restart. All of the above calls and some basic system service calls will be discussed in detail in the following sections.

Segment Search Arguments (SSAs)

For each segment accessed in a hierarchical path, one SSA can be provided. The purpose of the SSA is to identify by segment name and, optionally by field value, the segment to be accessed.

The basic function of the SSA permits the application program to apply three different kinds of logic to a call:

- Narrow the field of search to a particular segment type, or to a particular segment-occurrence.
- Request that either one segment or a path of segments be processed.
- Alter DL/I's position in the database for subsequent call.

Segment Search Argument (SSA) names represent the fourth (fifth for PL/I) through last arguments (SSA1 through SSAn) in the call statement. There can be 0 or 1 SSA per level, and, since DL/I permits a maximum of 15 levels per database, a call may contain from 0 to 15 SSA names. In our subset, an SSA consists of one, two or three elements: The segment name, command code(s) and a qualification statement, as shown in Table 11. Table 12 on page 168 shows the values of the relational operators described in Table 11.

Table 11. Segment Name, Command Code, and Qualifications

Operator	Description
Segment name	The segment name must be eight bytes long, left-justified with trailing blanks required. This is the name of the segment as defined in a physical and/or logical DBD referenced in the PCB for this application program.
Command codes	The command code are optional. They provide functional variations to be applied to the call for that segment type. An asterisk (*) following the segment name indicates the presence of one or more command codes. A blank or a left parenthesis is the ending delimiter for command codes. Blank is use when no qualification statement exists
Qualification statement	The presence of a qualification statement is indicated by a left parenthesis following the segment name or, if present, command codes. The qualification statement consists of a field name, a relational-operator, and a comparative-value.
Begin qualification character	The Left parenthesis, "(", indicates the beginning of a qualification statement. If the SSA is unqualified, the eight-byte segment name or if used, the command codes, should be followed by a blank.
Field name	The field name is the name of a field which appears in the description of the specified segment type in the DBD. The name is up to eight characters long, left-justified with trailing blanks as required. The named field may be either the key field or any data field within a segment. The field name issued for searching the database, and must have been defined in the physical DBD.
Relational operator	The relational operator is a set of two characters which express the manner in which the contents of the field, referred to by the field name, is to be tested against the comparative-value. See XREF TAB 13 for a list of the values.

Table 11. Segment Name, Command Code, and Qualifications (continued)

Operator	Description
Comparative value	The comparative value is the value against which the contents of the field, referred to by the field name, is to be tested. The length of this field must be equal to the length of the named field in the segment of the database. That is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required. A collating sequence, not an arithmetic, compare is performed.
End qualification character	The right parenthesis, ")", indicates the end of the qualification statement.

Table 12. Relational Operator Values

Operator	Meaning
b= or 'EQ'	Must be equal to
>= or 'GE'	Must be greater than or equal to
<= or 'LE'	Must be less than or equal to
'b>' or 'GT'	Must be greater than
'b<' or 'LT'	Must be less than
'<>' or 'NE'	Must be not equal to

Note: In Table 12, the lowercase b represents a blank character.

Qualification

Just as calls are "qualified" by the presence of an SSA, SSAs are categorized as either "qualified" or "unqualified", depending on the presence of absence of a qualification statement. Command codes may be included in or omitted from either qualified or unqualified SSAs.

In its simplest form, the SSA is unqualified and consists only of the name of a specific segment type as defined in the DBD. In this form, the SSA provides DL/I with enough information to define the segment type desired by the call. For example:

```
SEGNAMEbb last character blank to unqualified.
```

Qualified SSAs (optional) contain a qualification statement composed of three parts:

- A field name defined in the DBD
- A relational operator
- A comparative value

DL/I uses the information in the qualification statement to test the value of the segment's key or data fields within the database, and thus to determine whether the segment meets the user's specifications. Using this approach. DL/I performs the database segment searching. The program need process only those segments that precisely meet some logical criteria. For example:

```
SEGNAMEb (fieldxxx>=value)
```

The qualification statement test is terminated either when the test is satisfied by an occurrence of the segment type, or when it is determined that the request cannot be satisfied.

Command Codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to the call function or the segment qualification. The command codes are discussed in detail later in this chapter.

General characteristics of segment search arguments:

- An SSA may consist of the segment name only (unqualified). It may optionally also include one or more command codes and a qualification statement.
- SSAs following the first SSA must proceed down the hierarchical path. Not all SSAs in the hierarchical path need be specified. That is, there may be missing levels in the path. DL/I will provide, internally, SSAs for missing levels according to the rules given later in this chapter. However, it is strongly recommended to always include SSAs for every segment level.

Examples of SSAs will be given with the sample calls at each DL/I call discussion in “Handling Status Codes.”

Handling Status Codes

After each DL/I call, a two-byte status code is returned in the PCB which is used for that call. There are three categories of status codes:

- The blank status code, indicating a successful call
- Exceptional conditions and warning status codes from an application point of view
- Error status codes, specifying an error condition in the application program and/or DL/I

The grouping of status codes in the above categories is somewhat installation dependent. We will, however, give a basic recommendation after each specific call function discussion. It is also recommended that you use a standard procedure for status code checking and the handling of error status code. The first two categories should be handled by the application program after each single call. Figure 55 gives an example using COBOL.

```
CALL 'CBLTDLI' USING ....
IF PCB-STATUS EQ 'GE' PERFORM PRINT-NOT-FOUND.
IF PCB STATUS NE 'bb' PERFORM STATUS-ERROR.
everything okay, proceed...
```

Figure 55. Evaluating Status Codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving the real error situations to central status code error routine. A detailed discussion of the error status codes and their handling will be presented later in this chapter.

Sample Presentation of a Call

DL/I calls will be introduced in the following sections. For each call we will give samples. These samples will be in a standard format, as shown in Figure 56 on page 170.

```
77  GU-FUNC            PICTURE XXXX VALUE 'GUbb'


01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN   PICTURE ...
    02  ...
    02  ...

01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.
-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   succesfull call
     --:   exceptional but correct condition
  other:   error condition
```

Figure 56. Sample Call Presentation

All the calls in the samples are presented in COBOL format. The coding of a call in PI/I or Assembler will be presented later. Each call example contains three sections:

1. The first section presents the essential elements of working storage as needed for the call.
2. The second part, the processing section, contains the call itself. Note that the PCB-NAME parameter should see the selected PCB defined in the Linkage Section. Sometimes we will add some processing function description before and/or after the call, in order to show the call in its right context.
3. The third section contains the status codes and their interpretation, which can be expected after the call.

The last category of status code, labeled “other: error situation,” would normally be handled by a status code error routine. A discussion of those error status codes with the presentation of such a routine is later in this chapter.

Processing Against a Single Database Structure

This section discusses processing a single database record. A database record is a root segment and all of its physically dependent child segments.

DL/I Positioning

To satisfy a call, DL/I relies on two sources of segment identification:

- The established position in the database as set by the previous call against the PCB.
- The segment search arguments as provided with the call.

The database position is the knowledge of DL/I of the location of the last segment retrieved and all segments above it in the hierarchy. This position is maintained by DL/I as an extension of, and reflected in, the PCB. When an application program has multiple PCBs for a single database, these positions are maintained independently. For each PCB, the position is represented by the concatenated key of the hierarchical path from the root segment down to the lowest level segment accessed. It also includes the positions of non-keyed segments.

If no current position exists in the database, then the assumed current position is the start of the database. This is the first physical database record in the database. With HDAM this is not necessarily the root-segment with the lowest key value.

Retrieving Segments

There are two basic ways to retrieve a segment:

- Retrieve a specific segment by using a GU type call
- Retrieve the next segment in hierarchy by using a GN type call

If you know the specific key value of the segment you want to retrieve, then the GU call will allow to retrieve only the required segment. If you don't know the key value or don't care then the GN call will retrieve the next available segment which meets your requirements.

The Get Unique (GU) Call

The basic get unique (GU) call, function code "GUbb" normally retrieves one segment in a hierarchical path. The segment retrieved is identified by an SSA for each level in the hierarchical path down to and including the requested segment. Each should contain at least the segment name. The SSA for the root-segment should provide the root-key value. To retrieve more then one segment in the path, see "D Command Code" on page 176. Figure 57 shows an example of the get unique call.

```
77  GU-FUNC             PICTURE XXXX VALUE 'GUbb'

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.

01  IOAREA             PICTURE X(256).
------------------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.
------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   succesfull call
     GE:   exceptional but correct condition
  other:   error condition
```

Figure 57. Basic Get Unique Call

The main use of the GU call is to position yourself to a database record and obtain (a path of) segment (s). Typically, the GU call is used only once for each database record you wish to access. Additional segments within the database record would then be retrieved by means of get next calls (see "The Get Next (GN) Call" on page 172). The GU call can also be used for retrieving a dependent segment, by adding additional SSAs to the call.

For example, if you add a second SSA which specifies the stock location, you would retrieve a STOCK segment below the identified part. If the SSA did not provide a stock location number, this would be the first STOCK segment for this part.

The Get Next (GN) Call

The get next (GN) call, function code ‘GNbb’, retrieves the next segment in the hierarchy as defined in the PCB. To determine this next segment, DL/I relies on the previously established position.

The Unqualified Get Next Call

Figure 58 shows a get next call with no SSAs at all that will, if repeated, return the segments in the database in hierarchical sequence. Only those segments are returned to which the program is defined sensitive in its PCB.

```
77  GN-FUNC             PICTURE XXXX VALUE 'GNbb'


01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------

CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA.

-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   if previous call retrieved a PART, then a STOCK segment will be
           be retrieved
     GK:   a segment is returned in IOAREA, but it is a different type
           at the SAME level, for instance, a PURCHASE ORDER segment
           after the last STOCK segment.
     GA:   segment returned is IOAREA, but it is of a higher level than
           the last one, that is, a new PART segment
     GB:   possible end of database reached, no segment returned
  other:   error condition
```

Figure 58. Unqualified Get Next Call

If the call in Figure 58 was issued after the get unique call in Figure 57 on page 171, then it would retrieve the first STOCK segment for this part (if one existed). Subsequent calls would retrieve all other STOCK, PURCHASE ORDER, and DESCRIPTION segments for this part. After this, the next part would be retrieved and its dependent segments, etc., until the end of the database is reached. Special status codes will be returned whenever a different segment type at the same level or a higher level is returned. No special status code is returned when a different segment at a lower level is returned. You can check for reaching a lower level segment type in the segment level indicator in the PCB. Remember, only those segments to which the program is sensitive via its PCB are available to you.

Although the unqualified GN call illustrated in Figure 58 might be efficient, especially for report programs, you should use a qualified GN call whenever possible.

The Qualified Get Next Call

This qualified GN call should at least identify the segment you want to retrieve. In doing so, you will achieve a greater independence toward possible database structure changes in the future. Figure 59 on page 173 shows an example of a qualified GN call. If you supply only the segment name in the SSA, then you will retrieve all segments of that type from all database records with subsequent get next calls.

```
77  GN-FUNC             PICTURE XXXX VALUE 'GNbb'

01  SSA002-GN-SE1PPUR   PICTURE X(9) VALUE 'SE1PPURbb'

01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA002-GN-SE1PPUR.

-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   next PURCHACE ORDER has been move to the IOAREA
     GB:   end of database reached, no segment returned
  other:   error condition
```

Figure 59. Qualified Get Next Call

Repetition of the above GN call will retrieve all subsequent PURCHASE ORDER segments of the database, until the end of the database is reached. To limit this to a specific part, you could add a fully qualified SSA for the PART segment. This would be the same SSA as used in Figure 57 on page 171.

An example of a qualified get next call with a qualified SSA is shown in Figure 60.

```
77  GN-FUNC             PICTURE XXXX VALUE 'GNbb'

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.

01  SSA002-GN-SE1PPUR   PICTURE X(9)  VALUE 'SE1PPURb'.
01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------

CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
                     SSA002-GN-SE1PPUR.
-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   next PURCHASE ORDER segment is in IOAREA
     GE:   segment not found; no more purchase orders for this part,
           or part number in SSA001 does not exist
  other:   error condition
```

Figure 60. Qualified Get Next Call with Qualified SSA

This fully qualified get next call should be primarily used. It always clearly identifies the hierarchical path and the segment you want to retrieve.

The Get Hold Calls

To change the contents of a segment in a database through a replace or delete call, the program must first obtain the segment. It then changes the segment's contents and requests DL/I to replace the segment in the database or to delete it from the database.

This is done by using the get hold calls. These function codes are like the standard get function, except the letter 'H' immediately follows the letter 'G' in the code (that is, GHU, GHN). The get hold calls function exactly as the corresponding get calls for the user. For DL/I they indicate a possible subsequent replace or delete call.

After DL/I has provided the requested segment to the user, one or more fields, but not the sequence field, in the segment may be changed.

After the user has changed the segment contents, he can call DL/I to return the segment to, or delete it from the database. If, after issuing a get hold call, the program determines that it is not necessary to change or delete the retrieved segment, the program may proceed with other processing, and the “hold” will be released by the next DL/I call against the same PCB.

Updating Segments

Segments can be updated by application programs and returned to DL/I for restoring in the database, with the replace call, function code REPL’ Two conditions must be met:

- The segment must first be retrieved with a get hold call, (GHU or GHN), no intervening calls are allowed referencing the same PCB.
- The sequence field of the segment cannot be changed. This can only be done with combinations of delete and insert calls for the segment and all its dependents.

Figure 61 shows an example of a combination of GHU and REPL call. Notice that the replace call must not specify a SSA for the segment to be replaced. If, after retrieving a segment with a get hold call, the program decides not to update the segment, it need not issue a replace call. Instead the program can proceed as if it were a normal get call without the hold.

```
77  GHU-FUNC             PICTURE XXXX VALUE 'GHUb'.
77  REPL-FUNC            PICTURE XXXX VALUE 'REPL'.

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.
01  SSA002-GN-SE1PPUR   PICTURE X(9)  VALUE 'SE1PPURbb'.
01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
               SSA002-GN-SE1PPUR.
   the retrieved PURCHASE ORDER segment can now be changed by the program
   in the IOAREA.
CALL 'CBLTDLI' USING REPL-FUNC,PCB-NAME,IOAREA.
-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   segment is replaced with contents in the IOAREA
  other:   error condition
```

Figure 61. Basic Replace Call

Use the get hold call whenever there is a reasonable chance (about 5% or more) that you will change the segment because there is only a very small performance difference between the get and the get hold call.

Deleting Segments

To delete the occurrence of a segment from a database, the segment must first be obtained by issuing a get hold (GHU, GHN) call. Once the segment has been acquired, the DLET call may be issued.

No DL/I calls which use the same PCB can intervene between the get hold call and the DLET call, or the DLET call is rejected. Quite often a program may want to process a segment prior to deleting it. This is permitted as long as the processing does not involve a DL/I call which refers to the same database PCB used for the get hold/delete calls. However, other PCBs may be referred to between the get hold and DLET calls.

DL/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. The deletion of a parent, in effect, deletes all the segment occurrences beneath that parent, whether or not the application program is sensitive to those segments. If the segment being deleted is a root segment, that whole database record is deleted. The segment to be deleted must still be in the IOAREA of the delete call (with which no SSA is used), and its sequence field must not have been changed. Figure 62 gives an example of a DLET call.

```
77  GHU-FUNC             PICTURE XXXX VALUE 'GHUb'.
77  DLET-FUNC            PICTURE XXXX VALUE 'DLET'.

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.
01  SSA002-GN-SE1PPUR   PICTURE X(9)  VALUE 'SE1PPURbb'.
01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
               SSA002-GN-SE1PPUR.

   the retrieved PURCHASE ORDER segment can now be processed by the
    program in the IOAREA.

CALL 'CBLTDLI' USING DLET-FUNC,PCB-NAME,IOAREA.
-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   requested purchase order segment is deleted from the database;
           all its dependents, if any, are deleted also.
  other:   error condition
```

Figure 62. Basic Delete Call

Inserting Segments

Adding new segment occurrences to a database is done with the insert call, function code ‘ISRT’.

The DL/I insert call is used for two distinct purposes: It is used initially to load the segments during creation of a database. It is also used to add new occurrences of an existing segment type into an established database. The processing options field in the PCB indicates whether the database is being added to or loaded. The format of the insert call is identical for either use.

When loading or inserting, the last SSA must specify only the name of the segment being inserted. It should specify only the segment name, not the sequence field. Thus an unqualified SSA is always required.

Up to a level to be inserted, the SSA evaluation and positioning for an insert call is exactly the same as for a GU call. For the level to be inserted, the value of the sequence field in the segment in the user I/O area is used to establish the insert

position. If no sequence field was defined, then the segment is inserted at the end of the physical twin chain. If multiple non-unique keys are allowed, then the segment is inserted after existing segments with the same key value.

Figure 63 shows an example of an ISRT call. The status codes in this example are applicable only to non-initial load inserts. The status codes at initial load time will be discussed under “Loading Databases” on page 187.

```
77  ISRT-FUNC            PICTURE XXXX VALUE 'ISRT'.

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.
01  SSA002-GN-SE1PPUR   PICTURE X(9)  VALUE 'SE1PPURbb'.
01  IOAREA             PICTURE X(256).
-------------------------------------------------------------------------
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
MOVE PURCHASE-ORDER TO IOAREA.
CALL 'CBLTDLI' USING ISRT-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
               SSA002-GN-SE1PPUR.
-------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   new PURCHASE ORDER segment is inserted in database
     II:   segment to insert already exists in database
     GE:    segment not found; the requested part number (that is, a
            parent of the segment to be inserted) is not in the database
  other:   error condition
```

Figure 63. Basic Insert Call

Note: There is no need to check the existence of a segment in the database with a preceding retrieve call. DL/I will do that at insert time, and will notify you with an II or GE status code. Checking previous existence is only relevant if the segment has no sequence field.

Calls with Command Codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to either the call function or the segment qualification. Command codes in an SSA are always prefixed by an asterisk (*), which immediately follows the 8 byte segment name. Figure 64 illustrates an SSA with command codes D and P.

```
01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb*DP(FE1PGPNRb='.
    02  SSA001-FE1PGPNR PICTURE X(8).
    02  SS1001-END      PICTURE X     VALUE ')'.
```

Figure 64. Example of an SSA with D and P Command Codes

D Command Code

The ‘D’ command code is the one most widely used. It requests DL/I to issue path calls. A “path call” enables a hierarchical path of segments to be inserted or retrieved with one call. (A “path” was defined earlier as the hierarchical sequence of segments, one per level, leading from a segment at one level to a particular segment at a lower level.) The meaning of the ‘D’ command code is as follows:

- For retrieval calls, multiple segments in a hierarchical path will be moved to the I/C area with a single call. The first through the last segment retrieved are concatenated in the user's I/C area. Intermediate SSAs may be present with or without the 'D' command code. If without, these segments are not moved to the user's I/O area. The segment named in the PCB "segment name feedback area" is the lowest-level segment retrieved, or the last level satisfied in the call in case of a non-found condition. Higher-level segments associated with SSAs having the 'D' command code will have been placed in the user's I/O area even in the not-found case. The 'D' is not necessary for the last SSA in the call, since the segment which satisfies the last level is always moved to the user's I/O area. A processing option of 'P' must be specified in the PSBGEN for any segment type for which a command code 'D' will be used.
- For insert calls, the 'D' command code designates the first segment type in the path to be inserted. The SSAs for lower-level segments in the path need not have the D command code set, that is, the D command code is propagated to all specified lower level segments.

Figure 65 shows an example of a path call.

```
77  GU-FUNC             PICTURE XXXX VALUE 'GUbb'.

01  SSA004-GU-SE1PART.
    02  SSA004-BEGIN    PICTURE x(21) VALUE 'SE1PARTb*D(FE1PGPNRb='.
    02  SSA004-FE1PGPNR PICTURE X(8).
    02  SS1004-END      PICTURE X     VALUE ')'.
01  SSA005-GN-SE1PGDSC  PICTURE X(9)  VALUE 'SE1PGDSCb'.

01  IOAREA              PICTURE X(256).
------------------------------------------------------------------------

CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA004-GU-SE1PART
               SSA004-GN-SE1PGDSC.

------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   both segments (PART and DESCRIPTION) have been placed in IOAREA
     GE:   segment not found; PART segment may be retrieved in IOAREA;
           check segment name and level indicator in PCB.
  other:   error condition
```

Figure 65. Sample Path Retrieve Call

Figure 65 shows a common usage of the path call. Although we don't know if the requested part has a separate DESCRIPTION segment (SE1PGDSC), we retrieve it at almost no additional cost if there is one.

N Command Code

When a replace call follows a path retrieve call, it is assumed that all segments previously retrieved with the path call are being replaced. If any of the segments have not been changed, and therefore, need not be replaced, the 'N' command code may be set at those levels, telling DL/I not to replace the segment at this level of the path. The status codes returned are the same as for a replace call.

F Command Code

This command code allows you to back up to the first occurrence of a segment under its parent. It has meaning only for a get next call. A get unique call always starts with the first occurrence. Command code F is disregarded for the root segment.

L Command Code

This command code allows you to retrieve the last occurrence of a segment under its parent. This command code should be used whenever applicable.

Hyphen (-) Command Code

The hyphen is a null command code. It s purpose is to simplify the maintenance of SSAs using command codes.

Database Positioning After DL/I Calls

As stated before, the database position is used by DL/I to satisfy the next call against the PCB. The segment level, segment name and the key feedback areas of the PCB are used to present the database position to the application program.

The following basic rules apply:

- If a get call is completely satisfied, current position in the database is reflected in the PCB key feedback area.
- A replace call does not change current position in the database.
- Database position after a successful insert call is immediately after the inserted segment.
- Database position after return of an II status code is immediately prior to the duplicate segment. This positioning allows the duplicate segment to be retrieved with a GN call.
- Database position after a successful delete call is immediately after all dependents of the deleted segment. If no dependents existed, database position is immediately after the deleted segment.
- Database position is unchanged by an unsuccessful delete call.
- After an (partial) unsuccessful retrieve call, the PCB reflects the lowest level segment which satisfied the call. The segment name or the key feed back length should be used to determine the length of the relevant data in the key feedback area. Contents of the key feedback area beyond the length value must not be used, as the feedback area is never cleared out after previous calls. If the level-one (root) SSA cannot be satisfied, the segment name is cleared to blank, and the level and key feedback length are set to 0.

In considering ‘current position in the database’, it must be remembered that DL/I must first establish a starting position to be used in satisfying the call. This starting position is the current position in the database for get next calls, and is a unique position normally established by the root SSA for get unique calls.

The following are clarifications of ‘current position in the database’ for special situations:

- If no current position exists in the database, then the assumed current position is the start of the database.
- If the end of the database is encountered, then the assumed current position to be used by the next call is the start of the database.
- If a get unique call is unsatisfied at the root level, then the current position is such that the next segment retrieved would be the first root segment with a key value higher than the one of the unsuccessful call, except when end of the database was reached (see above) or for HDAM, where it would be the next segment in physical sequence.

You can always reestablish your database positioning with a GU call specifying all the segment key values in the hierarchical path. It is recommended that you use a get unique call after each not found condition.

Using Multiple PCBs for One Database

Whenever there is a need to maintain two or more independent positions in one database, you should use different PCBs. This avoids the reissue of get unique calls to switch forward and backward from one database record or hierarchical path to another. There are on restrictions as to the call functions available in these multiple PCBs. However, to avoid “position confusion” in the application program, you should not apply changes via two PCBs to the same hierarchical path. For simplicity reasons you should limit the updates to one PCB unless this would cause additional calls.

System Service Calls

Besides call functions for manipulating database segments, DL/I provides special system service calls. The most common ones are:

STATISTICS (STAT)
This call is used to obtain various statistics from DL/I.

CHECKPOINT (CHPK)
CHPK informs DL/I that the user has “checkpointed” his program and that thus may be restarted at this point. The current position is maintained in GSAM databases. For all other databases, you must reposition yourself after each checkpoint call with a get unique call.

RESTART (XRST)
XRST requests DL/I to restore checkpointed user areas and reposition GSAM database for sequential processing if a checkpoint ID for restarting has been supplied by the call or in the JCL.

The XRST and CHKP calls will be discussed under the topic “Using Batch Checkpoint/Restart” on page 192.

Processing GSAM Databases

All accessing to GSAM databases is done via DL/I calls. A check is made by DL/ to determine whether a user request is for a GSAM database. if so, control is passed to GSAM, which will be resident in the user region. If not, control is passed to DL/I, and standard hierarchical processing will result.

Calls to be used for GSAM accessing are:

```
CALL 'CBLTDLI' USING call-func,pcb-name,ioarea.
```

Where:

call-func
Is the name of the field that contains the call function. The function could be:

OPEN Open the GSAM database

CLSE Close the GSAM database

GN Retrieve the next sequential record

ISRT Insert a new logical record (at end of database only)

The open and close call are optional calls to be used to explicitly initiate or terminate database operations. The database will automatically be opened by the issuance of the first processing call used and automatically closed at “end-of-data” or at program termination.

Records may not be randomly added to GSAM data sets. The data set may be extended by opening in the load mode, with DISP=MOD, and using the ISRT function code.

pcb-name
Is the name of the GSAM PCB

ioarea Is the name of the I/O area for GN/ISRT calls

Table 13 contains the status codes associated with processing GSAM databases.

Table 13. Status Codes Associated with Processing GSAM Databases

Status Code	Meaning
bb	Successful call, Proceed
GL	End of input data (Get Next only)
other	error situation

Record Formats

Records may be fixed or variable length, blocked or unblocked. Records must not have a sequence key. The record in the IOAREA includes a halfword record length for variable length records.

The use of GSAM data sets in a checkpoint/restart environment is further discussed later in this chapter.

Language Specific Programming Considerations

The next few sections discuss programming considerations that are unique to different programming languages.

- “COBOL Programming Considerations”
- “Java Programming Considerations” on page 182
- “PL/I Programming Considerations” on page 182

COBOL Programming Considerations

There are a few considerations that apply when you are coding DL/I programs in COBOL. See Figure 66 on page 181 for this discussion as the numbers between parenthesis in the text below see the corresponding code lines. Specific parameter values and formats are explained elsewhere throughout this chapter

```
ID
 DIVISION.                                                                  000001
                                                                            000002
ENVIRONMENT DIVISION.                                                       000003
                                                                            000004
DATA DIVISION.                                                              000005
WORKING-STORAGE SECTION.                                                    000006
77          GU-FUNC            PIC XXXX   VALUE   'GU  '.                  000007
77          GN-FUNC            PIC XXXX   VALUE   'GN  '.                  000008
77          ERROPT             PIC XXXX   VALUE   '1      '.                000009
77          DERRID             PIC X(8)   VALUE   'DERROR01'.               000010
01          IOAREA             PIC X(256) VALUE SPACES.                    000011
01          SSA001-GU-SE1PART.                                              000012
              02 SSA001-BEGIN     PIC X(19) VALUE 'SE1PART (FE1PGPNR ='.    000013
              02 SSA001-FE1PGPNR  PIC X(8).                                 000014
              02 SSA001-END       PIC X     VALUE ')'.                     000015
                                                                            000016
LINKAGE SECTION.                                                            000017
01         D1PC.                                                            000018
              02   D1PCDBN  PIC X(8).                                       000019
              02   D1PCLEVL PIC 99.                                         000020
              02   D1PCSTAT PIC XX.                                         000021
              02   D1PCPROC PIC XXXX.                                       000022
              02   D1PCRESV PIC S9(5) COMP.                                 000023
              02   D1PCSEGN PIC X(8).                                       000024
              02   D1PCKFBL PIC S9(5) COMP.                                 000025
              02   D1PCNSSG PIC S9(5) COMP.                                 000026
              02   D1PCKFBA PIC X(20).                                      000027
                                                                            000028
PROCEDURE DIVISION.                                                         000029
  ENTRY 'DLITCBL' USING D1PC.                                               000030
  :                                                                         000031
  :                                                                         000032
  CALL 'CBLTDLI' USING GU-FUNC, D1PC, IOAREA,                               000033
       SSA001-GU-SE1PART.                                                   000034
  :                                                                         000035
  CALL 'CBLTDLI' USING GN-FUNC, D1PC, IOAREA.                               000036
  IF D1PCSTAT NOT = '  ',                                                   000037
     CALL 'ERRRTN' USING D1PC, DERRID, IOAREA, ERROPT.                      000038
     MOVE +4 TO RETURN-CODE.                                                000039
  :                                                                         000040
  CALL DFSOAST USING D1PC.                                                  000041
  :                                                                         000043
  :                                                                         000044
  GOBACK.                                                                   000045
```

Figure 66. Example of a COBOL Batch Program

- The DL/I function codes (7)(, IOAREA (11), and Segment Search Arguments (12) should be defined in the Working-Storage Section of the Data Division. Typically, either the IOAREA would be REDEFINED to provide addressability to the fields of each segment, or separate IOAREAs would be defined for each segment.
- The program Communication Blocks (PCBS) Should be defined in the Linkage Section of the Data Division (18). When there are multiple database structures (thus multiple PCBs) in a program, there must be one PCB defined in the Linkage Section for each PCB in the PSB. However, these PCBs need not be in any specific order.
- An ENTRY statement (30) should be coded at the entry to your program. A parameter of the USING clause should exist for each database structure (PCB) that is used in your program. The order of PCBs in this clause must be the same as specified in the Program Specification Block (PSB) for your program.

- Each DL/I CALL statement should be coded as in statement (33). The parameters of the DL/I call are explained elsewhere in this chapter, and differ in number for different functions.
- The status code in the PCB should be checked after each call (37). The status-code error routine is discussed below (38).
- At the end of processing, control must be returned to DL/I via a GOBACK statement (44). Optionally, you can set the COBOL 'RETURN-CODE' (39). If DL/I detects no errors, and thus does not set the return code, the COBOL 'RETURN-CODE' value will be passed on to the next job step.

Java Programming Considerations

The basic programming considerations for Java are discussed in Chapter 21, "Application Programming in IMS Java," on page 223.

PL/I Programming Considerations

This section refers to Figure 67 on page 183. The numbers between parenthesis in the text following the figure see the corresponding code line.

```
/*-------------------------------------------------------------------*/0000001
/*               SAMPLE PL/I PROGRAM                                 */0000002
/*------------------------------------------------------------------ */0000003
PE2PROD:                                                              0000005
PROCEDURE (DC PTR,DB_PTR) OPTIONS (MAIN);                             0000006
/*        DECLARE POINTERS AND PCBS.                                  */0000008
 DECLARE                                                              0000010
   PLITDLI ENTRY,                              /* DL/I WIlL BE CALLD*/ 0000012
   DFSOAST ENTRY OPTIONS (ASSEMBLER INTER),    /* STATISTICS PRINT  */ 0000013
   DFSOAER ENTRY OPTIONS (ASSEMBLER INTER),    /* STATUS COOE PRINT */ 0000014
   DC_PTR POINTER,                             /* CHPAT IN PSB      */ 0000015
   DB_PTR POINTER,                             /* ORDER DB PCB      */ 0000016
  01 C1PC BASED (DC_PTR),                      /* NOT USED IN       */ 0000018
     02 DUMMY CHAR (32),                       /* BATCH DL/I        */ 0000019
  01 D1PC BASED (DB_PTR),                      /* PHASE 2 ORDER DB  */ 0000021
     02 D1PCDBDN CHAR (8),                     /* DBD NAME          */ 0000022
     02 D1PCLEVL CHAR (2),                     /* SEGMENT LEVEL     */ 0000023
     02 D1PCSTAT CHAR (2),                     /* STATUS CODE       */ 0000024
     02 D1PCPROC CHAR (4),                     /* PROCESSING OPTN   */ 0000025
     02 O1PCRESV FIXED BINARY(31),             /* RESERVED          */ 0000026
     02 D1PCSEGN CHAR (8),                     /* SEGMENT NAME      */ 0000027
     02 D1PCKFBL FIXED BINARY(31),             /* KEY FEEOBACK LNG  */ 0000028
     02 D1PCNSSG FIXED BINARY(31),             /* N0. OF SENSEGS    */ 0000029
     02 D1PCKFBA CHAR (14);                    /* KEY FEEDBACK      */ 0000030
/*  DECLARE FUNCTION COOES, I/0 AREA, CALL ARG LIST LENGTHS         */ 0000032
  DECLARE                                                             0000034
    IO_AREA CHAR (256)                        /* I/0 AREA          */ 0000036
    GU_FUNC STATIC CHAR (4) INIT t'GU'I,      /* CALL FUNCTION     */ 0000037
    FOUR STATIC FIXED BINARY (31) INIT I4 ),  /* ARG LIST LENGTH   */ 0000038
    ERROPT1 CHAR (4) INIT ('0') STATIC,       /* OPTN FOR DFSOAER  */ 0000039
    ERROPT2 CHAR (4) INIT ('2') STATIC,       /* FINAL OPTN:DFSOAER*/ 0000040
    DERRID CHAR (8) INIT ('DERFORO1') STATIC; /* ID FOR DFSOAER    */ 0000041
/* DECLARE SEGMENT SEARCH AFGUMENT (SSA) - ORDER SEGMENT.           */ 0000043
  DECLARE                                                             0000045
   01 SSA007_GU_SE2OPDER,                                             0000047
      02 SSA007_BEGIN CHAR (19) INIT ('SE2ORDER(FE2OGPEF ='),         0000048
      02 SSA007_FE2OG2EF CHAR (6),                                    0000049
      02 SSA007_END CHAR (1) INIT ('1');                              0000050
/* PROCESSING PORTION OF THE PROGRAM                                */ 0000052
 SSACO7_FE2OGREF = 'XXXXXXX';                 /* SET SSA VALUE     */ 0000054
 CALL PLITDLI (FOUR,GU_FUNC,.DB_PTR,IO_AREA,  /* THIS CALL WILL    */ 0000055
             SSA007_GU_FE2ORDER);             /* RETURN 'GE' STAT  */ 0000056
 IF D1PCSTAT -- '  ' THEN                     /* CALL EROOR PRINT  */ 0000057
    CALL DFSOAER (D1FC,DERRID,IO_AREA,ERROPT1);                       0000058
    CALL DFSOAER (D1PC,DERRID,IO AREA,ERROPT2); /* FINAL CALL TO ERR*/ 0000059
/* RETURN TO CALLER.                                                */ 0000065
  END PE2PORD;                                                        0000067
```

Figure 67. Example of a PL/I Batch Program

When DL/I invokes your PL/I program it will pass the addresses, in the form of pointers, to each PCB required for execution. These will be passed in the same sequence as specified in PSB. To use the PCBs, you must code parameters in your PROCEDURE statement, and declare them to have the attribute POINTER.

In the example, DC_PTR and DB_PTR are specified in the PROCEDURE statement (6) and declared POINTER variables (15 and 16). These pointer variables should be used in declaring the PCBs as BASED structures (18 and 21), and in calling DL/I(55).

The format of the PL/I CALL statement to invoke DL/I (55) is:

```
CALL PLITDLI (parmcount, function, pcb-ptr, io-area,ssal,...,ssan):
```

Where:

parmcount Is the number of arguments in this call following this argument. It must have the attributes FIXED BINARY (31). See (38).

function Is the DL/I function code. It must be a fixed length character string of length 4. *pcb-ptr* is a pointer variable containing the address of the PCB. This is normally the name of one of the parameters passed to your program at invocation.

io-area Is the storage in your program into/from which DL/I is to store/fetch data. It can be a major structure, a connected array, a fixed-length character string (CHAR (n)), a pointer to any of these or a pointer to a minor structure. It cannot be the name of a minor structure of a character string with the attribute VARYING.

ssa1... Is one or more optional segment search arguments. Each SSA argument must be one of the same PL/I forms allowed for io-areas, described above. See (47) in the example.

Upon completion of your program, you should return either via a RETURN statement or by executing the main procedure END statement.

Processing Databases with Logical Relationships

Generally, there is no difference between the processing of physical databases and logical databases: all call functions are available for both. Some considerations do apply, however, when accessing a logical child of a concatenated segment.

Accessing a Logical Child in a Physical Database

When accessing a logical child in a physical DBD, you should remember the layout of the logical child. It always consists of the logical parent concatenated key (that is, all the consecutive keys from the root segment down to and including the logical parent) plus the logical child itself: the intersection data (see Figure 60 on page 173). This is especially important when inserting a logical child. You will also get an IX status code when you try to insert a logical child and its logical parent does not exist (except at initial load time). This will typically happen when you forget the LPCK in front of the LCHILD.

Note: In general, physical databases should not be used when processing logical relationships.

Accessing Segments in a Logical Database

The following considerations apply for each call function when accessing segments in logical DBDs.

Retrieve Calls

These calls function as before with the same status codes. Remember, however, that the concatenated segment always consists of the logical child segment plus, optionally (dependent on the logical DBD), the destination parent segment.

Replace Calls

In general, these calls function the same as before. When replacing a concatenated segment you may replace both the logical child segment and the destination parent. Remember, however, that you never can change a sequence field. The following sequence fields can occur in a concatenated segment:

- Destination parent concatenated key.

- Real logical child sequence field, (that is, the sequence of the physical twin chain as defined for the real logical child). This field can (partially) overlap the logical parent concatenated key.
- Virtual logical child sequence field, (that is, the sequence of the logical twin chain as defined for the virtual logical child). This field can (partially) overlap the physical parent concatenated key.
- The key of the destination parent itself.

If any of the above fields is changed during a replace operation, a DA status code will be returned, and no data will be changed in the database.

Delete Calls

In general, these calls function the same as before. If, however, you delete a concatenated segment (either of the two versions), only the logical child and its physical dependents (that is, the dependents of the real logical child) will be deleted. the destination parent can be deleted only via its physical path. In other words: “The delete is not propagated upwards across a logical relation.” You can delete only those dependents of concatenated segments which are real dependents of the logical child. Examples:

- If the logical DBD of Figure 13 on page 47, a PART segment was deleted, the associated STOCK and DETAIL segments are deleted, too. However, the associated CUSTOMER ORDER and SHIPMENT segments remain.
- If the logical DBD of Figure 13 on page 47, a CUSTOMER ORDER segment was deleted, the associated DETAIL and SHIPMENT segments are deleted too. However, the associated PART and, STOCK segments remain.

Notice the logical child (and its physical dependents) is always deleted whenever one of its parents is deleted.

Insert Calls

Whenever you insert a concatenated segment, the destination parent must already exist in the database. You can provide the destination parent together with the logical child in the IOAREA, but it is not used. Besides the normal status codes, an IX status code is returned when the destination parent does not exist.

Processing Databases with Secondary Indexes

Access segments via a secondary index allows a program to process segments in a order which is not the physical sequence of the database. One good example of this is the ORDER segment. To process an order when only the Customer order number is known, the ORDER segment can be access via the customer order number. This is the simplest from of secondary index.

Another basic use for a secondary index is to provide a method of processing a subset of the segments in a database without having to read the entire database.An example of this would be to provide a secondary index on a Balance owning field in the customer database. The secondary index database could be defined to only contain those database records for which a non-zero balance is owning.

Accessing Segments by Using a Secondary Index

The format of the CALL parameters for accessing segments via a secondary index are identical to those access through the primary path. The difference is in the PCB coded in the PSB. The second PCB in the PSB in Figure 68 on page 186 shows how to define a process using the secondary index.

```
*
* PSB with Secondary index PCB
*
          PCB    TYPE=DB,PROCOPT=G,
                      DBDNAME=BE2CUST,,KEYLEN=6

          PCB    TYPE=DB,PROCOPT=G,
                      DBDNAME=BE2CUST,,PROCSEQ=FE2CNAM,,KEYLEN=20
*
                     SENSEQ NAME=SE2PSCUST

   PSBGENG,LANG=COBOL,PSBNAME=SE2PCUST,CMPAT=YES
  END
```

Figure 68. Example of a PSB with a Secondary Index Defined

Retrieving Segments

The same calls are used as before. However, the index search field, defined by an XDFLD statement in the DBD will be used in the SSA for the get unique of the root segment. It defines the secondary processing sequence.

After the successful completion of this get unique call, the PCB and ICAREA look the same as after the basic GU of Figure 57 on page 171, except that the key feedback area now starts with the customer name field.

When using the secondary processing sequence, consecutive get next calls for the CUSTOMER ORDER segment will present the CUSTOMER ORDER segments in customer name sequence.

If both the primary and the secondary processing sequence are needed in one program, you should use two PCBs as shown in Figure 69.

```
77  GU-FUNC                          PICTURE XXXX   VALUE 'GUbb'

01  SSA002-GU-SE2PCUST.
    02  SSA002-BEGIN            PICTURE x(19) VALUE 'SE2PCUSTb(FE2PCNAMb='.
    02  SSA002-FE2PCNAM PICTURE X(20).
    02  SS1002-END                 PICTURE X     VALUE ')'.

01  IOAREA                            PICTURE X(256).
------------------------------------------------------------------------
MOVE CUSTOMER-NAME TO SSA002-FE2PCNAM.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA002-GU-SE2PCUST.
------------------------------------------------------------------------
STATUS CODES:
-------------
     bb:   succesfull call
     GE:   exceptional but correct condition
  other:   error condition
```

Figure 69. Example of a Get Unique Call Using a Secondary Index

Replacing Segments

To replace segments in the indexed database a combination of get hold and replace calls can be used as before. Again, no sequence fields may be changed. The index search fields, however, can be changed. If an index search field is changed, DL/I will automatically update the index database via a delete old and insert new pointer segment.

Note: When using a secondary processing sequence, this could result in the later re accessing of a database record.

Deleting Segments

When using a secondary processing sequence, you cannot delete the index target segment (that is, the root segment). If you have a need to do so, you should use a separate PCB with a primary processing sequence.

Inserting Segments

Again, when using a secondary processing sequence, you cannot insert the index target segment. In all other cases, the ISRT call will function as before.

Creating Secondary Indexes

A secondary index can be created during initial load of the indexed database or later. The secondary index database is created with the DL/I reorganization utilities. No application program requirements.

Loading Databases

Loading databases with information has some considerations for the application program and the PSB used.

Overview of Loading Databases

Basically the load program inserts segments into the database from some kind of input. It builds the segments and inserts them in the database in hierarchical order. Quite often the data to be stored in the database already exists in one or more files, but merge and sort operations may be required to present the data in the correct sequence.

The process of loading database is different than updating a database with segments already in the it. A database must be initialized before it can be used by most application programs. A database can be initialize in several ways:

- Data reloaded by the database recovery utility
- Data loaded by a database reload utility
- Data loaded by a program with the PROCOPT of L (full-function only)

Once the database is initialize it will remains so until it has been deleted and redefined. Therefore is it possible to have an empty initialize database. A database which is not empty can not be used by a PSB with a PROCOPT of L nor can it be recovered or loaded with the reload utility.

If the database has no secondary indexes or logical relationship, then the load process is very straight forward. Any program with a PROCOPT of L can load it. Once that program has completed and close the database, the database can then be used by any program for read or update.

The loading of database with logical relationships and secondary indexes are discussed next.

Loading a HDAM Database

When initially loading an HDAM database, you should specify PROCOPT=L in the PCB. There is no need for DL/I to insert the database records in root key order, but you must still insert the segments in their hierarchical order.

For performance reasons it is advantageous to sort the database records into sequence. The physical sequence should be the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms. This can be achieved by using a tool from the IMS/ESA System Utilities/Database Tools (WHAT TOOL? AN IMS UTILITY?). This tool provides a sort exit routine, which gives each root key to the randomizing module for address conversion, and then directs SORT to sort on the generated address + root key value.

Status Codes for Loading Databases: The status codes, as shown in Table 14, can be expected when loading basic databases after the ISRT call:

Table 14. Database Load Status Codes

Returned Status Code	Explanation
bb or CK	Segment is inserted in database
LB	The segment already exists in database
IC	The key field of the segment is out of sequence
LD	No parent has been inserted for this segment in the database
other	Error situation

Status Codes for Error Routines: There are essentially two categories of error status codes: those caused by application program errors and those caused by system errors. Sometimes, however, a clear split cannot be made immediately.

This listing is not complete, but does contain all the status codes you should expect using our subset of DL/I. You should see the DL/I status codes in the *IMS Version 9: Messages and Codes, Volume 1* if you should need a complete listing of all possible status codes.

Loading a HIDAM Database

When loading a HIDAM database initially, you must specify PROCPT=LS in the PCB. Also, the database records must be inserted in ascending root sequence, and the segment must be inserted in their hierarchical sequence.

Loading a Database with Logical Relationships

To establish the logical relationships during initial load of databases with logical relationships, DL/I provides a set of utility programs. These are necessary because the sequence in which the logical parent is loaded is normally not the same as the sequence in which the logical child is loaded. To cope with this, DL/I will automatically create a workflow whenever you load a database which contains the necessary information to update the pointers in the prefixes of the logically related segments.

Before doing so, the work file is sorted in physical database sequence with the prefix resolution utility (DFSURG10). This utility also checks for missing logical parents. Next, the segment prefixes are updated with the prefix update utility (DFSURGPO). After this, the database (s) are ready to use. The above database load, prefix resolution and update should be preceded by the Prereorganization utility (DFSURPRO). This utility generates a control data set to be used by database load, DFSURG10 and DFSURGP). Figure 70 on page 189 illustrates the process.

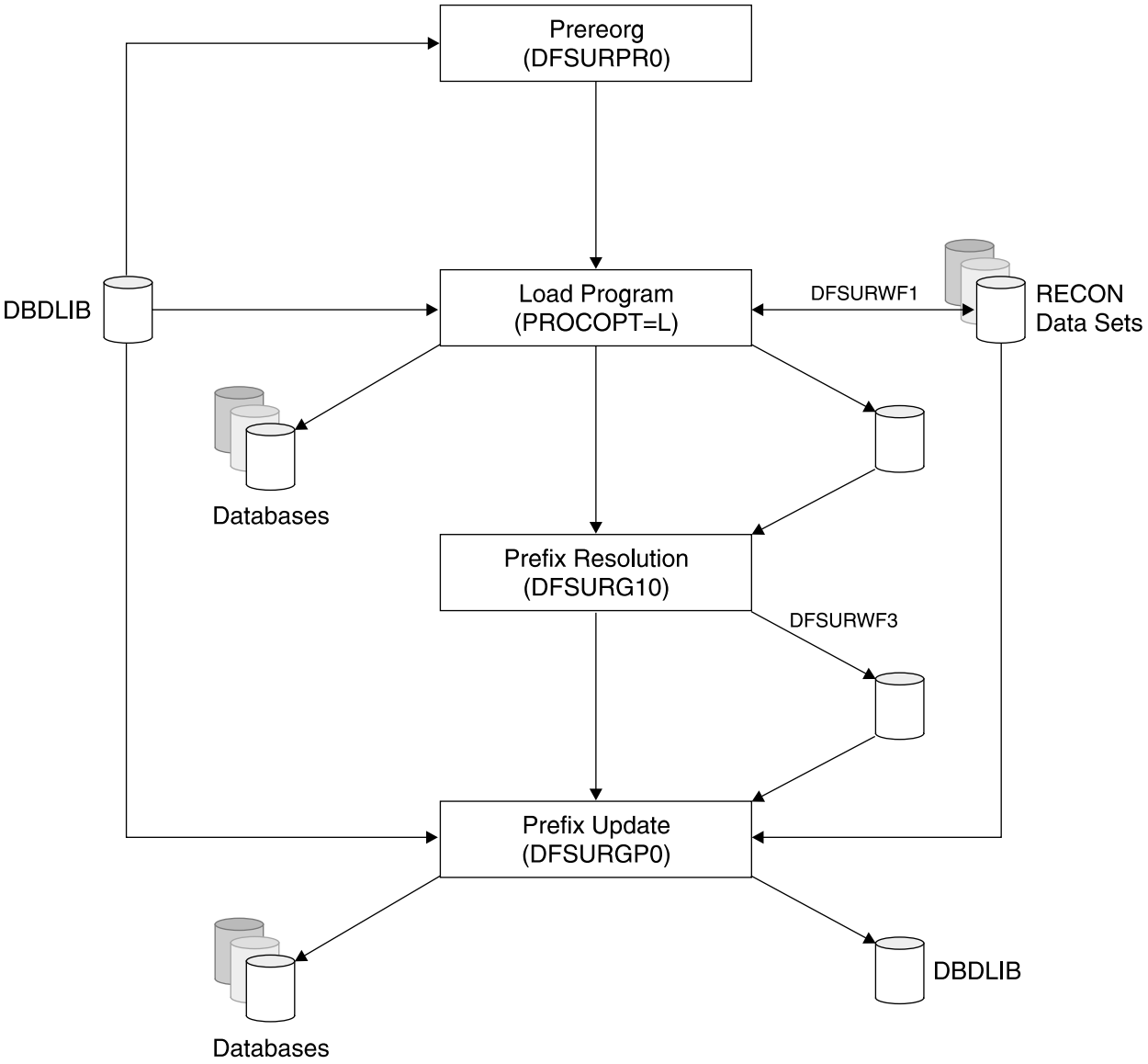

Figure 70. Overview of Loading a Database that has Logical Relationships

If both any of the databases involved in the logical relationship also has secondary indexes, then the process for loading a database with secondary indexes must be used as well. See Figure 72 on page 191 for an illustration of the complete process.

Notes:

1. You cannot use a logical DBD when initially loading a database (PROCOPT=L (S) in the PCB).
2. You must load all database involved in the logical relationship and pass the work files to the prefix resolution utility.

Loading a Database with Secondary Indexes

To load a database which has secondary indexes, the primary database must be uninitialized as shown in Figure 71 on page 190. IMS will extract the required information into the work file to build the secondary index database(s).

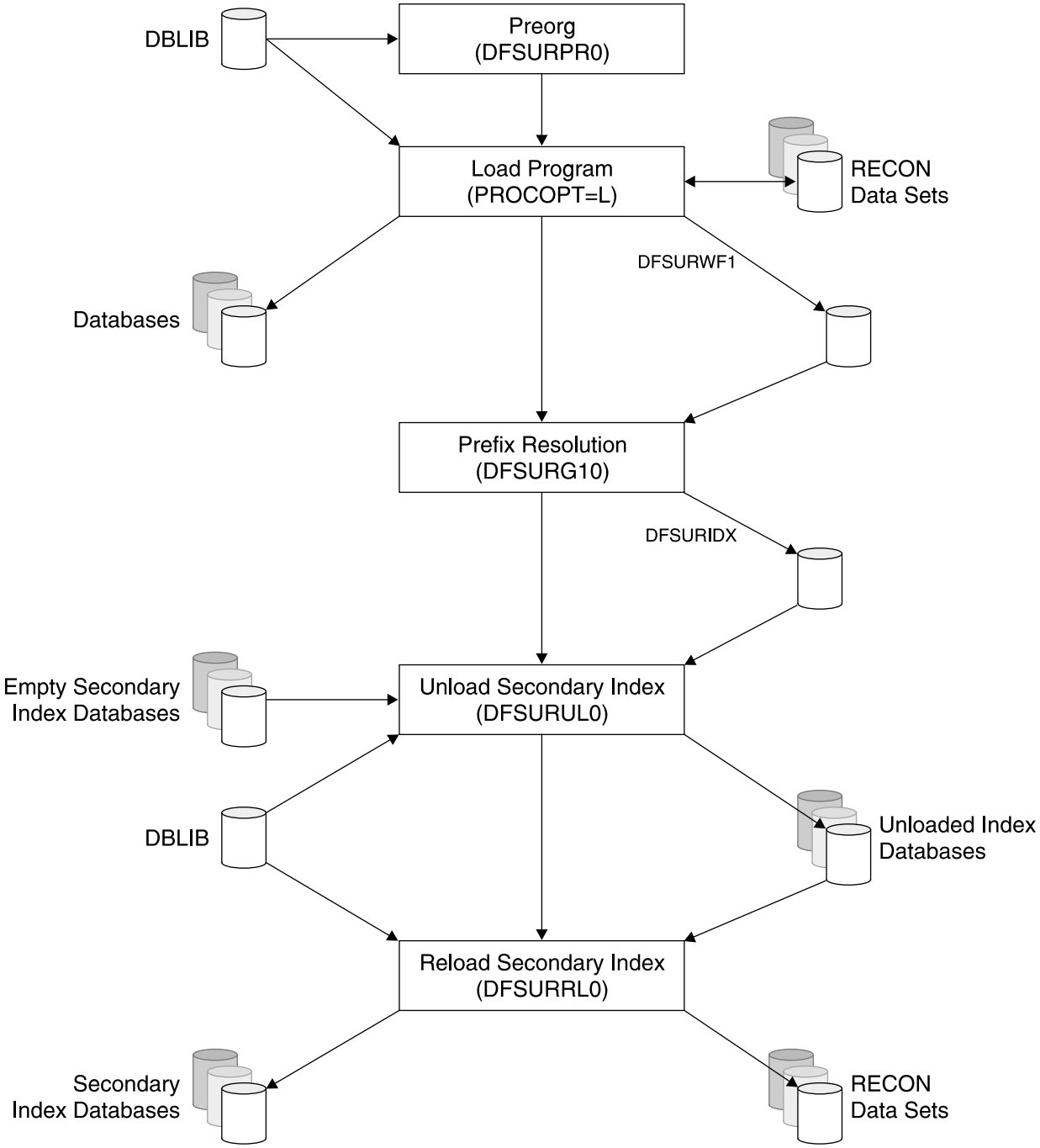

Figure 71. Overview of Loading a Database that has Secondary Indexes

Figure 72 on page 191 illustrates the process of loading a database that has logical relationships and secondary indexes.

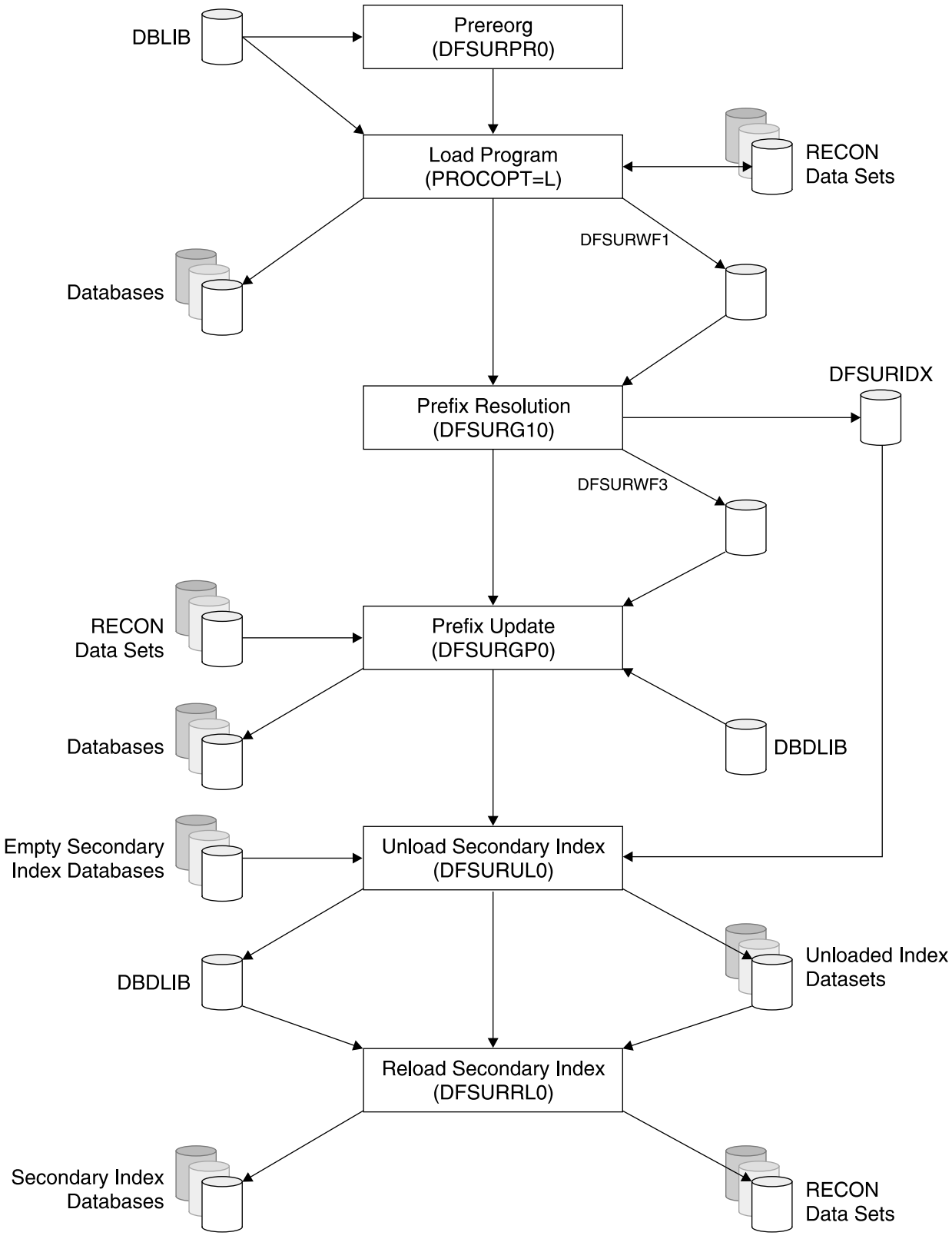

Figure 72. Overview of Loading a Database that has Logical Relationships and Secondary Indexes

Using Batch Checkpoint/Restart

The batch checkpoint/restart facility of DL/I allows long running programs to be restarted at an intermediate point in case of failure. At regular intervals (CHKP calls) during application program execution, DL/I saves on its log data set, designated working storage areas in the user's program, the position of GSAM databases, and the key feedback areas of non-GSAM databases

For each checkpoint, a checkpoint ID (message DFS681I) will be written to the z/OS system console and to the job system output.

At restart, the restart checkpoint ID is supplied in the PARM field of the EXEC statement of the job. DL/I will then reposition the GSAM databases and restore the designated program areas. This is accomplished with a special restart call (XRST) which must be the very first DL/I call in the program. At initial program execution, the XRST call identifies the potential program areas to be checkpointed by later CHKP calls.

To utilize the checkpoint/restart function of DL/I for batch programs, you should consider the following guidelines:

- All the data sets that the program uses must be DL/I databases. GSAM should be used for sequential input and output files, including SYSIN and SYSOUT. Any other file cannot be repositioned by DL/I and can result in duplicate or lost output.
- The GSAM output data sets should use DISP=(NEW,KEEP,KEEP) for the initial run and DISP=(OLD,KEEP,KEEP) at restart (s).
- SYSOUT should not be used directly. The output should be written to a GSAM file (as in 2) and be printed with the additional jobstep. IEBGENER can be used for this purpose.
- The first call issued to DL/I must be XRST call. Its format will be discussed later.
- The frequency of the checkpoint call is your choice. A basic recommendation is on checkpoint for every 50 to 500 update transactions. It is good practice to program for an easy adjustment of this frequency factor.
- After each checkpoint call, you must reposition yourself in the non-GSAM databases by issuing a get unique call for each of those databases. Repositioning of GSAM databases is done by DL/I, and you should proceed with a get next (input) or an insert (output) call.

The following sections discuss the restart call (see "Using the Restart Call") and the checkpoint call (see "Using the Checkpoint Call" on page 194).

Using the Restart Call

Upon receiving the restart call (XRST), DL/I checks whether a checkpoint ID has been supplied in the PARM field of the EXEC card or in the work area pointed to by the XRST call. If no ID has been supplied, a flag is set to trigger storing of repositioning data and user areas on subsequent CHKP calls (that is, DL/I assumes that this is the initial program execution, not a restart).

If the checkpoint at which restart is to occur has been supplied, the IMS batch restart routine reads backwards on the log defined in the //IMSLOGR DD card to locate the checkpoint records. User program areas are restored.

The GSAM databases active at the checkpoint are repositioned for sequential processing. Key feedback information is provided in the PCB for each database

active at the checkpoint. The user program must reposition itself on all non-GSAM databases, just as it must do after taking a checkpoint.

The format of the XRST call in COBOL is:

```
CALL 'CBITDLI'  using call-func,IOPCB-name, I/O-area-len,work-area
[,1st-area-len, 1st rea,...,nth-area-len,nth-area}.
```

The format of the XRST call in PL/I is:

```
CALL PLITDLI (parmcount,call-func,IOPCB-name. I/O-area-len,work-ar
[,1st-area-len,1st-area,...,nth-area-len,nth-area]):
```

The format of the XRST call in Assembler is:

```
CALL ASMTDLI,(call-func,IOPCB-name,I/O-area-len,work-area[,1st-area-len,
1st-area,...,nth-area-len,nth-rea]),
```

Where:

parmcount
Is the name of a binary fullword field containing the number of arguments following. PL/I only.

call-func
Is the name of a field which contains the call function ‘XRST’.

IOPCB-name
Is the name of the I/O PCB or the “dummy” I/O PCB supplied by the CMPAT option in PSEGEN (C1PCB in the sample programs).

I/O-area-len
Is the name of the length field of the largest I/O area used by the user program: must be a fullword.

work-area
Is the name of a 12-byte work area. This are should be set to blanks (X'40') before the call and tested on return. If the program is being started normally, the area will be unchanged. If the program is being restarted from checkpoint, the ID supplied by the user in that CHKP call and restart JCL will be placed in the first 8 bytes. If the user wishes to restart from a checkpoint using the method other than IMS Program Restart, he may use the XRST call to reposition GSAM databases by placing the checkpoint ID in this area before issuing the call. This ID is the 8-byte left-aligned, user supplied ID.

1st-area-len
Is the name of a field which contains the length of the first area to be restored. The field must be a fullword.

1st-area
Is the name of the first area to be restored.

nth-area-len
Is the name of a field which contains the length of the nth area to be restored (max n=7): must be a fullword. nth-area is the name of the nth area to be restored (max n=7).

Notes:

1. The number of areas specified on the XRST call must be equal to the maximum specified on any CHKP call.
2. The lengths of the areas specified on the XRST call must equal to or larger than the lengths of the corresponding (in sequential order) areas of any CHKP call.

3. The XRST call is issued only once and it must be the first request made to DL/I.
4. The only correct status code is bb: any other implies an error condition.
5. All “area-len” fields in PL/I must be defined as substructures. The name of the major structure should, however, be specified in the call.

Using the Checkpoint Call

When DL/I receives a CHKP call from a program which initially issued a XRST call, the following actions are taken:

- All database buffers modified by the program are written to DASD.
- A log record is written, specifying this ID to the OS/VS system console and job sysout.
- The user-specified areas (for example, application variables and control tables) are recorded on the DL/I log data set. They should be specified in the initial XRST call.
- The fully-qualified key of the last segment processed by the program on each DL/I database is recorded on the DL/I log data set.

The format of the CKPT call in COBOL is:

```
CALL 'OBLTDLI' using call-func,IOPCB-name, I/O-area-len,I/O=area
[,1st-area-len,1st-area,...,nth-area-len,nth-area]).
```

The format of the CKPT call in PL/I is:

```
CALL PLITDLI [parmcount, call-func,IOPCB-name,I/O-area-len, I/O-area
[,1st-area-len,1st-area,...,nth-area-len,nth-area]):
```

The format of the CKPT call in Assembler is:

```
CALL ASMTDLI, (call-func,IOPCB-name,I/O-area-len,I/O-area
[,1st-area-len,1st-area,...,nth-area-len,nth-area]):
```

Where:

parmcount
Is the name of a binary fullword field containing the number of arguments following. PL/I only.

call-func
Is the name of a field which contains the call function ‘CKPT’.

IOPCB-name
Is the name of the I/O PCB or the dummy I/O PCB in batch.

I/O-area-len
Is the name of the length field of the largest I/O area used by the application program: must be a fullword.

I/O-area
Is the name of the I/O area. The I/O area must contain the 8 byte checkpoint ID. This is used for operator or programmer communication and should consist of EBCDIC characters. In PI/I, this parameter should be specified as a pointer to a major structure, an array, or a character string.

The recommended format is MMMMnnnn where:

MMMM
Is the 4-character program identification.

nnnn Is the 4-character checkpoint sequence number, incremented at each CHKP call.

1st-area-len (optional)
Is the name of a field that contains the length of the first area to checkpoint: must be a fullword.

1st-area (optional)
Is the name of the first area to checkpoint.

nth-area-len (optional)
Is the name of the field that contains the length of the nth area to checkpoint (max n=7): must be a fullword.

nth-area (optional)
Is the name of the nth area to checkpoint (max n=7).

Notes:

1. The only correct status code in batch is bb: any other specifies an error situation.
2. Before restarting a program after failure, you always must first correct the failure and recover your databases. You must reestablish your position in all IMS database (except GSAM) after return from the checkpoint (that is, issue a get unique).
3. All “area-len” fields in PL/I must be defined as substructures see the example under note 5 of the XRST call.
4. Because the log tape is read forward during restart, the checkpoint ID must be unique for each checkpoint.

Chapter 19. Application Programming for the IMS Transaction Manager

This chapter, which deals with writing application programs in the IMS Transaction Manager environment, is divided into two major sections:

- "Application Program Processing"
- "Transaction Manager Application Design" on page 201

Application Program Processing

Basically, the MPP processing can be divided into five phases. Figure 73 on page 198 illustrates these phases and the list that follows Figure 73 on page 198 describes the phases.

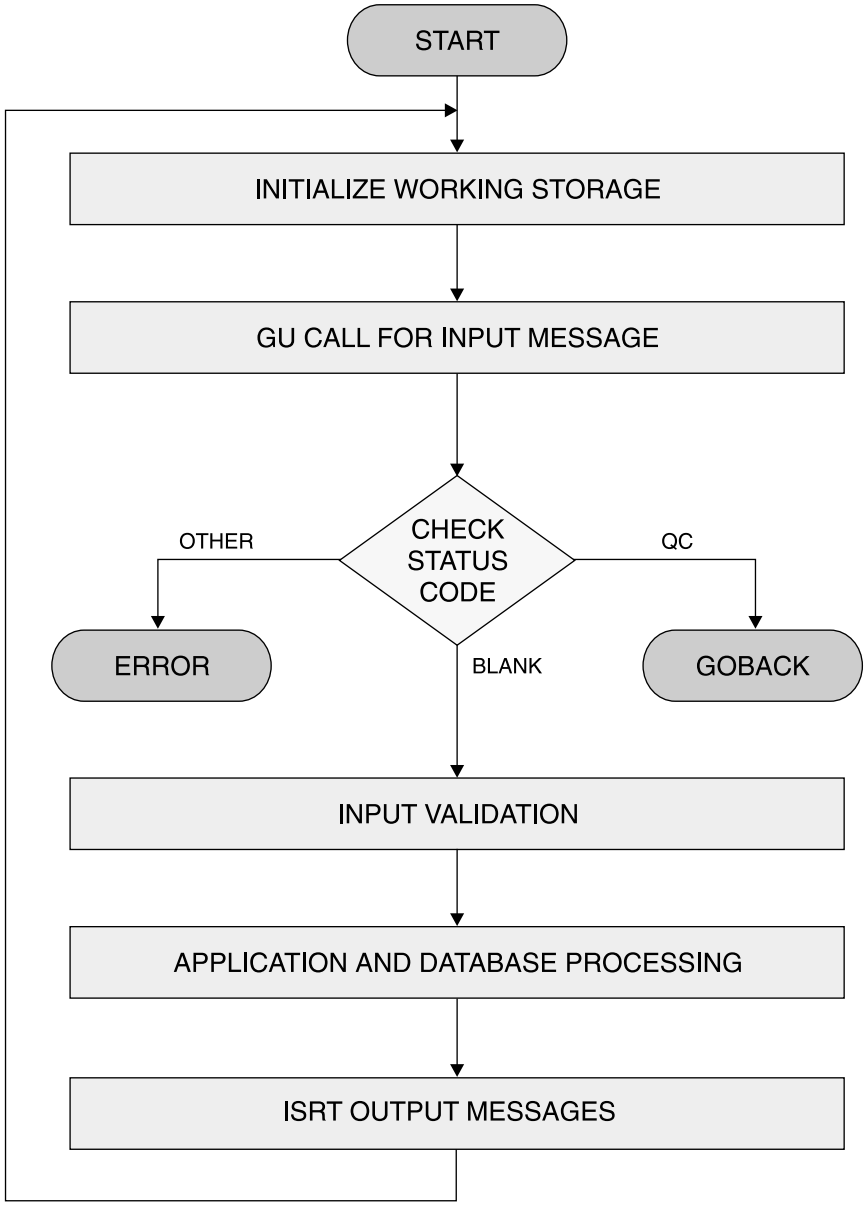

Figure 73. General MPP Structure and Flow

The following are the five phases of the flow of an MPP:

1. **Initialization**

   Initialization is the clearing of working storage, which may contain data left-over by the processing of a message from another terminal.

2. **Retrieval of the scratch pad area (SPA) and input message**

   The application issues a call to IMS TM to retrieve a message from the message queue. The application retrieves the SPA first if the transaction is conversational.

3. **Input syntax check**

   IMS TM checks the syntax of the input message. All checks which can be done without accessing the database, including a consistency check with the status of the conversation as maintained in the SPA.

4. **Database processing**

Database processing is performed preferably in one phase. This means that the retrieval of a database segment is immediately followed by its update. Compare this to an initial retrieve of all required segments followed by a second retrieve and then update.

5. **Output processing**

   The output message is built and inserted together with the SPA (only for conversational transactions).

After finishing the processing of one input message, the program should go back to step 1 and request a new input message. If there are no more input messages, IMS will return a status code indicating that. At that time, the MPP must return control to IMS.

Role of the PSB

The program specification block (PSB) for an MPP or a BMP contains, besides database PCBs, one or more PCB (s) for logical terminal linkage. The very first PCB always identifies the originating logical terminal. This PCB must be referenced in the get unique and get next message calls. It must also be used when inserting output messages to that LTERM. In addition, one or more alternate output PCBs can be defined. Their LTERM destinations can be defined in the PCBs or set dynamically with change destination calls.

DL/I Message Calls

The same DL/I language interface that is used for the access of databases is used to access the message queues.

The principal DL/I message calls are:

GU (get unique)
: This call must be used to retrieve the first, or only, segment of the input message.

GN (get next)
: This call must be used to retrieve second and subsequent message segments.

ISRT (insert)
: This call must be used to insert an output message segment into the output message queue. Note: these output message(s) will not be sent until the MPP terminates or requests another input message via a get unique.

CHNG (change destination)
: This call can be used to set the output destination for subsequent insert calls.

For a detailed description of the DL/I database calls and guidelines for their use, see Chapter 18, “Application Programming for the IMS Database Manager,” on page 165.

Conversational Processing

A transaction code can be defined as belonging to a conversational transaction during IMS system definition. If so, an application program that processes that transaction, can interrelate messages from a given terminal. The vehicle to accomplish this is the scratch pad area (SPA). A unique SPA is created for each physical terminal which starts a conversational transaction.

Each time an input message is entered from a physical terminal in conversational mode, its SPA is presented to the application program as the first message segment (the actual input being the second segment). Before terminating or retrieving another message (from another terminal), the program must return the SPA to IMS with a message ISRT call.

The first time a SPA is presented to the application program when a conversational transaction is started from a terminal, IMS will format the SPA with binary zeroes (X'00'). If the program wishes to terminate the conversation, it can indicate this by inserting the SPA with a blank transaction code.

Output Message Processing

As soon as an application reaches a synchronization point, its output messages in the message queue become eligible for output processing. A synchronization point is reached whenever the application program terminates or requests a new message from the input queue via a GU call.

In general, output messages are processed by message format service before they are transmitted via the telecommunications access method.

Different output queues can exist for a given LTERM, depending on the message origin. They are, in transmission priority:

1. Response messages, that is, messages generated as a direct response (same PCB) to an input message from this terminal.
2. Command responses.
3. Alternate output messages, messages generated via an alternate PCB.

Application Program Termination

The following sections discuss terminating your application program.

Normal Termination

The program returns control to IMS TM when it finishes processing. In a BMP, DLI, or DBB processing region, your program can set the return code and pass it to the next step in the job. If your program does not use the return code in this way, it is a good idea to set it to zero as a programming convention.

Restriction: MPPs cannot pass return codes.

Abnormal Termination

Upon abnormal termination of a message or batch-message processing application program for other reasons than deadlock resolution, internal commands are issued to prevent rescheduling. These commands are the equivalent of a /STOP command. They prevent continued use of the program and the transaction code in process at the time of abnormal termination. The master terminal operator can restart either or both stopped resources.

At the time abnormal termination occurs, a message is used to the master terminal and to the input terminal that identifies the application program, transaction code, and input terminal. It also contains the system and user completion codes. in addition, the first segment of the input transaction, in process by the application at abnormal termination, is displayed on the master terminal.

The database changes of a failing program are dynamically backed-out. Also, its output messages inserted in the message queue since the last synchronization point are cancelled.

Logging and Checkpoint/Restart Processing

To ensure the integrity of its databases and message processing IMS uses logging and checkpoint/restart. In case of system failure, either software or hardware, IMS can be restarted. This restart includes the repositioning of users' terminals, transactions, and databases.

Logging

During IMS execution, all information necessary to restart the system in the event of hardware or software failure, is recorded on a online log data sets (OLDS).

The following critical system information is recorded on the OLDS:

- The receipt of an input message in the input queue
- The start of an MPP or BMP
- The receipt of a message by the MPP for processing
- Before and after images of database updates by the MPP or BMP
- The insert of a message into the queue by the MPP
- The termination of an MPP or BMP
- The successful receipt of an output message by the terminal

In addition to the above logging, all previous database record unchanged data is written to the log data set. This log information is only used for dynamic back-out processing of a failing MPP/BMP. as soon as the MPP/BMP reaches a synchronization point, the dynamic log information of this program is discarded.

Emergency Restart

In case of failure, IMS is restarted with the log data set active at the time of failure. Restart processing will back-out the database changes of incomplete MPPs and BMPs. The output messages inserted by these incomplete MPPs will be deleted.

After back-out, the input messages are re-enqueued, the MPPs restarted, and the pending output messages are re-transmitted. If a BMP was active at the time of failure, it must be resubmitted by using a z/OS job. If the BMP uses the XRST/CHKP calls, it must be restarted from its last successful checkpoint. In this way, missing or inconsistent output is avoided.

Transaction Manager Application Design

We will distinguish between the following areas in the IMS database/data communication design process:

- Program design
- Message Format Service (MFS) design
- Database design

In "Online Program Design" on page 204, we will concentrate on the design of message processing programs (MPPs).

The MFS design will discuss the 3270 screen layouts and operator interaction.

Although we will cover each of the above areas in separate sections, it should be realized that they are largely dependent upon each other. Therefore, an overall system design must be performed initially and an overall system review must follow the design phase of each section

Online Transaction Processing Concepts

In an IMS online environment, one can view a transaction from three different points:

- The application, that is, its processing characteristics and database accesses.
- The terminal user.
- The IMS system.

Each of the above constitutes a set of characteristics. These characteristics are described in the following sections.

Application Characteristics

From an application point of view, we can identify:

- Data collection with no previous database access). This is not a typical IMS application but can be part of an IMS application system.
- Update. This normally involves database reference and the subsequent updating of the database. This is the environment of most IMS applications.

In typical IMS multi-application environment, the above characteristics are often combined. However, a single transaction normally has only one of the above characteristics.

Terminal User Characteristics

From the terminal user's point of view, we distinguish:

- Single-interaction transactions.
- Multi-interaction transactions.

The single interaction transaction does not impose any dependency between any input message and its corresponding output, and the next input message.

The multi-interaction transaction constitutes a dialogue between the terminal and the message processing program (s). Both the terminal user and the message processing rely on a previous interaction for the interpretation/processing of a subsequent interaction.

IMS Characteristics

From the IMS system point of view, we distinguish:

- Non-response transactions
- Response transactions
- Conversational transactions

These IMS transaction characteristics are defined for each transaction during IMS system definition.

With non-response transactions, IMS accepts multiple input messages (each being a transaction) from a terminal without a need for the terminal to first accept the corresponding output message, if any. These non-response transactions will not be further considered in our sample.

With response transactions, IMS will not accept further transaction input from the terminal before the corresponding output message is sent and interpreted by the user.

Conversational transactions are similar to response transactions, in that no other transactions can be entered from the terminal until the terminal is out of conversational mode. With response mode, the terminal is locked until a reply is received. This is not the case for conversational mode. Another difference is that for conversation transactions, IMS provides a unique scratch pad area (SPA) for each user to store vital information across successive input messages.

Transaction Response Time Considerations

In addition to the above characteristics, the transaction response time is often an important factor in the design of online systems. The response time is the elapsed time between the entering of an input message by the terminal operator and the receipt of the corresponding output message at the terminal.

Two main factors, in general, constitute the response time:

- The telecommunication transmission time, which is dependent on such factors as:
  - Terminal network configuration
  - Data communication access method and data communication line procedure
  - Amount of data transmitted, both input and output
  - Data communication line utilization
- The internal IMS processing time, which is mainly determined by the MPP service time. The MPP service time is the elapsed time required for the processing of the transaction in the MPP region.

Choosing the Correct Characteristics

Each transaction in IMS can and should be categorized by one characteristic of each of the previously discussed three sets.

Some combinations of characteristics are more likely to occur than others, but all of them are valid.

In general, it is the designer's choice as to which combination is attributed to a given transaction. Therefore, it is essential that this selection of characteristics is a deliberate part of the design process, rather than determined after implementation.

Following are some examples:

- Assume an inquiry for the customer name and address with the customer number as input. The most straightforward way to implement this is clearly a non-conversational response-type transaction.
- The entry of new customer orders could be done by a single response transaction. The order number, customer number, detail information, part number, quantity etc., could all be entered at the same time. The order would be processed completely with one interaction. This is most efficient for the system, but it may be cumbersome for the terminal user because she or he has to re-enter the complete order in the case of a an error.

Quite often, different solutions are available for a single application. Which one to choose should be based on a trade-off between system cost, functions, and user convenience. The following sections will highlight this for the different design areas.

Online Program Design

This area is second in importance to database design. We will limit the discussion of this broad topic to the typical IMS environment. We will first discuss a number of considerations so that you become familiar with them. Next, we will discuss the design of the two online sample programs. You will notice that some discussions are quite arbitrary and may not have to be adjusted for your own environment. Do remember, however, that our prime objective is to make you aware of the factors which influence these decisions.

Single versus Multiple Passes

A transaction can be handled with one interaction or pass, or with two or more passes (a pass is one message in and one message out). Each pass bears a certain cost in line time and in IMS and MPP processing time. So, in general, you should use as few passes as possible. Whenever possible you should use the current output screen to enter the next input. This is generally easy to accomplish for inquiry transactions, where the lower part of the screen can be used for input and the upper part for output. (See “Basic Screen Design” on page 205).

For update transactions, the choice is more difficult. The basic alternatives are:

One-pass update

After input validation, the database updates are all performed in the same pass. This is the most efficient way from the system point of view. However, correcting errors after the update confirmation is received on the terminal requires additional passes or re-entering of data. An evaluation n of the expected error rate is required.

Two-pass update

On the first pass, the input is validated, including database access. A status message is sent to the terminal. If the terminal operator agrees, the database will be updated in the second pass. With this approach, making corrections is generally much simpler, especially when a scratch pad area is used. However, the database is accessed twice.

You should realize, that, except for the SPA, no correlation exists between successive interactions from a terminal. So, the database can be updated by somebody else and the MPP may process a message for another terminal between two successive passes.

Multi-pass update

In this case, each pass does a partial database update. The status of the database and screen is maintained in the SPA. This approach should only be taken for complex transactions. Also, remember that the terminal operator experiences response times for each interaction. You also must consider the impact on database integrity. IMS will only back-out the database changes of the current interaction in the case of project or system failure.

Notes:

1. IMS emergency restart with a complete log data set will reposition the conversation. The terminal operator can proceed from the point where he or she was at the time of failure.
2. When a conversational application program terminates abnormally, only the last interaction is backed out.

The application must reposition the conversation after correction. For complex situations, IMS provides an abnormal transaction exit routine. This is not covered in our subset.

Conversational versus Non-Conversational

Conversational transactions are generally more expensive in terms of system cost than non-conversational ones. However, they give better terminal operator service. You should only use conversational transactions when you really need them. Also, with the proper use of MFS, the terminal operator procedures sometimes can be enhanced to almost the level of conversational processing.

Transaction or Program Grouping

It is the designer's choice how much application function will be implemented by one transaction and/or program. The following considerations apply:

- Inquiry-only. transactions should be simple transactions. These should be normally implemented as non-conversational transactions. Also, they can be defined as "non-recoverable inquiry-only"". If in addition, the associated MPPs specify PROCOPT= GO in all their database PCB's, no dynamic enqueue and logging will be done for these transactions.
- Limited-function MPPs are smaller and easier to maintain. However, a very large number of MPPs costs more in terms of IMS resources (control blocks and path lengths).
- Transactions with a long MPP service time (many database accesses). should be handled by separate programs.

Note: IMS provides a program-to-program message switch capability. This is not part of our subset. With this facility, you can split the transaction processing in two (or more) phases. The first (foreground) MPP does the checking and switches a message (and, optionally, the SPA) to a (background) MPP in a lower priority partition which performs the lengthy part of the transaction processing. In this way the foreground MPP is more readily available for servicing other terminals. Also, if no immediate response is required from the background MPP and the SPA is not switched, the terminal is more readily available for entering another transaction.

Basic Screen Design

Generally, a screen can be divided into five areas, top to bottom:

1. Primary output area, contains general, fixed information for the current transaction. The fields in this area should generally be protected.
2. Detail input/output area, used to enter and/or display the more variable part of the transaction data. Accepted fields should be protected (under program control): fields in error can be displayed with high intensity and unprotected to allow for corrections.
3. MPP error message area. In general, one line is sufficient. This can be the same line as 5 below.
4. Primary input, that is requested action and/or transaction code for next input, and primary database access information.
5. System message field, used by IMS to display system messages and by the terminal operator to enter commands.

For readability, the above areas should be separated by at least one blank line. The above screen layout is a general one, and should be evaluated for each individual application. IBM recommends that you develop a general screen layout and set of formats to be used by incidental programs and programs in their initial test.

This can significantly reduce the number of format blocks needed and maintenance. In any case, installation standards should be defined for a multi-application environment.

MFS Subset Restrictions

1. The maximum output length of a message segment is 1388 bytes: this is related to our long message record length of 1500 bytes.
2. A format is designated for one screen size. This can be later changed via additional MFS statements to support both screens and other devices with the same set of format blocks. A 1920 character format can be displayed on the top part of a 2560 or 3440 character display, and 480 character format can be displayed on the top of a 960 character display.
3. A segment is one physical page, which is one logical page.

General Screen Layout Guidelines

The following performance guidelines should be observed when making screen layouts:

- Avoid full-format operations. IMS knows what format is on the screen. So if the format for the current output is the same as the one on the screen, IMS need not retransmit all the literals and unused fields.
- Avoid unused fields, for example, undefined areas on the screen. Use the attribute byte (non-displayed) of the next field as a delimiter, or expand a literal with blanks. Each unused field causes additional control characters (5) to be transmitted across the line during a full-format operation.

  Note: This has to be weighed against user convenience. For example, our sample customer name inquiry format does not have consecutive fields but it is user convenient. Also, this application rarely needs a new format so we are not so much concerned with unused fields.

Including the Transaction Code in the Format

IMS requires a transaction code as the first part of an input message. With MFS, this transaction code can be defined as a literal. In doing so, the terminal operator always enters data on a preformatted screen. The initial format is retrieved with the /FORMAT command.

To allow for multiple transaction codes on one format, part of the transaction code can be defined as a literal in the MID. The rest of the transaction code can then be entered via a DFLD. This method is very convenient for the terminal operator because the actual transaction codes are not of his concern. Any example of such a procedure is shown in our sample customer order entry application.

Miscellaneous Design Considerations

The following design considerations should also be noted:

- The conversation will be terminated (insert blank transaction code in SPA) after each successful order entry. This is transparent to the terminal operator, because the output format is linked to a MID which contains the transaction code, so the operator need not re-enter it.
- Each output message should contain all the data (except the MOD-defined literals) to be displayed. You should never rely on already existing data on the screen, because a clear or (re) start operation may have destroyed it.
- Using secondary indexing can significantly increase the accessibility of online databases. Therefore, a wider use of this facility is discussed in “Secondary Indexing” on page 48.

Chapter 20. The IMS Message Format Service

The chapter contains an overview of the Message Format Service (MFS) function of IMS. MFS describes the screen input and output interaction with IMS online programs.

The sections in this chapter are:

Overview of MFS

Through the message format service (MFS), a comprehensive facility is provided for IMS users of 3270 and other terminals/devices. MFS allows application programmers to deal with simple logical messages instead of device dependent data. This simplifies application development. The same application program may deal with different device types using a single set of editing logic while device input and output are varied to suit a specific device. The presentation of data on the device or operator input may be changed without changing the application program.

Full paging capability is provided for display devices. This allows the application program to write a large amount of data that will be divided into multiple screens for display on the terminal. The capability to page forward and backward to different screens within the message is provided for the terminal operator. The conceptual view of the formatting operations for messages originating from or going to an MFS-supported device is shown in Figure 74.

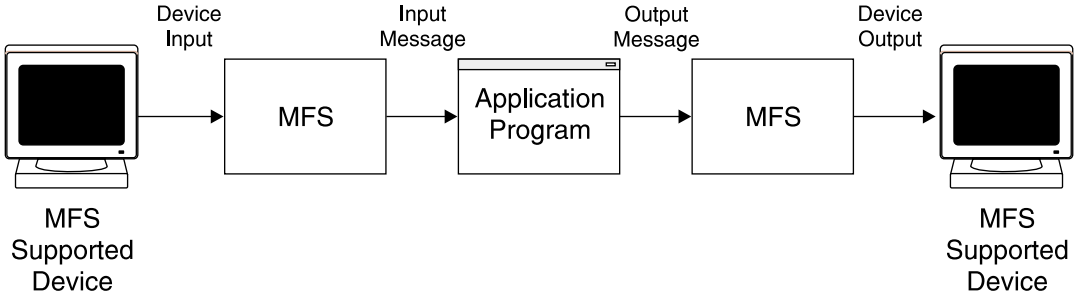

Figure 74. Message Formatting Using MFS

MFS has three major components:

- MFS Language utility
- MFS pool manager
- MFS editor

The MFS language utility is executed offline to generate control blocks and place them in a format control block data set named IMS.FORMAT. The control blocks describe the message formatting that is to take place during message input or

output operations. They are generated according to a set of utility control statements. There are four types of format control blocks:

- Message input descriptor (MID)
- Message output descriptor (MOD)
- Device input format (DIF)
- Device output format (DOF)

The MID and MOD blocks relate to application program input and output message segment formats, and the DIF and DOF blocks relate to terminal I/O formats. The MID and DIF blocks control the formatting of input messages, while the MOD and DOF blocks control output message formatting.

Notes:

1. The DIF and the DOF control blocks are generated as a result of the format (FMT) statement.
2. The MID and the MOD are generated as a result of the various message (MSG) statements.
3. The initial formatting of a 3270 display is done by issuing the `/FORMAT` *`modname`* command. This will format the screen with the specified MOD, as if a null message was sent.

Figure 75 on page 209 provides an overview of the MFS operations.

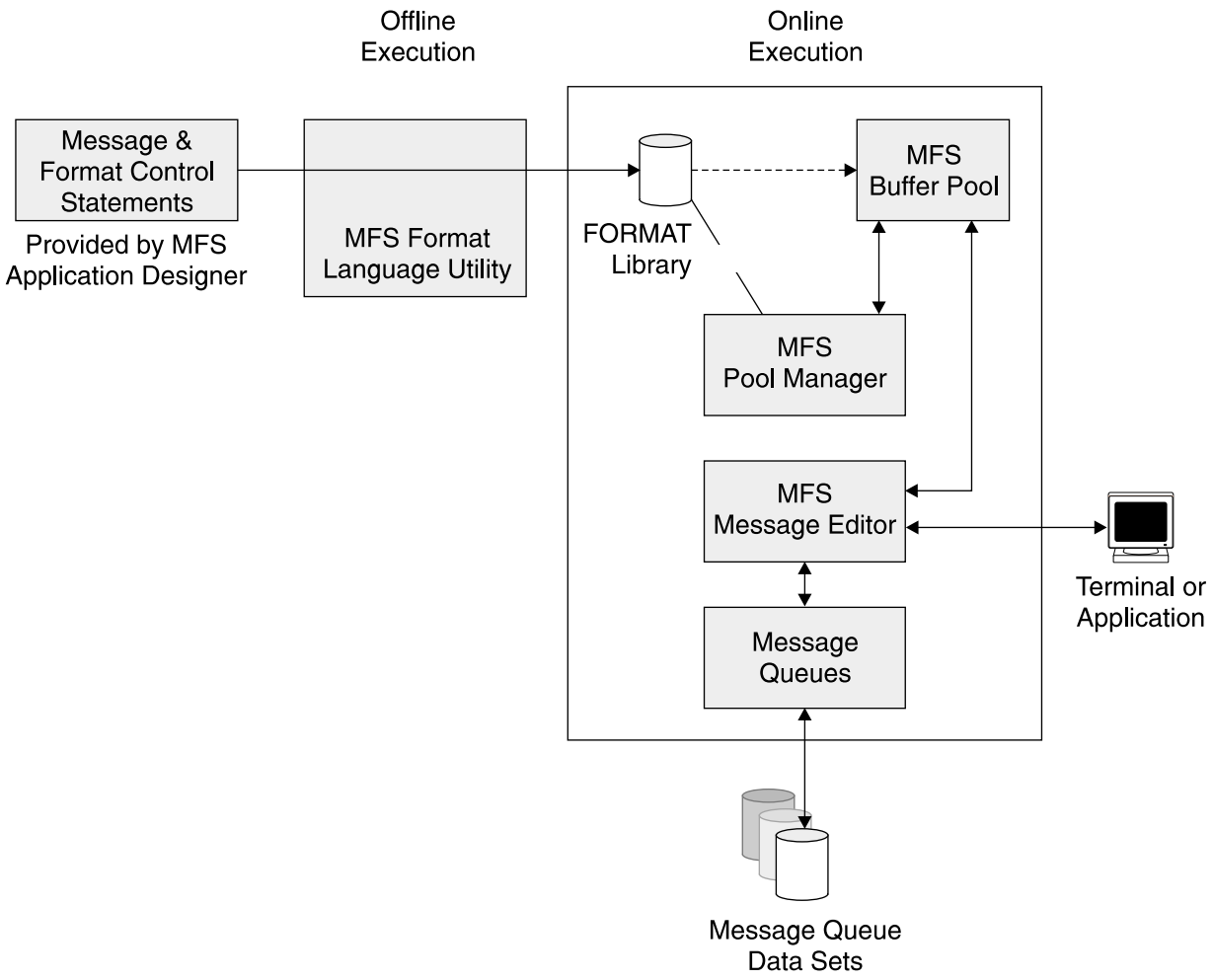

Figure 75. Overview of Message Format Service Functions

MFS and 3270 Devices

The IMS Message Format Service (MFS), described in “Overview of MFS” on page 207, is always used to format data transmitted between IMS and the devices of the 3270 information display system. MFS provides a high level of device independence for the application programmers and a means for the application system designer to make full use of the 3270 device capabilities in terminal operations. Although our subset only considers the 3270 devices, its use of MFS is such that it is open-ended to the use of other MFS supported terminals when required.

Relationships between MFS Control Blocks

Several levels of linkage exist between MFS control blocks, as described in the following sections.

MFS Control Block Chaining

Figure 76 shows the highest-level linkage, that of chained control blocks.

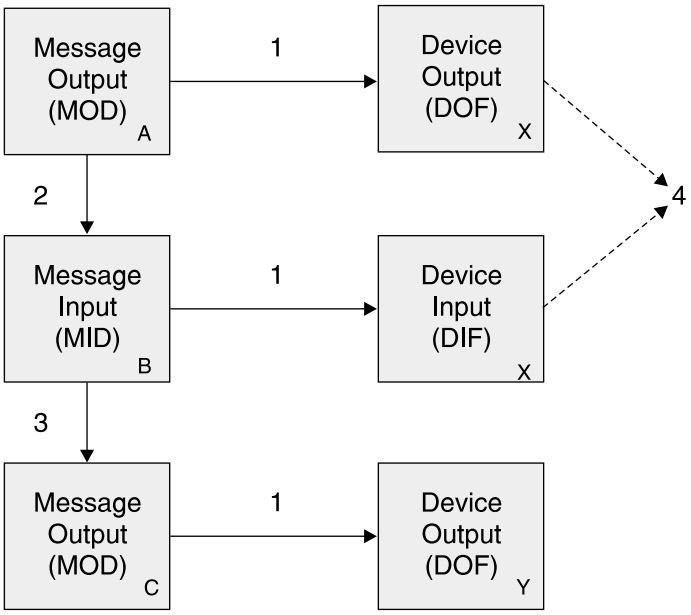

Figure 76. Chained Control Block Linkage

Legend:

1. This linkage must exist
2. If the linkage does not exist, device input data from 3270 devices is not processed by MFS. It is always used in our subset.
3. This linkage is provided for application program convenience. It provides a MOD name to be used by IMS if the application program does not provide a name via the format name option of the insert call. The default MOD, DFSMO2, will be used if none is specified at all, or if the input is a message switch to an MFS-supported terminal.
4. The user-provided names for the DOF and DIF used in one output/input sequence are normally the same. The MFS language utility alters the internal name for the DIF to allow the MFS pool manger to distinguish between the DOF and DIF.

The direction of the linkage allows many message descriptions to use the same device format if desired. One common device format can be used for several application programs whose output and input message formats, as seen at the application program interface, are quite different.

Linkage Between Device Fields and Message Fields

Figure 77 on page 211 shows the second level of linkage, that between message fields and device fields. The arrows show the direction of reference in the MFS language utility control statements, not the direction of data flow.

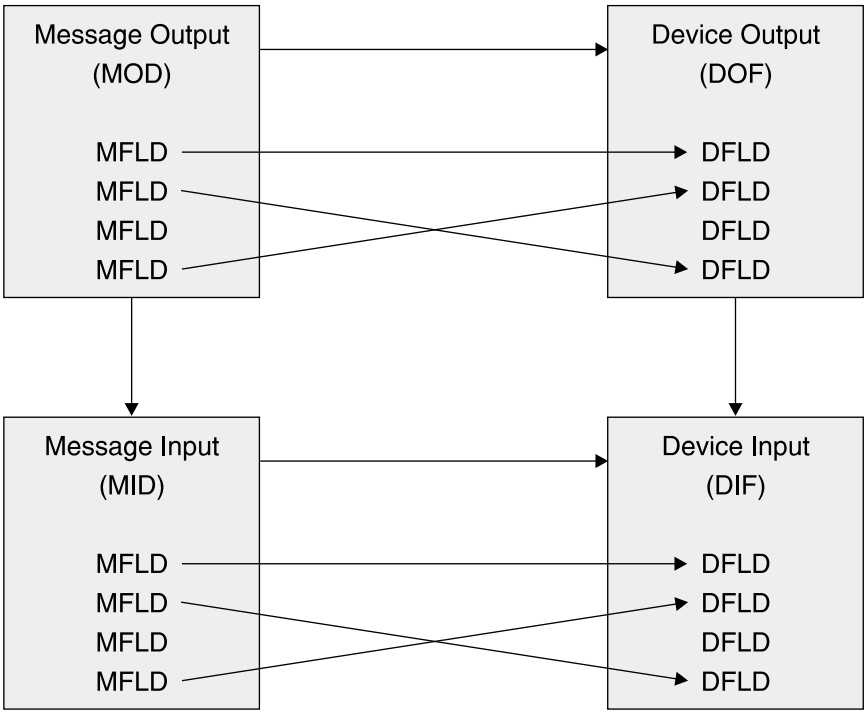

Figure 77. Linkage Between Message Fields and Device Fields

References to device fields by message fields need not be in any particular sequence. An MFLD need not see any DFLD, in which case it simply defines space in the application program segment to be ignored if the MFLD is for output, and padded if the MFLD is for input. Device fields need not be referenced by message fields, in which case they are established on the device, but no output data from the output message is transmitted to them. Device input data is ignored if the DFLD is not referenced by the input MFLD.

Linkage Between Logical Pages and Device Pages

Figure 78 shows a third level of linkage, one which exists between the LPAGE and the DPAGE.

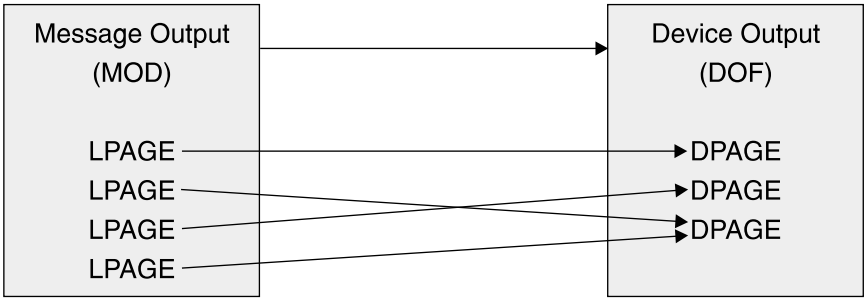

Figure 78. LPAGE - DPAGE Linkage

The LPAGE in the MOD must see a DPAGE in the DOF. However, all DPAGEs need not be referred to from a given MOD.

Because we will always have single segment input in our subset, the defined MFLDs in the MID can refer to DFLDs in any DPAGE. But input data for any given input message from the device is limited to fields defined in a single DPAGE.

Message Description Linkage

Figure 79 shows a fourth level of linkage. It is optionally available to allow selection of the MID based on which MOD LPAGE is displayed when input data is received from the device.

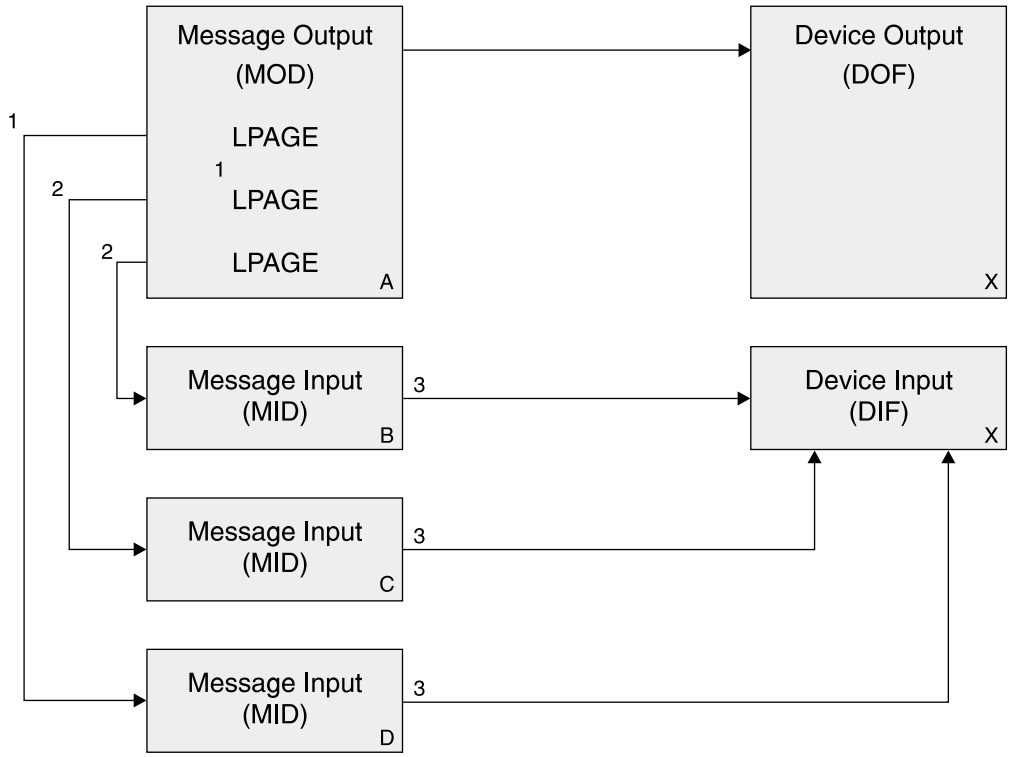

Figure 79. Optional Message Description Linkage

Legend:

1. The next MID name provided with the MSG statement is used if no name is provided with the current LPAGE.
2. If the next MID name is provided with the current LPAGE, input will be processed using this page.
3. For 3270 devices, all MIDs must refer to the same DIF. This is the same user-provided name used to refer to the DOF when the MOD was defined.

3270 Device Considerations Relative to Control Block Linkage

Since output to 3270 display devices establishes fields on the device using hardware capabilities, and field locations cannot be changed by the operator, special linkage restrictions exist. Because formatted input can only occur from a screen formatted by output, the LPAGE and physical page description used for formatting input is always the same as that used to format the previous output. The MFS language utility enforces this restriction by ensuring that the format name used for input editing is the same as the format name used for the previous output editing. Furthermore, if the DIF corresponding to the previous DOF cannot be fetched during online processing, an error message is sent to the 3270 display.

MFS Functions

The following sections contain a description of the basic MFS functions.

- “Input Message Formatting”
- “Output Message Formatting” on page 214
- “MFS Formats Supplied by IBM” on page 218

Input Message Formatting

All device input data received by IMS is edited before being passed to an application program. The editing is performed by either IMS basic edit or MFS. It tells how the use of MFS is determined and how, when MFS is used, the desired message format is established based on the contents of two MFS control blocks — the device input format (DIF) and the message input descriptor (MID).

All 3270 devices included in an IMS system use MFS. The 3270s always operate in formatted mode except when first powered on, after the CLEAR key has been pressed, or when the MOD used to process an output message does not name a MID to be used for the next input data. While in unformatted mode, you can still enter commands and transactions, but they will not be formatted by MFS.

Input Data Formatting Using MFS

Input data from terminals in formatted mode is formatted based on the contents of two MFS control blocks, the MID and the DIF. The MID defines how the data should be formatted for presentation to the application program and points to the DIF associated with the input device. See Figure 80.

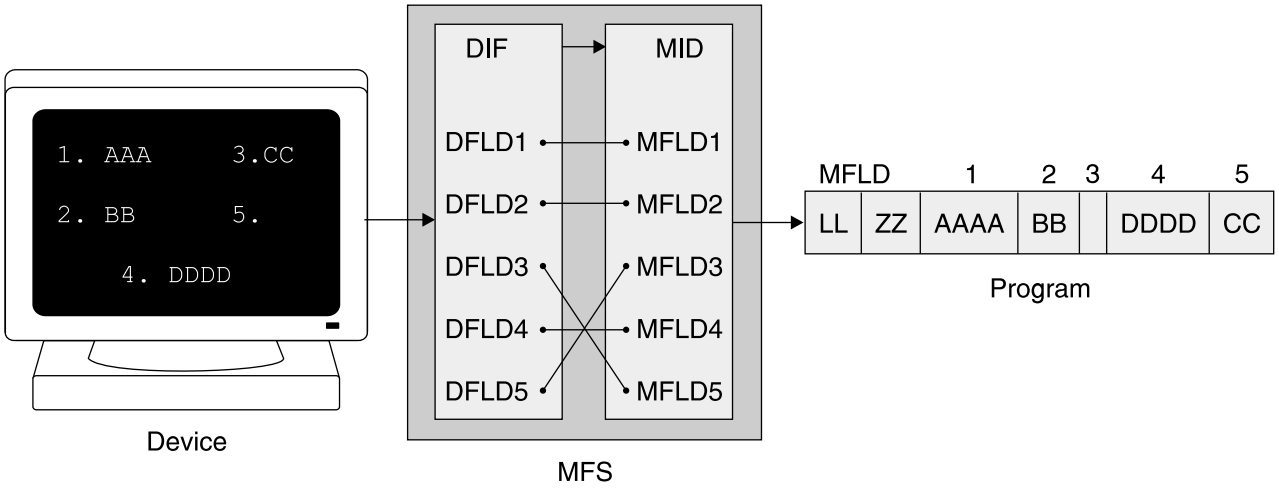

Figure 80. MFS Input Formatting

The MID contains a list of message descriptor fields (MFLDs) which define the layout of the input segment as is to be seen by an application program. The DIF contains a list of device descriptor fields (DFLDs) which define what data is to be expected from which part of the device (that is, the location on the screen). MFS maps the data of the DFLDs into the corresponding MFLDs. The application program is largely device independent because different physical inputs can be mapped into the same input segment.

MFLD statements are to define:

- The device fields (DFLDs) defined in the DIF which contents will be included in the message presented to the application program.
- Constants, defined as literals to be included in the message: a common use of literals is to specify the transaction code.

In addition, the MFLD statement defines:

- The length of the field expected by the application program
- Left or right justification and the fill character to be used for padding the data received from the device.
- A 'nodata' literal for the MFLD if the corresponding DFLD does not contain any input data.

It should be noted that all message fields as defined by MFLD statements will be presented to the application program in our subset. Furthermore, there will always be only one input message segment, except for conversational transaction, in which case the first segment presented to the program is the SPA. The SPA is never processed by MFS, however.

Input Message Field Attribute Data

Sometimes input messages are simply updated by an application program and returned to the device. In such a case, it may simplify message definition layouts in the MPP if the attribute data bytes are defined in the message input descriptor as well as the message output descriptor.

Non-literal input message fields can be defined to allow for 2 bytes of attribute data. When a field is so defined, MFS will reserve the first 2 bytes of the field for attribute data to be filled in by the application program when preparing an output message. In this way, the same program area can be conveniently used for both input and output messages. When attribute space is specified, the specified field length must include the 2 attribute bytes.

IMS Passwords

If the input data is for a password protected transaction, a device field should be designated for the password. The device field in which the operator keys in the password will not be displayed on the screen.

Output Message Formatting

All output messages for 3270 devices are processed by MFS in a way similar to input.

Output Data Formatting Using MFS

All MFS output formatting is based on the contents of two MFS control blocks -- the message output descriptor (MOD) and the device output format (DOF). See Figure 81 on page 215, the MOD defines output message content and optionally, literal data to be considered part of the output message. Message fields ((MFLDs) refer to device field locations via device field (DFLD) definitions in the DOF. The DOF specifies the use of hardware features, device field locations and attributes, and constant data considered part of the format.

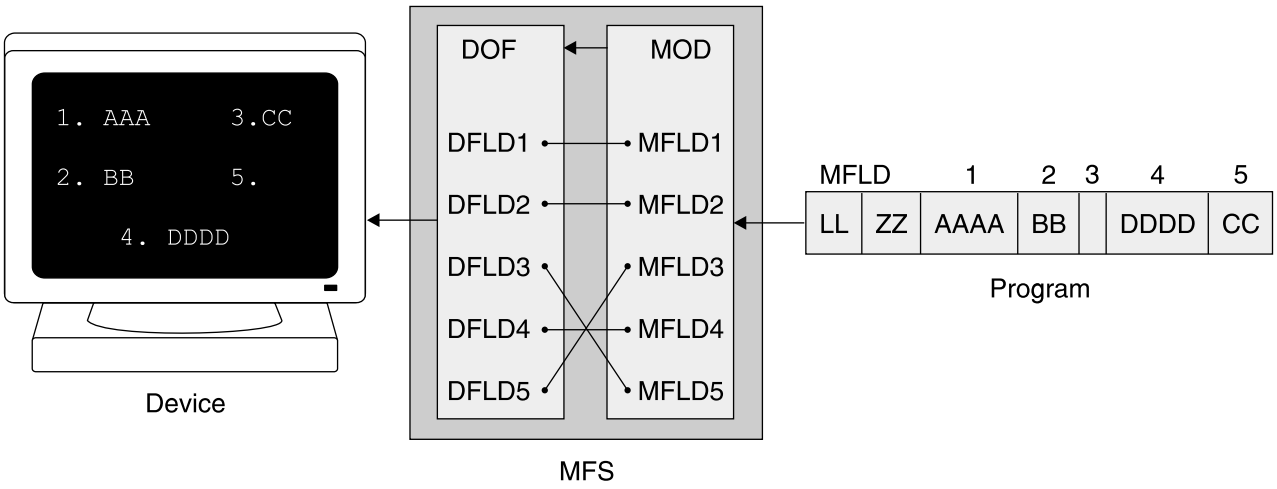

Figure 81. MFS Output Formatting

The layout of the output message segment to be received by MFS from the program is defined by a list of MFLDs in the MOD. The DOF in turn contains a list of DFLDs which define where the data is to be displayed/printed on the output device. MFS maps the data of the MFLDs into the corresponding DFLDs.

All fields in an output message segment must be defined by MFLD statements. Fields can be truncated or omitted by two methods. The first method is to insert a short segment. The second method is to place a NULL character (X3f') in the field. Fields are scanned left (including the attribute bytes, if any) to right for NULL character. The first NULL character encountered terminates the field. If the first character of a field is a NULL character, no data is sent to the screen for this field. This means that if the field is protected and the same device format is used, the old data remains on the screen. To erase the old data of a protected field, the application program must send X'403F' to that field.

Positioning of all fields in the segment remains the same regardless of NULL characters. Truncated fields are padded with a program tab character in our subset. Furthermore, we always specify erase-unprotected-all in the display device format. This erases all old data in unprotected fields on the screen.

Notes:

1. Device control characters are invalid in output message fields under MFS. The control characters HT, CR, LF, NL, and BS will be changed to null characters (X'CC'). All other nongraphic characters are X'40' through X'FE'.
2. With MFS, the same output message can be mapped on different device types with one set of formats. This will not be covered in our subset. The formatting discussed will cover one device type per device format, not a mixture. However, the mixture can be implemented later by changing the formats.

In addition to MFLD data, constants can be mapped into DFLDs. These constants are defined as literals in DFLD or MFLD statements.

Multiple Segment Output Messages

MFS allows mapping of one or more output segments of the same message onto a single or multiple output screens. In our subset, we will limit ourselves to a one-to-one relationship between output message segments and logical output pages. Also one logical output page is one physical output page (one screen).

Logical Paging of Output Messages

Logical paging is the way output message segments are grouped for formatting. when logical paging is used, an output message descriptor is defined with one or more LPAGE statements. Each LPAGE statement relates a segment produced by a application program to corresponding device page.

Using logical paging, the simplest message definition consists of one LPAGE and one segment description. As shown in Figure 82, each segment produced by the application program is formatted in the same manner using the corresponding device page.

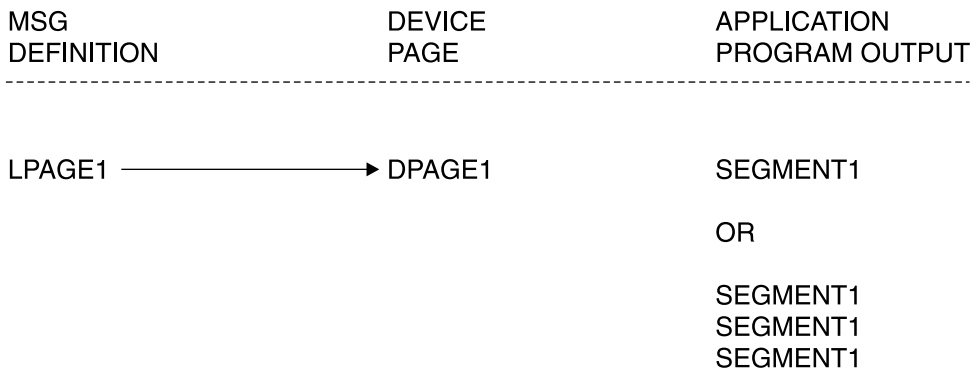

Figure 82. An Output Message Definition with One LPAGE

With the definition shown in Figure 82, each output segment inserted by the MPP will be displayed with the same and only defined MOD/DOF combination.

If different formats are required for different output segments, one LPAGE and SEG statement combination is required for each different format. Each LPAGE can link to a different DPAGE if desired. This would not be required if only defined constants and MFLDs differ in the MOD.

The selection of the DPAGE to be used for formatting is based on the value of a special MFLD in the output segment. This value is set by the MPP. If the LPAGE to be used cannot be determined from the segment, the last defined LPAGE is used. See also the description of the COND parameter of the LPAGE statement. Each LPAGE can refer to a corresponding DPAGE with unique DFLDs for its own device layout. See Figure 83.

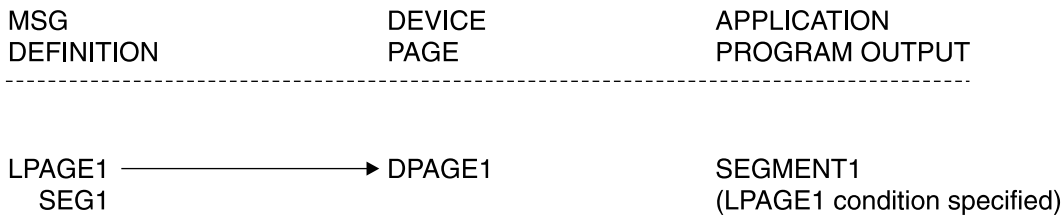

Figure 83. An Output Message Definition with Multiple Pages

Operator Paging of Output Messages

If an output message contains multiple pages, the operator requests the next one with the program access key 1 (PA1). If PA1 is pressed after the last page is received, IMS will send a warning message in our subset. If PA1 is then pressed again, IMS will send the first page of the current output message again.

The operator can always request the next output message by pressing the PA2 key. Also, in our subset, when the operator enters data, the current output message is dequeued.

Output Message Literal Fields

Output message fields can be defined to contain literal data specified by the user during definition of the MOD. MFS will include the specified literal data in the output message before sending the message to the device.

MFS users can define their own literal field or select a literal from a number of literals provided by MFS. The MFS-provided literals are referred to as system literals and include various date formats, a time stamp, the output message sequence number, the logical terminal name, and the number of the logical page.

Output Device Field Attributes

Device field attributes are defined in DFLD statements. For 3270 display devices, specific attributes may be defined in the ATTR= keyword of the DFLD statement. If not, default attributes will be assumed. The message field definition (MFLD) corresponding to the device field (DFLD) may specify that the application program can dynamically modify the device field attributes.

When a field is so defined, the first 2 data bytes of the field are reserved for attribute data. Any error in the 2-byte specification causes the entire specification to be ignored, and the attributes defined or defaulted for the device field are used.

Note: The two attribute bytes should not be included in the length specification of the device field (DFLD) in the DOF.

The default attributes for non-literal 3270 display device fields are alphabetic, not-protected, normal display intensity, and not-modified. Literal device fields have forced attributes of protected and not-modified and default attributes of numeric and normal display intensity. Numeric protected fields provide an automatic skip function on display terminals.

Cursor Positioning

The positioning of the cursor on the 3270 display device is done in either of two ways:

- The DPAGE statement defines the default cursor position.
- The program can dynamically set the cursor to the beginning of a field via its attribute byte.

System Message Field (3270 Devices)

Output formats for 3270 display devices may be defined to include a system message field. If so defined, all IMS messages except DFS057 REQUESTED FORMAT BLOCK NOT AVAILABLE are not sent to the system message field whenever the device is in formatted mode. Providing a system message field avoids the display of an IMS message elsewhere on the screen, thereby overlaying the screen data.

When MPS sends a message to the system message field, it activates the device alarm (if any) but does not reset modified data tags (MDTs) or move the cursor. Since an IMS error message is an immediate response to input, MDTs remain as they were at entry and the operator merely has to correct the portion of the input in error.

In our subset we will always reserve the bottom line of the screen for the system message field. This field can also be used to enter commands, for example, /FORMAT.

Printed Page Format Control

The 3270 printer devices are also supported via MFS. Three basic options can be specified in the DEV statement (PAGE=operand):

- A defined fixed number of lines should always be printed for each page (SPACE). This is the recommended option because it preserves forms positioning.
- Only lines containing data should be printed. Blank lines are deleted (FLOAT).
- All lines defined by DFLDs should be printed, whether or not the DFLDs contain data (DEFN).

MFS Formats Supplied by IBM

Several formats are included in the IMS.FORMAT library during IMS system definition. They are used mainly for the master terminal, and for system commands and messages. All these formats start with the characters DFS. One of the most interesting in our subset is the default output message format. This format is used for broadcast messages from the master terminal and application program output messages with no MOD name specified. It permits two segments of input, each being a line on the screen. DFSDF2 is the format name, DFSMO2 the MOD and DFSMI2 the MID name.

When the master terminal format is used, any message whose MOD name begins with DFSMO (except DFSMO3) is displayed in the message area. Any message whose MOD name is DFSDPO1 is displayed in the display area.

Messages with other MOD names cause the warning message USER MESSAGE WAITING to be displayed at the lower portion of the display screen.

MFS Control Statements

This section describes the control statements used by the MFS language utility. There are two major categories of control statements:

- Definition statements are used to define message formats and device formats.
- Compiler statements are used to control the compilation and listings of the definition statements

The definition of message formats and device formats is accomplished with separate hierarchical sets of definition statements. The following sections list some of the components of these statements.

- “Definition Statement for Message Formats”
- “Definition Statement for Device Formats” on page 219
- “Compiler Statement Definitions” on page 219
- “Relationships Between Source Statements and Control Blocks” on page 219

Definition Statement for Message Formats

The statement set used to define message formats consists of the following statements:

MSG Identifies the beginning of a message definition.

LPAGE Identifies a related group of segment/field definitions.

PASSWORD Identifies a field to be used as an IMS password

SEG Identifies a message segment.

MFLD Defines a message field. Iterative processing of MFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish Iterative processing, the DO statement is placed before the MFLD statement (or statements) and the ENDDO after the MFLD statement (or statements). For more information about the DO and ENDDO statements, see “Compiler Statement Definitions.”

MSGEND Identifies the end of a message definition.

Definition Statement for Device Formats

The statement set used to define message formats consists of the following statements:

FMT Identifies the beginning of a format definition.

DEV Identifies the device type and operational options.

DIV Identifies the format as input, output, or both.

DPAGE Identifies a group of device fields corresponding to an LPAGE group of message fields.

DFLD Defines a device field. Iterative processing of DFLD statements can be invoked by specifying DO and ENDDO statements. To accomplish iterative processing, the DO statement is placed before the DFLD statement (or statements) and the ENDDO after the DFLD statement (or statements). For more information about the DO and ENDDO statements, see “Compiler Statement Definitions.”

FMTEND Identifies the end of the format definition.

Compiler Statement Definitions

Compilation statements have variable functions. The most common ones are:

DO Requests iterative processing of MFLD or DFLD definition statements.

EJECT Ejects SYSPRINT listing to the next page.

END Defines the end of data for SYSIN processing.

ENDDO Terminates iterative processing of MFLD or DFLD.

PRINT Controls SYSPRINT options.

SPACE Skips lines on the SYSPRINT listing.

TITLE Provides a title for the SYSPRINT listing.

Compilation statements are to be inserted at logical points in the sequence of control statements. For example, TITLE could be placed first, and EJECT could be placed before each MSG or FMT statement.

Relationships Between Source Statements and Control Blocks

In general, the following relations exists between the MFS source statements and control blocks:

- One MSG statement and its associated LPAGE, SEG, and MFLD statements generate one MID or MOD.

- One FMT statement and its associated DEV, DIV, DPAGE and DFLD statements generate one DIF and/or DOF. For displays, both the DIF and DOF are generated, because the output screen is used for input too.

In addition the MFS utilities will establish the linkages between the MID, MOD, DIF, and DOF. These are the result of the symbolic name linkages defined in the source statements.

Generating MFS Control Blocks

MFS control blocks are generated by running the MFS language utility program. This is a two-stage process. See Figure 84. The sections that follow the figure describe this process.

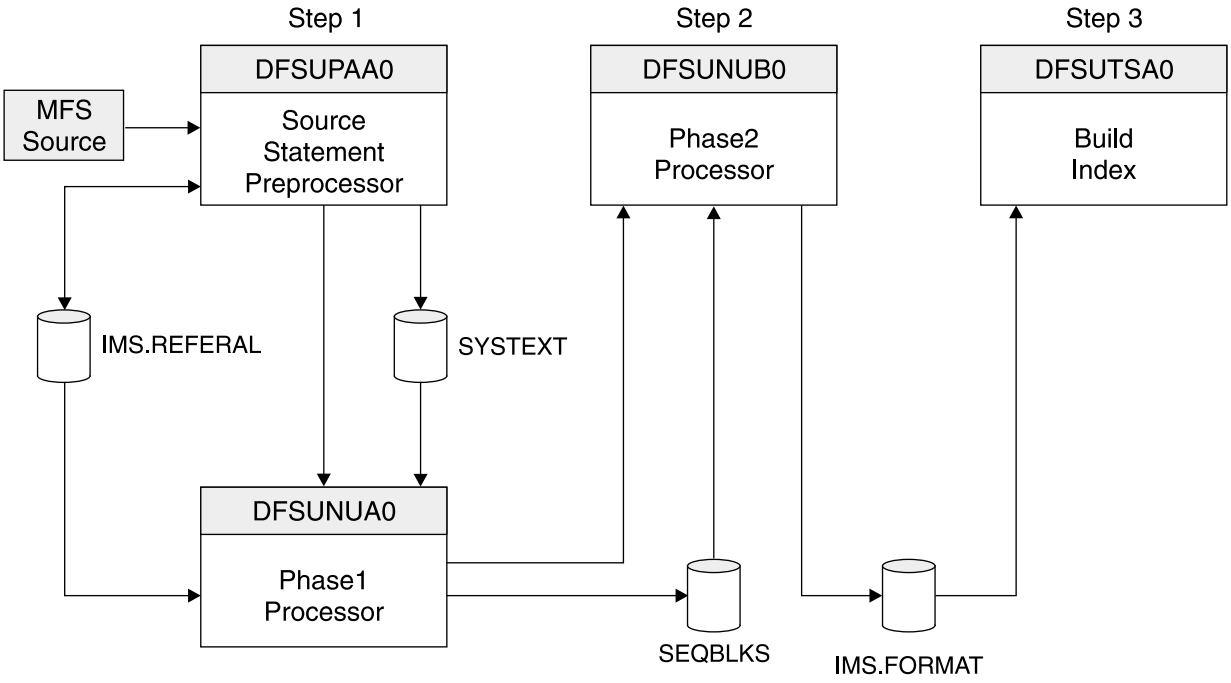

Figure 84. Overview of Process for Creating MFS Control Blocks

The MFS control block generated can be executed by an IMS supplied cataloged procedure: MFSUTL. Multiple formats can be generated with one execution. In general you would process a complete format set, that is, the related message and format descriptions, in one execution of MFSUTL. Three executions of MFSUTL are involved to process the three sample format sets.

"Steps for Generating MFS Control Blocks" describes the three steps of generating MFS control blocks.

Steps for Generating MFS Control Blocks

Generating MFS Control Blocks (Step 1)

Preprocessor

The MFS language utility preprocessor generates intermediate text blocks (ITBs), based on the MFS language source statement. Definitions of the MFS language utility source input are discussed in "MFS Control Statements" on page 218. The primary function of the preprocessor is to

perform syntax and relational validity checks on user specifications and generate ITBs. The ITBs are then processed by phase 1 of the utility to generate message (MSG) and format (FMT) descriptors. An ITB generated for each MSG or FMT description can be re-used by the same or another format set, once it has been successfully added to the IMS.REFERAL data set. Each such description must start with a MSG or FMT statement and end with a MSGEND or FMTEND statement.

Phase 1

The preprocessor invokes phase 1 if the highest return code generated by the preprocessor is less than 16. Phase 1 places the newly constructed descriptors on the SEQBLKS data set. Each member processed has a control record placed on the SEQBLKS data set identifying the member, its size, and the date and time of creation. This control record is followed by the image of the descriptor as constructed by phase 1. Alternatively, if an error is detected during descriptor building, an error control record is placed on the SEQBLKS data set for the description in error, identifying the member in error, and the date and time the error control record was created. In addition, phase 1 returns a completion code of 12 to z/OS. If execution of step 2 is forced, phase 2 will delete descriptors with build errors.

Generating MFS Control Blocks (Step 2)

Phase 2

Phase 2 receives control as a job step following phase 1. After final processing, it will place the new descriptors into the IMS.FORMAT library. Phase 2 passes a completion code to z/OS for step 2 based on all the descriptor maintenance to IMS.FORMAT for a given execution of the MFS language utility.

Generating MFS Control Blocks (Step 3)

In our subset, we will always execute the MFS service utility after MFS control block generation. This utility will build a new index directory block which will eliminate the need for directory search operations during the IMS online operation.

Maintaining the MFS Library

The IMS.FORMAT and IMS.REFERAL libraries are standard, z/OS partitioned data sets. Backup and restore operations can be done with the proper z/OS utility (IEBCOPY). However, care must be taken that both the IMS.FORMAT and IMS.REFERAL data sets are dumped and restored at the same time.

Chapter 21. Application Programming in IMS Java

IMS Java application programs use JDBC or IMS Java hierarchic database interface. JDBC is the SQL-based standard interface for data access in the Java 2 SDK Standard Edition and Enterprise Edition. IMS Java's implementation of JDBC supports a selected subset of the full facilities of the JDBC 2.0 API. The IMS Java hierarchic database interface is more closely related to the standard DL/I database call interface used with other languages, and provides a lower-level access to IMS database functions than the JDBC interface.

IMS Java provides class libraries that allow you to easily develop applications that can access IMS's broad range of database types and options, including:

- Full-function databases
- High Availability Large Databases (HALDBs)
- Fast Path Data Entry Databases (DEDBs)
- Logical relationships
- Secondary indexes

IMS Java application programs can be message-driven or non-message-driven and can handle a variety of message processing:

- Conversational and non-conversational transactions
- Multi-segment and single-segment messages
- Message Formatting Services (MFS)
- Alternate PCB program switching

Regardless of what environment the IMS Java application runs in, it accesses the IMS databases the same way.

The following sections are covered in this chapter:

Environments that Support IMS Java

The following sections are overviews of the support for IMS Java application programs from various environments.

IMS Environment Overview

IMS Java application programs run in one of two IMS dependent regions that provide a Java Virtual Machine (JVM) environment for the Java applications:

- Java Message Processing (JMP) region for message-driven Java applications. JMP applications process input messages from the message queue, similar to Message Processing Programs (MPPs), in a DB/DC environment.
- Java Batch Processing (JBP) region for non-message-driven Java applications. JBP applications run in an online batch mode and do not process input

messages, similar to non-message-driven Batch Message Processing (BMP) applications, in a DBCTL or a DB/DC environment.

Restriction: JBP applications cannot be message driven.

Related Reading:

- For guidance information on designing an IMS application, see the *IMS Version 9: Application Programming: Design Guide*.
- For information on configuring JMP and JBP regions, see the *IMS Version 9: IMS Java Guide and Reference* and the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

WebSphere Application Server for z/OS Environment Overview

You can write Java applications (WebSphere Application Server for z/OS Enterprise Java Beans [EJBs]) that run on a WebSphere Application Server for z/OS and OS/390 J2EE server and access IMS databases.

Related Reading: For information on configuring WebSphere Application Server for z/OS to run Java applications that access IMS databases, see *IMS Version 9: IMS Java Guide and Reference*.

CICS Environment Overview

You can write IMS Java applications that run on CICS Transaction Server for z/OS and access IMS databases.

Related Reading: For information on configuring CICS to run Java applications that access IMS databases and for information on developing an Java application that runs on CICS and accesses IMS databases, see *IMS Version 9: IMS Java Guide and Reference*.

DB2 UDB for z/OS Environment Overview

You can write DB2 UDB for z/OS stored procedures that access IMS databases.

Related Reading: For information on configuring DB2 UDB for z/OS to run Java applications that access IMS databases and developing a DB2 UDB for z/OS stored procedure that accesses IMS databases, see *IMS Version 9: IMS Java Guide and Reference*.

Describing an IMS Database to IMS Java

Processing IMS databases with an IMS Java application requires that you describe the database view of your application's PSB to IMS Java. You must do this by providing the name of a metadata class when establishing the JDBC database connection.

There are two ways you can prepare the metadata class for an application:

- Provide the application PSB source and any related DBD source to the DLIModel utility, and specify the generation of the IMS Java metadata class.

  This is the recommended technique and is described in this chapter.
- Code the metadata class manually.

  This is described further in the *IMS Version 9: IMS Java Guide and Reference*.

You can use the DLIModel utility to:

- Create IMS Java metadata classes to describe a PSB's view of IMS databases, from PSB and DBD source.
- Incorporate additional field information form XMI input files that describe COBOL copybook members.
- Incorporate additional PCB, segment, and field information, or overrides of existing information, into the generated class from user-prepared input control statements.
- Create a DLIModel Java Report (designed to assist Java application programmers), which describes the IMS Java view of the PSB and its databases.
- Create an XMI description of the PSB and its databases.

The DLIModel utility can process most types of PSBs and databases. For example, IMS Java supports:

- All database organizations except MSDB, HSAM, SHSAM, and GSAM
- All types and implementations of logical relationships
- Secondary indexes except for shared secondary indexes
- Secondary indexes processed as stand-alone databases
- PSBs that specify field-level sensitivity

Figure 85 on page 226 shows the inputs and outputs of the DLIModel utility. The actions of the utility are directed by control statements that you supply. PSB and DBD source members are read from their PDS or PDSE data sets and parsed by the utility to build an in-memory object model of the database structure and the PSB's view of that structure. Multiple PSBs may be processed in a single run of the utility.

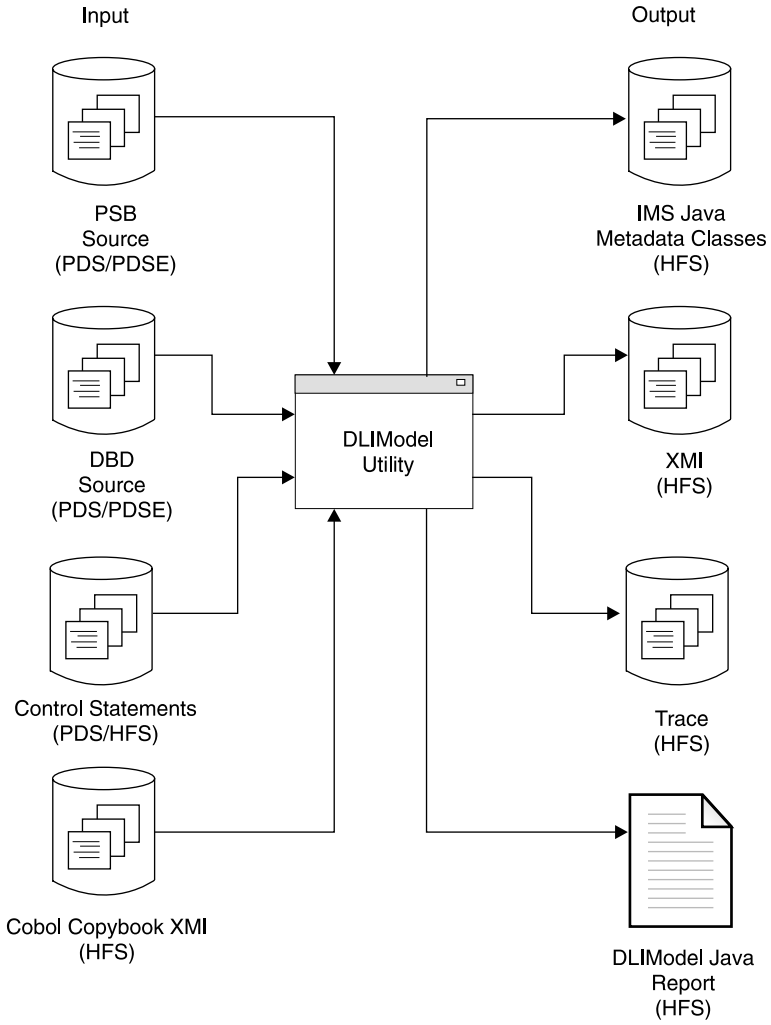

Figure 85. DLIModel Utility Inputs and Outputs

Related Reading: For more information about the DLIModel utility, see the *IMS Version 9: IMS Java Guide and Reference*.

Accessing an IMS Database with IMS Java

IMS Java supports two styles of database programming:

JDBC JDBC is the SQL-based standard interface for data access in the Java 2 SDK Standard Edition and Enterprise Edition. IMS Java's implementation of JDBC supports a selected subset of the full facilities of the JDBC 2.0 API. This is the recommended style where sufficient for the application.

Related Reading: For more information about using the JDBC interface, see "Using JDBC to Access an IMS Database" on page 227.

IMS Java hierarchic database interface
The IMS Java hierarchic database interface is more closely related to the standard DL/I database call interface used with other languages, and provides a lower-level access to IMS database functions than the JDBC interface. Using this style of programming, you can build segment search arguments (SSAs) and call the functions of the DLIConnection object to

read, insert, update, or delete segments. With this style, the application has full control to navigate the segment hierarchy.

Related Reading: For details of using the IMS Java hierarchic database interface, see the *IMS Version 9: IMS Java Guide and Reference*.

Note: Both styles require that you first describe your IMS databases to the IMS Java classes through a metadata class. See "Describing an IMS Database to IMS Java" on page 224 for more information about creating a metadata class.

Recommendation: Before accessing an IMS database, you should have a basic understanding of hierarchical databases. For more information about hierarchical databases, see Chapter 5, "Overview of the IMS Hierarchical Database Model," on page 41.

Using JDBC to Access an IMS Database

The following sections discuss the JDBC access method.

Comparing Hierarchical and Relational Databases

For the purpose of writing JDBC calls, a database segment definition defines the fields for a set of segment instances similar to the way a relational table defines columns for a set of rows in a table. In this way, segments relate to tables, and fields in a segment relate to columns in a table. Therefore, the name of an IMS segment from the summary report becomes the table name in an SQL query, and the name of a field becomes the column name in the query.

Writing an Application that Uses JDBC

To use JDBC to read, update, insert, and delete segment instances, an application must:

1. Load the DLIDriver and retrieve a Connection object from the DriverManager. This step is highlighted in bold text in Figure 86 on page 228.
2. Retrieve a Statement or PreparedStatement object from the Connection and execute it.
3. Iterate the ResultSet returned from the Statement or PreparedStatement object to retrieve specific field results.

```
package dealership.application;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;
import com.ibm.ims.db.*;
import java.sql.*;

public class IMSAuto extends IMSApplication {
    IMSMessageQueue messageQueue = null;
    InputMessage inputMessage = null;
    ModelOutput modelOutput = null;
    Connection connection = null;

    public  IMSAuto() {
        super();
    }

    public static void main(String args[]){
        IMSAuto imsauto = new IMSAuto();
        imsauto.begin();
    }

    public void doBegin() throws IMSException {
        messageQueue = new IMSMessageQueue();
        inputMessage = new InputMessage();
        modelOutput = new ModelOutput();

        try {
            Class.forName("com.ibm.ims.db.DLIDriver");
            connection = DriverManager.getConnection
                ("jdbc:dli:dealership.application.DealerDatabaseView");
        }

        catch (Exception e){
            reply("Connection not established");
        }

        while(messageQueue.getUniqueMessage(inputMessage)) {
            if (!inputMessage.getString("ModelTypeCode").trim().equals("")){
               if (getModelDetails(inputMessage, modelOutput))
                   messageQueue.insertMessage(modelOutput);
            }

            else {
                reply("Invalid Input");
            }

            IMSTransaction.getTransaction().commit();
        }
    }

    public void reply(String errmsg) throws IMSException{
        ErrorMessage errorMessage = new ErrorMessage();
        errorMessage.setString("MessageText",errmsg);
        messageQueue.insertMessage(errorMessage);
    }
}
```

Figure 86. JDBC Application

Part 5. IMS System Administration

Chapter 22. The IMS System Definition Process

The IMS system definition process:

- Is used to define the IMS resources:
  - At installation time.
  - When maintenance is applied.
  - When standard application definition changes are required.
- Assumes a working knowledge of SMP/E. SMP/E is required for installation and is used as part of the IMS system definition process.
- Is evolving to be more dynamic (versus static) over time.

Extended Terminal Option (ETO), a separately-priced feature of IMS TM, is the first IMS offering for defining resources dynamically (without requiring you to shut down IMS). With ETO, you can dynamically add or delete VTAM terminals and users to your IMS outside of IMS the system definition process. For more information about ETO, see “The Extended Terminal Option (ETO)” on page 238.

The following sections are covered in this chapter:

Overview of the IMS System Definition Process

The IMS system definition process is made up of many steps. Some steps only occur for certain types of IMS definitions (see “Types of IMS System Definitions” on page 232). Some of the steps involved in IMS system definition are:

- You modify or tailor IMS-supplied macros and procedures (see “IMS System Definition Macros” on page 235)
- Stage1 assembly (see “Stage 1” on page 233)
- Stage2 assembly, including the optional building of MFS default formats and PROCLIB updates (see “Stage 2” on page 234)
- JCLIN (see “JCLIN” on page 234)
- Use the SMP/E `APPLY` command to process any maintenance that has not been processed using the SMP/E `ACCEPT` command (see “SMP/E Maintenance” on page 234)
- “IMS Security Maintenance Utility Generation” on page 235

For on overview of the Stage1 and Stage2 components of the system definition process, please see Figure 87 on page 232.

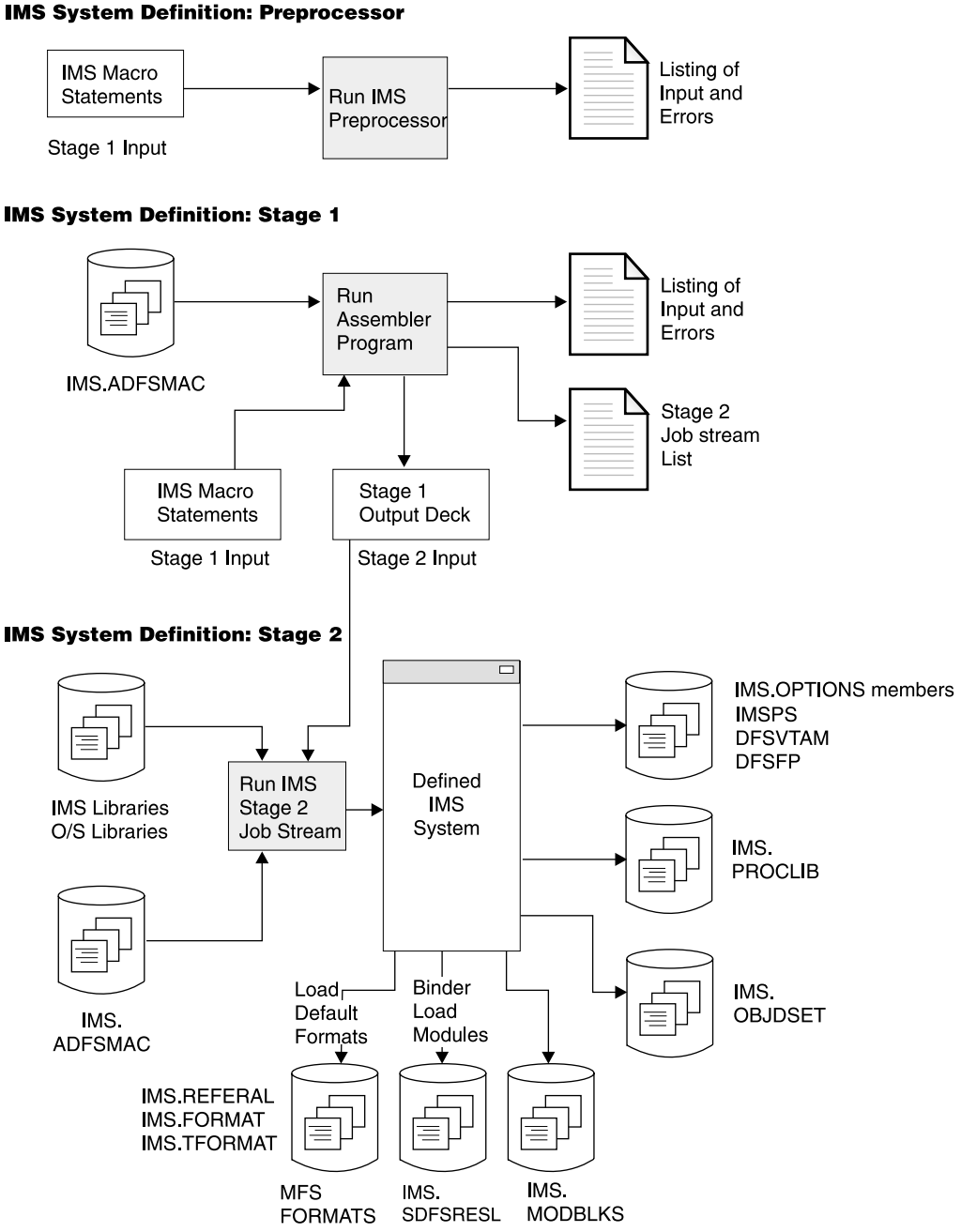

Figure 87. Overview of the Two Stages of System Definition Processing

Types of IMS System Definitions

There are 7 different levels of IMS system definitions, as documented in Table 15.

Table 15. Types of IMS System Definitions

IMSCTRL option	When used	Comments
BATCH	Only for the batch environment.	Generates modules and procedures needed to build a complete batch IMS system.

Table 15. Types of IMS System Definitions (continued)

IMSCTRL option	When used	Comments
MSVERIFY	Only appropriate for MSC.	Builds control blocks for the MSC Verification utility.
MODBLKS	Used when online changes to the IMS system is required (such as programs, transactions, and database definitions).	Generates control blocks members for resources to be added online (for example, APPLCTN, DATABASE, TRANSACT, and RTCODE macros).
CTLBLKS	Used to rebuild the existing IMS nucleus and to create communications definitions.	Generates modules for all IMS control blocks (for example, TERMINAL and LINE macros). This type of system definition includes MODBLKS and MSVERIFY.
NUCLEUS	Used when performing major maintenance that affects the contents of the IMS nucleus or when a new nucleus with a new suffix is required.	Generates an IMS nucleus for the control region. This type of system definition includes CTLBLKS.
ON-LINE	Major update, or initial system definition. Often required for maintenance.	Builds all the modules and procedures needed for the online IMS environment. This type of system definition includes NUCLEUS.
ALL	Typical initial system definition, usually needed at maintenance time.	Builds most IMS modules and procedures. Includes BATCH and ON-LINE options.

After your initial system definition the ON-LINE, CTLBLKS, and NUCLEUS types of system definition can be used to implement changes. These system definitions require a cold start of the IMS online system to take effect.

However, for certain changes to your IMS system, you can take advantage of the online change method using the MODBLKS system definition. The changes are made active during the execution of the online system and do not require a restart operation.

Related Reading: For all the details about the different types of system definition and what they are used for, see “Using the Macro Table” in the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Stage 1

The IMS Stage1 job uses the assembler, with the input being the IMS macros, as discussed in “IMS System Definition Macros” on page 235. Other references are to the IMS distribution macro libraries (IMS.ADFSMAC)

The output of the IMS Stage 1 includes:

- Standard assembler listing output with any appropriate error messages.
- IMS Stage 2 input JCL, also used for JCLIN.

The output from Stage 1 is then used as the JCL to run the Stage 2 of the system definition process.

Depending on the Stage 1 definitions within the IMSGEN macro, the Stage 2 can be divided up into a single job with multiple steps, or many jobs with fewer steps. This depends on how your site prefers to run this process.

Stage 2 will do all the module assembling and binding as required to build all the necessary load modules, depending on what type of system definition is being run.

Stage 2

As for the Stage 2, these steps will all refer to the IMS distribution macro library (IMS.ADFSMAC) at assembly time, and distribution load library (IMS.ADFSLOAD) at bind time.

The output of the IMS system definition process includes:

- Executable load modules in data sets IMS.SDFSRESL, IMS.MODBLKS
- IMS Options definitions in data set IMS.OPTIONS.
- Assembled object code for use in later system definition steps in data sets IMS.OBJDSET.
- Optionally create the runtime IMS. PROCLIB data set. See “IMS.PROCLIB Update.”
- Optionally create the runtime IMS default MFS screens in data sets IMS.FORMAT, IMS.TFORMAT, IMS.REFERAL. See “Building MFS Default Formats.”

IMS.PROCLIB Update

A parameter in the IMS Stage1 macro IMSGEN (PROCLIB=YES/NO) determines whether or not the IMS.PROCLIB data set is to be populated by this system definition, or not. The IMS.PROCLIB library contains IMS started tasks and JCL procedures, as well as the IMS.PROCLIB members required by IMS and IMS utilities to provide options.

Building MFS Default Formats

A parameter in the IMS Stage1 macro IMSGEN (MFSDFMT=YES/NO) determines whether or not the default message format screens are built as part of Stage 2 of the system definition process.

JCLIN

JCLIN is a an SMP/E process that tells SMP/E how to assemble and bind modules.

As the IMS Stage 2 actually assembles and binds the IMS modules based on the definitions for that particular system (and is run outside of SMP/E control), the system definition Stage 2 input JCL must be used as input to the JCLIN process, so that SMP/E will know how to manage any maintenance added to the system following this IMS system definition.

JCLIN should be run following any IMS system definition, to ensure that SMP/E is always synchronized with the updated IMS.

SMP/E Maintenance

All IMS system definitions use the IMS SMP/E distribution libraries and the IMS Stage 1 macros as input.

As a result, any SMP/E maintenance (SYSMODs - PTFs, APARs or USERMODs) that were processed using the SMP/E `APPLY` command (but not processed using the SMP/E `ACCEPT` command) prior to an IMS system definition, might be regressed as a result of that IMS system definition, depending on the type of IMS system definition, and the impact of the SYSMOD.

Recommendation: Any maintenance that has been processed using the SMP/E `APPLY` command, but not processed using the SMP/E `ACCEPT` command, should be reprocessed (using both commands) following an IMS system definition, unless further investigations have shown specific cases where this is not necessary.

Related Reading: For more information about performing SMP/E maintenance on IMS, see Chapter 8, "IMS Service Considerations" in the *IMS Version 9: Installation Volume 1: Installation Verification*.

IMS Security Maintenance Utility Generation

For security beyond that provided by default terminal security, you can use the various security options specified with the Security Maintenance utility (SMU). The utility is executed offline after completion of IMS Stage 2 processing for system definition. Its output is a set of secured-resource tables placed on the MATRIX data set. The tables are loaded at system initialization time, and, for certain options, work with exit routines and/or RACF during online execution to provide resource protection.

Requirement: The SMU generation must ALWAYS be run after any system definition. If the SMU generation is not run after an IMS system definition, IMS might not start.

Related Reading: For further details, see

- The *IMS Version 9: Utilities Reference: System*.
- The *IMS Version 9: Administration Guide: System*.
- The *IMS Security Guide* redbook.

Note: IMS Version 9 is the last version of IMS to support the SMU.

IMS System Definition Macros

As these change from release to release of IMS, please see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring* for details of the various macros. They have been summarized here very briefly:

APPLCTN

The APPLCTN macro allows you to define the program resource requirements for application programs that run under the control of the IMS DB/DC environment, as well as for applications that access databases through DBCTL. An APPLCTN macro combined with one or more TRANSACT macros defines the scheduling and resource requirements for an application program. Using the APPLCTN macro, you only describe programs that operate in message processing regions, Fast Path message-driven program regions, batch message processing regions, or CCTL threads. You do use the APPLCTN macro to describe application programs that operate in batch processing regions. When defining an IMS data communication system, at least one APPLCTN macro is required.

BUFFPOOLS

The BUFPOOLS macro statement is used to specify default storage buffer pool sizes for the DB/DC and DBCTL environments. The sizes specified are used unless otherwise expressly stated for that buffer or pool at control program execution time for an online system.

COMM

The COMM macro is used to specify general communication requirements

that are not associated with any particular terminal type. COMM is always required for terminal types supported by VTAM. It can also be required to specify additional system options, such as support for MFS on the master terminal.

CONFIG
The CONFIG macro statement provides the configuration for a switched 3275 terminal. Because the configuration provided by CONFIG is referenced when the named 3275 dials into IMS, differently configured 3275s can use the same communication line. All CONFIG macro statements must be between the LINEGRP macro and the LINE macros. LINE macros can refer to named CONFIG macros defined previously in this line group or in previously defined line groups.

CTLUNIT
The CTLUNIT macro statement specifies 2848, 2972, and 3271 control unit characteristics. CTLUNIT is valid only for 3270 remote line groups.

DATABASE
The DATABASE macro statement is used to define the set of physical databases that IMS is to manage. One DATABASE macro instruction must be specified for each HSAM, HISAM, HDAM, or PHDAM database. Two DATABASE macro instructions are required for each HIDAM or PHIDAM database: one for the INDEX DBD and one for the HIDAM or PHIDAM DBD. One DATABASE macro instruction must be included for each secondary index database that refers to any database defined to the online system. For Fast Path, a DATABASE macro statement must be included for each Main Storage Database (MSDB) and Data Entry Database (DEDB) to be processed.

FPCTRL
The FPCTRL macro statement defines the IMS Fast Path options of the IMS control program, and the DBCTL environment. It is ignored when the IMSCTRL statement specifies that only a BATCH or MSVERIFY system definition is to be performed.

IDLIST
The IDLIST macro statement is used to create a terminal security list for switched 3275s.

IMSCTF
The IMSCTF macro statement defines parameters to IMS, and to the DBCTL environment.

IMSCTRL
The IMSCTRL macro statement describes the basic IMS control program options, the z/OS system configuration under which IMS is to execute, and the type of IMS system definition to be performed. **The IMSCTRL macro instruction must be the first statement of the system definition control statements.**

IMSGEN
IMSGEN specifies the assembler and binder data sets and options, and the system definition output options and features. **The IMSGEN must be the last IMS system definition macro, and it must be followed by an assembler END statement.**

LINE The LINE macro statement describes both switched and nonswitched communication lines.

LINEGRP
The LINEGRP macro statement defines the beginning of a set of macro instructions that describe the user's telecommunications system.

MSGQUEUE
This macro defines the characteristics of the three message queue data sets (QBLKS, SHMSG, and LGMSG). The information you specify in this macro is also used in a shared-queues environment. The MSGQUEUE macro is required for all DB/DC and DCCTL systems.

MSLINK
The MSLINK macro statement defines a logical link to another system.

MSNAME
The MSNAME macro statement provides a name for the remote and local system identifications that it represents.

MSPLINK
The MSPLINK macro statement defines a physical MSC link.

NAME The NAME macro statement defines a logical terminal name (LTERM) associated with a physical terminal. Preparation of the NAME macro can be required for each of the following macros: TERMINAL, SUBPOOL, MSNAME.

POOL The POOL macro statement describes a pool of logical terminals that are to be associated with a set of switched communication lines.

RTCODE
The RTCODE macro statement is used one or more times with the APPLCTN macro statement that defines an IMS Fast Path application program. It specifies the routing codes that identify the program named in the preceding APPLCTN macro statement. A TRANSACT macro statement that specifies an IMS Fast Path-exclusive transaction builds an internal RTCODE macro statement with a routing code identical to the transaction code.

SECURITY
The SECURITY macro statement lets you specify optional security features to be in effect during IMS execution unless they are overridden during system initialization.

STATION
The STATION macro statement describes the physical and logical characteristics of the System/3 or System/7.

SUBPOOL
The SUBPOOL macro statement, when used in a VTAM macro set, is a delimiter between groups of NAME macro statements to create LU 6.1 LTERM subpools.

When used in a switched communication device macro set, the SUBPOOL macro defines a set of logical terminals.

TERMINAL
The TERMINAL macro statement defines physical and logical characteristics of VTAM nodes and non-VTAM communication terminals.

TRANSACT
The TRANSACT macro statement is used one or more times with each APPLCTN macro statement to identify transactions as IMS exclusive, IMS Fast Path potential, or IMS Fast Path exclusive. It specifies the transaction

codes that cause the application program named in the preceding APPLCTN macro to be scheduled for execution in an IMS message processing region. It also provides the IMS control program with information that influences the application program scheduling algorithm.

TYPE The TYPE macro statement defines the beginning of a set of communication terminals and logical terminal description macro statements which include TERMINAL and NAME. The TYPE macro statement begins a description of one set, that contains one or more terminals of the same type. TYPE defines terminals attached to IMS through VTAM. It is equivalent to the LINEGRP/LINE macro set used to define terminals attached to IMS by means other than VTAM.

VTAMPOOL
The VTAMPOOL macro, required for parallel session support, begins the definition of the LU 6.1 LTERM subpools.

The Extended Terminal Option (ETO)

The IMS Extended Terminal Option (ETO) allows you to add VTAM terminals and users to your IMS without predefining them during system definition. ETO is separately priced feature of the IMS Transaction Manager (TM) and provides additional features such as output security, automatic logoff, and automatic signoff.

By installing ETO, you can achieve each of the following:

- Improved system availability by reducing scheduled down time associated with adding or deleting VTAM terminals.
- Faster system availability to users, because they can establish an IMS session from any VTAM terminal in the network.
- Improved IMS security by relating output messages to users, rather than to terminals.
- Reduced number of macros required to define the terminal network. This reduces system definition time and storage requirements.
- Reduced checkpoint and restart time. For ETO terminals and user structures, resources are not allocated until they are actually required; similarly, when they are no longer required, they are deleted.
- Reduced number of skilled system programmer resources that are required for maintaining static terminal definitions.

ETO Terminology

The following sections describe the terms that have ETO-specific meanings. These meanings are important for understanding and administering ETO.

Terminals

A terminal is a physical VTAM logical unit (LU) that establishes a session with IMS. A physical terminal is represented using a control block.

When terminals are not built by ETO but are defined at system definition, they are called *static terminals*. A static terminal can be a VTAM node. When messages are sent to a static terminal they are queued to a logical terminal (LTERM) message queue, where they await retrieval by the recipient.

When a terminal is not defined at system definition and ETO builds a terminal, that terminal is called a *dynamic terminal*, or an ETO terminal. For dynamic terminals, the logical terminal (LTERM) is known as a dynamic user message queue, LTERM

associates the messages with the user, rather than with the physical terminal. Associating messages with the users provides more security for these users, because they can access their messages only when they sign on using their unique user ID. In addition, all users in the network can access their messages from any physical terminal, instead of being restricted to using a particular physical terminal.

Dynamic User

An ETO dynamic user is a user who signs on to a dynamic terminal and who has a unique identification (user ID) that IMS uses for delivering messages. The user is usually associated with a person but can also be associated with another entity, such as a printer.

Terminal Structure

A terminal structure is a control block that represents a specific terminal that is known to IMS. A terminal structure is created when the individual terminal logs on to IMS. It is deleted when the terminal logs off with no remaining associated activity (such as status that must be retained for the next connection to IMS).

User Structure

A user structure is a set of control blocks, including a user block and one or more LTERM blocks. The message queues are associated with the dynamic user, as opposed to the physical terminal, and they are queued to the user ID.

The dynamic user structure connects to the physical terminal only when the user signs on. This provides a secure environment, because different users accessing the same terminal cannot receive each other's messages.

IMS creates a user structure when either of the following events take place:

- A dynamic user signs on to IMS
- Output messages that are destined for a dynamic user are sent to the user, but the user has not signed on to IMS.

Usually, a user structure represents a person who uses IMS. The user structure name is usually the same as the user ID. A user structure can also represent a logical destination, such as a printer. In this case, the user structure name can be the same as or different from the LTERM name that your installation uses in its application programs and its exit routines. For example, you can assign the same name to a user structure for a printer that you assign to its LTERM destination node name. However, output is then queued according to the terminal, and not to the user.

Figure 21 and Figure 22 show the differences between static resources and ETO dynamic resources.

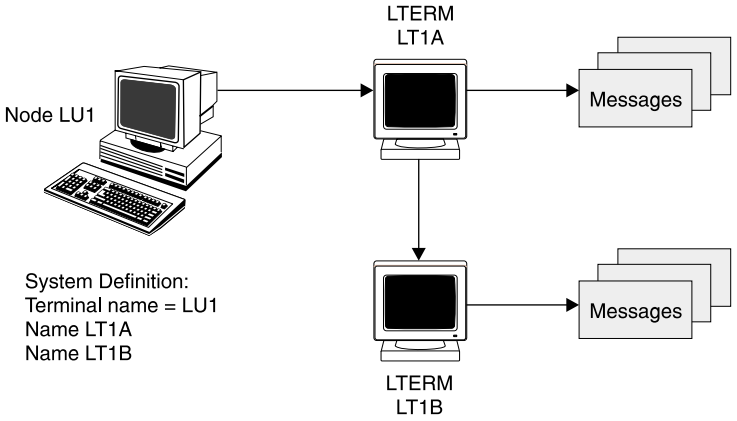

Figure 88. Static Resources

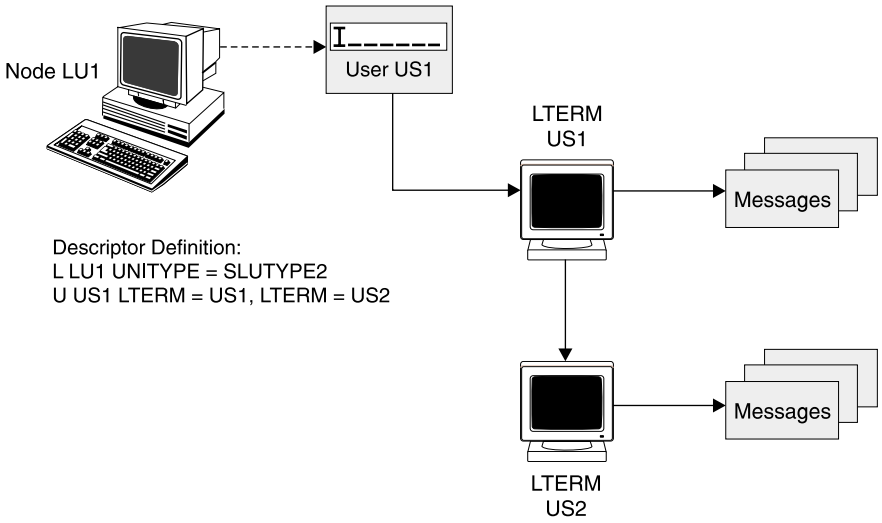

Figure 89. ETO Dynamic Resources

Descriptors

A descriptor provides information to IMS when IMS builds a dynamic resource for a logon or a sign-on. IMS stores the descriptors in two IMS.PROCLIB members, DFSDSCMx and DFSDSCTy.

Related Reading: For more information about these PROCLIB members, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

The next few sections describe the four types of descriptors:

- Logon descriptors
- User descriptors
- MSC descriptors
- MFS device descriptors

Logon Descriptors: A logon descriptor is a skeleton that IMS uses to build an ETO dynamic terminal. It provides information regarding a terminal's physical characteristics. IMS uses logon descriptors in conjunction with exit routines to create terminal structures.

There are three types of logon descriptors:

Generic
A generic logon descriptor provides characteristics for all terminals of a particular type. For example, all SCS printers might share a single generic descriptor. Similarly, all 3270 terminals might share a generic descriptor.

Group A group logon descriptor provides characteristics for a collection of terminals, each of which has compatible hardware characteristics and is defined to IMS in the same manner. The actual characteristics for these terminals are usually identical, but they can differ. IMS uses the group descriptor to derive their characteristics.

Specific
A specific logon descriptor provides characteristics for a single terminal, and these characteristics apply only to that terminal. In this case, the descriptor name matches the name of the terminal that it describes.

Recommendation: Although you might need to use specific logon descriptors during the actual migration to ETO, use generic or group logon descriptors after you have migrated to ETO; these kinds of descriptors ease network administration.

User Descriptor: A user descriptor is a skeleton from which a user structure is built. A user descriptor can provide user options and queue names.

MSC Descriptor: An MSC descriptor is used to create a remote LTERM, which is an LTERM that does not exist on the local IMS. The physical terminal definition (either static or dynamic) for the remote LTERM is in the remote IMS.

Each MSC descriptor for a remote LTERM is loaded during IMS initialization and tells IMS which MSC link to use for output destined for that remote LTERM.

Related Reading: For more information about MSC, see “Intersystem Communications (ISC)” on page 117.

MFS Device Descriptor: MFS device descriptors allow you to add new device characteristics for MFS formatting without requiring an IMS system definition. The MFSDCT utility (DFSUTB00) uses MFS device descriptors to update default formats in the MFS library.

IMS also uses MFS device descriptors to update the MFS device characteristics table. IMS loads this table only during initialization; therefore, updates are not effective until the next IMS initialization.

ETO Concepts

The following sections describe important ETO concepts.

When Structures Are Created

Structures are created in the following situations:

- Logon
- Sign-on
- Output is queued to your LTERM
- /ASSIGN command is used to assign an LTERM to a non-existent user
- /ASSIGN command is used to assign a non-existent LTERM to a user
- /CHANGE USER user AUTOLOGON command is directed to a non-existent user

In all cases, IMS searches for an existing structure (terminal or user) before creating a new one.

IMS creates and deletes user structures in the sequence described in the following list. This sequence applies only to terminal logon and logoff and to user sign-on and sign-off. When asynchronous output is queued to a user, IMS creates the user structure, as needed.

1. When you establish a session between IMS and an undefined terminal, IMS selects a logon descriptor.
2. Using the information in the logon descriptor, the customization defaults, and VTAM information, IMS builds an IMS terminal control block (called a VTAM terminal control block - VTCB) that describes the new terminal.
3. When you sign on, if a user structure does not exist, IMS builds one, using information from a user descriptor that it selects, and then connects this user structure to the terminal structure.
4. IMS deletes terminal or user structures when they are no longer needed to maintain sessions. User structures are typically deleted when you sign off, if no special status needs to be maintained and if no messages remain queued. IMS deletes terminal structures when no terminal status exists (such as trace mode), no user is signed on, and the terminal is idle.

   If you are using Resource Manager and a resource structure, IMS normally maintains status in the resource structure instead of the local control blocks. Therefore, IMS deletes the structures.

   Related Reading: For more information about the Resource Manager, see "Resource Manager" on page 339 or the *IMS Version 9: Common Service Layer Guide and Reference*. For more information about how IMS manages terminal and user structures, see the *IMS Version 9: Administration Guide: Transaction Manager*.

Descriptors and Exit Routines

The main purpose of ETO is to dynamically define terminals to IMS. Using descriptors and exit routines, you can assign characteristics to these dynamic terminals and assign user structures to be associated with those terminals.

A descriptor provides the basic information for the dynamic terminal. An exit routine completes or changes this information. Two methods of using descriptors and exit routines are:

- You can use many descriptors and code little or no processing logic in exit routines.
- You can use few descriptors and code exit routines to perform much of the processing.

How Descriptors are Created and Used

All descriptors are created during IMS initialization, prior to IMS startup. You must specify that you want ETO support and ensure that the ETO initialization exit routine (DFSINTX0) does not disable ETO support.

During IMS initialization, IMS reads and validates all ETO descriptors. IMS initialization then continues, and the descriptors remain in storage for the duration of IMS execution. Any changes you make to descriptors become effective after the next initialization of IMS.

IMS uses descriptors to create both terminal and user structures. IMS rebuilds structures during an IMS restart, if appropriate. For example, if messages are

queued for a structure and IMS shuts down, the structures are rebuilt when IMS restarts. IMS rebuilds these structures to be the same as they were before the IMS restart. IMS does not use the descriptors or exit routines to rebuild these structures. Therefore, any changes you make to descriptors are only reflected in new structures that are built after IMS restart, and the changes are not reflected in structures that are rebuilt during IMS restart.

Example: USERA signs on using descriptor DESCA which specifies ASOT=20. USERA starts an IMS conversation, and then IMS abnormally terminates. The system programmer changes DESCA to ASOT=10. After the IMS restart, USERB signs on using DESCA. USERA was rebuilt during the IMS restart. USERA still has ASOT=20, and USERB has ASOT=10.

Administering ETO

The tasks involved in administering ETO are as follows:

- Planning for ETO
- Coding ETO descriptors
- Coding ETO exit routines
- Specifying system definition parameters for ETO
- Starting ETO
- Signing on and queue LTERM allocation
- Implementing ETO printer support
- Assigning output
- Tuning ETO for performance

Related Reading: For more information about administering ETO, see the *IMS Version 9: Administration Guide: Transaction Manager*.

Chapter 23. Customizing IMS

Customizing IMS is the process whereby you tailor IMS functions to fit the specific needs of your installation.

Some customization is required before you can use IMS, but most is optional. If you elect not to do optional customization on a specific IMS function, that function then works according to IBM-provided defaults. If IMS is new to your installation, you initially might want to do a minimum amount of optional customization. You can use IMS and you can do additional customization when performance or IMS capabilities indicate the need to do so.

There are a number of ways for you to customize IMS. IMS provides:

- Initialization values you can change.
- Macros you can use.
- Procedures you can change.
- Exits (places in its logic flow) for which you can write exit routines to perform special processing. IMS calls exit routines at various points and allows you to control how IMS performs its work.

One example of customizing might be to interrogate an in-flight message (in an MSC environment) to determine if it originated from a test or production system. If it came from a test system, you might want to route that message to an application that collects data about the test system. If the message originated from a production system, you probably would not want to change its destination.

To perform the interrogation of the message, you would code a TM and MSC Message Routing and Control User Exit routine. If you name this routine DFSMSCE0, place it in the IMS.SDFSRESL library, and bind it to that library (or a concatenated library), IMS will call your exit routine when IMS receives the message. When your exit routine is done with its processing, the exit routine returns control to IMS and then IMS resumes processing the message to either the original destination or the test application.

Certain exit routines are required and others are optional. Some IBM-supplied exit routines can be used as is and some require modification before using.

Some of the functions you can perform using exit routines are:

- To edit messages
- To check security
- To edit transaction code input, message switching input, and physical terminal input and output
- To perform additional application clean-up
- To initialize dependent regions
- To control the number of buffers the RECON data sets use
- To keep track of segments that have been updated

Table 16 on page 246, Table 17 on page 246, and Table 18 on page 250 list most of IMS's exits and briefly describe what the exits can be used for.

Related Reading: For all the details pertaining to all the IMS exits, see the *IMS Version 9: Customization Guide*.

Table 16. IMS DB Exit Routines and Their Uses

Exit Name	Exit Use Description
Data Capture Exit Routine	Receives control whenever a segment, for which the exit routine is defined, is updated. Possible use is to enable replication of that data to a relational DB2 database.
Data Conversion User Exit Routine (DFSDBUX1)	Receives control at the beginning of a DL/I call and at the end of the call. In the exit routine, you can modify segment search arguments, the key feedback area, the I/O area, and the status code.
Data Entry Database Randomizing Routine (DBFHDC40/ DBFHDC44)	Required for placing root segments in or retrieving them from a DEDB.
Data Entry Database Resource Name Hash Routine (DBFLHSH0)	Is used with the Internal Resource Lock Manager (IRLM) and enables IMS and DBCTL to maintain and retrieve information about the control intervals (CIs) used by sharing subsystems.
Data Entry Database Sequential Dependent Scan Utility Exit Routine (DBFUMSE1)	Allows you to copy and process a subset of the number of segments that the utility scans at a particular time.
HALDB Partition Selection Exit Routine (DFSPSE00)	Used to select partitions by some criteria other than high key.
HDAM and PHDAM Randomizing Routines (DFSHDC40)	Required routine for HDAM and PHDAM access methods for placing root segments in, or retrieving them from, HDAM or PHDAM databases.
Secondary Index Database Maintenance Exit Routine	Used to selectively suppress secondary indexing.
Segment Edit/Compression Exit Routine (DFSCMPX0)	Used to compress and expand segments of data.
Sequential Buffering Initialization Exit Routine (DFSSBUX0)	Used to dynamically control the use of Sequential Buffering (SB) for online and batch IMS subsystems, as well as DBCTL.

Table 17. IMS TM Exits and Their Uses

Exit Name	Exit Use Description
Build Security Environment Exit Routine (DFSBSEX0)	Tell IMS whether or not to build the RACF or equivalent security environment in an IMS dependent region for an application that has received its input message from neither OTMA nor an LU 6.2 device.
Conversational Abnormal Termination Exit Routine (DFSCONE0)	Used to clean up, if required, when a conversation is prematurely terminated.
Fast Path Input Edit/Routing Exit Routine (DBFHAGU0)	Used to determine the eligibility of an incoming message for Fast Path processing.
Front-End Switch Exit Routine (DFSFEBJ0)	Allows you to keep the input terminal in response mode while it is waiting for the reply from the processing system for messages entered in an IMS system by a front-end switchable VTAM node and processed in another system (such as IMS or CICS).

Table 17. IMS TM Exits and Their Uses (continued)

Exit Name	Exit Use Description
Global Physical Terminal (Input) Edit Routine (DFSGPIX0)	Message segments are passed one at a time to this edit routine so that the routine can process the segments in one of the following ways: • Accept the segment and release it for further editing by the IMS Basic Edit routine. • Modify the segment (for example, change the transaction code or reformat the message text) and release it for further editing by the IMS Basic Edit routine. • Cancel the segment. • Cancel the message and request that IMS send a corresponding message to the terminal operator. • Cancel the message and request that IMS send a specific message from the User Message Table to the terminal operator.
Greeting Messages Exit Routine (DFSGMSG0)	Allows you to tailor how IMS handles messages issued during the logon and signon process.
IMS Adapter for REXX Exit Routine (DFSREXXU)	Has the ability to: • Override the exec name to be executed. This name defaults to the IMS program name. • Choose not to execute any exec and have the IMS adapter for REXX return to IMS. • Issue DL/I calls using the AIB interface as part of its logic in determining what exec to execute. • Set REXX variables (through IRXEXCOM) before the exec is started. • Extract REXX variables (through IRXEXCOM) after the exec ends. • Change the initial default IMSRXTRC tracing level.
Initialization Exit Routine (DFSINTX0)	Used to create two user data areas that can be used by some of your installation's exit routines.
Input Message Field Edit Routine (DFSME000)	Used to perform common editing functions such as numeric validation or conversion of blanks to numeric zeros.
Input Message Segment Edit Routine (DFSME127)	Used to perform common editing functions such as numeric validation or conversion of blanks to numeric zeros
Logoff Exit Routine (DFSLGFX0)	Used to perform processing that complements the Logon exit routine (DFSLGNX0)
Logon Exit Routine (DFSLGNX0)	Enables you to control the way logons are processed.
LU 6.2 Edit Exit Routine (DFSLUEE0)	Enables you to edit input and output LU 6.2 messages for IMS-managed LU 6.2 conversations.
Message Control/Error Exit Routine (DFSCMUX0)	Used to control transactions, responses, and message switches that are in error.
Message Switching (Input) Edit Routine (DFSCNTE0)	Similar to the Transaction Code (Input) Edit and is capable of message switching.
Non-Discardable Messages Exit Routine (DFSNDMX0)	Used to tell IMS what to do with the input message associated with an abended application program.

Table 17. IMS TM Exits and Their Uses (continued)

Exit Name	Exit Use Description
OTMA Destination Resolution Exit Routine (DFSYDRU0)	Used to determine and change the final destination of OTMA member names or Tpipe names that are used for OTMA asynchronous output messages.
OTMA Input/Output Edit Exit Routine (DFSYIOE0)	Used to modify or cancel OTMA input and output messages.
OTMA Prerouting Exit Routine (DFSYPRX0)	Used to determine whether an asynchronous output message needs to be routed to an OTMA destination or a non-OTMA destination.
Output Creation Exit Routine (DFSINSX0)	Used to validate both an unknown destination for a message and the creation of an unknown user.
Physical Terminal (Input) Edit Routine (DFSPIXT0)	Message segments are passed one at a time to the Physical Terminal Input edit routine, and the edit routine can handle them in one of the following ways: • Accept the segment and release it for further editing by the IMS basic edit routine. • Modify the segment and release it for further editing by the IMS basic edit routine. • Cancel the segment. • Cancel the message and request that the terminal operator be notified accordingly. • Cancel the message and request that a specific message from the User Message Table be sent to the terminal operator.
Physical Terminal (Output) Edit Routine (DFSCTTO0)	Used to edit output messages immediately before they are sent to a terminal.
Queue Space Notification Exit Routine (DFSQSPC0/DFSQSSP0)	This exit routine is activated when a logical record is assigned to or released from a message queue data set. This routine causes a message to be issued when one of following occurs: • The number of records currently in use exceeds upper threshold percentage value of the maximum number assignable before initiation of automatic shutdown. • The number of records currently in use falls below the lower threshold percentage value of the same maximum.
Security Reverification Exit Routine (DFSCTSE0)	Used to reevaluate transaction authorization checking on the DL/I CHNG Call.
Shared Printer Exit Routine (DFSSIML0)	Used to decide whether a terminal that is unavailable can be automatically acquired by IMS or an AOI application program.
Sign-On Exit Routine (DFSSGNX0)	Used for sign-on processing if ETO=Y is specified.
Signoff Exit Routine (DFSSGFX0)	Used for sign-off processing.
Sign On/Off Security Exit Routine (DFSCSGN0)	Used to verify a user's ID and password. This exit routine can conflict with the Sign-On exit routine (DFSSGNX0).
Time-Controlled Operations (TCO) Exit Routine (DFSTXIT0)	Used to insert messages that are the commands, transactions, and message switches that you specify in the time schedule requests and message sets that make up a script member.

Table 17. IMS TM Exits and Their Uses (continued)

Exit Name	Exit Use Description
TM and MSC Message Routing and Control User Exit Routine (DFSMSCE0)	Used to: • Provide maximum routing control for TM and MSC messages. • Ease TM and MSC coding and maintenance requirements, and reduce the number of exit modules. • Support a consistent set of routing capabilities across all of the exit entry points (or functions). • Provide a common parameter list interface and linkage interface to the various entry points (or functions). • Provide the ability to append an optional user prefix segment to TM and MSC messages which TM and MSC user exit routines can use to communicate and control user-customized routing needs. • Logs routing errors and footprints in the message to indicate those exit routines that reroute the message.
Transaction Authorization Exit Routine (DFSCTRN0)	Works with the Security Reverification exit routine (DFSCTSE0) and the Sign On/Off Security exit routine (DFSCSGN0) to check an individual user ID for authority to use a transaction.
Transaction Code (Input) Edit Routine (DFSCSMB0)	Used to edit input messages before they are enqueued for scheduling.
Type 1 Automated Operator Exit Routine (DFSAOUE0)	This AO exit routine is called continuously for system messages destined for the master terminal, operator-entered commands, and command responses. Use it to: • Ignore selected segments or an entire message. • Send a copy of a system message, command, or command response to an alternate destination. • Send a new message to an alternate destination for a system message, command, or command response. • Change a system message. • Change a system message and send a copy to an alternate destination. • Change a copy of a command or command response and send the copy to an alternate destination. • Delete a system message. • Delete a system message and send a copy to an alternate destination. • Request the edited command buffer (when the input is a command).
2972/2980 Input Edit Routine (DFS29800)	Required to perform terminal-related functions inherent in the design of the 2972/2980 General Banking Terminal system.
4701 Transaction Input Edit Routine (DFS36010)	Appends a blank and the eight-byte node name to a transaction input message.

Table 18. IMS System Exits and Their Uses

Exit Name	Exit Use Description
Application Group Name (AGN) Exit Routine (DFSISIS0)	Provides users without RACF a mechanism for checking authorization to IMS application group names (AGNs). Recommendation: Use the Resource Access Security Exit Routine (DFSRAS00) instead of this exit routine for AGNs.
Buffer Size Specification Facility (DSPBUFFS)	Allows you to control the number of buffers used for RECON data sets when either the local shared resource (LSR) or the nonshared resource (NSR) buffering option is used.
Command Authorization Exit Routine (DFSCCMD0)	Used to verify that a command is valid from a particular origin.
DBRC Command Authorization Exit Routine (DSPDCAX0)	Used in conjunction with RACF or another security product to determine the success or failure of DBRC command authorization.
Dependent Region Preinitialization Routines	Dependent Region Preinitialization routines enable you to perform any application-unique dependent region initialization.
Dump Override Table (DFSFDOT0)	Used to either force or suppress dumps for specified abends.
ESAF Indoubt Notification Exit Routine (DFSFIDN0)	Used to resolve in-doubt work before restarting a failed IMS.
IMS Command Language Modification Facility (DFSCKWD0)	Used to modify the command keyword table.
Large SYSGEN Sort/Split Input Exit Routine (DFSSS050)	Enables you to alter the resource data for user-generated resources.
Large SYSGEN Sort/Split Output Exit Routine (DFSSS060)	Enables you to alter the resource data for user-generated resources.
Log Archive Exit Routine	Used to produce an edited subset of the complete IMS log.
Log Filter Exit Routine (DFSFTFX0)	Allows you to control the amount of log data sent to an RSR tracking subsystem, by acting as a filter.
Logger Exit Routine (DFSFLGX0)	Used to process log data for recovery purposes.
Partner Product Exit Routine (DFSPPUE0)	Used to initialize products that run with IMS.
RECON I/O Exit Routine (DSPCEXT0)	Tracks changes to the RECON data set, which you can log in a journal.
Resource Access Security Exit Routine (DFSRAS00)	Used to authorize IMS resources such as transactions, PSBs, or output LTERM names. Recommendation: Use this exit routine instead of the Application Group Name Exit Routine (DFSISIS0) for AGNs.
SCI Registration Exit Routine (DSPSCIX0)	Used by DBRC to perform an authorization check before allowing a potential SCI client to register with SCI.
System Definition Preprocessor Exit Routine (Input Phase) (DFSPRE60)	Used to alter, insert, or delete data from stage 1 input before the Preprocessor record scan occurs.
System Definition Preprocessor Exit Routine (Name Check Complete) (DFSPRE70)	Used to build tables that contain resource names that have been cross-checked.

Table 18. IMS System Exits and Their Uses (continued)

Exit Name	Exit Use Description
Type 2 Automated Operator Exit Routine (DFSAOE00)	Used to: • Modify the text of IMS system messages. • Delete IMS system messages. • Direct any message, command, or command response to an Automated Operator (AO) application. • Start a BMP job (for example, an AO application).
User Message Table (DFSCMTU0)	Used to create your own messages and list them in your own message table.

Chapter 24. IMS Security

This chapter covers some of the issues regarding IMS security.

Related Reading For more information about IMS security, see:

- Chapter 4, “Establishing IMS Security”, in the *IMS Version 9: Administration Guide: System*
- *IMS Version 9: Installation Volume 2: System Definition and Tailoring*
- *IMS Security Guide*
- *z/OS V1R4 Security Server RACF Security Administrator’s Guide*

The following sections are covered in this chapter:

History of IMS Security

When IMS was developed, security products like the Resource Access Control Facility (RACF), had not been developed, or were not in use by most installations. It was common during this period to have each subsystem implement its own security. Therefore, the IMS product offered some basic levels of protection for IMS resources. These internal IMS security facilities (for example, the Security Maintenance Utility or SMU) are available for protecting many IMS resource types and are used by some IMS installations today.

Recommendation: IBM recommends that you implement security using only RACF or an equivalent security product because IMS Version 9 is the last version of IMS to support the SMU.

With the development and introduction of security products, like RACF, more and more installations have implemented security for IMS resources using security products. Two advantages of using a security product for securing access to resources are:

- One product can be used to implement the security requirements for multiple subsystems, such as IMS, CICS, and other subsystems.
- All of the security information can be kept and maintained in one place, like the RACF database. One centralized database repository containing all the installations’ security specifications eliminates, or significantly minimizes, the problems inherent with using individual products’ security functions, namely:
  - Duplicating and distributing security information among several subsystems, and
  - Coordinating the security enforcement functions implemented in multiple products

RACF offers a wide range of security choices to the installation. For example, RACF contained new security features, such as user identification (userid) and verification based security, which is not available through IMS internally provided SMU security.

Security Overview

When you initiate security safeguards, you must balance requirements between those responsible for the security of resources and those users who legitimately need access to those resources. Because an individual assigned to resource security is held responsible for resources that might be compromised, that person should not allow easy access to dominate protection measures. On the other hand, users performing their assigned tasks need convenient access to the resources. The users and the security specialist should work out a balanced approach between the ease of resource access and the complexity of protecting that resource.

In an IMS system, you should consider various facets of the security implementation:

- The resource name. For example, a user might be allowed access to the Part database but not to the Customer Order database.
- Level of access: what the user can do to the resource. For example, a user might be allowed to read a file but not to update it.

IMS provides a system definition macro (the SECURITY macro) that allows the installation to code all of the security specifications on one macro. The SECURITY macro is used to specify security options for IMS internally provided SMU security, RACF security, an installation provided security exit routine, or any combination of these facilities.

IMS provides ample flexibility in allowing the installation to secure any type of resource.

Before you decide what security facilities to use in designing a secure IMS system, you should know which resources within the system need protection. In other words, you should decide what to protect before you decide how to protect it.

Securing Resources

Table 19 lists the:

- Resources you can protect
- Valid security options for that resource
- Facilities available to protect that resource
- Applicable environments

Table 19. Resources and the Facilities to Protect Them

Resources	Security Options/Type of Protection	Facilities	Valid Environments
Command	Default terminal security	System definition	DB/DC, DCCTL
	LTERM security[1]	SMU	DB/DC, DCCTL
	Password security[1]	SMU	DB/DC, DCCTL
	Transaction command security	SMU	DB/DC, DCCTL
	Input access security	RACF	DB/DC, DCCTL
	IMSplex command security	RACF	DB/DC, DCCTL
	DBRC command authorization[3]	RACF or exit routine	DB/DC, DCCTL

Table 19. Resources and the Facilities to Protect Them (continued)

Resources	Security Options/Type of Protection	Facilities	Valid Environments
Database	Segment sensitivity	PSBGEN RACF	DB/DC, DCCTL, DBCTL
	Field sensitivity	PSBGEN RACF	DB/DC, DCCTL, DBCTL
	Password security (for `/LOCK`, `/UNLOCK` commands)	SMU *or* RACF	DB/DC, DCCTL, DBCTL
Dependent region	Application group name (AGN) security	SMU and exit routine *or* SMU and RACF	DB/DC, DCCTL, DBCTL
	APSB security	RACF	DB/DC, DCCTL, DBCTL
	Resource Access Security (RAS)	RACF	DB/DC, DCCTL, DBCTL
IMS online system (control region)	Extended resource protection (using APPL resource class)	RACF	DB/DC, DCCTL, DBCTL
LTERM[1]	Password security (for `/IAM`, `/LOCK`, `/UNLOCK` commands)	SMU *or* RACF	DB/DC, DCCTL
	AGN security	SMU and exit routine *or* SMU and RACF	DB/DC, DCCTL
	RAS security	RACF	DB/DC, DCCTL
LU 6.2 inbound and IMS-managed outbound conversations	Allocate verification security	RACF and exit routine	DB/DC, DCCTL
	Input access security	RACF and exit routine	DB/DC, DCCTL
Online application program	Password security (for `/IAM`, `/LOCK`, `/UNLOCK` commands)	SMU *or* RACF	DB/DC, DCCTL
	Extended resource protection (using APPL keyword)	RACF	DB/DC, DCCTL
PSB	AGN security	SMU and exit routine *or* SMU and RACF	DB/DC, DCCTL, DBCTL
	RAS	RACF	DB/DC, DCCTL, DBCTL
	APSB security	RACF[2]	DB/DC, DCCTL
PTERM[1]	Signon verification security	SMU and Exit Routine *or* RACF and Exit Routine	DB/DC, DCCTL
	Terminal-user security	RACF	DB/DC, DCCTL
	Password security (for `/IAM`, `/LOCK`, `/UNLOCK` commands)	SMU *or* RACF	DB/DC, DCCTL
System data set	OS password protection	OS/390	DB/DC, DCCTL, DBCTL
	Data set protection (VSAM) (using PERMIT, RDEFINE classes)	RACF	DB/DC, DCCTL
Terminals defined with ETO	Signon verification security	RACF and Exit Routine	DB/DC, DCCTL
	Input access security	RACF and Exit Routine	DB/DC, DCCTL

Table 19. Resources and the Facilities to Protect Them (continued)

Resources	Security Options/Type of Protection	Facilities	Valid Environments
Transaction	LTERM security[1]	SMU	DB/DC, DCCTL
	AGN security	SMU and exit routine *or* SMU and RACF	DB/DC, DCCTL
	Input access security	RACF	DB/DC, DCCTL
	RAS	RACF	DB/DC, DCCTL
	Password security[1] (for `/LOCK`, `/UNLOCK` commands)	SMU *or* RACF	DB/DC, DCCTL
Type 1 Automated Operator Interface (AOI) applications	Transaction command security	SMU *or* RACF and Command Authorization exit routine	DB/DC, DCCTL
Type 2 AOI applications	Transaction command security	RACF and Command Authorization exit routine	DB/DC, DCCTL

Notes:

1. Static terminals only. Not applicable to ETO-defined terminals.
2. Using RACF to secure APSBs applies to CPI-C driven applications only.
3. DBRC Command Authorization is an additional command security option for DBRC commands only. DBRC commands are also subject to any other command security options active in the IMS system.

Chapter 25. IMS Logging

During IMS execution, all information necessary to restart the system in the event of hardware or software failure is recorded on a system log data set. The following critical system information is recorded on the logs:

- The receipt of an input message in the input queue
- The start of an MPP or BMP
- The receipt of a message by the MPP for processing
- Before and after images of data base updates by the MPP or BMP
- The insert of a message into the queue by the MPP
- The termination of an MPP or BMP
- The successful receipt of an output message by the terminal

The following sections are covered in this chapter:

- "Checkpoints"
- "Database Recovery Control (DBRC)"
- "IMS Log Components"

Checkpoints

At regular intervals during IMS execution, checkpoints are written to the log without having to wait to do any physical I/O. A checkpoint is taken after a specified number of log records are written to the log since the previous checkpoint, or after a checkpoint command. Special checkpoint commands are available to stop IMS in an orderly manner.

Database Recovery Control (DBRC)

DBRC is an integral part of IMS logging and, as time moves on, DBRC is becoming a prominent part of IMS's daily operation.

DBRC keeps information about all of IMS's logging activities in the recovery control (RECON) data sets.

Related Reading: For more information about DBRC, see Chapter 26, "Database Recovery Control (DBRC)," on page 263 and the *IMS Version 9: DBRC Guide and Reference*.

IMS Log Components

The IMS logs are made up of a number of components, which are described in the following sections:

IMS Log Buffers

The log buffers are used for IMS to write any information required to be logged, without having to do any real I/O.

Whenever a log buffer is full, the complete log buffer is scheduled to be written out to the OLDS as a background, asynchronous task. In a busy system, IMS will generally chain these log buffer writes together.

Should any application or system function require a log record to be externalized (that is, IMS believes that for recoverability, this log record must be physically written to DASD before proceeding), then the WADS data set is used. See "Write-Ahead Data Sets (WADS)" on page 260.

The OLDS buffers are used in such a manner as to keep available as long as possible the log records that may be needed for dynamic backout. If a needed log record is no longer available in storage, one of the OLDS buffers will be used for reading the appropriate blocks from the OLDS.

The number of log buffers is an IMS start-up parameter, and the maximum is 999. The size of each log buffer is dependent on the actual blocksize of the physical OLDS. The IMS log buffers now reside in extended private storage, however, there is a log buffer prefix that still exists in ECSA.

Online Log Data Sets (OLDS)

The OLDS are the data sets which contain all the log records required for restart and recovery. These data sets must be pre-allocated (but need not be pre-formatted) on DASD and will hold the log records until they are archived.

The OLDS is written by BSAM. OSAM is used to read the OLDS for dynamic backout

The OLDS are made up of multiple data sets which are used in a wrap around manner. At least 3 data sets must be allocated for the OLDS to allow IMS to start, while an upper limit of 100 is supported.

Only complete log buffers are written to the OLDS, to enhance performance. Should any incomplete buffers need to be written out, they are written to the WADS. The only exceptions to this are at IMS shutdown, or in degraded logging mode, when the WADS are unavailable, then the WADS writes will be done to the OLDS.

All OLDS should be dynamically allocated, by using the DFSMDA macro, and not hardcoded in the IMS control region JCL.

Dual Logging of OLDS

Dual logging can also be optionally implemented, with a primary and secondary data set for each defined OLDS.

- A primary and secondary data set will be matched and, therefore, the pair should have the same space allocation. Because an OLDS pair will contain the same data, extra space allocated to one will not be used in the other.
- Secondary extent allocation cannot be used
- OLDS can be allocated on different supported DASD
- All OLDS must have the same blocksize, and be a multiple of 2Kb (2048 bytes). the maximum allowable blocksize is 30kb.

Dynamic Backout

In addition to the above logging, all previous database record images are written to the OLDS, and can also be used for dynamic back-out processing of a failing MPP or BMP. As soon as the MPP or BMP reaches a synchronization point, the dynamic log information of this program is discarded.

Archiving

The current OLDS (both primary and secondary) is closed and the next OLDS is used whenever one of the following situations occurs:

- OLDS becomes full
- I/O error occurs
- MTO command is entered to force a log switch (such as /SWI OLDS)
- MTO command is issued to close a database (such as /DBR DB) without specifying the NOFEOV parameter

DBRC is automatically notified that a new OLDS is being used. When this occurs, IMS can automatically submit the archive job to IMS Log Archive utility by using an IMS startup parameter (ARC=).

IMS can define whether the log archive process will occur with every log switch, or every second log switch, and the DBRC skeletal JCL that controls the archiving, can be defined to also create 1 or 2 System Log data sets, and 0, 1 or 2 Recovery Log Data sets. After the last allocated OLDS has been used, the first OLDS will again be used in a wrap around fashion, as long as it has been archived.

The IMS log archive JCL is in DBRC skeletal JCL, and can be tailored to create the required SLDS, and optionally dual SLDS, 1 or 2 RLDS data sets, and any user data sets. Figure 90 shows the data sets for the Log Archive utility.

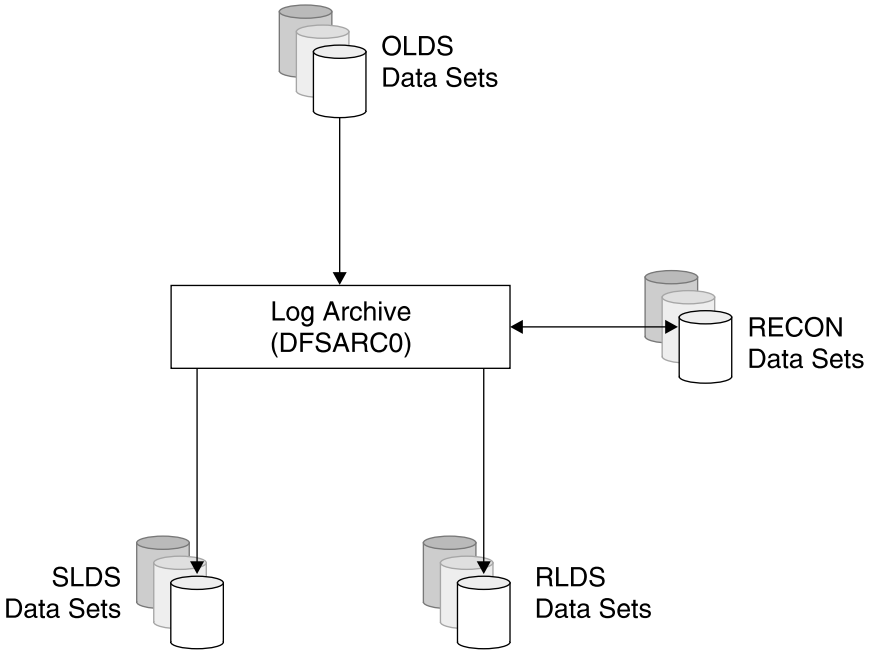

Figure 90. Inputs and Outputs of the IMS Log Archive Utility

Related Reading: For the details of the IMS Log Archive utility, see the *IMS Version 9: Utilities Reference: System*.

OLDS I/O Errors

In the case of a write error, the subject OLDS (or pair of OLDS) will be put into a stopped status and will not be used again. This is equivalent to a user issuing the command /STO OLDS.

If using dual OLDS, then the data set without error will be used for IMS archives.

If data set errors result in only a single OLDS remaining, a /CHE FREEZE command is internally scheduled by IMS. If an error occurs on the very last OLDS, IMS will abend with a U0618.

Information is kept in the RECON data set about the OLDS for each IMS system. The data in the RECON indicates whether an OLDS contains active log data which must be archived, or whether it is available for use.

Lack of OLDS

IMS issues messages when it is running out of OLDS.

During the use of the last available OLDS, IMS will indicate that no spare OLDS are available

When all the OLDS are full, and the archives have not successfully completed, then IMS will stop, and have to wait until at least 1 OLDS has been archived. The only thing IMS will do is repeatedly issue messages to indicate that it is has run out of OLDS, and is waiting.

Write-Ahead Data Sets (WADS)

The WADS is a small direct access data set which contains a copy of committed log records which are in OLDS buffers, but have not yet been written to the OLDS.

When IMS processing requires writing of a partially filled OLDS buffer, a portion of the buffer is written to the WADS. If IMS or the system fails, the log data in the WADS is used to terminate the OLDS, which can be done as part of an Emergency Restart, or as an option on the IMS Log Recovery Utility.

The WADS space is continually reused after the appropriate log data has been written to the OLDS. This data set is required for all IMS systems, and must be pre-allocated and formatted at IMS start-up when first used.

In addition, the WADS provide extremely high performance. This is achieved primarily through the physical design of the WADS. Each WADS track is divided into 2080 byte blocks with a 1 byte key. Each block has the same key (key value = 0). This was done for efficiency on conventional rotational DASD, and is still valid for newer types of DASD.

All WADS should be dynamically allocated by using the DFSMDA macro, and not hardcoded in the control region JCL.

All the WADS must be on the same device type and have the same space allocation.

Dual WADS

Dual WADS is supported to provide backup in the event of a read error while terminating the OLDS from the WADS. The primary and secondary WADS will contain the same data. Single or Dual WADS logging is determined from an IMS start-up parameter.

Redundant WADS

Regardless of whether there are single or dual WADS, there can be up to 10 WADS defined to any IMS. (WADS0, WADS1,...., WADS9).

WADS0 (and WADS1 if running dual WADS) are active, and the rest remain as spares in case any active WADS has an I/O error. The next spare will then replace the one with the error.

System Log Data Sets

The SLDS is created by the IMS log archive utility, possibly after every OLDS switch. It is usually placed on TAPE or CARTIDGE, but can reside on DASD. The SLDS can contain the data from one or more OLDS data sets.

The SLDS can also be used as input to all IMS log utilities, and IMS restart.

Information about SLDS is maintained by DBRC in the RECON data set. Calls to DBRC are made by the Archive Utility identifying the OLDS being archived and the SLDS being created. OLDS that have been archived are then available for reuse by IMS.

Dual SLDSs

Dual archiving to 2 SLDS data sets (primary and secondary) is supported.

When archiving to TAPE or CARTRIGE, the user can also force the primary and secondary volumes to contain the same data by specifying the number of log blocks per volume. When this number is reached, a force-end-of-volume (FEOV) will occur on both the primary and secondary SLDS. In this way, both primary and secondary SLDS are identical and interchangeable should a subsequent I/O error occur on one of them.

The user can also specify which records are copied from the OLDS to the SLDS. Generally, the SLDS should contain all the log records from the OLDS, but if the user wants to omit types of log records from the SLDS, these can be specified within the log archive utility.

The SLDS must always contain those log records required for database recovery, batch backout or system recovery.

The blocksize of the SLDS is independent of the OLDS blocksize, and can be specified to maximize space on the SLDS device type.

Recovery Log Data Sets

When the IMS log archive utility is run, the user can request creation of an output data set that contains all of the log records needed for database recovery. This is the RLDS and is also known to DBRC.

The RLDS is preferred by many instillations. All database recoveries and change accumulation jobs will always use the RLDS if one exists, and this can considerably speed up any of these processes because the only contents of these data sets are

database recovery log records. All other IMS TM, application scheduling and checkpoint log records are not included on the RLDSs.

The RLDS is optional, and you can also have dual copies of this, in a similar way to the SLDS.

Chapter 26. Database Recovery Control (DBRC)

DBRC includes the IMS functions that:

- Ensure IMS system and database integrity.
- Control log and database recovery.
- Control access to databases by various IMS subsystems sharing those databases.

DBRC is responsible for:

- Controlling logs for an online IMS.
- Recording database recovery information.
- Verifying that database utilities have the correct input.
- Preventing or allowing use of a database (authorization).
- Optionally, generating JCL for some of the database utilities.

Related Reading: This chapter contains an overview of DBRC. For all the details associated with DBRC, see *IMS Version 9: DBRC Guide and Reference*.

The following sections are covered in this chapter:

Overview of DBRC

DBRC records information in a set of VSAM data sets called the recovery control (RECON) data sets (see "Overview of the RECON Data Sets" on page 266).

IMS records the following information in the RECON data sets:

- Log data set information
- Database data set information
- Event information, such as:
  - Allocation of a database
  - Update of a database
  - Image copy of a database
  - Abend of a subsystem
  - Recovery of a database

- Reorganization of a database
- Archive of a OLDS data set

Using DBRC

The aspects pertaining to DBRC usage are discussed in the following sections:

DBRC Options

The first option is whether the DBRC function is active in address spaces executing IMS and the second option is whether databases must be registered in the RECON data set.

1. DBRC is always active in an IMS control region (DBCTL, DCCTL, or DB/DC). This cannot be overridden.

   For batch and utility regions, module DFSIDEF0 can be used to specify whether or not DBRC is active. In DFSIDEF0, you can set DBRC= to YES, NO, or FORCE.

   YES | NO
   This sets the default for DBRC usage for batch execution, it can be overridden at execution time on the DBRC EXEC parm (unless, of course, you defined DBRC=FORCE).

   FORCE
   This forces DBRC usage in all other address spaces. It cannot be overridden in the JCL. Any job attempting to run with DBRC=N abends. There are also YES,NO options, but these are only valid for a Batch IMSGEN, not a DBCTL IMSGEN, for DBCTL you must have DBRC support generated (even if its not forced for batch).

   A BMP does not have a DBRC execution parameter. DBRC is always active in the IMS control region that the BMP connects to.

   The above parameters control whether DBRC is active in an address space. The level of functions available is controlled by information in the RECON.

2. The FORCER/NOFORCER (force registration or not) option in the RECON header controls whether or not databases **must** be registered in the RECON.
   - If NOFORCER is specified, databases might, or might not be registered in the RECON. If a database is not registered in the RECON and DBRC is active, you get a warning message each time the database is opened.
   - If FORCER is specified, then, if DBRC is active in the address space, all databases must be registered in the RECON. If the database is not registered, DBRC rejects authorization and the job abends (in DBCTL environments, the PSB fails to schedule, but the DBCTL region stays up).

   When running with DBRC=N, there is an incomplete record of updates, which is useless for recovery purposes.

   If a database is registered in the RECON and you run a job with DBRC=N, the next time you run a job with DBRC=Y a warning message is issued flagging the fact the database has been accessed outside of DBRC control. You might want to take an image copy at that point.

Communicating with DBRC

Use DBRC commands or DBRC API requests to obtain services from DBRC. The following sections discuss these commands and requests in more detail.

DBRC Commands

Use DBRC batch and online commands to:

- Add to, change, and delete information in the RECON data sets
- Generate the JCL and the control statements necessary to run the various IMS utilities used in database recovery

The following is a list of the DBRC batch commands:

- `BACKUP.RECON`
- `CHANGE`
- `DELETE`
- `GENJCL`
- `INIT`
- `LIST`
- `NOTIFY`
- `RESET.GSG`

These batch commands are issued to DBRC by including them in the JCL job that runs the Database Recovery Control utility (DSPURX00).

A variation of some of the DBRC batch commands can be issued online using the `/RMxxxxxx` command (for example, `/RMCHANGE.RECON`). Although most of the command examples in this book feature the DBRC batch commands, the online version (where appropriate) of the example command can be used instead.

Related Reading: For more information about the DBRC commands, see the chapter titled “DBRC Commands” in the *IMS Version 9: DBRC Guide and Reference*.

DBRC Application Programming Interface (API)

Use DBRC API requests to:

- Start and stop DBRC
- Query information from the RECON

The following is a list of the DBRC API requests:

- `STARTDBRC`
- `QUERY`
- `RELBUF`
- `STOPDBRC`

Related Reading: For more information about the DBRC application programming interface and the API requests, see the chapter titled “Using the DBRC API” in the *IMS Version 9: DBRC Guide and Reference*.

Database Authorization

A DBRC sharing environment introduces the concept of database authorization. This process determines if an online IMS or batch IMS can have access to the requested databases. DBRC authorizes or refuses to authorize access to the databases depending on the current authorizations and the access intent of the IMS system.

Access Intent

Access intent is determined by DBRC when IMS tries to allocate a database:

- For a batch job, DBRC uses the processing option (PROCOPT) of the PSB for each database to determine the access intent. If the PSB has multiple PCBs for the same database, the highest intent for that database is used.
- For an IMS TM online system, the ACCESS parameter of the DATABASE macro sets the access intent. This access intent can be changed by issuing a `/STA DB` command.

There are four processing intent attributes. These attributes are listed below in reverse order, from the highest access intent (the most restrictive) to the lowest (the least restrictive):

EX (exclusive)
The IMS system requires exclusive access of the database and no sharing is allowed regardless of the share options registered in DBRC.

- PROCOPT of L or xE (batch) (where x = A,D,G,I,D)
- ACCESS of Ex (online)

UP (update)
The IMS system can update the database. Even if no updates actually take place, the database is held in update mode. Any logs created with actual changes during this process are required for recovery or change accumulation.

- PROCOPT of A,I,R,D (batch)
- ACCESS of UP (online)

RD (read with integrity)
The IMS system only reads the database, but it also checks any enqueue or lock held by other IMS systems. It waits for the lock to be released before processing.

- PROCOPT of G (batch)
- ACCESS of RD (online)

RO (read without integrity)
The IMS system only reads the database and it does not check for any lock or enqueue held by other IMS systems.

- PROCOPT of GO (batch)
- ACCESS of GO (online)

Overview of the RECON Data Sets

The RECON data set is the most important data set for the operation of DBRC and data sharing. The RECON data set holds all resource information and event tracking information that is used by IMS.

The RECON data set can consist of one, two, or three data sets:

1. The original data set
2. The copy of the original data set
3. The spare data set

The original data set and the copy are a pair of VSAM clusters that work as a pair to record information. One is a duplicate of the other. A third RECON can be used

as a spare. IMS normally works with two active RECON data sets. If one becomes unavailable, the spare will be activated if it is available.

Important: The best solution, from an availability point of view, is to use all three data sets. This is strongly recommended. Using three data sets for the RECON causes DBRC to use them in the following way:

- The first data set is known as copy1. It contains the current information. DBRC always reads from this data set and when some change has to be applied, the change is written first to this data set.
- The second data set is known as copy2. It contains the same information as the copy1 data set. All changes to the RECON data sets are applied to this copy2 only after the copy1 has been updated.
- The third data set (the spare) is used in the following cases:
  - A physical I/O error occurs on either copy1 or copy2.
  - DBRC finds, when logically opening the copy1 RECON data set, that a spare RECON has became available, and that no copy2 RECON data set is currently in use.
  - The following command is executed:

    ```
    CHANGE.RECON REPLACE(RECONn)
    ```

    When the third RECON data set is used, the remaining valid data set is copied to the spare. When the copy is finished the spare becomes whichever of the data sets was lost, missing, or in error.

Note: From the RECON point of view, the copy1 and the copy2 data sets are normally identified by a 1 or a 2 in a field of the RECON header information.

RECON Records

The individual records in the RECON contain the information pertaining to the various items that DBRC keeps track of. The following section (“Types of RECON Records”) briefly introduces the records kept in the RECON.

Related Reading: For complete information about the RECON records, see the *IMS Version 9: DBRC Guide and Reference*.

Types of RECON Records

There are six general types of RECON records:

Control records

Control records are used for controlling the RECON data set and the default values used by DBRC. This class of records includes the RECON header record and header extension record.

Log records

Log records are used for tracking the log data sets used by all subsystems. This class of records includes:

- Primary Recovery Log (PRILOG) and Secondary Recovery Log (SECLOG) records (including interim log records) that describe a recovery log data set (RLDS) created by an IMS TM system, a CICS online system, a batch DLI job, or the Log Archive utility (DFSUARC0).
- Log Allocation (LOGALL) record that lists the DBDSs for which database change records have been written to a particular log.
- Primary OLDS (PRIOLD) and Secondary OLDS (SECOLD) records (including interim OLDS records) that describe the IMS TM online data sets (OLDS) that are defined for use.

- Primary System Log (PRISLDS) and Secondary System Log (SECSLDS) records (including interim SLDS records) that describe a system log SLDS created by an IMS TM system.

Change accumulation records
Change accumulation records are used for tracking information about change accumulation groups. This class of records includes change accumulation group, execution, and data set records.

DBDS group records
Database data set group (DBDSGRP) records are used to define the members of a DBDS group. The only record type in this class is a DBDS group record.

Subsystem records
Subsystem record contains information about the subsystem and related recovery information including:

- Subsystem name and type (online or batch)
- IRLM identification
- Abnormal-end flag and the recovery-process start flag
- List of authorized databases
- Time stamp that correlates the subsystem entry with the appropriate log records

Database records
Database records are used to track the state of databases, DBDSs, and resources required for recovery of DBDSs. This class of records includes:

- Database record (IMS, HALDB, or PARTition)
- Area authorization record
- DBDS record (non-Fast Path or Fast Path)
- Allocation record
- Image copy record
- Reorganization record
- Recovery record

Database Related Information

A database and its associated data sets should only be defined in one RECON data set.

The fundamental principle behind the RECON data set is to store all recovery-related information for a database in one place. It is not possible to use multiple RECON data sets in the recovery processing for the same database.

IMS Systems and the RECON

An IMS system can only be connected to one set of RECON data sets.

All databases that are accessed by IMS TM systems under the control of DBRC must be registered in the RECON referenced by the online IMS system only if the RECON has the FORCER option set on.

All batch IMS systems that access any database accessed by the online IMS system should reference the same RECONs that are referenced by the online IMS system.

Database Names in the RECON

The database names (DBD names) defined in one RECON data set must all be unique.

The database records, stored in the RECON data set, are registered with a key based on the DBD name. Therefore, DBRC cannot be used to control both test and production databases, using the same RECON data sets, unless some naming convention is adopted.

As a general rule, more than one set of RECON data sets are necessary if all the following conditions are true:

- Multiple versions of the same database exist (for example, test and production).
- The same DBD name is used for the different versions of the database.
- More than one version of the databases can be used, but only one can be registered to DBRC in the RECON data set. The other versions are treated as not registered (unless FORCER has been set in the RECON).

The application of the previous rules usually results in the need for at least two different sets of RECON data sets: one shared between the production systems and one for the test systems.

Note: On the `INIT.DBDS` command, which is used to create the database data set record in the RECON, you must supply the database data set name (DSN). When IMS opens the database, DBRC checks the DSN against the name that is registered in the RECON. If this name does not match, DBRC treats this database as if it was not registered. In this case, the test database (with a DSN different than the production database, even if with the same DBD name) and data set name, can coexist with the production environment, but not under the control of the DBRC.

Defining and Creating the RECON Data Sets

The RECON data sets are VSAM KSDSs. They must be created by using the VSAM AMS utilities.

The same record size and CI size must be used for all the RECON data sets.

The RECON data sets should be given different FREESPACE values so that CA and CI splits do not occur at the same time for both active RECON data sets.

For availability, all three data sets should have different space allocation specifications. The spare data set should be at least as large as the largest RECON data set. Figure 91 on page 270 shows an example of a RECON data set definition.

```
DELETE STIMS220.RECONB

 SET LASTCC=0

 DEFINE CLUSTER (NAME(STIMS220.RECONB) -
        VOLUMES (SBV010)                   -
        INDEXED                          -
        KEYS (24 0)                      -
        CYLINDERS C5 2)                  -
        RECORDSIZE (128 32600)           -
        SPANNED                          -
        FREESPACE (30 80)                -
        CISZ(4096)                       -
        NOREUSE                          -
        NERAS SPEED REPL IMBD            -
        UNORDERED                        -
        SHAREOPTIONS (3 3))              -
 INDEX (NAME(STIMS220.RECONB.INDEX))     -
  DATA (NAME(STIMS220.RECONB.DATA))
```

Figure 91. Example of a RECON Data Set Definition

Initializing the RECON Data Sets

After the RECON data sets are created, they must be initialized by using the `INIT.RECON` command (issued with the DBRC Recovery Control utility). This causes the RECON header records to be written in both current RECON data sets.

The RECON header records must be the first records written to the RECON data sets because they identify the RECON data sets to DBRC.

Allocating RECON Data Sets to IMS Systems

There are two methods to allocate the RECON data set to an IMS system:

- Point to the RECON data sets by inserting the DD statements in the start-up JCL for the various IMS systems.
- Use dynamic allocation.

If a DD statement is specified for RECON, DBRC does not use dynamic allocation. Otherwise, DBRC uses dynamic allocation.

Recommendation: With multiple IMS systems sharing the same databases and RECON data sets, use dynamic allocation for both the RECON data sets and the associated databases. This ensures that:

- The correct and current RECON data sets are used.
- The correct RECON data sets are associated with the correct set of databases.

Dynamic allocation also makes the recovery of a failed RECON data set easier because DBRC dynamically de-allocates a RECON data set if a problem is encountered with it.

To establish dynamic allocation, a special member that names the RECON data sets must be added to IMS.SDFSRESL or to an authorized library that is concatenated to IMS.SDFSRESL. This is done using the IMS DFSMDA macro.

Figure 92 shows an example of the required macros for dynamic allocation of the RECON data sets.

```
//DYNALL JOB..
//STEP     EXEC IMSDALOC
//SYSIN    DD  *
     DFSMDA TYPE=INITIAL
     DFSMDA TYPE=RECON,DSNAME=PROD.RECON01,
                DDNAME=RECON1
     DFSMDA TYPE=RECON,DSNAME=PROD.RECON02,
                DDNAME=RECON2
     DFSMDA TYPE=RECON,DSNAME=PROD.RECON03,
                DDNAME=RECON3
```

Figure 92. Example JCL for Allocating RECON Data Sets Dynamically

RECON data sets are always dynamically allocated with DISP=SHR specified.

When using multiple RECON data sets (for example, test and production), be sure that each IMS uses the correct RECON data set group. This can be done by altering the SYSLMOD DD statement in the IMSDALOC procedure to place the dynamic allocation parameter lists for the various RECON data set groups in different IMS.SDFSRESL libraries. The appropriate IMS.SDFSRESL or concatenated IMS.SDFSRESL libraries must be included for each IMS start-up JCL.

Important: When multiple IMSs running on different processors are accessing the same RECON data set, the dynamic allocation parameter lists must be kept synchronized in the IMS.SDFSRESL libraries being used by the different processors. This does not happen automatically. Also, using dynamic allocation in some IMS systems and JCL allocation in others is not recommended.

Placement Considerations for the RECON Data Sets

The placement of the RECON data sets in the DASD configuration is very important. The primary rule is to configure for availability (or put another way, to isolate possible failures). This means, for example, to place all three RECON data sets on:

- Different volumes
- Different control units
- Different channels
- Different channel directors

Maintaining RECON Data Sets

There are several procedures and commands that can be used to maintain the RECON data set.

Backing Up the RECON

Operational procedures should be set up to ensure that regular backups of the RECON data set are taken.

These backups should be performed using the DBRC `BACKUP.RECON` DBRC utility command. The command includes a reserve mechanism to ensure that no updating of the RECON takes place during the backup. If possible, the backup should be taken when there are no IMS systems active.

The backup copy is created from the copy1 RECON data set. The command to create the backup copy invokes the AMS REPRO command, with its normal defaults and restrictions. For instance, the data set that is receiving the backup copy must be empty.

Deleting Inactive Log Records from the RECON

When DBRC becomes aware that an image copy has been taken of a database data set (DBDS), DBRC automatically deletes, reuses, or updates the records in the RECON that are associated with that particular DBDS. After this automatic processing, certain log records are considered inactive, but are not deleted from the RECON.

A log is considered inactive when the following conditions are all true:

- The log volume does not contain any DBDS change records that are more recent than the oldest image copy data set known to DBRC. This check is performed on a DBDS basis.
- The log volume was not opened in the last 24 hours.
- The log has either been terminated (nonzero stop time) or has the ERROR flag in the PRILOG and SECLOG record set on.

The only recovery-related records in the RECON that are not automatically deleted are the log records (for example, the PRILOG and LOGALL records). These records can be deleted using the `DELETE.LOG INACTIVE` command. This command can be added to the job that takes a backup of the RECON data set.

Monitoring the RECON

In addition to the regular backups, you should monitor the status of the individual RECON data sets on a regular basis. There are two ways to do this: using the `LIST.RECON STATUS` command and using the DBRC Query request.

Monitoring the RECON with the LIST.RECON STATUS Command

Regular use should be made of the `LIST.RECON STATUS` command to monitor the status of the individual RECON data sets.

Using the`LIST.RECON` command produces a formatted display of the contents of RECON. The copy1 RECON data set is used as a source. DBRC ensures that the second RECON data set contains the same information as the first RECON data set.

The optional parameter STATUS can be used to request the RECON header record information and the status of all RECON data sets. The use of this parameter suppresses the listing of the other records.

Issue this command two or three times a day during the execution of an online system to ensure that no problems have been encountered with these data sets.

Monitoring the RECON with the DBRC Query Request

Use the DBRC API Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve the following types of information from the RECON:

- Backout (TYPE=BACKOUT)
- Database (TYPE=DB). This variation of QUERY returns database registration and status information for:
  - Full-function databases

  - Fast Path databases
  - HALDB databases
  - DBDS or area information and supporting recovery-related information for each DBDS or area (allocation, image copy, recovery, and reorganization)
- Group and member information for the following group types:
  - Change Accumulation (TYPE=CAGROUP). CA execution information can also be returned.
  - DBDS (TYPE=DBDSGROUP)
  - Database (TYPE=DBGROUP)
  - Recovery (TYPE=RECOVGROUP)
  - Global Service Group (TYPE=GSGROUP)
- Log, recovery and system log data set (TYPE=LOG).
- Online log data set (TYPE=OLDS).
- RECON status (TYPE=RECON). This variation of the Query request returns RECON header information, as well as the status of the RECON configuration.
- Subsystem (TYPE=SUBSYS).

Reorganizing RECON Data Sets

Because all current levels of VSAM support CI reclaim (and DBRC does not turn it off), the requirement to reorganize RECONs to reclaim space has diminished. For instance, when all the records in a CI have been erased, the CI is returned to the free CI pool in the CA. Some users have decided to perform a monthly reorganization.

A plan for reorganizing the RECON data sets to reclaim this space on a regular basis must be considered. The RECON data sets can be reorganized while the IMS online systems are active.

The RECON data sets can be reorganized easily and quickly with the use of a few DBRC and AMS commands. The AMS `REPRO` command copies one RECON data set to another, reorganizing it during the process. This command, combined with a DELETE and a DEFINE of the RECON data sets, is enough to complete a reorganization.

Additional information to consider when designing the RECON reorganization procedures, related to the IMS TM status, are as follows:

- If the online system is active, a reorganization of the RECON data sets should be scheduled:
  - During a period of low RECON activity.
  - When no batch, DL/I or DBB jobs or utilities are running.

  A `LIST.RECON STATUS` command must be issued from each online system that uses the RECON data sets, after the `CHANGE.RECON REPLACE` command is issued, in order to de-allocate the RECON before deleting and defining it again.
- If the online system is not active, a reorganization of the RECON data sets should be scheduled:
  - After the RECON has been backed up (using the `BACKUP.RECON` command).
  - When no subsystems are allocating the RECON data sets.

Recreating RECON Data Sets

The RECON data sets may need to be recreated, for instance:

- In a disaster recovery site
- After the loss of all the RECON data sets when no current backup is available

Recreating the RECON can be a long and slow process. When designing procedures to handle this process, there are two basic alternatives:

- Restore the RECON from the last backup (if available) and update it to the current status required
- Recreate and re initialize the RECON data sets

Both of these procedures have advantages and disadvantages. Which alternative is best suited for an installation depends on:

- The time frame in which the system must be recovered and available
- The point in time to which it is acceptable to recover
- The type of processing environment (24-hour online availability or batch)

PRILOG Record Size

One PRILOG record is created for each instance of IMS. This record must contain all the information about the log data sets created during the life of this IMS.

The record size can be large if spanned records are used; however, the following limitations should be considered before using spanned records:

- The maximum size of a record to be used by the VSAM REPRO command is 32,760 bytes if the output is a non-VSAM data set.
- RECON backup and transfer to off-site storage is normally performed with a sequential data set.
- PRILOG records are only deleted when every RLDS and SLDS data set within that record is no longer required.

This is a problem only for those installations which have a high volume of log data sets and the requirement for a continuous operation environment.

Use the following formula to calculate the size of the maximum required PRILOG record.

```
S = 52 + (120 D) + (32 V)
```

Where:

S The size for the PRILOG/PRISLDS record (in bytes)

52 The required prefix of the PRILOG record

120 The required number of bytes for each SLDS/RLDS entry

D The number of SLDS/RLDS data sets created from archive for this execution of the subsystem

32 The required number of bytes for each volume that contain SLDS/RLDS data sets

V The number of volumes that can contain SLDS/RLDS data sets

The following are two examples of calculating the PRILOG record size.

Example 1: For the first example, assume that an installation has the following characteristics:

- An online IMS is running for 23 hours a day.
- The IMS fills up an OLDS every 30 minutes.
- Each OLDS is archived to one RLDS and one SLDS.
- There are 2 volumes that can contain RLDS or SLDS data sets.
- There are 46 RLDS and 46 SLDS data sets each day.

Using the formula discussed in “PRILOG Record Size” on page 274, the size of the PRILOG record for this example is:

```
S = 52 + (80 D)  + (14 V)
  = 52 + (80 92) + (14 2)
  = 52 + (7360)  + (28)
S = 7440
```

This is well under the maximum size, so there is no problem with this subsystem.

Example 2: For the second example, assume that the environment changes to allow the IMS to run 24 hours a day for 6 days before being stopped. There are 48 RLDS and 48 SLDS data sets each day, and a total of 576 for the 6 days. The calculation now looks like this:

```
S = 52 + (80 D)   + (14 V)
  = 52 + (80 576) + (14 2)
  = 52 + (46,080)  + (28)
S = 46,160
```

This is now over the suggested maximum record size. One solution is to switch to archiving after two OLDS are full. This reduces the number of RLDS and SLDS data sets by half. This brings the PRILOG record size well below the maximum size.

Summary of Recommendations for RECON Data Sets

Keep the following recommendations in mind when planning for your RECON data sets:

- Use three RECON data sets: two current and one spare.
- Define the three RECON data sets with different space allocations.
- Separate the RECON data sets (for example, put them on different devices and channels).
- Use dynamic allocation.
- Do not mix dynamic allocation and JCL allocation.
- Define the RECON data sets for *availability*, but keep performance implications in mind.

DBRC Support for Remote Site Recovery

DBRC assists you in the installation of IMS DB and IMS TM, as well as with the definition and management of IMS components, in the Remote Site Recovery (RSR) complex. In support of RSR, DBRC provides:

- Commands to define, update, and display the status of the RSR complex.

  The RECON contains the definition of an RSR complex. You define the elements of the RSR complex with DBRC commands, and you can modify and display the RSR complex definition with other DBRC commands.

- Services that are used by an active IMS to identify the tracking IMS and the databases covered by RSR.

  An active IMS obtains the identity of its tracking IMS from DBRC. As databases are updated by the active IMS, DBRC tells the database component whether the database is covered by RSR. And the active IMS sends its log data to the tracking IMS.
- Services used by a tracking IMS to record information about log data that is received from an active IMS.

  As logs are received and stored at the tracking site, DBRC records the receipt of the log data. When begin-update records are received for registered databases, DBRC records the database update.
- Tracking IMS database support:
  - Two types of tracking (called shadowing): DB level tracking (DBTRACK) or Recovery level tracking (RCVTRACK).
  - Maintains log data set information for online forward recovery.
  - Records which database change records have actually been applied to the covered databases.
- Services to assist in the takeover process

  During a remote takeover, DBRC changes the state of the registered databases at the new active site to indicate that they are now the master databases.

Related Reading: See the *IMS Version 9: Administration Guide: System* and the *IMS Version 9: Operations Guide* for more information on RSR.

Chapter 27. Controlling IMS

Controlling IMS consists of many tasks. These tasks are discussed in the following sections:

Related Reading: For more detailed information on the above topics, see the *IMS Version 9: Operations Guide*.

Monitoring the System

You need to monitor the status of the system on a regular schedule to gather problem determination and performance information. For example, to determine if you should start an extra message region, you might monitor the status of the queues during peak load.

You can determine the current status of the system by issuing `/DISPLAY` commands, specifying the appropriate keywords. You can monitor the status of the IRLM by using the z/OS `MODIFY` *`irlmproc`*`,STATUS` command.

You can use the `/TRACE` command to help diagnose system operation problems. This command turns on or off various IMS traces, which record use of IMS control blocks, message queue and I/O line buffers, and save area sets. IMS records trace information on the IMS log unless you request that the traces be recorded on an external trace data set. See "Trace Facility" on page 286 for more information. You can also use `/TRACE` to trace locking activities and to start and stop the IMS Monitor (see "IMS Monitor" on page 282).

Related Reading: For more information on monitoring, see the *IMS Version 9: Administration Guide: Database Manager*.

Processing IMS System Log Information

The system log data sets are a basic source for statistics about the processing performed by the online system. Individual log record types contain data that can be analyzed in many ways. For example, you can select and format all activity pertaining to a specified user ID or about IMS pools.

Using IMS System Log Utilities

IMS provides several utilities to assist with extracting log records from the system log. These utilities also assist with reducing and merging data that is spread across several log data sets. The sections that follow describe several of these utilities.

File Select and Formatting Print Program

If you want to examine message segments or database change activity in detail, you can use the IMS File Select and Formatting Print Program (DFSERA10). This utility prints the contents of log records contained in the OLDS, SLDS, or the CQS log stream. Each log record is presented as one or more segments of 32 bytes. The printed output gives each segment in both character and hexadecimal formats.

You can specify selection criteria for a subset of the records rather than printing all records. You can also specify a starting record number and the number of records to process. You can use an exit routine to customize the selection and formatting of the log records.

Although you can use the File Select and Formatting Print Program to copy entire input logs, you can more conveniently use the Log Archive utility (DFSUARC0). You use one or more SLDSs as input and specify a user data set as output. Also, you need to specify `DBRC=NO` in the `EXEC` statement to prevent DBRC from making entries in the RECON data set about your backup log. Making backup copies of the system log data sets can be useful to obtain an alternative input source for statistics and other monitoring activities occurring in parallel with production use of the system log.

Fast Path Log Analysis Utility

Use the Fast Path (FP) Log Analysis utility (DBFULTA0) to prepare statistical reports for Fast Path, based on data recorded on the IMS system log. This utility is an offline utility and produces three data sets, one of which contains six formatted reports:

- Detail Listing of Exception Transactions
- Summary of Exception Detail by Transaction Code for IFP Regions
- Overall Summary of Transit Times by Transaction Code for IFP Regions
- Overall Summary of Resource Usage and Contentions for All Transaction Codes and PSBs
- Summary of Region Occupancy for IFP Regions by PST
- Summary of VSO Activity
- Recapitulation of the Analysis

These reports are useful for system installation, tuning, and trouble shooting. This utility is not related to the IMS Monitor or the Log Transaction Analysis utility.

Log Transaction Analysis Utility

In an IMS DB/DC or DCCTL environment, you can collect information about individual transactions, based on records on the system log, using the Log Transaction Analysis utility (DFSILTA0). Many events are tabulated in the Log Analysis report produced by this utility including total response time, time on the input queue, processing time, and time on the output queue.

You can select a start time for the report tabulation; analysis begins at the first checkpoint after the start time. To control how much transaction activity is tabulated, you can specify an interval (in minutes) of elapsed time from the start time before the utility ends the tabulation, or you can relate the activity reported to a number of IMS checkpoints.

You can retitle a Log Analysis report or change the sequence in which the detailed transaction lines are printed. You can sort by transaction code or by any of the fields in the report. You can also suppress printing so that the output is stored on a DASD data set.

Using this utility, you can create an output data set, in system log format, that is a copy of all or part of the input system logs. By having a copy of the system log, you can monitor system activity without impacting the use of the OLDS for recovery.

The Statistical Analysis Utility

In an IMS DB/DC or DCCTL environment, you can produce several summary reports using the IMS Statistical Analysis utility (DFSISTS0). You can use these reports to obtain actual transaction loads and response times for the system. The statistics produced are dependent on the input system log data sets. The following set of reports is produced:

- Telecommunication line and terminal (distributed traffic over 24-hour day)
- Transaction (distributed activity over 24-hour day)
- Transaction response
- Messages queued but not sent (listing by destination and by transaction code)
- Program-to-program messages (listing by destination and by transaction code)
- Application accounting
- IMS accounting

Knowledge-Based Log Analysis

IMS Version 9 provides enhanced log-formatting routines that help you examine and display data from IMS log data sets:

- The Knowledge-Based Formatting Print routine (DFSKBLAK) provides a clear, simple description of each event represented by a log record, including the meaning of the various fields and flags. It does not display null fields.
- The Knowledge-Based Summary Formatting Print routine (DFSKBLAS) prints and displays the header and description of a log record, but not the record itself. This information can be useful when you need to have a general understanding of the log records associated with a particular resource.
- The Knowledge-Based Basic Formatting Print routine (DFSKBLA3) is similar in function to the Record Format and Print Module (DFSERA30), but its output also provides a brief description of the log record identifier (or log record type). This information can be useful if you are unfamiliar with the record types present in an IMS log. The Knowledge-Based Basic Formatting Print routine also interprets the prefix fields for the IMS type X'01' and type X'03' log records.

With knowledge-based log analysis, IMS Version 9 provides the following new functions for log analysis and diagnosis:

Knowledge-Based Log Record Analysis

The enhanced log formatting routines perform basic knowledge-based log record analysis by returning all meaningful log records that contain the indicated search criteria. The search criteria can include: a specific unit of work (UOW), recovery token, LTERM name, node name, transaction name, or program name. When you use dynamic search, the knowledge-based log analysis enhances the search to include criteria that you did not explicitly request, but that was “discovered” during the search process.

DL/I Trace Analysis

The enhanced log formatting routines perform an analysis on the log records produced by IMS as result of the `/TRACE SET ON TABLE xxxx`

OPTION LOG (ID of X'67FA') command. Using supplied search criteria, the knowledge-based log analysis extracts only those entries of interest.

MSC Trace Analysis

The enhanced log formatting routines use IMS MSC log records to measure the overall performance of each link defined in the system. You can specify either of the following types of analysis:

Detail The Detail Selection routine (DFSKBST0) produces a report that contains the individual response times in milliseconds of every send data and receive data for each MSC link that is traced.

Summary

The Summary Selection routine (DFSKMSCD) produces a report that contains the average response time in milliseconds of the total number of send data and receive data values for each link trace.

The enhanced log formatting routines produce formatted output only if the IMS input logs contain X'6701' records generated using the /TRACE SET ON LINK*link#* command.

IRLM Lock Trace Analysis

The enhanced log formatting routines create several output reports for IRLM lock traces, including one based on “wait time order”. This particular report lists databases in order of the total lock wait time during the trace. You can save the report to a data set, which you can then sort using either the knowledge-based log analysis ISPF tools or an editor’s SORT command. The IRLM Lock Trace Analysis panel provides some pre-defined sort options, such as Sorted by Database Name and Sorted by RBA.

DBCTL Transaction Analysis

The enhanced log formatting routines perform the functions of both the Log Transaction Analysis utility (DFSILTA0) and the Fast Path Log Analysis utility (DBFULTA0). The enhanced log formatting routines also perform a sort of the data. The DBCTL Transaction Analysis uses the X'07' and X'5937' log records orthe X'5938' log record to gather statistics.

IMS Record User Data Scrub

The enhanced log formatting routines perform a scan of all of the IMS logs. The routines delete those record parts that might contain sensitive or confidential customer business transaction information. The deletion does not compromise the integrity or the content of the vital IMS system data. This scrub is useful when you must send IMS log data to an outside organization for analysis.

Statistics Log Record Analysis

The enhanced log formatting routines call a routine (DFSKBST0) to produce output that is a verbal description and interpretation for all of the fields contained in the X'45' log record, including its subcodes. You must use the /CHECKPOINT STATISTICS command to ensure that the X'45' log record contains the information required for this enhanced formatting.

IMS Log Content Summary and IMS System Configuration Overview

The enhanced log formatting routines creates a summary of the log data sets’ content, characteristics of the IMS system that produced the log data (when statistical log records are present), and some statistical information related to transactions, programs, and databases. The KBLA Log Summary function also includes the following information:

- Input IMS logs used for the utility execution
- IMSID

- First and last LSN (log sequence number) in the log
- Time stamp (UTC) and local time of the first and last log record
- Difference between UTC and local time (HHMM format)
- Elapsed time on selected logs
- Total number of log records in the log data set
- Presence of internal trace record, system restarts, dump log record, and system checkpoint
- Number of log records present for each record ID
- Programs running during the period covered by the log
- Transactions running during the period covered by the log
- Databases accessed during the period covered by the log

Related Reading: For detailed information about the KBLA routines and utilities, see the *IMS Version 9: Utilities Reference: System*.

Using the IMS Performance Analyzer for z/OS

The IMS Performance Analyzer (IMS PA), program number 5697-B89, gives you the information you need to increase your IMS system performance. IMS PA provides more than 30 reports that can help optimize IMS system and application performance.

Specifically, IMS PA:

- Offers an ISPF CUA®-compliant user interface to build, maintain, and submit report requests
- Provides revised, enhanced, and new reports
- Supports IMS Versions 7, 8, and 9 from a single load library
- Allows the optional use of GDDM® for selected graphical reports
- Saves selected report data for reporting using PC tools
- Requires no dependency on GPAR

IMS PA produces a comprehensive set of reports (from management summaries to detailed program traces), organized by level of detail and area of analysis, to meet a broad range of IMS system analysis objectives, such as:

- **System performance evaluation.** IMS PA features help you monitor and evaluate IMS system performance on a daily basis. Management level summary reports express key values in terms of rates, ratios, and percentages. These reports help you use IMS PA for trend analysis, comparative analysis of systems, and evaluation of a system against installation standards. IMS PA's fast and efficient processing of monitor output data lets you produce long and frequent traces to obtain an accurate view of your IMS system.
- **System tuning.** Reports that help you enhance IMS system performance through system tuning are a key feature of IMS PA. Monitor output is summarized and categorized to help you rapidly identify problem areas. Detailed analysis reports help you investigate and evaluate these problem areas and also the effect of changes to the system.
- **Application and program evaluation.** IMS PA reports program activity in message processing or batch regions. IMS PA can be a valuable tool for evaluating existing applications and programs and validating whether new applications and programs conform to installation standards. Program activity reports and program traces add greatly to system documentation.

IMS PA produces alphanumerically collated report items in terms of ratios, rates, and percentages to facilitate comparison of results without additional computations. Schedules in progress, including wait-for-input (WFI) and BMPs, are reported. Reports on IMS batch programs are also provided.

IMS PA is a functional replacement for IMS/VS Performance Analysis Reporting (IMSPARS) and IMS Monitor Summary and System Analysis Program (IMSASAP).

Related Reading: IBM also offers a number of other IMS database productivity tools. IMS Tools is a set of database performance enhancements for your IMS environment. These tools can help you automate and speed up your IMS utility operations. They can also assist you in analyzing, managing, recovering, and repairing your IMS databases. You can learn more about these tools on the DB2 and IMS Tool Web site at http://www.ibm.com/software/data/db2imstools/.

Choosing Tools for Detailed Monitoring

Many of the monitoring tools you can use to collect detailed data are also used for general diagnostics. The principal tool provided by IMS is the IMS Monitor, which allows you to monitor online subsystems. For a stand-alone IMS DB batch system driven by an SLDS, use the Database Batch Monitor. You can also use IMS PA, program isolation and lock traces, and the external trace facility.

For information about IMS PA, see “Using the IMS Performance Analyzer for z/OS” on page 281.

IMS Monitor

The IMS Monitor collects data while the online IMS subsystem is running. It gathers information for all dispatch events and places it (in the form of IMS Monitor records) on a sequential data set. You use the IMSMON DD statement in the IMS control region JCL to specify the IMS Monitor data set. IMS adds data to this data set when you activate the Monitor using the `/TRACE` command. The IMS MTO can start and stop the Monitor to obtain snapshots of the system at any time. But, remember that the IMS Monitor adds to system overhead and generates considerable amounts of data.

Controlling Monitor Output

Plan to run the IMS Monitor for short intervals and to control its operation carefully. Shorter intervals also prevent the overall averaging of statistics, so that problems within the system can be more readily identified. The IMS Monitor’s output can be constrained by:

- Type of activity monitored
- Database or partition or area
- Dependent region
- Time interval

IMS Monitor Output Data Sets

The IMS Monitor output can be either a tape or a DASD data set. Using DASD eliminates the need to have a tape drive allocated to the online system. If you want to use the Monitor frequently, you might find that permanently allocated space for a DASD data set is convenient. One technique is to code `DISP=SHR` on the IMSMON DD statement so that the reports can be generated as each Monitor run is completed.

You must coordinate the report generation with the operator because each activation of the monitor writes over existing data. Although this does not occur for tape data sets, new volumes must be mounted. The volume is rewound, and a mount request is issued each time you start the IMS Monitor.

Recommendations:

- Do not catalog IMS Monitor data sets. The IMS Monitor can produce multiple output volumes while IMS is running if the data sets are not cataloged.

  If you want to have IMS dynamically allocate the IMS Monitor data set, do not include the IMSMON DD statement in the IMS control region JCL.
- Allow IMS to dynamically allocate IMS Monitor tape data sets. A tape drive is not permanently reserved for the control region for dynamically allocated data sets.

Related Reading: For details of how to specify dynamic allocation for the IMS Monitor data set, see the *IMS Version 9: Utilities Reference: System*.

Selecting Monitor Traces

After monitor requirements have been established, you might be able to restrict the scope of the IMS Monitor activity. Restricting the scope has the advantage of reducing the impact of the IMS Monitor on system throughput. However, you should not compromise the collection of useful data. You can control what specific types of events are traced by using specific keywords on the `/TRACE` command. For example, you can monitor line activity, scheduling and termination events, activity between application programs and message queues, activity between application programs and databases, or all activity. You can also limit monitoring to:

- Particular databases, partitions, or areas
- Particular dependent regions
- A specified interval of time

Obtaining IMS Monitor Reports

You can obtain reports based on the IMS Monitor's output by using the IMS Performance Analyzer (IMS PA) or the IMS Monitor Report Print utility (DFSUTR20). For information on IMS PA, see "Using the IMS Performance Analyzer for z/OS" on page 281.

The IMS Monitor Report Print utility summarizes and formats the raw data produced by IMS and presents the information in a series of reports. You can suppress the reports pertaining to DL/I calls and tabulated frequency distributions.

The duration of the monitored events is determined by the entries for start and stop of the IMS Monitor. You cannot select a different time period for reporting, because many of the timed events are not captured continuously: only when the IMS Monitor is started and stopped. For this reason, you should ensure that the IMS Monitor is stopped before taking any action to stop the IMS control region.

Related Reading: For a description of the JCL requirements and utility control statements, see the *IMS Version 9: Utilities Reference: System*.

//DFSSTAT Reports

The //DFSSTAT reports give you the number of database and data communications calls issued by an application program and describe the buffering activity. These reports are described in the *IMS Version 9: Utilities Reference: System*.

GTF Trace

You can use the z/OS Generalized Trace Facility (GTF) to record a wide range of system-level events. The trace activity is controlled from the z/OS system console using the `MODIFY` command. Output is spooled to a sequential data set that is used by a generalized formatting utility. You can write exit routines that are called by the formatting utility to edit the trace records and present the data as desired.

z/OS Component Trace (CTRACE)

IRLM 2.1 uses the z/OS component trace (CTRACE) facility to trace IRLM activity. Because the trace output is in z/OS CTRACE format, you can use IPCS CTRACE format, merge, and locate routines to process the buffer data.

Use the z/OS `TRACE CT` command to start, stop, or modify an IRLM diagnostic trace. This command can be entered only from the z/OS master console. Entering the commands requires an appropriate level of z/OS authority. IRLM does not support all the options available on the command. You can also start the IRLM tracing by placing `TRACE=YES` in the IRLM procedure.

Related Reading: For information on the `TRACE CT` command for IRLM, see the *IMS Version 9: Command Reference*. For complete information on the command, see *z/OS MVS System Commands*.

See “Tracing IRLM Activity” on page 296 for more information on IRLM traces.

Obtaining Program Isolation and Lock Traces

In an IMS DB/DC or DBCTL environment, you can detect contention for a database segment by examining the output produced by the Program Isolation Trace Report utility (DFSPIRP0). To get the source data for the utility, issue the `/TRACE SET ON PI OPTION ALL` command. To stop gathering source data, issue the `/TRACE SET OFF PI` command. A control statement for the utility can select a start or stop time relative to a specified date.

Tracing the program isolation function can create additional log records. These records contain the enqueue or dequeue requests issued by the program isolation function between sync points as a result of database updates, checkpoint, and message handling events.

The Program Isolation Trace Report utility only reports those events that required wait time. The report identifies the data management block (DMB) name, database control block (DCB) number, relative byte address (RBA), program specification block (PSB) name, and transaction code. The utility sorts all activity by RBA number (shown as ID in the report). The report lists elapsed times for enqueues that required a wait (during the trace interval) and totals the number of enqueues for each ID, DCB, and DMB. The requesting PSB or transaction is considered the holding PSB or transaction of the next enqueue waiting for the same segment. A sample report is illustrated in Figure 93 on page 285. In this report, no elapsed wait time is recorded for Fast Path.

Related Reading: For details of how to run the report utility, see the *IMS Version 9: Utilities Reference: Database and Transaction Manager*.

```
                    P R O G R A M   I S O L A T I O N   T R A C E   R E P O R T          PAGE   1
 DATE:  08/10/03
 TIME:  16:36  TO  16:37
 DCB *** REQUESTING *** ELAPSED   **** HOLDING *** ID TOTAL DCB TOTAL DMB      TOTAL
 DMB NAME NUM  ID  TRAN AND PSB NAMES   TIME       TIME    TRAN AND PSB NAMES ENQ'S ENQ'S ENQ'S
TABLEDBQ 1 0022D020  DE1Q     PROGDE1Q  16:36:54   0:00.061  DE2Q     PROGDE2Q
                                                                                1
           003BE00C  DE2Q     PROGDE2Q  16:36:51   0:00.027  DE1Q     PROGDE1Q
                                                                                1
           007D901C  DE2Q     PROGDE2Q  16:36:34   0:00.036  DE1Q     PROGDE1Q
                                                                                1
           008EF014  DE2Q     PROGDE2Q  16:36:49   0:00.038  DE1Q     PROGDE1Q
                                                                                1
           0090401C  DE1Q     PROGDE1Q  16:36:50   0:00.072  DE2Q     PROGDE2Q
                                                                                1
           00A06010  DE2Q     PROGDE2Q  16:36:38   0:00.046  DE1Q     PROGDE1Q
                                                                                1
           00A1401C  DE1Q     PROGDE1Q  16:36:50   0:00.008  DE2Q     PROGDE2Q
                                                                                1    7    7
TABLEDBR 1 002A901C  DE2R     PROGDE2R  16:36:40   0:00.034  DE1R     PROGDE1R
                                                                                1
           0045801C  DE2R     PROGDE2R  16:36:41   0:00.043  DE1R     PROGDE1R
                                                                                1
           0072F024  DE1R     PROGDE1R  16:36:30   0:00.053  DE2R     PROGDE2R
```

Figure 93. Sample Program Isolation Trace Report

You can use the File Select and Formatting Print utility to select and print trace table and PI entries in the log records in the following ways:

- Specify an `OPTION` statement with the `PRINT` parameter and `COND=E` and `EXITR=DFSERA40` keyword parameters. The output is a report containing the program isolation (PI) trace records formatted in sequential order.

  Related Reading: For an example of this report and an explanation of the headings, see the *IMS Version 9: Utilities Reference: System*.
- Select only the log records that contain the trace using the IMS Trace Table Record Format and Print Module (DFSERA60). Specify an `OPTION` statement with the `PRINT` parameter and `COND=E` and `EXITR=DFSERA60` keyword parameters. The output is a report containing the PI trace entries, the DL/I trace entries, and the lock trace entries formatted to show these entries in sequential order. For an explanation of the headings, see an assembly listing of the macro IDLIVSAM TRACENT.

You can use a Program Isolation Trace Record Format and Print Module output report to find out more information:

- The level of control (LEV) column shows read only, share, exclusive control, and single update activity.
- The return code (RC) column shows return codes from DFSFXC10 or the IRLM. You can determine whether the caller had to wait for the requested resource, or if the transaction caused a deadlock situation.
- The PST post code (PC) column shows the cause of the wait. If the entry is X'60', a deadlock occurred.

You can reduce the number of records examined by specifying an additional `OPTION` statement to the File Select and Formatting Print utility so that only records confirming deadlock are printed.

IMS automatically resolves deadlock situations by using dynamic backout. But the detection of deadlocks is important so you can modify your application design to prevent future deadlocks.

The advantage of the PI trace records report is that it shows where contention for a particular segment or range of segments occurs. The report also shows which transactions are competing within a database. It also shows high wait times that might explain a delay in response time. One way to handle the segment contention might be change the database design to separate some of the fields into an additional segment type.

Trace Facility

You can use the IMS Trace facility to write IMS trace tables internally or to an external trace data set. IMS can write this external trace data set to either DASD or tape:

- DASD data sets can be allocated by JCL or can be dynamically allocated.
- Tape data sets must be dynamically allocated.

Related Reading: For information on how to use the DFSMDA macro to create the dynamic allocation members, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

You can also write the trace tables to the OLDS, but this could adversely affect OLDS performance. The external trace data sets are independent of the OLDS, so you can write trace tables to the external trace data sets even if the OLDS is unavailable.

To display the status of traces, use the `/DISPLAY TRACE` command. This command can be used to determine the status of the IMS traces in effect and the status of any external trace data sets in use.

Related Reading:

- See the *IMS Version 9: Diagnosis Guide and Reference* for information about:
  - When and why trace tables are used
  - The DFS2867A message when using external tracing to OLDS
- See the *IMS Version 9: Installation Volume 2: System Definition and Tailoring* for information about defining and setting up trace facilities.

Executing Recovery-Related Functions

While IMS is running, an IMS system programmer or operator might need to execute functions relating to the recoverability of the system. These tasks include:

- “Using DBRC Commands”
- “Dumping the Message Queues” on page 287
- “Recovering the Message Queues” on page 287
- “Archiving the OLDS” on page 287
- “Making Databases Recoverable or Nonrecoverable” on page 288
- “Running Recovery-Related Utilities” on page 288

Using DBRC Commands

You can use the `/RMxxxxxx` commands to use DBRC functions.

Recommendation: Allow operators to use the /RMLIST and /RMGENJCL commands Restrict the use of /RMCHANGE, /RMDELETE, and /RMNOTIFY commands, because they update the RECON data set.

Related Reading: For information on using DBRC commands in a data-sharing environment, see “Online DBRC Commands” on page 300.

Dumping the Message Queues

If you want to save the message queues in a nonshared-queues environment, use the /CHECKPOINT SNAPQ command. This command dumps the message queues to the log without terminating the online system.

Recommendation: Schedule the /CHECKPOINT SNAPQ regularly because it shortens the time required for emergency restart if a problem occurs on the message queue data sets. Consider the following intervals:

- Whenever the OLDS is switched
- Once each hour
- Once each shift
- Twice each day (midnight and noon)
- Once each day

For a shared-queues environment, use the /CQCHKPT SHAREDQ command to dump the shared queues.

Recovering the Message Queues

In a non-shared-queues environment, you can recover the message queues during an IMS restart if the previous shutdown included the DUMPQ or the SNAPQ keyword. Specify the BUILDQ keyword on the /NRESTART or /ERESTART command to restore the messages to the message queue data sets from the IMS log. Specify the FORMAT keyword on the /NRE or /ERE command if you also want to reinitialize the message queue data sets.

In order to use the /NRE BUILDQ command, the system must be shut down using a /CHECKPOINT DUMPQ | PURGE command. To use the /ERE BUILDQ command, you need only a prior /CHECKPOINT SNAPQ command.

Restriction: If a /NRE BUILDQ or /ERE BUILDQ command fails and you cold start IMS, messages are lost and are not processed.

You can use the Queue Control Facility (QCF) program product (5697-E99) to select messages from the OLDS (or SLDS) and reinsert them into the IMS message queues after an IMS cold start.

Related Reading: See *IMS Queue Control Facility for z/OS* and the *IMS Version 9: Diagnosis Guide and Reference* for more information about QCF.

For a shared-queues environment, CQS automatically rebuilds the message queues if the coupling facility fails. You can also use the SETXCF START,REBUILD command to rebuild the queues manually.

Archiving the OLDS

As mentioned in “Archiving” on page 259, you should archive the OLDS to an SLDS at regular intervals. If you are not using automatic archiving, the MTO should use

the DBRC GENJCL command at regular intervals to generate the JCL for the Log Archive utility, and should execute the utility.

Making Databases Recoverable or Nonrecoverable

You can change recoverable full-function databases and DEDBs to nonrecoverable (after deleting recovery-related records from the RECON data set) by using the CHANGE.DB NONRECOV command. You can change to recoverable again by using the CHANGE.DB RECOVABL command.

Use the LIST.DB command to display whether a database is recoverable.

Running Recovery-Related Utilities

Depending on your recovery strategy, the MTO might be responsible for executing various recovery-related utilities at regular intervals. These could include:

- Database Image Copy utility (DFSUDMP0)
- Database Image Copy 2 Utility (DFSUDMT0)
- Online Database Image Copy utility (DFSUICP0)
- Database Change Accumulation utility (DFSUCUM0)

The MTO should also run these utilities when a database changes from nonrecoverable to recoverable.

Related Reading: For complete information about these utilities, see the *IMS Version 9: Utilities Reference: Database and Transaction Manager*.

Modifying and Controlling System Resources

You establish the initial settings of IMS resources during IMS system definition. The MTO, and other operators authorized to do so, can change various system resources using IMS commands.

You can use many IMS commands to perform similar control functions for different types of resources. “List of Commands with Similar Functions for Multiple Resources” shows the relationship between these commands and resources. The tables in “List of Commands with Similar Functions for Multiple Resources” provide answers to a series of specific questions. For example, after a command is issued, can a resource:

- Receive input?
- Send output?
- Perform output message queueing?

Related Reading: For details of these commands, see the *IMS Version 9: Command Reference*.

List of Commands with Similar Functions for Multiple Resources

The following tables show what IMS commands affect certain resources. The resources are:

- Telecommunication Line, Physical Terminal, or Node (Table 20)
- Logical Terminal (Table 21)
- Logical Link (Table 22)
- Logical Link Path (Table 23)
- Transaction (Table 24)

- Transaction Class (Table 25)
- Program (Table 26)
- Database (Table 27)
- Subsystem (Table 28)

Telecommunication Line, Physical Terminal, or Node

Table 20. IMS Commands That Affect Telecommunications Line, Physical Terminal, or Node Resources

IMS Command	Resource: Telecommunications Line, Physical Terminal, or Node		
	Receive Input	Send Output	Output Message Queuing
/ASSIGN	Y	Y	Y
/LOCK	N	N	Y
/MONITOR	Y	N	Y
/PSTOP	N	N	Y
/PURGE	N	Y	Y
/RSTART	Y	Y	Y
/START	Y	Y	Y
/STOP	N	N	Y
/UNLOCK	Y		Y
Note: /MONITOR, /PSTOP, /PURGE, and /RSTART refer to the telecommunication line or physical terminal, not to the node.			

Logical Terminal

Table 21. IMS Commands That Affect Logical Terminal Resources

IMS Command	Resource: Logical Terminal			
	Receive Input	Send Output	Queuing from Other Terminals	User
/ASSIGN	Y	Y	Y	Y
/LOCK	N	N	N	
/PSTOP	N	N	Y	
/PURGE	N	Y	N	
/RSTART				Y
/START	Y	Y	Y	Y
/STOP	N	N	N	Y
/UNLOCK		Y	Y	

Logical Link

Table 22. IMS Commands That Affect Logical Link Resources

IMS Command	Resource: Logical Link	
	Receive Input	Receive Output
/PSTOP	N	N

Table 22. IMS Commands That Affect Logical Link Resources (continued)

IMS Command	Resource: Logical Link	
	Receive Input	**Receive Output**
/RSTART	Y	Y

Logical Link Path

Table 23. IMS Commands That Affect Logical Link Path Resources

IMS Command	Resource: Logical Link Path	
	Queue Primary Requests Not Continuing Conversation	**Transmit Queue Message to Partner Systems**
/ASSIGN	Y	Y
/PURGE	N	Y
/START	Y	Y
/STOP	N	N

Transaction

Table 24. IMS Commands That Affect Transaction Resources

IMS Command	Resource: Transaction	
	Message Scheduling by Transaction	**Message Queuing by Transaction**
/ASSIGN	Y	Y
/LOCK	N	Y
/MSASSIGN	Y	
/PSTOP	N	Y
/PURGE	Y	N
/START	Y	Y
/STOP	N	N
/UNLOCK	Y	Y

Transaction Class

Table 25. IMS Commands That Affect Transaction Class Resources

IMS Command	Resource: Transaction Class
	Transaction Scheduling by Class
/ASSIGN	Y
/MSASSIGN	Y
/START	Y
/STOP	N

Program

Table 26. IMS Commands That Affect Program Resources

	Resource: Program
IMS Command	**Execute**
/ASSIGN	Y
/LOCK	N
/START	Y
/STOP	N
/UNLOCK	Y

Database

Table 27. IMS Commands That Affect Database Resources

	Resource: Database
IMS Command	**Use**
/ASSIGN	Y
/START	Y
/STOP	N
/UNLOCK	Y

Subsystem

Table 28. IMS Commands That Affect Subsystem Resources

	Resource: Subsystem
IMS Command	**Attach**
/ASSIGN	Y
/START	Y
/STOP	Y

You can also use other commands to affect the operating state of specific resources, as described in the following sections.

Related Reading: For more information about using IMS commands, see the *IMS Version 9: Command Reference*.

Modifying Dependent Regions

Use the `/ASSIGN` command to modify the assignment of classes to regions. Do this to adjust the processing load among message regions.

Modifying Telecommunication Lines

Use the `/DEQUEUE` command to discard response-mode output messages before you enter an `/RSTART LINE` command.

Modifying Terminals

Use the `/ASSIGN LTERM` command to modify the assignment of logical terminals to physical terminals or nodes. The new assignment remains in effect until the next cold start or until you issue another `/ASSIGN` command.

Use the /DEQUEUE command to discard response-mode output so that the /RSTART command can reset terminal response mode.

Use the /COMPT command for VTAM terminals (nodes) to notify IMS that a terminal component is operable or inoperable.

IMS provides a VTAM I/O Timeout facility to detect VTAM hung nodes and determine what action, if any, should be taken. Use the /TRACE command to start and stop the VTAM I/O Timeout facility. Use the /IDLE command to deactivate a node and the /ACTIVATE command to activate a node. Use the /DISPLAY command to display all nodes that have I/O outstanding for a time period greater than that specified during system definition.

Modifying Transactions

Use the /ASSIGN command to reassign the scheduling priorities established for transactions during system definition. The new assignments remain in effect until the next cold start or until you issue another /ASSIGN command.

In a shared-queues environment, you can use the /ASSIGN command to control which IMS subsystems can run certain types of transactions by assigning transactions to particular classes.

Example: You can define TRANA to class 4 on IMSA and to class 255 on IMSB and IMSC, so that only IMSA can run TRANA. If IMSA fails, you can reassign TRANA on either IMSB or IMSC to a class that these IMS subsystems can run.

Recommendation: Do not use the /STOP TRANSACTION command to control which IMS subsystems can run certain types of transactions.

Modifying Databases

Use the /DBDUMP command to stop online update access to a database. This lets you produce an offline dump of the database.

Use the /DBRECOVERY command to stop all online access to a database. Use it to recover a database offline.

Normally, IMS switches to using the next OLDS when you enter the /DBDUMP or /DBRECOVERY command. This switch does not occur if you specify the NOFEOV keyword on either command.

Specify the GLOBAL keyword on the /DBDUMP or /DBRECOVERY command to have the command apply to all subsystems sharing the database. The IRLM must be active if you use this keyword. The default is LOCAL, which specifies that the command applies only to the subsystem on which you enter the command.

Note: IMS must be restarted after issuing a command with the GLOBAL keyword.

Modifying ISC Users (Subpools)

Use the /ASSIGN command to change the assignment of a static LTERM to an ISC user (also called a subpool). The new assignment remains in effect until the next cold start or until you issue another /ASSIGN command.

Modifying ETO Users

For dynamic user IDs, use the /ASSIGN command to change the assignment of a user ID to another user or to an LTERM. The new assignment remains in effect until the next cold start or until you issue another /ASSIGN command.

Use the /DISPLAY USER DEADQ command to list all message queues that are eligible for dead letter status. Use the /ASSIGN command to assign a dead letter queue to another user ID. Use the /DEQUEUE command to discard a dead letter queue.

In a shared-queues environment, use the /DISPLAY QCNT MSGAGE command to determine which messages, if any, are eligible for dead letter status.

Modifying MSC Resources

Use the /MSVERIFY command to verify the consistency of MSC system identifications (SYSIDs) and logical link paths (MSNAMEs) across two systems. You can use the /MSASSIGN command to change the assignment of MSNAMEs and SYSIDs to logical links.

All changes made by an /MSASSIGN command remain in effect until the next cold start or until you issue another /MSASSIGN command.

After using the /MSASSIGN command, you should use the /MSVERIFY command to ensure that the assignment produced a valid configuration.

Modifying Security Options

Use the /CHANGE command to update a current password with a new password. The current password must be known to IMS.

Restriction: IMS does not allow different user IDs to have the same passwords.

Use the /MODIFY PREPARE RACF and /MODIFY COMMIT commands to reinitialize RACF information if you are not using a RACF data space. If you are using a RACF data space, use the RACF SETROPTS RACLIST command rather than the IMS /MODIFY command.

Use the /DELETE command to delete terminal or password security for the specified system resource.

Use the /SECURE APPC command to control the RACF security level for input from LU 6.2 devices. Use the /DISPLAY APPC command to show the security level that is currently in effect. When IMS starts, the default is full security.

Use the /SECURE OTMA command to control the RACF security level for input from OTMA clients. Use the /DISPLAY OTMA command to show the security level that is currently in effect. When IMS starts, the default is full security.

Modifying Conversations

Use the /DISPLAY CONV command to show the status of all conversations, held or active. You can terminate a conversation if necessary with the /EXIT command, but you should only do this after warning the end user.

Modifying Subsystems

Use the /CHANGE command to delete an invalid network identifier (NID). If you need to disconnect from a specific subsystem, use the /STOP command. If the /STOP command does not work, use the z/OS MODIFY command.

Gathering Performance-Related Data

IMS provides the DB Monitor and the IMS Monitor, which gather and format IMS performance-related data and record this data on a statistics log.

The DB Monitor is available to IMS batch systems. It can monitor the activity between application programs and databases. For more information about the DB Monitor, see "DB Monitor."

The IMS Monitor is available to IMS online systems; In addition to performing all of the functions of the DB Monitor, the IMS Monitor can track and record information about activities occurring in the IMS control region and data communication activities. For more information about the IMS Monitor, see "IMS Monitor" on page 295.

IMS uses the statistics produced by each monitor to generate reports. The report programs run offline and print reports that summarize and categorize IMS activities.

Related Reading: For additional information about the monitor report programs, see:

- *IMS Version 9: Utilities Reference: Database and Transaction Manager*
- *IMS Version 9: Utilities Reference: System*

DB Monitor

The DB Monitor records performance data during execution of an IMS DB batch system. The DB Monitor can be active during the entire execution of an IMS batch job, or you can stop and restart it from the system console.

Activating and Controlling the Monitor

To activate the DB Monitor, specify MON=Y in the PROC statement of the batch job. When you submit the job, IMS uses parameter substitution to update the PARM field of the EXEC statement with a Y in the appropriate position.

To stop the DB monitor, the system console operator can use the MODIFY *jobname*,STOP command. Message DFS2215A displays on the system console when the Monitor is inactive.

To reactivate the DB monitor, the console operator can use the MODIFY *jobname*,START command.

Message DFS2216A displays on the console when the monitor is active again.

Logging the Data

IMS records the data produced by the DB Monitor on either the OLDS or a separate DB monitor log. Use the //IMSMON DD statement in the batch procedure to control where the data is logged:

- To store the monitor records on the OLDS, either include a //IMSMON DD DUMMY statement or omit the //IMSMON DD statement entirely.

- To store the monitor records on a separate DB Monitor log, include a valid //IMSMON DD statement.

If, for any reason, IMS cannot open the DB Monitor log data set specified on the //IMSMON DD statement, IMS displays message DFS2217I on the system console. Batch execution continues, but the Monitor is inactive.

If the DB Monitor log device encounters I/O errors, IMS displays message DFS2219I on the system console. Batch execution continues, but the Monitor is inactive.

If you want to stop the Monitor and force an end-of-volume for the DB Monitor log, use the MODIFY *jobname*,STOPEOV command. When you use the STOPEOV keyword, the batch region does not continue executing until the z/OS mount request for a new data set is satisfied.

Modify Command Errors

If you enter an incorrect job name on the MODIFY command, z/OS issues an error message. If you make some other error while entering the MODIFY command, IMS issues message DFS2218I, followed by either message DFS2215A or message DFS2216A.

IMS Monitor

The IMS Monitor records performance-related data during execution of the IMS online subsystem. When a significant event occurs within IMS while the monitor is active, IMS passes relevant data to it. The monitor formats a record describing the event, including the time stamps, and logs the event.

Activating and Controlling

If you create a DFSDCMON member in the IMS.SDFSRESL data set, you do not need a DD statement in the IMS, DBC, or DCC procedures for the Monitor because IMS dynamically allocates and deallocates the monitor data set. Otherwise, to activate the IMS Monitor, you must include a DD statement (using IMSMON as the data set name) in the IMS, DBC, or DCC procedures to specify the IMS Monitor log data set. When you include this DD statement, the IMS Monitor becomes available. If the monitor is available but inactive, processor usage is unaffected.

To start and stop the IMS Monitor, use the /TRACE command.

Using the Trace Command

In addition to starting and stopping the IMS Monitor, you can specify the types of events to be monitored using the /TRACE command. You can use the /TRACE command to monitor some or all of the following:

- Telecommunication line and logical link activity
- Scheduling and termination events
- Activity between application programs and message queues
- Activity between application programs and databases (full function and Fast Path)

You can also use the /TRACE command to limit the monitoring to:

- Particular databases, partitions, or areas
- Particular dependent regions
- A particular interval of time

Logging the Data

If the IMS Monitor log data set is on tape, IMS issues a tape mount request each time you start the monitor, and IMS rewinds the tape each time you stop the Monitor. If the IMS Monitor log data set is on DASD, IMS uses the same data set each time you start the monitor. Therefore, you should process the log after you stop the monitor before restarting it.

Recommendation: Start the IMS Monitor and allow it to run for a period of time, and then stop it to write a "snapshot" of current activity on the IMS Monitor log.

You must stop the IMS Monitor before you take a shutdown checkpoint in order for the report program to produce usable output.

I/O Errors

If a permanent I/O error occurs on the IMS monitor log data set, IMS stops the Monitor and issues message `DFS2202`. In this situation, you cannot restart the Monitor until you restart IMS because IMS does not close the IMS Monitor log data set until you shut down IMS.

If the problem that caused the error has not been corrected when you restart IMS, you should specify a different volume or unit for the new execution.

Tracing IRLM Activity

The IMS Monitor does not collect IRLM trace activity. IRLM uses the z/OS component trace (CTRACE) facility. Use the `TRACE CT` command to run the following types of sublevel traces:

DBM Trace interactions with the identified DBMS.

EXP Trace any exception condition.

INT Trace member and group events other than normal locking activity.

SLM Trace interactions with the z/OS locking component.

XCF Trace all interactions with the z/OS cross-system coupling services.

XIT Trace only asynchronous interactions with the z/OS locking component.

Related Reading: For a complete description of the z/OS `TRACE CT` command for IRLM, see the *IMS Version 9: Command Reference*.

Controlling Data Sharing

Controlling data sharing involves:

- "Monitoring the System"
- "Controlling Data Sharing Using DBRC" on page 300

Monitoring the System

To monitor data sharing, you obtain information on the status of the following: IRLM, IMS subsystems and databases, the RECON data set, and coupling facility structures.

Obtaining the Status of IRLM Activity

To display the status of an IRLM on either your system or on another connected system, enter the following z/OS command:

`MODIFY irlmproc,STATUS,irlmx`

where *irlmproc* is the name of the procedure that you used to start the IRLM, and *irlmx* is the name of the IRLM whose status you want to display. This command gives:

- The IMS IDs of IMS subsystems using this IRLM.
- The number of locks that are held and waiting for each subsystem on this IRLM.
- Identification of this IRLM: its subsystem name and IRLM number.

You can use the `ALLD` keyword to display the names and status of every IMS identified to an IRLM in a data-sharing group. Or, you can use the `ALLI` keyword to display the names and status of every IRLM in a data-sharing group.

You can also trace IRLM activity. See “Tracing IRLM Activity” on page 296.

Related Reading: For a complete description of the commands for the IRLM, see the *IMS Version 9: Command Reference*.

Displaying Components and Resources

Monitoring components and resources in a data-sharing environment requires the same kinds of procedures as in a non-sharing environment. For information about monitoring, see “Monitoring the System” on page 277.

Table 29 lists keywords for the `/DISPLAY` command that you can use to obtain information about various IMS resources.

Table 29. /DISPLAY Command Keywords That Provide Information about IMS Resources

Resources	/DISPLAY Command Keywords
Active control regions	ACTIVE REGION
Active jobs	ACTIVE
Programs, transactions, and conversations	CONVERSATION PROGRAM PSB STATUS PROGRAM STATUS TRANSACTION SYSID TRANSACTION TRANSACTION
Databases	DATABASE AREA STATUS DATABASE
Terminals, lines, links, and nodes	ACTIVE DC ASSIGNMENT LINE ASSIGNMENT LINK ASSIGNMENT NODE LINE LINK LTERM MASTER MSNAME NODE PTERM STATUS LINE STATUS LINK STATUS LTERM STATUS MSNAME STATUS NODE STATUS PTERM

Table 29. /DISPLAY Command Keywords That Provide Information about IMS Resources (continued)

Resources	/DISPLAY Command Keywords
External subsystems and connections to external subsystems	CCTL OASN SUBSYS SUBSYS
VTAM	TIMEOVER

For example, you can use the /DISPLAY DATABASE command after you enter a /START DATABASE command to determine whether the database is started.

Related Reading: For detailed information about the /DISPLAY command, see the *IMS Version 9: Command Reference*.

Monitoring Structures on a Coupling Facility

The following z/OS operator commands are especially useful in monitoring structure activity:

- DISPLAY XCF,STRUCTURE
- DISPLAY XCF,STRUCTURE,STRNAME=

These commands let you look at structures on a coupling facility to determine resource status and, for failures, gather information for problem determination.

Related Reading: For detailed information on these commands, see *z/OS: MVS System Commands*.

DISPLAY XCF,STRUCTURE Command: Use this command to display the status of structures defined in your active policy.

Example: Figure 94 shows an example of output from this command.

```
IXC359I  11.09.26  DISPLAY XCF 376
STRNAME           ALLOCATION TIME  STATUS
CF01             02/17/96 17:03:49 ALLOCATED
CF02                --       --    NOT ALLOCATED
CF03                --       --    NOT ALLOCATED
CF04                --       --    NOT ALLOCATED
OSAMSESXI        02/17/96 17:02:54 ALLOCATED
                                   REBUILDING
                                   REBUILD PHASE: QUIESCE
VSAMSESXI        02/17/96 17:03:03 ALLOCATED
```

Figure 94. Output from a DISPLAY XCF,STRUCTURE Command

Observe the following in Figure 94:

STRNAME
Is the name of a structure.

ALLOCATION TIME
Is a timestamp indicating when the structure was allocated in the coupling facility.

STATUS
Is the current status of the structure. A structure can be in a number of different states, such as ALLOCATED, NOT ALLOCATED, or ALLOCATED REBUILDING.

The displayed information can help you determine whether the appropriate IRLM, OSAM, VSAM, shared MSGQ, and shared EMHQ structures have been defined and allocated. If not, check that the structure names on your CFNAMES control statement match the structure names that are displayed. If they do not match, change the names in either the coupling facility resource manager (CFRM) policy or the CFNAMES control statement.

In Figure 94 on page 298, six structures are defined (CF01-CF04, OSAMSESXI, and VSAMSESXI). Three of the structures are allocated (CF01, OSAMSESXI, and VSAMSESXI). The status of the OSAM structure is that it is currently being rebuilt after a structure failure.

DISPLAY XCF,STRUCTURE,STRNAME= Command: Use the DISPLAY XCF,STRUCTURE,STRNAME= command to display detailed information about a specific structure. The three structures of importance to IMS are the IRLM, OSAM, VSAM, shared MSGQ, and shared EMHQ structures.

Example: Figure 95 shows an example of output from this command, specifying an OSAM structure named OSAMSESXI.

```
IXC360I  17.30.46  DISPLAY XCF 677
STRNAME: OSAMSESXI
 STATUS: ALLOCATED
  POLICY SIZE    : 2048 K
  PREFERENCE LIST: CF02     CF01
  EXCLUSION LIST IS EMPTY

 ACTIVE STRUCTURE
 ----------------
  ALLOCATION TIME: 02/17/96 17:02:54
  CFNAME         : CF02
  COUPLING FACILITY: ND02...
                     PARTITION: 0    CPCID: 00
  ACTUAL SIZE    : 2048 K
  STORAGE INCREMENT SIZE: 256 K
  VERSION        : A8DAC970 15774B04
  DISPOSITION    : DELETE
  ACCESS TIME    : 0
  MAX CONNECTIONS: 32
  # CONNECTIONS  : 5

 CONNECTION NAME  ID VERSION  SYSNAME  JOBNAME  ASID STATE
 ---------------- -- -------- -------- -------- ---- ------
 DLI11            04 00040001 MVS1     DLI11    0031 ACTIVE
 DLI12            05 00050001 MVS1     DLI12    0033 ACTIVE
 IMS1             01 00010013 MVS1     DLIOCSA8 0036 ACTIVE
 IMS2             02 00020012 MVS2     DLIOCSB8 0035 ACTIVE
 IMS3             03 00030011 MVS3     DLIOCSC8 0038 ACTIVE
```

Figure 95. Output for DISPLAY XCF,STRUCTURE,STRNAME= Command

There are three parts to the output.

- The first part shows the status of the structure and information about the active policy.

- The second part of the output shows more detailed status information for the structure.
- The third part of the output shows which z/OS systems are connected to the structure and gives information about each use.

Controlling Data Sharing Using DBRC

DBRC allows you to control access to data by IMS subsystems that participate in data sharing. Using DBRC, you can modify, initiate, and delete the current status indicators in the RECON data set to change:

- The access intent of online IMS subsystems
- The share level of registered databases

Your data sharing environment depends on the status of the databases and subsystems indicated in the RECON data set.

You can modify the access intent indicator using a form of the /START command. For a description of this command, see “Changing Database Access Intent” on page 301.

You can modify the share level indicator using a form of one of the DBRC online change commands, /RMCHANGE.

Online DBRC Commands

The /RMCHANGE command is one of a set of IMS commands that control DBRC utility functions; IMS responds to them by issuing corresponding commands directly to DBRC. Table 30 shows these commands and their recommended uses.

Related Reading: For full descriptions of these commands and their functions, see the *IMS Version 9: DBRC Guide and Reference*.

Table 30. DBRC Commands and Functions

Command	Recommended Use
/RMCHANGE	Alter a database sharing level Prevent other subsystem authorization Set or remove backout-required status
/RMDELETE	Remove database record and associated status Delete subsystem with unauthorized databases Remove area data set (ADS) record and associated status
/RMGENJCL	Generate online recovery jobs
/RMINIT	Register a new database Start control of an area data set
/RMLIST	Obtain recovery and authorization information for a database or ADS
/RMNOTIFY	Reset a system status

Example: If you want to change the share level of a registered database (named ORDERDB) from intraprocessor block-level data sharing (share-level 2) to interprocessor block-level data sharing (share-level 3) to allow data sharing with another IMS subsystem in another processor, enter:

```
/RMCHANGE DBRC='DB DBD(ORDERDB) SHARELVL(3)'
```

DBRC replies with its corresponding command input (CHANGE.DB DBD(ORDERDB) SHARELVL(3)) and a series of DBRC messages.

Do not enter the /RMCHANGE, /RMDELETE, or any command that alters the status of database records in the RECON data set while a job accesses the database. Stop the database using a /DBRECOVERY GLOBAL command before modifying any RECON record for that database. Otherwise, IMS rejects the command with an error.

Denial of Authorization

If, in a data sharing environment, DBRC is responding to authorization requests but fails to obtain authorization for a program:

- You receive message DFS047A identifying the database.
- IMS schedules the program's PSB without database access. IMS abnormally terminates a BMP or MPP with abend U3303 only if it tries to access the database.
- IMS abnormally terminates batch and utility regions with abend U0047.

Changing Database Access Intent

Use the following command to change the access intent that you declare during system definition: /START DATABASE *dbx* ACCESS=*xx*, where *dbx* is the database name and *xx* is the new access intent. Values for *xx* are:

EX Exclusive use

UP Update access

RD Read access

RO Read-only access

This command is local; it affects only the subsystem on which you enter it. The GLOBAL keyword is not valid with the ACCESS= keyword. If you need to change the access level for a shared database across the sysplex, you must enter this command on each subsystem that shares the database.

In order to change the access intent for a DEDB, you must stop all PSBs that access any of the areas in the DEDB. You might also have to stop regions that have wait-for-input (WFI) transactions scheduled for the DEDB.

Controlling Log Data Set Characteristics

From time to time, you need to tune and modify log data set characteristics, for example, in the following circumstances:

- After monitoring
- After changing your requirements for system availability, integrity, or operator handling

For summaries of actions required for changing log data set design, see:

- "Controlling the Online Log Data Set"
- "Controlling the Write-Ahead Data Set" on page 304
- "Controlling the System Log Data Set" on page 305
- "Controlling the RECON Data Sets" on page 305

Controlling the Online Log Data Set

Because you can restart IMS (warm start or emergency restart) with all input on SLDS, you can reallocate the OLDSs between a shutdown (or failure) and a subsequent restart. To restart IMS using SLDSs as input, you must delete the PRIOLDS and SECOLDS records from the RECON data sets.

Table 31 lists the actions required to change OLDS characteristics.

Table 31. Changing OLDS Characteristics

Modification	Actions Required
BLKSIZE[1]	1. Shut down IMS. 2. Archive all OLDSs. 3. Delete PRIOLDS and SECOLDS records from the RECON data sets, using the DELETE.LOG command. 4. Scratch all OLDSs. 5. Reallocate OLDSs with the new BLKSIZE. 6. Verify WADS space allocation. 7. Restart IMS (from SLDS).
Single to Dual[2]	1. Shut down IMS. 2. Archive all OLDSs. 3. Allocate dual OLDSs. 4. Delete the OLDS records from the RECON data sets, using the DELETE.LOG command. The primary OLDS records will be deleted. 5. Change OLDSDEF specification to dual. 6. Change IMS startup procedure (OLDS DD statements), if required. 7. Compile DFSMDA macros, if required. 8. Modify operating procedures. 9. Restart IMS.
Dual to Single[2]	1. Shut down IMS. 2. Archive all OLDSs. 3. Delete OLDS record from the RECON data sets, using the DELETE.LOG command. The primary and secondary OLDS records will be deleted. 4. Delete secondary OLDSs. 5. Change IMS startup procedure. 6. Modify operating procedures. 7. Restart IMS.
BUFNO[3]	1. Shut down IMS. 2. Change the OLDSDEF specification for BUFNO. 3. Verify CSA size. 4. Verify WADS space allocation. 5. Restart IMS.
Space, Location, or Allocation[4]	1. Shut down IMS. 2. Archive all OLDSs. 3. Delete PRIOLDS and SECOLDS records in the RECON data sets, using the DELETE.LOG command. 4. Scratch and reallocate OLDSs. 5. Restart IMS (from SLDS).

Notes:

1. **Changing OLDS Block Size**: Changing OLDS block size affects WADS space allocation. For information on how to calculate WADS space requirements, see

the *IMS Version 9: Installation Volume 1: Installation Verification*. To change WADS space allocation, see Table 32 on page 304. All OLDSs must have the same block size.

2. **Changing the Mode**: You must change your OLDSDEF specification in the DFSVSMxx member in IMS.PROCLIB. IMS initialization requires that at least three pairs of OLDSs be available. You must also reconsider data set placement.

   When changing from single to dual OLDS, each data set in a pair of OLDSs must have the same space allocation (number of blocks).

   Changing the mode from single to dual or from dual to single requires changes in the following operating procedures:

   - Skeletal JCL for archive (ARCHJCL member)
   - Skeletal JCL for log recovery (LOGCLJCL member)
   - Batch JCL for Log Recovery utility
   - Batch backout JCL for online transactions and BMPs

   If you warm start the IMS subsystem after changing from single to dual OLDSs, the `/DISPLAY OLDS` command does not show the secondary OLDSs as IN USE until they have been archived once. The command does, however, show dual OLDS logging.

3. **Changing the Number of OLDS Buffers**: Change BUFNO by changing the OLDSDEF specification for BUFNO in the DFSVSMxx member in IMS.PROCLIB.

   When you modify BUFNO, you should consider also modifying the region size for the VSAM common segment area (CSA). The amount of storage fixed for OLDS buffers is [`BUFNO * BUFFERSIZE`].

   WADS space is also affected by BUFNO. To change WADS space allocation, see Table 32 on page 304..

   Related Reading: For information on how to calculate WADS space requirements, see the *IMS Version 9: Installation Volume 1: Installation Verification*.

4. **Changing Space, Location, or Allocation**: For the recommended method of changing the space or location of your OLDS, or for reallocating an OLDS on the same volume and with the same space, follow the procedure for changing the BLKSIZE in note 1 above.

   You can modify space, location, or allocation without shutting down IMS in a non-XRF environment by using the following procedure:

   a. `/STOP OLDS` *nn*
   b. Archive all OLDSs.
   c. Delete PRIOLDS and SECOLDS records in the RECON data sets, using the `DELETE.LOG` command.
   d. Scratch and reallocate OLDSs.
   e. `/START OLDS` *nn*

Using Newly Initialized Volumes for OLDS

If a newly initialized (or reinitialized, but unformatted) volume is to contain an OLDS, you must format either the volume or the space occupied by the OLDS before the online system uses it. If you do not format the volume, or if the block size of the new OLDS data set is not the same as the existing OLDS data set, you can expect severe performance degradation and excessive device and channel usage until IMS completely fills the OLDS once. This problem is especially noticeable during emergency restart and during XRF tracking and takeover.

You can use any of the following techniques for formatting a volume for an OLDS:

- Copy an existing OLDS (of the same size) into the new OLDS.
- Copy an existing volume into the new volume, scratch the volume table of contents (VTOC), and allocate the new OLDS.
- Use another IMS subsystem to fill the OLDS (turn on all traces to the log and issue checkpoint commands until the OLDS fills).
- Write a program that either writes at least one byte of data into each track on the volume or fills the OLDS with maximum logical record length (LRECL) blocks.

Controlling the Write-Ahead Data Set

Table 32 lists the actions required to change WADS characteristics.

Table 32. Changing WADS Characteristics

Modification	Actions Required
Single to Dual[1]	1. Shut down IMS. 2. Allocate new WADS. 3. Define a DFSMDA member. 4. Add DD statement in IMS JCL, if necessary. 5. Code `WADS=D` in IMS JCL. 6. Modify operating procedures. 7. Restart IMS with `FORMAT WADS` keywords.
Dual to Single[1]	1. Shut down IMS. 2. Code `WADS=S` in IMS JCL. 3. Delete DFSMDA member. 4. Remove DD statement in IMS JCL, if necessary. 5. Modify operating procedures. 6. Restart IMS.
Adding Spare	1. Allocate a spare WADS. 2. Update WADSDEF specification in the DFSVSMxx member of IMS.PROCLIB. 3. Define DFSMDA member. 4. Add DD statement in IMS JCL, if necessary. 5. Modify operating procedures. 6. `/START WADS` *n* (or wait until IMS restart).
Removing Spare	1. `/STOP WADS` *n* (and wait for dynamic deallocation). 2. Scratch spare WADS. 3. Update WADSDEF statement in the DFSVSMxx member of IMS.PROCLIB. 4. Remove DD statement in IMS JCL. 5. Modify operating procedures.
Space, Location, or Allocation[2]	1. Shut down IMS. 2. Scratch and reallocate WADSs. 3. Restart IMS with `FORMAT WADS` keywords.

Table 32. Changing WADS Characteristics (continued)

Modification	Actions Required
Notes:	
1. **Changing the Mode**: WADS can be dynamically allocated and deallocated. To reflect the new mode for WADSs, you must update the following: • Skeletal JCL for the Log Recovery utility (LOGCLJCL member). • IMS startup procedure (`WADS=` execution parameter). • All recovery procedures implemented to recover WADS errors or to close unclosed OLDSs using WADS. 2. **Changing the Space, Location, or Allocation**: All WADS must have the same space allocation (number of tracks) and be on the same type of device.	

Controlling the System Log Data Set

Converting from single to dual SLDSs requires modification in the skeletal JCL (ARCHJCL member) and in all your operational procedures using SLDSs. In the operational procedures, consider using the secondary SLDS when you experience errors in the primary one. No modification is required for online processing because IMS dynamically allocates SLDSs.

Changing the BLKSIZE requires modification in the skeletal JCL (ARCHJCL member). All SLDSs required for online processing must have the same BLKSIZE.

Controlling the RECON Data Sets

Table 33 lists the actions required to change RECON data set characteristics.

Table 33. Changing RECON Data Set Characteristics

Modification	Actions Required
Adding Spare[1]	1. Define a spare RECON data set. 2. Compile DFSMDA macro for the spare data set or add DD statement in IMS JCL and batch JCL if you do not use dynamic allocation.
Removing Spare	1. Delete cluster. 2. Delete DFSMDA member in IMS.SDFSRESL or remove DD statement from IMS JCL and batch JCL if you do not use dynamic allocation.
Replacing Active	**Recommendation:** Stop all IMS subsystems and batch jobs accessing the RECON data sets. 1. Define a spare data set with new space or allocate a spare data set at a new location. 2. `CHANGE.RECON REPLACE (RECON`*n*`)` 3. Define a new spare data set. 4. Continue normal processing.
Single to Dual	**Recommendation:** Stop all IMS subsystems and batch jobs accessing the RECON data sets. 1. Define a spare RECON data set. 2. `CHANGE.RECON DUAL` 3. Define a new spare. 4. Continue normal processing.

Table 33. Changing RECON Data Set Characteristics (continued)

Modification	Actions Required
Note:	
1. **Adding a Spare RECON Data Set**: The spare data set must be in VSAM CREATE mode.	

Recommendation: For both online and batch, use dynamic allocation for RECON data sets, and run with at least three RECON data sets.

Connecting and Disconnecting Subsystems

Before an IMS subsystem can access databases in an external subsystem, (another program executing in an z/OS address space), such as DB2, you must connect the IMS subsystem to this other subsystem. IMS can only connect to another subsystem if that subsystem is identified in the subsystem member in IMS.PROCLIB. Specify the subsystem member name in the `SSM` EXEC parameter. When specified to and accessed by IMS, the subsystem member name cannot be changed without stopping IMS.

Related Reading: For information on the `SSM` EXEC parameter, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Connections between an IMS subsystem and another subsystem can occur with or without operator intervention, and without regard to which subsystem is available first.

- Automatic connection: When the SSM EXEC parameter is specified, IMS will automatically establish the connection when it processes the parameter.
- Operator-controlled connection: When the SSM EXEC parameter is not specified, the connection will be established when the /START SUBSYS SSM command is entered. The /START SUBSYS SSM command specifies the subsystem member in IMS.PROCLIB that IMS will use in connecting to subsystems.

Disconnecting IMS from another subsystem is initiated with the /STOP SUBSYS command, and normally completes without operator intervention. IMS will quiesce all dependent region external subsystem activity prior to stopping the subsystem.

Chapter 28. IMS System Recovery

IMS has many features that provide high availability and complete recovery of the IMS system in all operating environments. The major features are:

- Extended Recovery Facility (XRF)
- Remote Site Recovery (RSR)
- IMS running in a parallel sysplex environment and using shared queues, shared data, or both

The following sections in this chapter discuss XRF and RSR. For more information about IMS in a parallel sysplex environment, see Part 6, “IMS in a Parallel Sysplex Environment,” on page 313.

XRF and RSR are two features of IMS that can, optionally, be used to increase the availability of IMS systems and the data in IMS databases. Both rely on duplicating IMS subsystems and data on another z/OS system.

The first of these is the Extended Recovery Facility (XRF). The XRF functions are delivered as an integral part of the IMS program product. It is intended to provide increased availability for IMS subsystems. There is an overhead, both in machine usage and support, in using XRF. However, if you have an application that can only tolerate minimal outages, then you might consider using XRF.

The second of these features is Remote Site Recovery (RSR). RSR is a separately priced component available with IMS. It provides similar facilities to XRF, but with some differences.

Both features rely on having another IMS subsystem, situated on another z/OS system, that tracks the update activity of the primary IMS subsystem (only one for XRF, one or more for RSR) to provide a backup.

The differences between the two features is discussed in “Comparison of XRF and RSR” on page 309, but to summarize:

- XRF is suitable for situations where you have a single IMS DB/DC system that requires very high system availability (greater that 99.5%). However, the second z/OS containing the tracking IMS system must be channel attached to the z/OS system that the first IMS is running on.
- RSR is suitable for situations where you have one or more IMS subsystems, running in a number of address spaces on a single z/OS system, where you wish to minimize data loss in a failure situation, but can tolerate outages of approximately one hour. RSR uses network connections between the two z/OS systems, so there are no restrictions on the distance separating them.

The following sections discuss XRF and RSR in more detail.

Overview of Extended Recovery Facility (XRF)

XRF works by having a second, alternate, IMS running. The alternate IMS runs on a separate z/OS image that preferably is on a physically separate machine. The alternate IMS tracks the work of the active IMS system by using the IMS log data sets. XRF gives you the ability to perform hardware maintenance and maintenance on other system software products without interrupting the availability of the IMS application.

The principal drawbacks of XRF are:

- It will not protect against application errors. If the outage is caused by an application error, the same application message may be re-presented on the alternate IMS and cause it to fail.
- It will not, in itself, protect against network outages. You will have to plan for this separately.
- XRF does not support DB2 databases. However, if you are designing an application of this sort, it would be better to use IMS databases, particularly the Data Entry Database (DEDB). The DEDB has provisions for performing most database maintenance with the databases remaining available. It will also automatically maintain multiple copies of the data sets containing the data to guard against media failure.
- Some maintenance to the IMS software requires it to be applied to both the active and standby IMS systems at the same time.

Although XRF can prevent most unplanned and planned outages, it cannot keep the IMS system available indefinitely. You will eventually need planned outages for software maintenance and upgrades and some changes to the IMS configuration. IMS systems running with XRF have achieved continuous availability for years.

Overview of Remote Site Recovery (RSR)

RSR is a separately priced feature available with IMS. It provides similar facilities to XRF, but with some differences.

RSR can track details of IMS full-function databases, Fast Path DEDBs, IMS TM message queues and the current IMS TM telecommunication network, all on an alternate machine. This machine is connected to the machine with the active systems on by a network connection using the VTAM APPC protocol. The VTAM connection is between separate transport manager subsystems (TMS) on the active and tracking machines.

The transport manager subsystem on the active machine collects all log data from all IMS systems (DB/DC, DCCTL, DBCTL and batch) that are defined for RSR tracking and sends this data across to the tracking machine.

The TMS on the tracking machine receives this data and passes it to a single IMS DB/DC region. This processes the data and logs it using normal IMS logging. Depending on what level of tracking has been requested, the IMS region may also apply the updates to the IMS databases.

If there are any interruptions to the network connection, RSR will note the gaps in the logging and perform catch up processing when the link is re-established.

The IMS system on the tracking machine normally can only process input from the TMS. It only becomes a fully functioning system if it has to take over.

Not all databases are tracked. You define the databases that are to be tracked by specifying this when you define them to DBRC.

Related Reading: For more information about RSR, see *IMS Version 9: Administration Guide: System*.

Comparison of XRF and RSR

Table 34 gives a comparison of the features of XRF and RSR.

Table 34. Comparison on XRF and RSR Features

XRF	RSR
Uses same physical log data sets and database data sets for active IMS and alternate IMS.	Uses completely separate log data sets and database data sets.
Active and alternate IMSs must be within the distance restrictions of the channel-to-channel connection between them.	Active and tracking systems are connected by network. Only limit on separation is network response.
Active IMS and alternate IMS must use IMS TM.	Active IMS can be any system updating IMS resources DB/DC, TM only, DB only, or batch. The tracking IMS must be DB/DC.
One-to-one relationship between active IMS and alternate IMS.	One tracking IMS tracks many active IMSs.
All committed updates recorded on alternate IMS.	Possible for gap in data at tracking IMS after unplanned takeover.
Switching to or from alternate IMS comparatively simple.	Takeovers more complex than XRF.
Switch to alternate IMS in order of one minute.	Switch to alternate can take an hour or more.

Chapter 29. IBM IMS Tools

This chapter will be filled in when we get closer to the IMS V9 GA in order to have the most recent information.

Part 6. IMS in a Parallel Sysplex Environment

Chapter 30. Introduction to Parallel Sysplex

In 1990, the *sysplex* was announced as the strategic direction for large systems computing environments and described it as "... a collection of z/OS systems that cooperate, using certain hardware and software products, to process work". The term sysplex is derived from the words SYStems comPLEX. At the time of this first announcement, and for several years thereafter, there was no Parallel Sysplex—only a base sysplex. The base sysplex provided improvements in inter-processor communications between systems and any subsystems wishing to exploit those services, but no data sharing services.

The Parallel Sysplex was introduced later in the 1990s and added hardware and software components to provide for sysplex data sharing. In this context, data sharing means the ability for sysplex member systems and subsystems to store data into, and retrieve data from, a common area. In short, a Parallel Sysplex can have multiple central processor complexes (CPCs) and multiple applications (like IMS) that can directly share the workload.

Although the Parallel Sysplex environment is complex, it basically consists of three elements:

- Two z/OS components: cross-system coupling facility (XCF) and cross-system extended services (XES)
- One hardware component called the coupling facility (CF)

Since Version 5, each release of IMS adds more features that are based on Parallel Sysplex technology. Most of these features are discussed in the following sections of this chapter.

- Data sharing
- Shared queues
- VTAM Generic Resources
- VTAM multi-node persistent sessions
- Automatic restart management
- Sysplex communications
- Operations manager and single point of control
- Resource Manager and sysplex terminal management
- Coordinated global online change
- Automatic RECON loss notification

Future releases of IMS will continue to exploit the Parallel Sysplex environment.

This book uses the term sysplex as being synonymous with Parallel Sysplex.

Related Reading: For a further discussion of IMS in a sysplex environment, see:

- "Goals of a Sysplex Environment" on page 316
- "IMS and the Sysplex Environment" on page 316
- "IMS DB and the Sysplex Environment" on page 316
- "IMS TM and the Sysplex Environment" on page 320
- "Other Advantages of Running IMS TM in a Sysplex Environment" on page 329
- Chapter 31, "IMSplexes," on page 337

- *IMS Version 9: Administration Guide: System*

For a detailed discussion of IMS in a sysplex environment, see:

- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*

Goals of a Sysplex Environment

The goals of a sysplex environment are:

- High availability
- Capacity
- Workload balancing

High availability means that the end user has access to the facilities and resources needed to perform some business function when it is needed. Parallel Sysplex allows you to have multiple IMS systems that can do the same work. These systems are called clones (meaning each IMS system definition is the same). The clones must have access to the same data and the capability to process the same transactions or work requests. High availability requires that you survive both planned outages, such as upgrades, and unplanned outages, such as abends. Even with clones, you need the capability to quickly restore any failed system.

Adequate capacity means that the servers have the resources to satisfy all of the work requests from its end users within the needed time frame. Parallel Sysplex allows you to meet capacity needs by easily adding and deleting cloned IMS systems. When you change your configuration in this way, you can route the work to the systems that can handle it.

Workload balancing is the spreading of work across the systems that can do it and spreading that work appropriately. Parallel Sysplex provides capabilities to automatically balance workloads. Proper balancing allows you to more easily handle unexpected peaks in workloads. For balancing to work best, it must be able to dynamically adjust to changes in the workload and changes in your hardware and software configurations.

IMS and other products, when used in a Parallel Sysplex, provide the facilities to increase availability, enhance capacity, and more easily balance workloads.

IMS and the Sysplex Environment

Use the following procedure to enable IMS to make the most of the sysplex environment:

1. Implement data sharing among multiple IMS DBs within the sysplex. For more information on this topic, see “IMS DB and the Sysplex Environment.”
2. Distribute the transaction workload of multiple IMS TMs within the sysplex. This involves distributing connections and transactions. For more information on this topic, see “IMS TM and the Sysplex Environment” on page 320.

IMS DB and the Sysplex Environment

The foundation in using Parallel Sysplex with IMS is the implementation of data sharing by the Database Manager. Data sharing allows multiple IMS subsystems to access and update the same IMS databases.

Block-level data sharing means that each IMS server has the authority to access and update all shared data. As many as 255 IMS subsystems on up to 32 logical partitions (LPARs) or processors can share IMS databases. Each IMS has full capability to access and update with integrity.

Figure 96 shows a data sharing configuration with four IMS systems running on four z/OS images sharing the same set of IMS databases. Each IMS system has its own database buffer pools. Each IMS reads and updates the databases. To support the integrity requirements, IMS utilizes structures in coupling facilities. A lock structure is used to hold locking information that is shared by the lock managers (IRLMs) used by the IMS systems. Information about database blocks and their locations in the buffer pools is stored in cache structures. These locks are used to maintain the integrity of the buffer pools when an IMS updates a block.

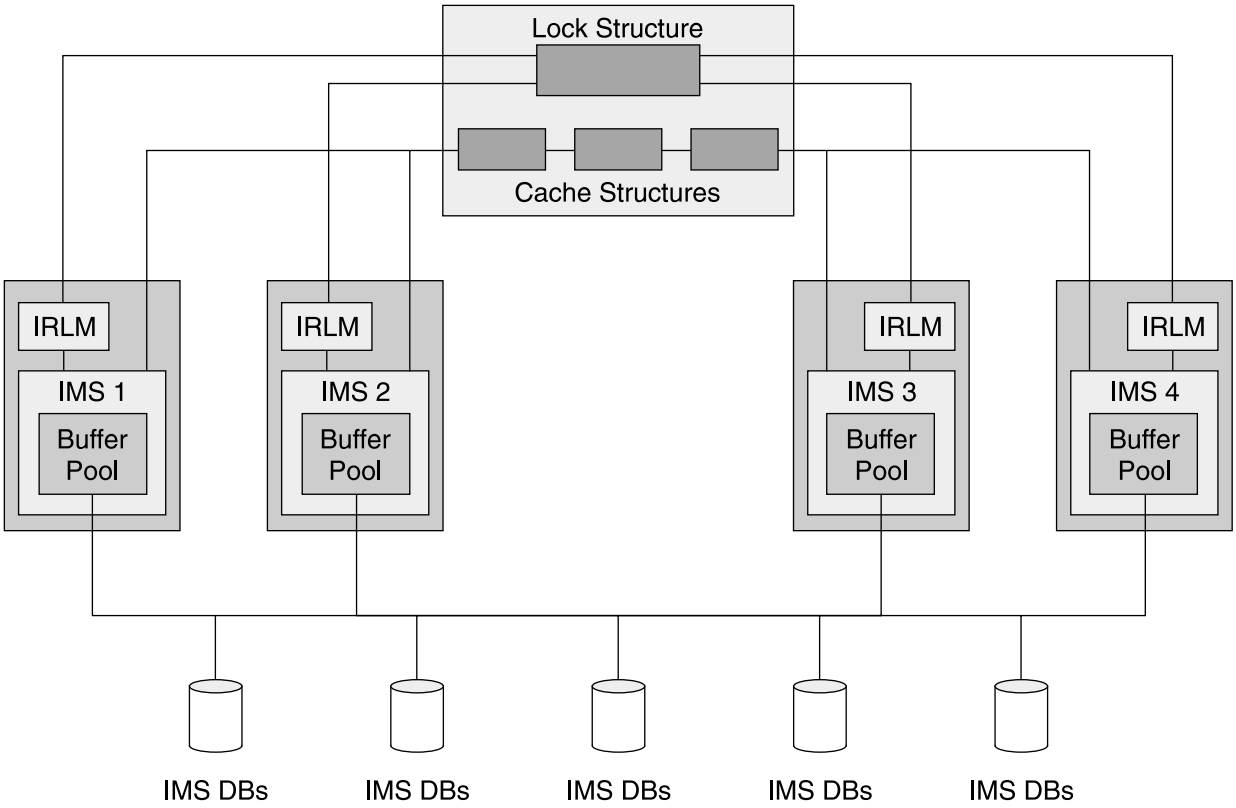

Figure 96. Example of a Data Sharing Configuration with IMS DC/DB, DBCTL, and IMS Batch Jobs

There are no restrictions on what IMS data can be shared. If you are using MSDBs, they must be converted to DEDBs.

IMS does not force data affinities on an installation because all IMS data can be shared. A data affinity occurs when some data, such as a database, is not shared. Its access can only occur from one system. Without data affinities, a transaction or batch process is capable of running on any system. It does not have to be routed to a particular IMS because that is the only one with access to the data.

Dependent Regions and Grouped IMSs in a Sysplex

Outside of the sysplex environment, a dependent region runs with only one control region. You do not move a dependent region from one IMS to another. With Parallel

Sysplex you might want this ability, especially for BMPs. Use the IMSID parameter in the dependent region to specify to which control region the dependent region will connect. Each IMS has a unique IMSID. This makes the movement of a dependent region, such as a BMP, difficult. It seems we have to change the JCL to specify a different IMSID before we can move the dependent region. In the example shown in Figure 97, we want to execute dependent region BMPY (that has IMSID=IMS2 specified) on z/OS3 where IMS3 is running. This will not work. z/OS fails because BMPY is associated with IMS2 (IMSID=IMS2) and we can't dynamically change the IMSID that is specified in BMPY. But, there is a solution.

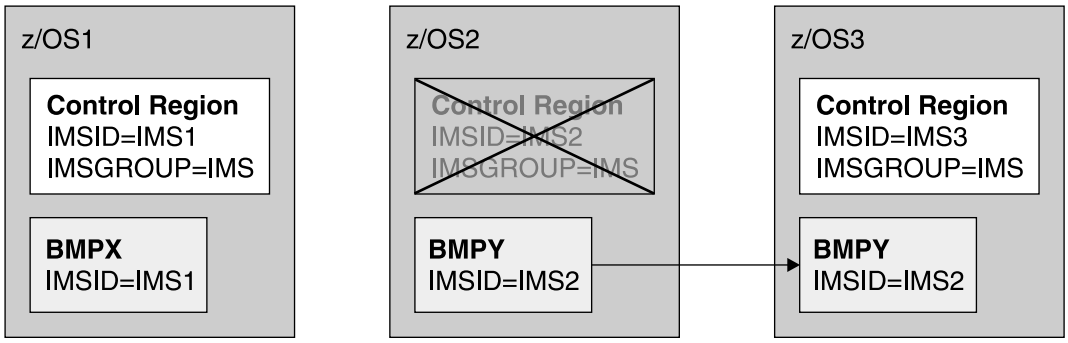

Figure 97. Moving a Dependent Region Between IMSs

The IMSGROUP parameter allows IMS to be known by both its IMSID and its IMSGROUP name. All the IMSs must have unique Musics, but they may have the same IMSGROUP name. They register this group name using z/OS token services and their own unique IMSID. The BMP has an IMSID equal to the IMSGROUP name. The BMP uses token services to see if there is an IMSID using the" IMS" as an IMSGROUP parameter. In the example shown in Figure 97, BMPY finds that it is running on the same z/OS with IMS3. It would connect to IMS3. In this way, if IMS2 is not running or that system is very busy, the BMP could be routed to any z/OS that has any IMS in the group.

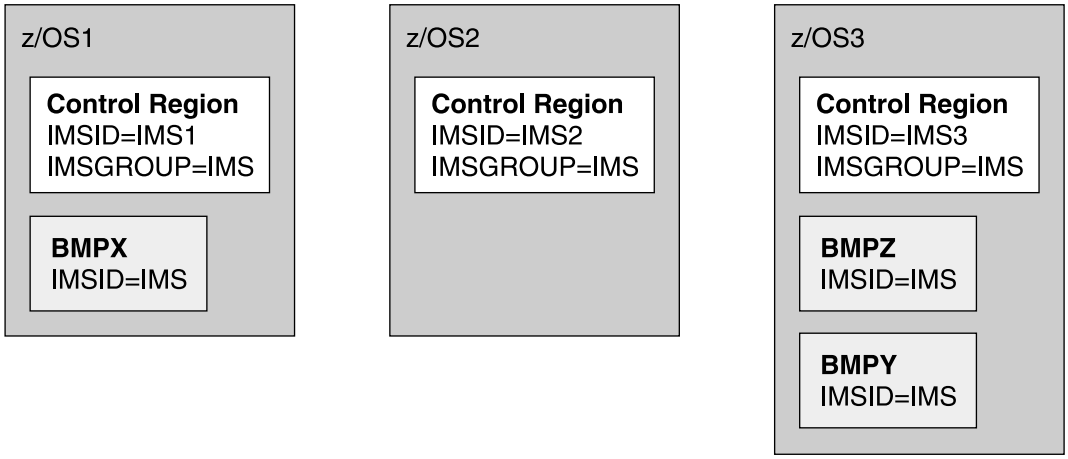

Figure 98. Example of a Dependent Region Running with A Different Control Region

The Program Restart Facility (5655-E14) may be used to easily restart a failed BMP on another IMS system. This further extends the capabilities to use BMPs with Parallel Sysplex.

Fast Database Recovery

Fast Database Recovery (FDBR) can be used to greatly reduce the effect of the failure of an IMS system on data availability to other IMS systems.

If an IMS system fails, the update locks are retained. They must be kept until the inflight work of the failed system is backed out. Without FDBR, this is done during emergency restart. Locked records cannot be accessed by other IMS systems. These systems do not wait for the release of the locks. Instead, their applications get a lock reject condition when they ask for a lock that is retained for the failed system. This lock reject condition typically causes an application abend. So, the failure of one IMS system affects the other IMS systems. They do not have access to some data until the inflight transactions are cleaned up.

FDBR is the solution for the locked records problem. FDBR is an independent region. It runs in its own address space. An FDBR region tracks one IMS control region. For maximum effectiveness, the tracking FDBR region should run on a different operating system than where the tracked IMS is running. Figure 99 illustrates a potential FDBR configuration.

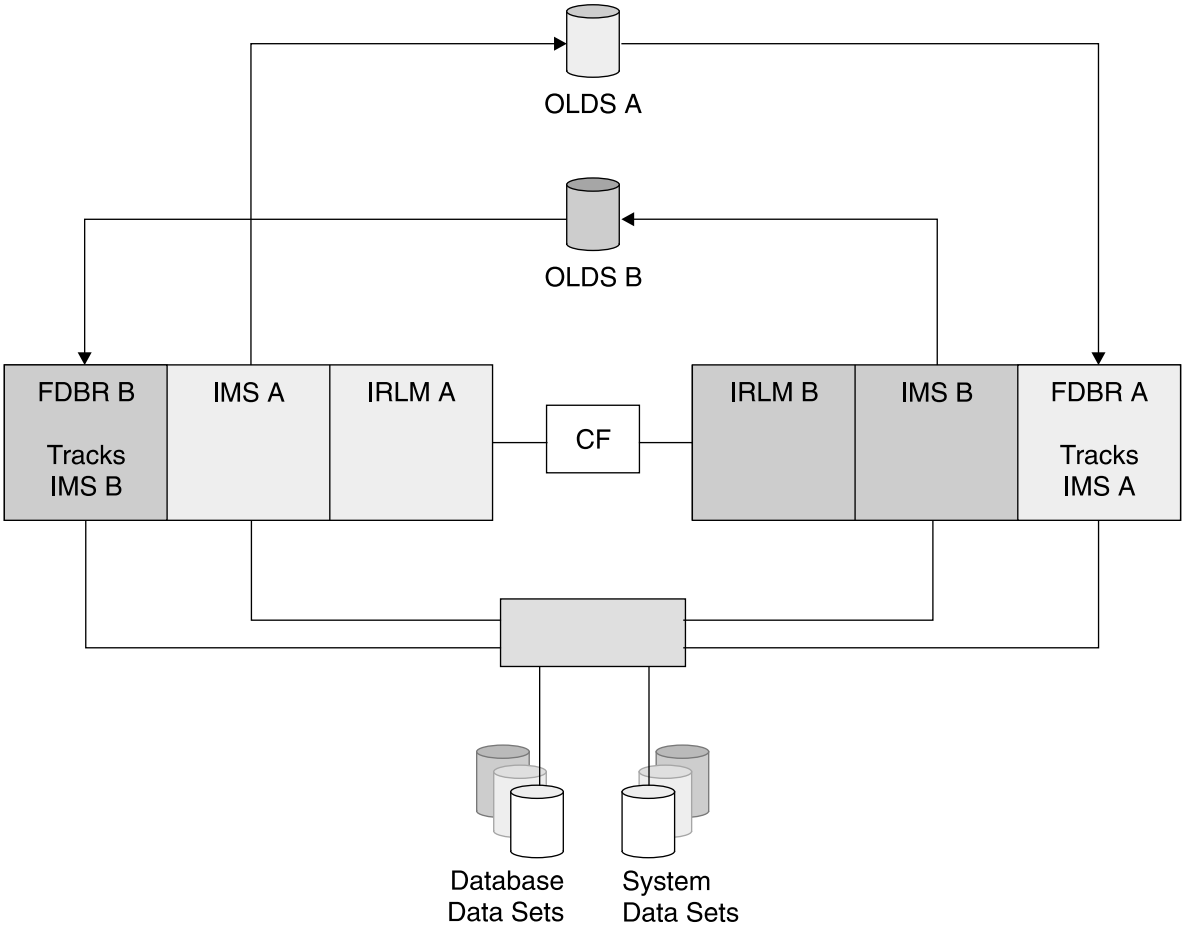

Figure 99. Sample FDBR Configuration

In Figure 99, IMS A and its FDBR run on different systems. IMS B and its FDBR run on different systems. If SYSTEM A or IMS A fails, FDBR A backs out all of the inflight work from IMS A. It also releases the retained locks held for the inflight work of IMS A. This allows IMS B to access all of the IMS data.

Tracking is accomplished by implementing either one of the following methods:

1. An FDBR and its IMS system join the same XCF group as the IMS that is being tracked. This allows FDBR to be immediately aware when the tracked IMS's address space or z/OS terminates.
2. FDBR continually reads the tracked IMS's log (OLDS). If IMS abends, its ESTAE routine writes a failure log record (type X'06'). FDBR can read this log record before IMS's address space terminates.

When either of these tracking methods makes FDBR aware of the IMS failure, FDBR restores the databases to the last point of consistency. For full-function databases, this means it backs out inflight units of work. For DEDBs, this means that it invokes redo processing for incomplete output threads. These are the same actions that emergency restart would have done. When these actions are complete, FDBR releases the locks held by the failed IMS, emulating what emergency restart would have done.

FDBR is much quicker than emergency restart. FDBR does not have to wait for the restart of IMS. It does not have to wait for the loading of the IMS modules. It does not have to wait for the reading of the log except for the last few records. FDBR is much quicker because it eliminates many of the potential lock rejects and application abends on the surviving IMSs.

Summary of IMS DB and the Sysplex Environment

Higher availability is provided by the data sharing configuration. It allows work that might otherwise have to wait for the restart of a failed IMS to run on a surviving IMS. FDBR reduces the impact of the failure by monitoring an active IMS and performing dynamic backout or DEDB redo processing (or both) sooner than an IMS emergency restart would do.

Work can run on any IMSs in the sysplex data sharing group because multiple IMSs have access the same data. Up to 32 processors can be used to provide maximum capacity.

If every IMS can access all the data, then every IMS can process any of the work. This allows an installation to create IMS clones. In fact, a single system definition can be used for all the IMSs. Cloning allows you to distribute the work to the systems that have the capacity to handle it.

IMS TM and the Sysplex Environment

After data sharing is in place, the next logical step in using Parallel Sysplex with IMS Transaction Manager is the distribution of connections. For VTAM, connections are sessions. For TCP/IP, they are socket connections. Distributing connections is one of the methods for distributing the workload across multiple IMSs and multiple processors. This is static distribution. That is, after a user is connected to an IMS, the user remains connected until the connection is broken. Another connection is required to use this method for distributing the workload to another IMS.

Distributing Transaction Workload

There are several ways to distribute the transaction workload. Two basic techniques are:

- Distribute the logons so that not all users are logged on to the same IMS. Whichever IMS they are logged on to is the one that processes the transaction.

- Distribute the transactions between IMSs after a transaction has been received from the network.

There is a combination whereby users' logons are distributed and the transaction submitted by these users are also distributed after they are entered.

It does not matter where the transaction is processed if you are data sharing.

Distributing Logons Manually

The earliest approach to distributing logons was to tell the end user which IMS to log on to. There are several problems with this technique:

- Balancing the logons becomes an administrative responsibility which must be monitored continuously as users come and go.
- As new IMSs join the group, either no users log on to that IMS (because they do not know about it), or the administrator must reassign users to the new IMSs.
- If an IMS fails, users have to be instructed either to wait for it to restart or to log on to another IMS. After a user knows of another IMS, the user might decide arbitrarily to log on to it instead of his primary IMS, defeating the balancing goal.

Distributing Logons Automatically

Distributing logons can be done automatically using several techniques.

For SNA networks, VTAM Generic Resources can be used to dynamically route a logon request to an active IMS. The IMS is chosen based on Workload Manager (WLM) information or the number of users currently logged on. This capability is available with IMS Version 6 and later releases.

Prior to IMS Version 6, logons could be distributed automatically using a VTAM USERVAR exit. This exit can be used to direct the logon request to one of several IMSs in the group. Although this method is still supported, IBM recommends using VTAM Generic Resources.

For TCP/IP, connection distribution can be accomplished using such tools as DNS/WLM, IND, or the Sysplex Distributor.

The following sections discuss these possibilities.

VTAM USERVAR Exit: USERVAR is a VTAM capability that can change the value specified for the VTAM application name in a logon request. USERVAR support includes an optional exit routine. The exit routine can choose from multiple application names. So, a USERVAR exit routine can be used to route a logon to any IMS that it knows about. But, it might not know of configuration changes or of the availability of any particular IMS in the group. For example (see Figure 100 on page 322), if IMS1 fails, the exit routine might continue to route logons to IMS1. Similarly, if IMS4 is added to the configuration, the exit routine might not route any logons to it. A sophisticated routine might be able to modify its decisions, but there is no automatic notification to the routine of changes in the configuration.

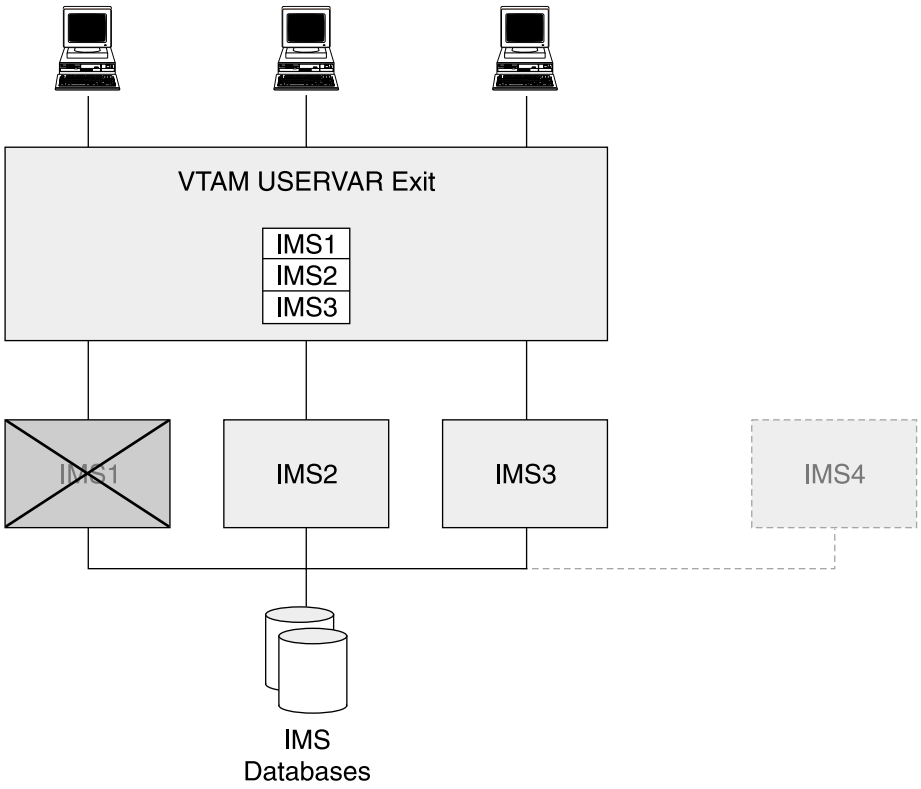

Figure 100. Example of VTAM USERVAR Exit Routing IMS Logons

VTAM Generic Resources: VTAM Generic Resources (VGR) is a service provided by VTAM in a Parallel Sysplex. It minimizes the knowledge that an end user needs to log on to one of several like instances of an application, such as IMS. Each instance of an application, joins a Generic Resource Group by specifying both the group name and its specific VTAM application name. End users specify the group name when logging on. VTAM selects a member of the group for the session.

Generic Resource Groups are dynamic. When a new IMS opens its VTAM ACB, it joins the group and is a candidate for subsequent logons. When an IMS terminates, it is removed from the group. It is then no longer a candidate for logons.

Information about the members of a group is kept in a coupling facility structure (see Figure 101 on page 323).

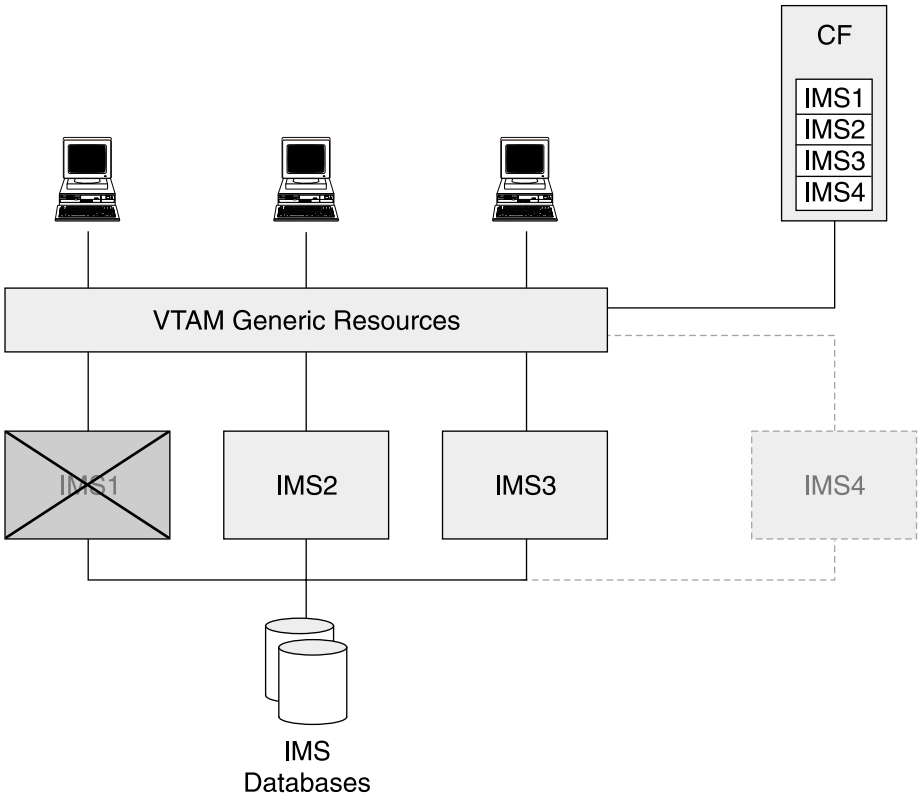

Figure 101. VTAM Generic Resources Distributing Logons In a Sysplex

APPC/IMS can use VGR, but this does not require direct IMS support. Instead, this support is provided by APPC/MVS.

There are many benefits of VGR over other techniques for distributing logon requests. Some of these benefits are:

Availability
VTAM knows by looking in the CF structure which IMSs are active. It routes requests only to active IMSs. If an IMS fails, its users can immediately log on again using the same generic name. They will be connected to one of the active IMSs.

Capacity
If another IMS is needed to handle the workload, it immediately becomes eligible for user logons. User procedures do not have to be modified.

Workload Balancing
VTAM attempts to balance logons across the available IMSs. It has two ways of doing this.

- If Workload Manager (WLM) goal mode is used, VGR routes a logon to the system with the most available capacity.
- If WLM goal mode is not used, VGR attempts to balance the number of logons for each IMS system.

You can implement a VGR user exit routine to override the VGR decision.

Web and TCP/IP Connections to IMS

Many installations access their IMS systems using TCP/IP. This includes connections from the Web. Web servers can use many different ways of connecting to IMS. The most typical connections are:

APPC If the Web server sends requests to z/OS using APPC protocols, then the connections to IMS can be distributed using APPC/MVS support for VTAM Generic Resources.

TCP/IP Telnet
TN3270 allows 3270 users to use TCP/IP protocols. The end user is a TN3270 client. The TN3270 client communicates with a TN3270 server using TCP/IP. The TN3270 server uses LU2 (3270) protocols to communicate with IMS through VTAM. VGR can be used with TN3270 servers to provide connection balancing.

TCP/IP Sockets
If the Web server uses sockets, the server can communicate with IMS Connect for z/OS, which communicates with IMS. IMS Connect executes in its own address space. It communicates with its client, in this case the Web server, using TCP/IP socket protocols. It communicates with IMS using the IMS Open Transaction Manager Access (OTMA) protocol. OTMA uses z/OS Cross System Coupling Facility (XCF), which allows programs running in different address spaces, possibly on different z/OS images in the Parallel Sysplex, to send and receive messages from each other. The distribution of connections from Web servers to IMS Connect must be done with TCP/IP socket protocols.

Domain Name Server/Workload Manager (DNS/WLM) can be used in conjunction with TN3270 to distribute the connections requests across multiple TN3270 servers in the Parallel Sysplex. The TN3270 client request goes to one DNS/WLM, which then uses the WLM to decide which TN3270 server should get the connection request. After the DNS/WLM chooses a TN3270 server, it is no longer involved (communications goes directly between the TN3270 client and the TN3270 server). The TN3270 server can then use VTAM Generic Resources to distribute sessions across the IMS members. VGR will always use a local IMS if one is available. A local IMS is an IMS that is using the VTAM that the TN3270 server is using. However, if the TN3270 server is on an z/OS image without an IMS, VGR can send the logon request to an available IMS on any z/OS.

Figure 102 on page 325 is an illustration of Telnet 3270 use. The diagram shows four systems. The system in the upper portion of Figure 102 on page 325, there is a DNS/WLM, a TN3270 Server, and an IMS. The other three systems in the Parallel Sysplex have second copies of these. The second DNS/WLM in the diagram is a backup for the first (in case the first one fails).

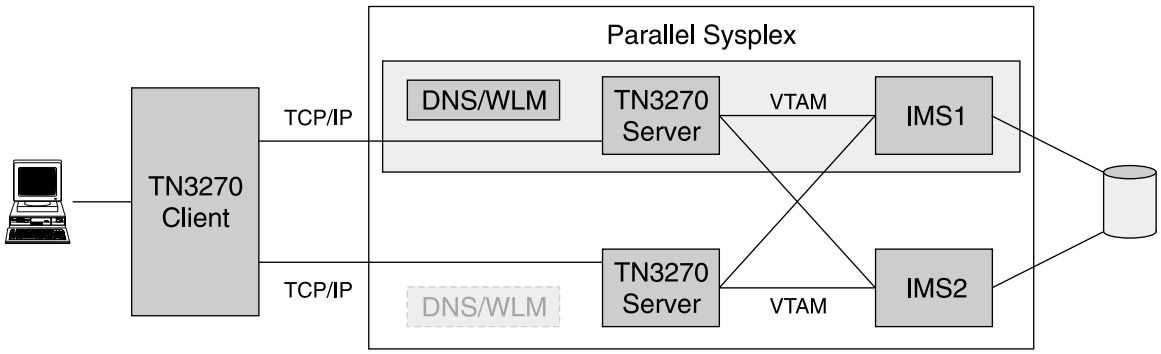

Figure 102. TN3270 Client Connecting to IMS

While the configuration illustrated in Figure 102 provides good connection balancing, it is fairly expensive (in CPU usage) to establish and terminate the connection. So, this is not a good configuration for connections that are short term.

The Interactive Network Dispatcher (IND) can be used to distribute connection requests from a Web server to one of several IMS Connect address spaces in the Parallel Sysplex. IND is much more efficient than DNS/WLM at handling connection requests, but requires a separate hardware box (such as a 2216 router) and so can be more expensive. IMS Connect then sends work to one of several IMSs using XCF services and IMS OTMA. This is illustrated in Figure 103.

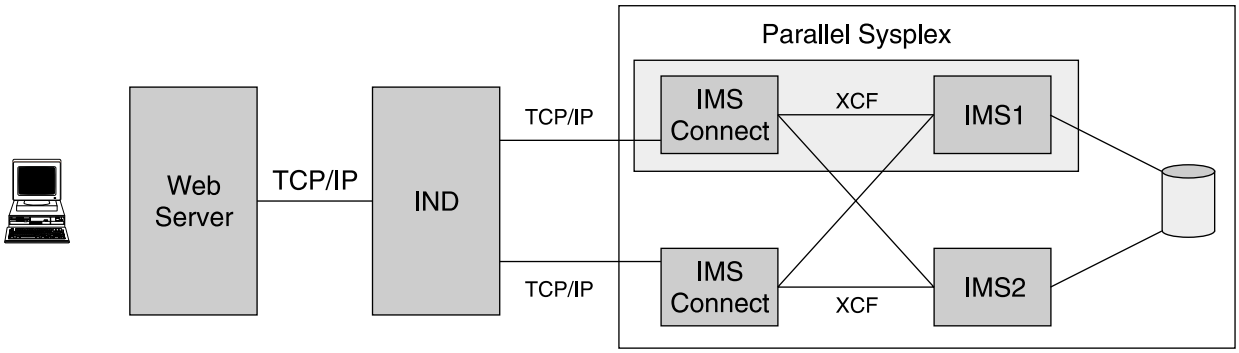

Figure 103. IND Connecting to Multiple IMSs via IMS Connect

In Figure 103, both IMSs can be reached through either IMS Connect. Exit routines in IMS Connect can be used to choose to which IMS the request will be sent. These routines may have available to them information about which IMSs are currently active.

The network dispatching function of IND is included in WebSphere Edge Server. There are WebSphere Edge Servers for LINUX, AIX®, Windows®, and Sun Solaris.

A better product for both long and short connections is the Sysplex Distributor. It is software that runs on the host system and distributes sockets across multiple target systems. Like DNS/WLM, a backup Sysplex Distributor can be running on another z/OS in the sysplex.

In the case illustrated in Figure 104 on page 326, there are multiple instances of IMS Connect. Inputs go through the Sysplex Distributor. Responses do not go

through the Sysplex Distributor. In Figure 104, the responses go directly from IMS Connect to the Web Server.

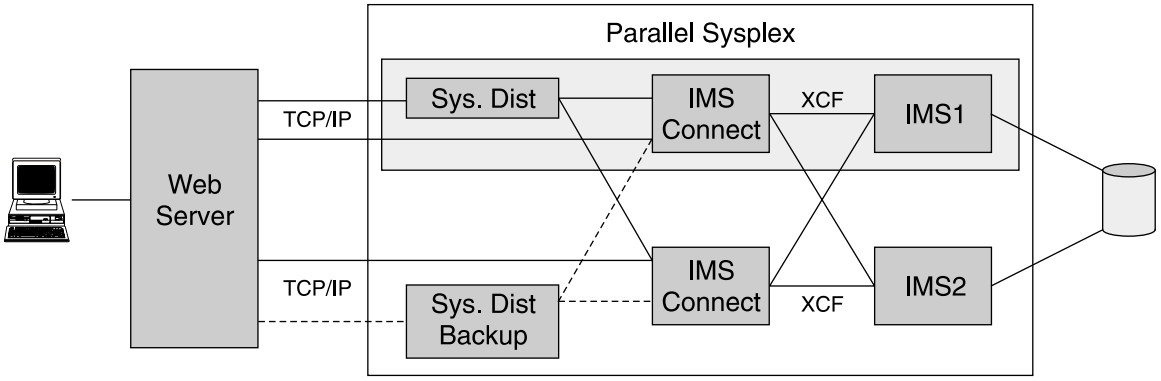

Figure 104. Web Connections to IMS Using the Sysplex Distributor and IMS Connect

Distributing Transactions

After the connections are distributed within the sysplex, the next step in using Parallel Sysplex with IMS Transaction Manager is the distribution of transactions. This involves processing a transaction on a system other than the one that initially received the input message

The techniques used to distribute connections from users across IMS systems might not balance the workload. Several things can cause this:

- A large batch workload on one system might overload it. Users who are already connected to that system remain connected to it. Their response times could be affected by this overload.
- The volumes of inputs from connected users might vary. This could result in peak loads on one system while another system has a lull in inputs.
- Other factors could also cause unbalanced workloads across the systems.

The techniques to address this imbalance involve distributing transactions to other IMS TM systems.

When you distribute transactions, an input transaction can be routed from one IMS to another for processing. There are two methods of doing this in IMS.

Multiple Systems Coupling (MSC)
With MSC, multiple IMS systems are connected by communication links. An IMS system may send a message across one of these links to another IMS. The receiving system processes the transaction and sends its reply to the original system. The original system sends the reply to the user.

IMS Shared Message Queues
With shared queues, IMS systems share one set of message queues. They are stored in list structures in coupling facilities. Any IMS system can process a transaction because the queues are available to all the IMS systems. Those with more processing capacity tend to process more transactions.

With either of these implementations, a user can be connected to any system and have an input message processed on another IMS system. There are some limitations for APPC and OTMA connections. In some cases, input messages from these connections must be processed by the system where the connection exists.

Routing Messages with MSC

As illustrated in Figure 105, MSC is comprised of VTAM sessions between multiple IMS systems. When an IMS transaction is received by any IMS, its definitions determine where that transaction is to execute. The transaction might be processed locally (on the IMS system that receives the transaction). However, the transaction might be processed remotely (on another system). If the transaction is defined to run remotely, the IMS system that receives the message sends it to the remote system.

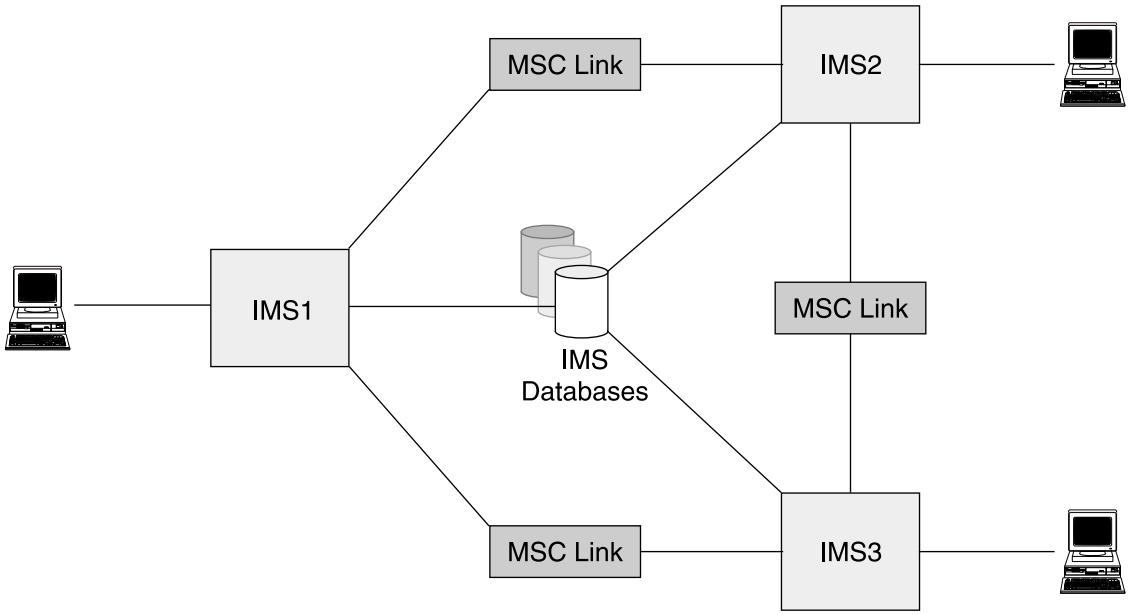

Figure 105. VTAM Sessions of 3 IMSs Connected to Each Other Using MSC

MSC can be used to distribute transactions across multiple IMSs. The definitions are static. An IMS makes its decision about whether or not to process a transaction based on the transaction code. This decision is not dynamic. Decisions are not based on workloads. It is relatively difficult to add a new IMS system to the complex. A new IMS system requires changes in the definitions for the other existing IMS systems.

MSC definitions can be used to distribute the workload, but they do not balance the workload.

A link failure or an IMS failure can mean that a transaction cannot be processed until the failure is corrected.

MSC users can include MSC exit routines to override the definitions of where transactions are processed. This adds some dynamic capabilities to MSC routing. The IMS Workload Router (product number 5697-B87) provides a set of these exit routines.

The Workload Router (WLR) uses MSC Directed Routing to distribute transactions across multiple IMSs without regard to how they have been defined. For example, if TRANA is received at IMS1, it can be processed on IMS1, IMS2, or IMS3. The WLR can be directed to process a certain percentage of transactions locally and to send others to remote IMSs. This provides some workload balancing capability.

Related Reading: For more information about the IMS Workload Router product, see the *IBM IMS Workload Router for z/OS User's Guide*.

Distributing Transactions Using Shared Message Queues

IMS TM is a queue-driven system. Transaction input messages can be received from terminals or programs. Output messages can be sent to terminals or programs. All of these messages are placed in message queues.

In conventional (non-shared queues) systems, each IMS has its own set of queues (see Figure 106). These queues are accessible only by the IMS system that owns them. When MSC is used, one IMS sends a message from its queues to another IMS system, which places the message in its queues. In any case, a message may be processed only by the IMS system on whose queues it resides.

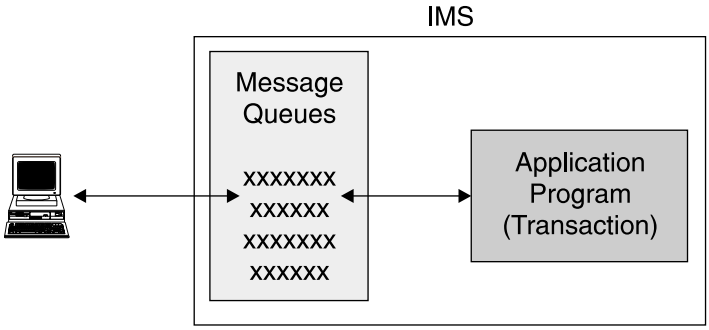

Figure 106. A Single IMS with a Single Message Queue

With shared queues, the message queues are moved to list structures in coupling facilities where they are available to any IMS in the shared queues group (see Figure 107). A terminal is connected to one IMS system. Input messages from the terminal are placed in the shared queues. They are accessible from any IMS using those queues. This means that another IMS system, not the one to which the terminal is connected, can process the input message.

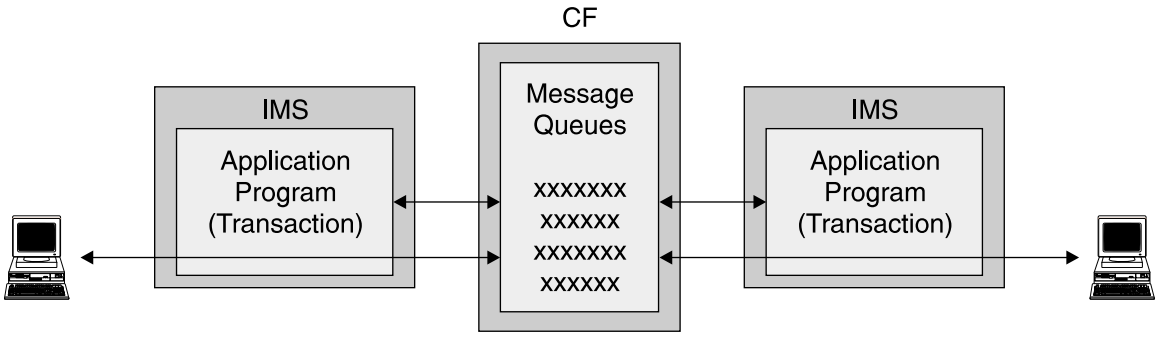

Figure 107. Two IMSs Accessing One Message Queue on a Coupling Facility

All messages (input and output) go into the shared queues. IMS subsystems register interest in specific queues, such as the queue for transaction TRANA or for terminal TERMX. IMS systems register interest for queues it can process. This includes the queues for the terminals connected to it and the transactions which are defined to it.

When there is work on a registered queue, the IMS systems that have registered interest in the queue are notified. When an IMS has the resources available to process the transaction, such as an available dependent region, it attempts to read a message from the shared queue. If it receives a message, it processes it and puts the response back on a shared queue. Multiple IMSs can attempt to retrieve messages from a queue, but only one will receive an individual message. Terminal output messages are retrieved from the queue by the IMS to which the terminal is connected. This IMS sends the output message to the terminal.

Only those IMS systems with available resources will ask for and process the transaction because any IMS can process a shared queues message. This tends to distribute the workload to the systems best able to handle it. Those systems with the most free resources will ask for work most frequently.

So, by using shared queues, the application workload is balanced dynamically. If there is any processing capacity available anywhere in the shared queues group, queued transactions will be scheduled and processed. The user is not forced to wait because a single IMS is overloaded.

Shared queues can also be used in conjunction with connection balancing provided by VTAM Generic Resources or one of the techniques used for TCP/IP.

Summary of IMS TM and the Sysplex Environment

In a Parallel Sysplex, you can take advantage of multiple capabilities:

- VTAM Generic Resources provides connection balancing for SNA sessions. It improves availability for users connected via VTAM and makes it easy to add new systems without changing user logon procedures.
- The various TCP/IP distributors provides connection balancing for users of TCP/IP. They provide improved availability for these users and make it easy to add capacity for systems with these users.
- The IMS Workload Router provides workload balancing capabilities for users of conventional queuing.
- Shared queues provides dynamic workload balancing by allowing an IMS system with available capacity to process any transaction in the shared queues group. Shared queues provides availability benefits by allowing any active IMS to process transactions when another IMS fails. Capacity is easily added because no modifications have to be made to the previously existing IMS systems.

Other Advantages of Running IMS TM in a Sysplex Environment

The next few sections discuss additional functions available with IMS that take advantage of the sysplex environment.

These sections are:

- “Rapid Network Reconnect”
- “Sysplex Failure Recovery” on page 332

Rapid Network Reconnect

Rapid Network Reconnect (RNR) is IMS’s support for VTAM persistent sessions for non-XRF systems. RNR can provide great benefits for some environments, but it is not appropriate for all environments. RNR is optional.

VTAM persistent session support eliminates session cleanup and restart when a host failure occurs. There are two kinds on persistent session support.

- Single node persistent sessions (SNPS) provide support only for IMS failures. With SNPS, the VTAM instance must not fail.
- Multinode persistent sessions (MNPS) provide support for all types of host failures. These include failures of IMS, VTAM, z/OS, or the processor.

With persistent sessions, end users do not lose their sessions for the supported failures. In fact, they remain logged on. Even though their IMS system fails, their sessions are not terminated. This means that the unbind traffic does not flow through the network when the failure occurs. Secondly, when their IMS system is restarted, their sessions do not have to be reestablished and the bind traffic does not flow. For LU types that typically have human users, such as SLUTYPE2 and SLUTYPE1 (CONSOLE), sign-on is required. For LU types that typically are programmable, such as SLUTYPEP, or do not have direct human users, such as LU1 (PRINTER1), sign-on is not required.

Single Node Persistent Sessions

When using RNR with SNPS, only outages due to IMS abends are supported. The VTAM used by this IMS must not fail.

The scenario illustrated in Figure 108 shows how SNPS works. The numbers in Figure 108 refer to the descriptions in the list that follows the figure.

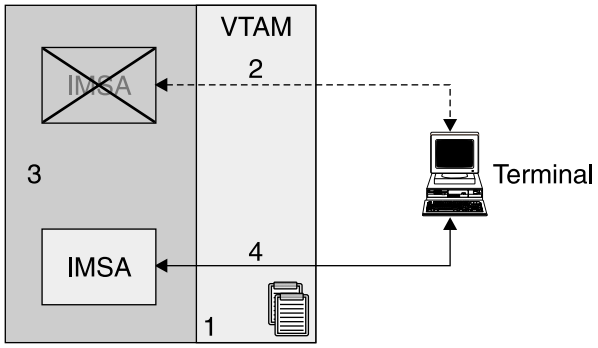

Figure 108. An SNPS Example Scenario Where a Logon is Not Terminated When Its IMS Fails

1. When a session is established, session data is stored in an z/OS data space associated with the VTAM address space.
2. If the IMS system abends but VTAM does not, the session stays active and the session data remains.
3. When the IMS is restarted, the users' sessions are given to the restarted IMS. The users have remained logged on even though their IMS system had failed.

Multinode Persistent Sessions

With MNPS, the session data is stored in a CF structure where it is available to other systems in the sysplex. All types of failures are supported with MNPS. As with SNPS support, when IMS restarted, the users are automatically reconnected in a "logged on" state.

When using RNR with MNPS, all outages of the IMS, VTAM, or processor are supported.

The scenario illustrated in Figure 109 shows how MNPS works. The numbers in Figure 109 refer to the descriptions in the list that follows the figure.

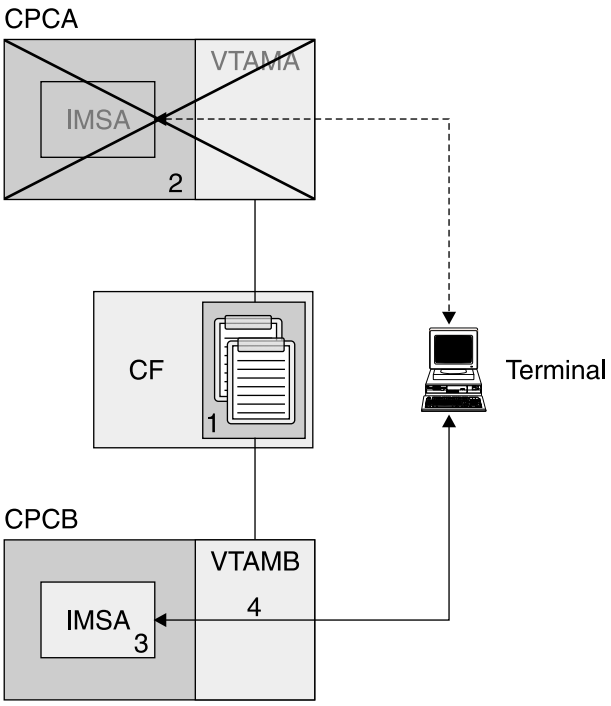

Figure 109. An MNPS Example Scenario Where a Logon is Not Terminated When Its IMS Fails

1. When a session is established, session data is stored in a coupling facility structure.
2. CPC A fails and IMSA also fails because it is running on CPCA. The session data is not lost, however, because it is on the coupling facility. Another VTAM in the sysplex detects the error and the session survives the failure of CPCA.
3. IMSA is restarted on another processor (CPCB) in the sysplex.
4. When IMSA is restarted on CPCB, the users sessions are given to the restarted IMS. These users have remained logged on even though their IMS system failed.

Benefits of Rapid Network Reconnect

The benefit of RNR is the maintenance of the sessions when IMS fails. Most of this benefit is the elimination of the time required to terminate and reestablish the sessions. This eliminates the bind and unbind traffic which would otherwise flow through the network. This traffic can be time consuming. Service to the end users is reestablished more quickly.

Of course, the IMS system has to be restarted. When using RNR, the end user does not have the option of logging on to another IMS in the Parallel Sysplex. The value of RNR depends on how quickly IMS is restarted. If the restart is slow, there is not much benefit. If the restart is quick, the benefit can be substantial. However, if another system with the same capabilities is available, the users would get quicker restoration of service by logging onto it. This means that RNR probably will not be used for IMS systems with clones.

Persistent session support for IMS users of APPC (LU 6.2) is provided by APPC/MVS, not IMS. With APPC, the sessions are persistent, but the conversations are not.

Sysplex Failure Recovery

Parallel Sysplex adds more components to a system. These include clones of systems and subsystems and new components such as coupling facilities and CF links. Even though you might have another component available to do your work when one component fails, you want to restore the sysplex to full robustness as soon as possible. Recoveries from most failures in a sysplex can be automated.

Advantages of Having Multiple Servers

The main advantage in a sysplex is that you have multiple copies of your servers. When one fails, another is available to do its work. This applies to subsystems, processors, and coupling facilities.

- If IMS fails, other IMS instances are available. You can use the routing and balancing capabilities to distribute the work to the active IMS systems.
- If a processor or LPAR fails, Automatic Restart Management (ARM) can be used to restart failed subsystems on surviving processors or LPARs.
- If a coupling facility fails, there are two ways of surviving the loss:
  - You can rebuild the CF structures on another CF.
  - You can use multiple copies of the structures.

Recovery Using Automatic Restart Management (ARM)

When IMS fails, you need to restart it as quickly as possible. Even though other IMS systems might be available to do work, the failed IMS might have inflight or indoubt work that needs to be resolved. This resolution releases locks on database resources and releases DBRC authorizations. This allows new work to have access to all of the data.

ARM can be used to provide rapid restarts of IMS. ARM is a sysplex capability that allows an automatic restart of subsystems like IMS, DB2, CICS, and IRLM. If the subsystem abends, the restart is on the same z/OS instance (LPAR). If the z/OS (LPAR) fails, the restart is on another z/OS in the sysplex.

Figure 110 illustrates the actions of ARM when an IMS abends.

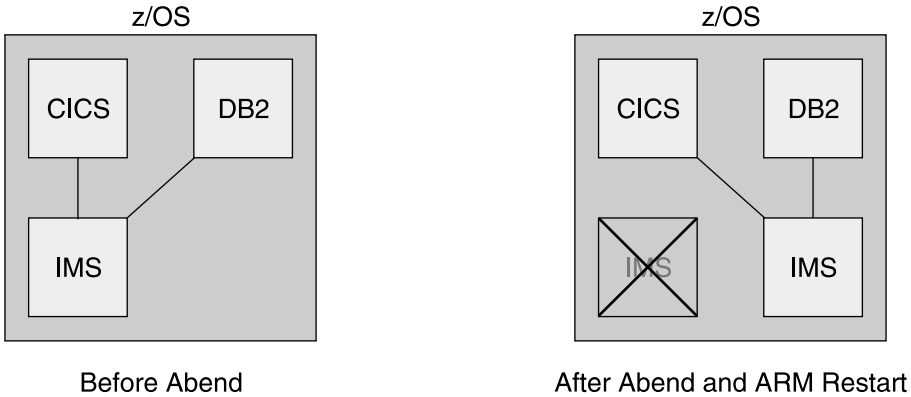

Figure 110. ARM Restarting an IMS that Abended

In Figure 110, IMS is restarted on the same z/OS system. This IMS was providing DBCTL services to a CICS and was using a DB2 subsystem. You must restart IMS

on the same z/OS so that services between these subsystems can be restored. For example, indoubt threads must be resolved.

In the case of an z/OS or processor failure, IMS will be restarted on another candidate z/OS (see Figure 111). The z/OS is chosen according to a user defined ARM policy. Subsystems that must remain together can be restarted as a group on the same z/OS. In the example illustrated in Figure 111, an IMS subsystem is using DB2 for database services. A CICS AOR (Application Owning Region) is using the same DB2 and the IMS for database services. When the z/OS system fails, the IMS, the DB2, and the CICS AOR must be moved together, but the CICS TOR (Terminal Owning Region) can be restarted on another z/OS in the sysplex because CICS AORs and TORs can communicate with other z/OS systems.

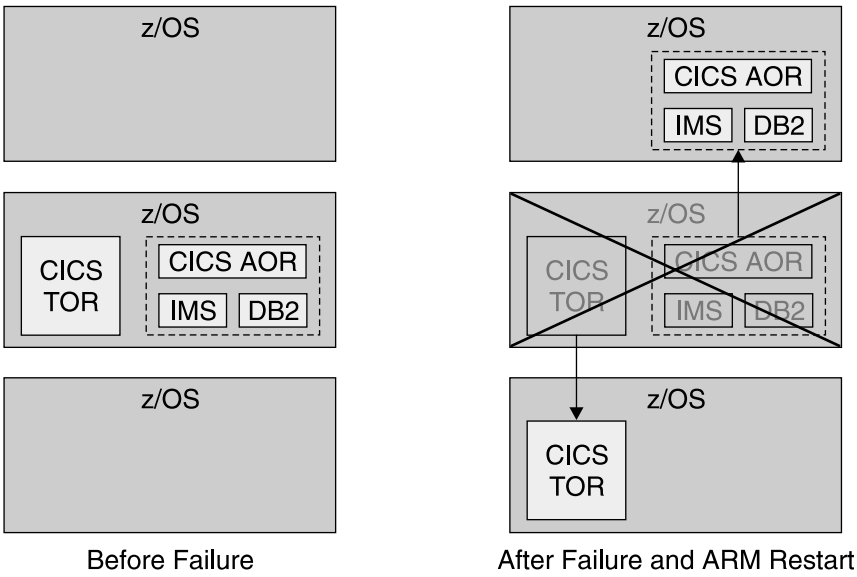

Figure 111. ARM Restarting IMS, CICS, and DB2 After a z/OS Failure

For ARM to restart subsystems, it must be active in the sysplex. ARM is controlled by a policy that the user defines. The policy is stored in an ARM couple data set. The policy is used to group subsystems for restart together. It also controls whether or not a subsystem will be restarted. For example, an installation might not want to restart test subsystems.

ARM only restarts subsystems that register with ARM. This is done when they initialize. IMS has a parameter (ARMRST) that controls whether or not IMS registers with ARM. ARMRST=Y is the default.

IMS has full ARM support. ARM can be used to restart IMS control regions, Common Queue Server regions, Fast Database Recovery regions, Common Service Layer components, and IRLMs. ARM does not directly restart IMS dependent regions. These are typically started by automation when the control region is started.

ARMWRAP is a program that registers an address space for ARM restarts. In is used for a step in a job. If the following step fails, ARM will restart the job. IMS Connect does not register with ARM. ARMWRAP can be used to get ARM support for IMS Connect.

Recovery After Coupling Facility Failures

Much of the sysplex support is provided through the use of coupling facility structures. If a CF is lost, it is important to have access to structures elsewhere.

If a CF survives, but you lose all of the links from a processor to the CF, you need to resolve the problem. This can be treated like the loss of the CF itself. You can either rebuild its structures on CFs that have connectivity to the processors that require it or you can use duplicate structures.

Recovery Using Structure Rebuild: Some structures can be rebuilt automatically when either a CF failure or a CF link failure occurs. These include IRLM lock structures, OSAM and VSAM cache structures, and IMS shared queues structures. The following example will use a IRLM lock structure.

Figure 112 shows an IRLM structure on CF1 before a CF failure.

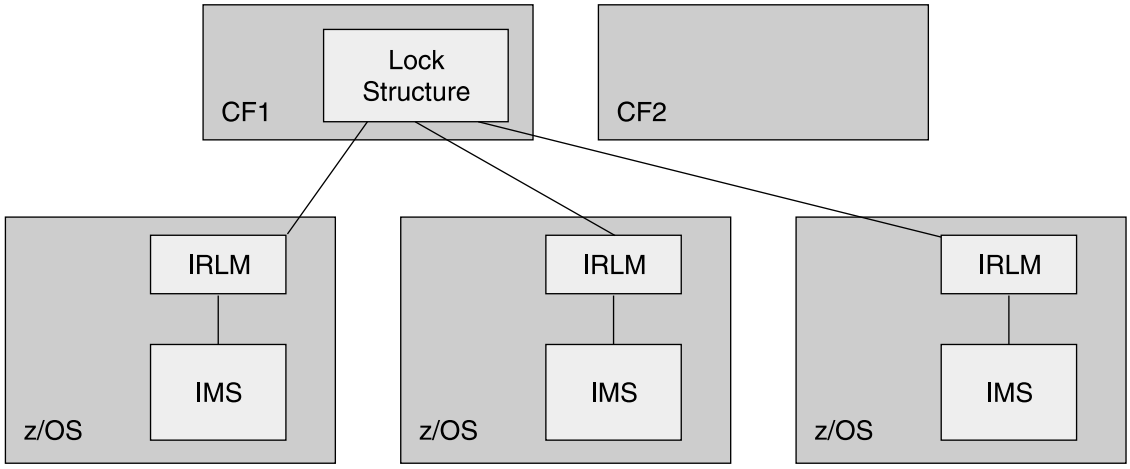

Figure 112. Three IMSs on Three z/OSs Sharing One IRLM Structure on a Coupling Facility

Figure 113 on page 335 shows a scenario where CF1 (on which the IRLM structure resides) fails. When the CF fails, the system automatically recognizes the loss and rebuilds the lock structure on another CF (CF2). Each IRLM retains the information necessary to restore its lock information in the structure. The IRLMs together rebuild the lock structure on another CF. Data sharing is resumed. Similar rebuild and recovery occurs for OSAM, VSAM, and shared queue structures.

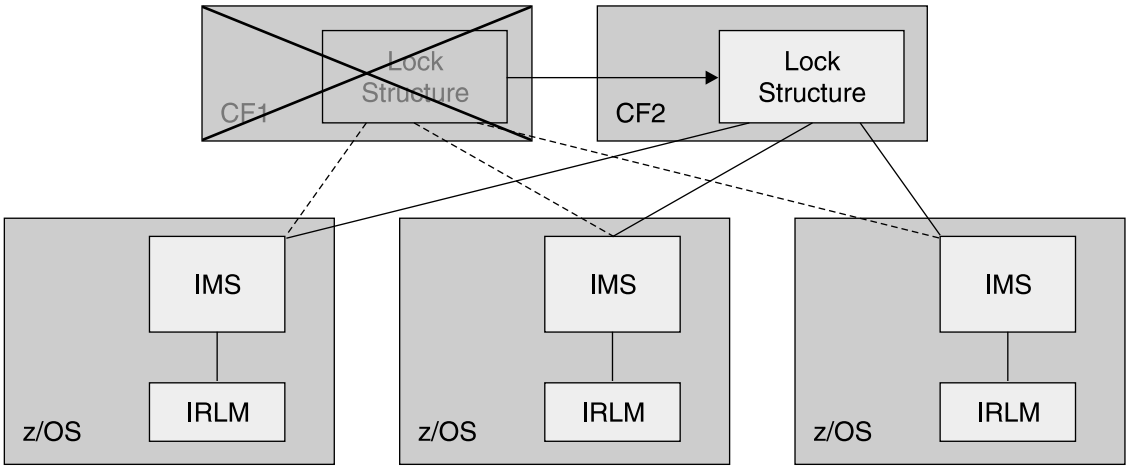

Figure 113. IRLM Structure on Failed Coupling Facility is Rebuilt on Another Coupling Facility

In Figure 114, the CF does not fail. Instead, the connectivity between one of the processors and the CF fails. This case is treated the same as the loss of the CF. That is, the system automatically rebuilds the structure on another CF. All processors have connectivity to this CF. This means that data sharing can continue. Similar rebuild and recovery occurs for OSAM, VSAM, and shared queue structures.

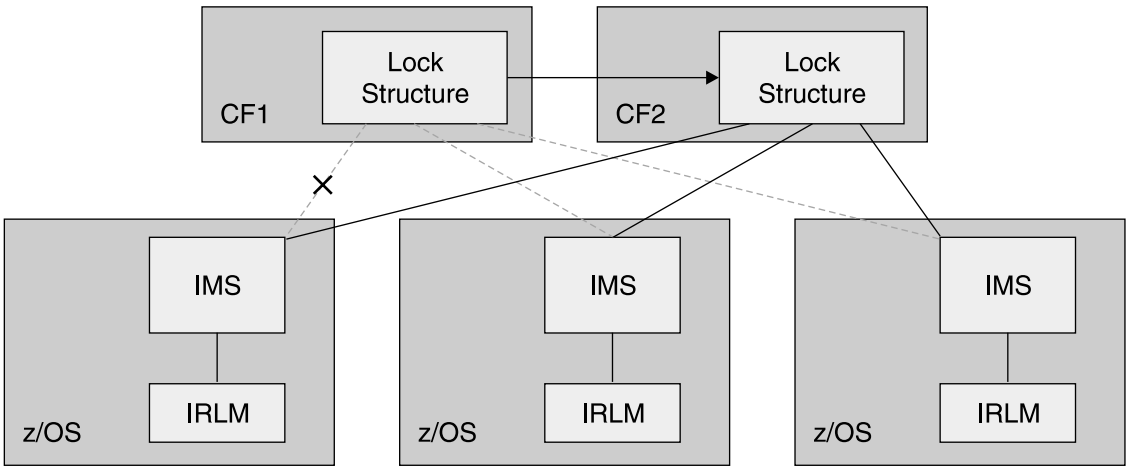

Figure 114. IRLM Structure Rebuilt on Another Coupling Facility After a Connectivity Failure

Recovery Using Structure Duplexing: Fast Path shared VSO does not rebuild its cache structures. Instead, it relies on a duplicate copy to provide failure survival. The duplicate copy can be created in either of two ways.

- Fast Path can build two structures. This is user-managed duplexing.
- By using appropriate hardware prerequisites, you can have the system build duplexed structures. This is system-managed duplexing. System-managed duplexing is also available for IRLM lock structures and shared-queues structures.

Figure 115 on page 336 shows a duplexed DEDB VSO structure on two coupling facilities that are being shared by three IMSs.

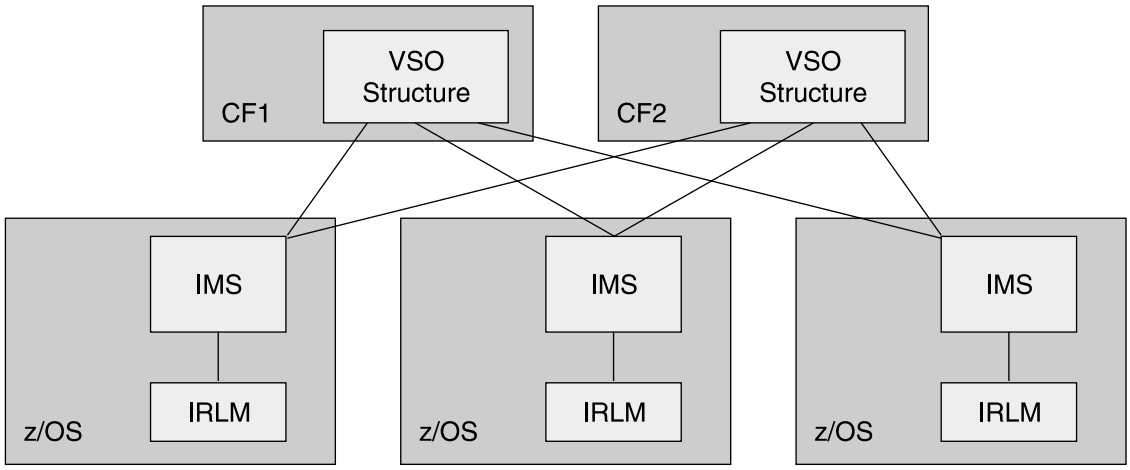

Figure 115. Shared VSO Structure Duplexed on Two Coupling Facilities

If a CF is lost (as shown in Figure 116), then a duplicate structure on another CF is used. With system-managed duplexing, a duplicate is immediately built if another CF is available. If another CF is not available, a duplicate structure is built when another CF becomes available.

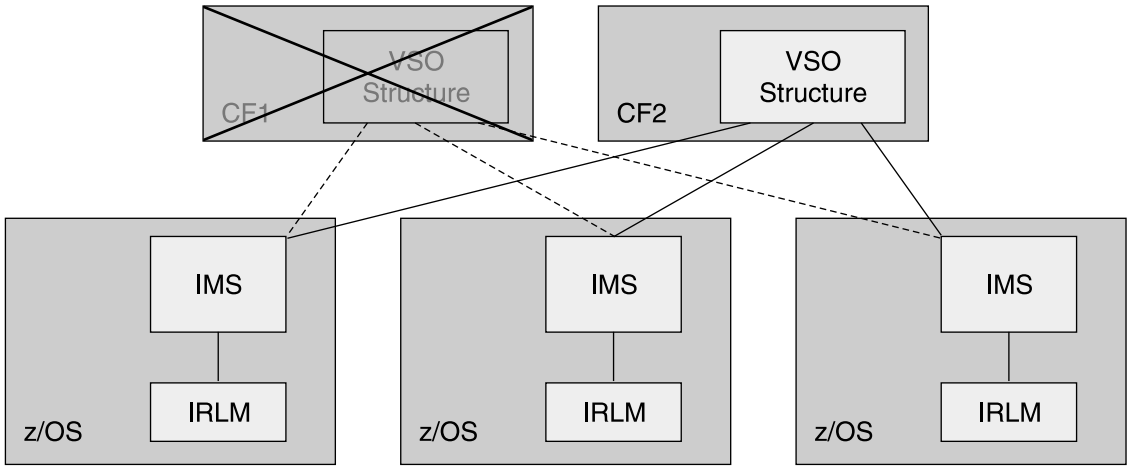

Figure 116. System-Managed Duplicate Shared VSO Structure is Used After a Coupling Facility Failure

Similarly, if connectivity to a CF is lost, then the use of its structure is discontinued. The duplicate structure on another CF is used instead.

Chapter 31. IMSplexes

This chapter introduces and discusses an IMSplex, which is defined in the following paragraph.

Definition: An IMSplex is one or more IMS address spaces (control, manager, or server) that work together as a unit. Typically (but not always), these address spaces:

- Share either databases or resources or message queues (or any combination)
- Run in a z/OS Parallel Sysplex environment
- Include an IMS Common Service Layer (CSL) - see “Common Service Layer” on page 338

The address spaces that can participate in the IMSplex are:

- Control region address spaces
- IMS manager address spaces (OM, RM, SCI)
- IMS server address spaces (Common Queue Server - CQS)

An IMSplex provides you with the ability to manage multiple IMS systems as if they were one system (a single-system perspective). An IMSplex can exist in a non-sysplex environment or can consist of multiple IMS subsystems (in sharing groups) in a sysplex environment.

Related Reading:

- For complete details about setting up a sysplex, see *z/OS MVS Setting Up a Sysplex*.
- For the details of IMSplex components, see the *IMS Version 9: Common Service Layer Guide and Reference*.

This chapter discusses an IMSplex with a CSL included.

The following sections are covered in this chapter:

- “Components of an IMSplex”
- “Requirements for an IMSplex” on page 339
- “Operating an IMSplex” on page 340

Components of an IMSplex

The following sections discuss the components of an IMSplex:

- “Common Queue Server”
- “Common Service Layer” on page 338

Common Queue Server

Common Queue Server (CQS) is a generalized server (delivered with IMS) that manages data objects on a coupling facility list structure, such as a queue structure or a resource structure, on behalf of multiple clients. CQS receives, maintains, and distributes data objects from shared queues on behalf of multiple clients. Each client has its own CQS access the data objects on the coupling facility list structure. IMS is one example of a CQS client that uses CQS to manage both its shared queues and shared resources.

Related Reading: For complete information about CQS, see “Common Queue Server (CQS) Address Space” on page 14.

Common Service Layer

The CSL is an IMS feature that provides the infrastructure for improving IMS systems management. The CSL is made up of three IMS address spaces. They are:

- The Operations Manager (OM)
- The Resource Manager (RM)
- The Structured Call Interface (SCI)

The CSL components and IMS subsystems in an IMSplex can be called IMSplex components.

Related Reading: For complete information about CSLs, see the *IMS Version 9: Common Service Layer Guide and Reference*.

Figure 117 illustrates a sample IMSplex configuration that includes the CSL, a single point of control (SPOC), and automated procedures.

- The OS image includes address spaces for OM, SCI, RM, an IMS control region, and IMS CQS.
- The OS image shares a coupling facility and databases.
- A SPOC application, an automation application, a master terminal, and an end-user terminal all access the z/OS image.

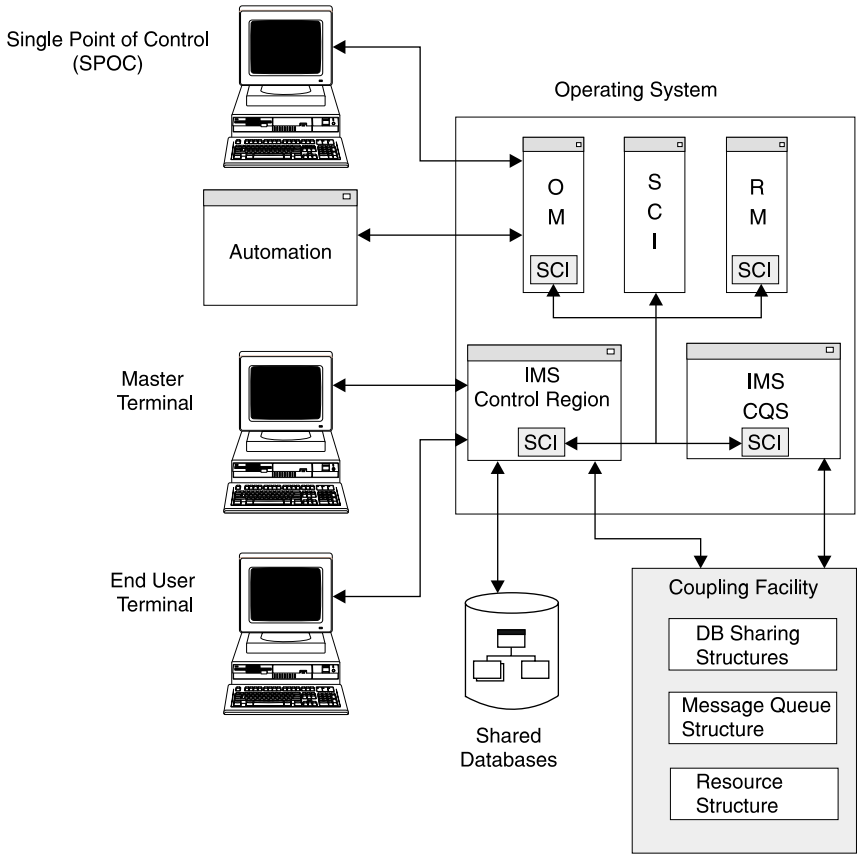

Figure 117. Sample IMSplex Configuration with a CSL

Operations Manager

The Operations Manager (OM) controls the operations of an IMSplex. OM provides an application programming interface (the OM API) through which commands can be issued and responses received. With a single point of control (SPOC) interface, you can submit commands to OM. The SPOC interfaces include the TSO SPOC, the REXX SPOC API, and the IMS Control Center. You can also write your own application to submit commands.

Specifically, OM can:

- Route IMS commands to IMSplex members registered for the command.
- Consolidate command responses from individual IMSplex members into a single response and provides that response to the originator of the command.
- Provide an API for automated operator commands.
- Provide a general use interface to register commands to support any command-processing client.
- Provide user exits for command and response edit and command security.

One OM must be defined in the IMSplex to use OM functions. Each z/OS image can have more than one OM. If multiple OMs are defined in the IMSplex, any OM defined can perform work from any z/OS image in the IMSplex.

Resource Manager

The Resource Manager (RM) is the component of the CSL that manages global resources and coordinates IMSplex-wide processes on behalf of its clients. IMS is an example of one such client. IMS uses RM to:

- Manage the following resources: transactions, LTERMs, MSNAMEs, nodes, users, user IDs, and APPC descriptor names
- Ensure that a resource that is defined as a transaction, LTERM, or msname is defined as the same resource type for all the IMSs in the IMSplex
- Coordinate IMSplex-wide processes, such as performing global online changes

RM uses the Common Queue Server (CQS) to maintain global resource information in a resource structure. A resource structure is actually a coupling facility list structure that all CQSs in the IMSplex can access.

Structured Call Interface

The Structured Call Interface (SCI) is the component of the CSL that provides the communication between IMSplex components, whether they are on one z/OS image or multiple z/OS images.

Requirements for an IMSplex

The minimum configuration for an IMSplex is an IMS control region and one set of CSL components (an OM, an RM, and an SCI) in one z/OS image and another (one or more) z/OS image (forming the sysplex) with a single SCI and a single IMS. Figure 118 on page 340 illustrates a similar configuration: one z/OS image with an IMS and one each of the CSL components and two other z/OS images each with a single IMS and a single SCI.

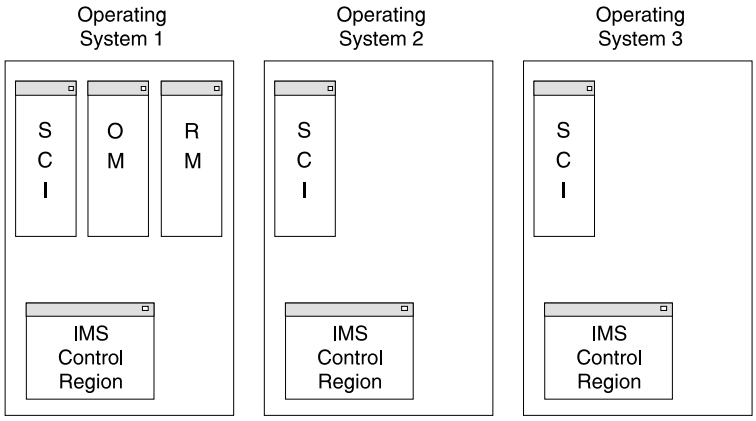

Figure 118. Minimum CSL Configuration for an IMSplex

All IMSs in an IMSplex must have a unique IMSID.

A single point of control (SPOC) application is required if you want to use certain CSL functions (for example, performing a global online change).

Restriction: The maximum configuration for an IMSplex is 32 control regions, 32 CQSs, and a maximum of 1023 total system address spaces.

Operating an IMSplex

Operating an IMSplex differs from operating a single IMS. The individual IMS subsystems in an IMSplex must be defined and set up individually; however, once this up-front work is done, they can be operated as a group (the IMSplex).

In general, you operate an IMSplex by issuing commands through the Operations Manager and analyzing the responses to those commands. You cannot issue commands directly to OM, however. OM has an application programming interface (API) that is designed to accept commands from and pass command responses to an application program. The IMS TSO SPOC application and the IMS Control Center are examples of such application programs.

The IMS Control Center (which supports IMS Version 8 and above and is part of the DB2 Universal Database (UDB) Control Center), is another application program that accesses the OM API. With the IMS Control Center, you can manage your IMSplex using a graphical interface. The IMS Control Center uses the SPOC functions of the OM API. For more information about the IMS Control Center, go to www.ibm.com/ims and link to the IMS Control Center page.

You can design an application program that allows an operator to enter the commands and view the responses or you can design an automated operator program that will issue commands and make decisions based on the responses. The commands that are issued are the same in both instances. The differences are:

- The automated application program must anticipate what the responses to the commands will be and have logic to deal with those responses.
- The operator (issuing commands through an application) must have procedures to follow.

Recommendation: When designing the automated operator programs or creating operator procedures, keep in mind that there might be IMSplex-wide ramifications

for some actions. For example, if you tell an operator (or code your AOP) to shut down a particular SCI for a particular reason, the IMSs that used that particular SCI can no longer communicate with the IMSplex after that SCI is shut down. This might not be the solution you were hoping for, so plan accordingly.

The following list briefly cover the operations of the IMSplex:

Starting or restarting an IMSplex

Start an IMSplex in the following manner:

1. Start the CSL components that will be local to the IMS control region with a z/OS `START` command.
2. Start the local IMS control region with the appropriate parameters specified on the appropriate PROCLIB members.
3. Start the other individual IMSplex components (other IMSs, SCIs, OMs, and RMs).

An IMSplex or its components can be restarted (after a failure or shutdown) by either manually starting the individual components (with a z/OS `START` command) or by using the z/OS Automatic Restart Manager (ARM). If ARM is to be used for restart purposes, it is specified on the IMSplex components' startup procedures or in their individual initialization PROCLIB members.

Querying statistics from an IMSplex

Any IMSplex member (for example, an Automated Operations Program - AOP) can query statistics about the components of a CSL using a CSLZQRY request.

Shutting down an IMSplex

Shutting down an IMSplex is accomplished in two basic steps:

1. Shutting down the IMS components that participate in that IMSplex (issuing a `/CHE FREEZE` or similar command to the individual IMSs)
2. Shutting down the CSL and its components.

Part 7. Appendixes

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
CICS
DataPropagator
DB2
DB2 Universal Database
DFSMS/dss
DFSMS
IBM
IMS
IMS/ESA
MVS
OS/390
Parallel Sysplex
RAA
RACF
SAA
System/360
VTAM
WebSphere
WebSphere MQ
z/OS

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.

Other company, product, and service names may be the trademarks or service marks of others.

Product Names

In this book, the licensed program “DB2 Universal Database for z/OS” is referred to as “DB2”.

Bibliography

This bibliography includes all the publications cited in this book, including the publications in the IMS library.

- *DB2 for z/OS Installation Guide*, GC26-9936
- *DFSMS/MVS V1R5 DFSMSdss Storage Administration Guide*, SC26-4930-04
- *DFSMS/MVS V1R5 DFSMSdss Storage Administration Reference*, SC26-4929-04.
- *IBM IMS Workload Router for z/OS User's Guide*, SC26-8945
- *IBM Systems Journal: IMS celebrates thirty years as an IBM product*, ISBN G321-5693
- *IMS Fast Path Solutions Guide*, SG24-4301
- *IMS in the Parallel Sysplex: Volume I: Reviewing the IMSplex Technology*, SG24-6908
- *IMS in the Parallel Sysplex: Volume II: Planning the IMSplex*, SG24-6928
- *IMS in the Parallel Sysplex: Volume III: IMSplex Implementation and Operations*, SG24-6929
- *IMS Connect Guide and Reference*, SC18-7260
- *IMS Fast Path Solutions Guide*, SG24-4301
- *IMS Performance Guide*, SG24-4637
- *IMS Primer*, SG24-5352
- *IMS Queue Control Facility for z/OS*, SC26-9685
- *IMS Security Guide*, SG24-5363
- *IMS Version 7 Release Planning Guide*, GC26-9437
- *MS Version 8 Release Planning Guide*, GC27-1305
- *The Complete IMS HALDB Guide All You Need to Know to Manage HALDBs*, SG24-6945
- *z/OS MVS Setting Up a Sysplex*, SA22-7625
- *z/OS MVS System Commands*, SA22-7627
- *z/OS V1R4 Security Server RACF Security Administrator's Guide*, SA22-7683

IMS Version 9 Library

ZES1-2330	ADB	*IMS Version 9: Administration Guide: Database Manager*
ZES1-2331	AS	*IMS Version 9: Administration Guide: System*
ZES1-2332	ATM	*IMS Version 9: Administration Guide: Transaction Manager*
ZES1-2333	APDB	*IMS Version 9: Application Programming: Database Manager*
ZES1-2334	APDG	*IMS Version 9: Application Programming: Design Guide*
ZES1-2335	APCICS	*IMS Version 9: Application Programming: EXEC DLI Commands for CICS and IMS*
ZES1-2336	APTM	*IMS Version 9: Application Programming: Transaction Manager*
ZES1-2337	BPE	*IMS Version 9: Base Primitive Environment Guide and Reference*
ZES1-2338	CR	*IMS Version 9: Command Reference*
ZES1-2339	CQS	*IMS Version 9: Common Queue Server Guide and Reference*
ZES1-2340	CSL	*IMS Version 9: Common Service Layer Guide and Reference*
ZES1-2341	CG	*IMS Version 9: Customization Guide*
ZES1-2342	DBRC	*IMS Version 9: DBRC Guide and Reference*
ZES1-2343	DGR	*IMS Version 9: Diagnosis Guide and Reference*
ZES1-2344	FAST	*IMS Version 9: Failure Analysis Structure Tables (FAST) for Dump Analysis*
ZES1-2346	OLR	*IMS Version 9: HALDB Online Reorganization Guide and Reference*
ZES1-2347	JGR	*IMS Version 9: IMS Java Guide and Reference*
ZES1-2348	IIV	*IMS Version 9: Installation Volume 1: Installation Verification*
ZES1-2349	ISDT	*IMS Version 9: Installation Volume 2: System Definition and Tailoring*
ZES1-2350	INTRO	*IMS Version 9: An Introduction to IMS*
ZES1-2351	MIG	*IMS Version 9: Master Index and Glossary*
ZES1-2352	MC1	*IMS Version 9: Messages and Codes, Volume 1*
ZES1-2353	MC2	*IMS Version 9: Messages and Codes, Volume 2*

ZES1-2354	OTMA	*IMS Version 9: Open Transaction Manager Access Guide and Reference*
ZES1-2355	OG	*IMS Version 9: Operations Guide*
GC17-7831	RPG	*IMS Version 9: Release Planning Guide*
ZES1-2358	URDBTM	*IMS Version 9: Utilities Reference: Database and Transaction Manager*
ZES1-2359	URS	*IMS Version 9: Utilities Reference: System*

Supplementary Publications

GC17-7825	LPS	*IMS Version 9: Licensed Program Specifications*
ZES1-2357	SOC	*IMS Version 9: Summary of Operator Commands*

Publication Collections

LK3T-7213	CD	IMS Version 9 Softcopy Library
LK3T-7144	CD	IMS Favorites
LBOF-7789	Hardcopy and CD	Licensed Bill of Forms (LBOF): IMS Version 9 Hardcopy and Softcopy Library
SBOF-7790	Hardcopy	Unlicensed Bill of Forms (SBOF): IMS Version 9 Unlicensed Hardcopy Library
SK2T-6700	CD	OS/390 Collection
SK3T-4270	CD	z/OS Software Products Collection
SK3T-4271	DVD	z/OS and Software Products DVD Collection

Accessibility Titles Cited in this Book

SA22-7787		z/OS V1R1.0 TSO Primer
SA22-7794		z/OS V1R1.0 TSO/E User's Guide
SC34-4822		z/OS V1R1.0 ISPF User's Guide, Volume 1

Index

Special characters

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Program Number: 5655-J38

ZES1-2350-01

Spine information:

IMS An Introduction to IMS Version 9