IMS

An Introduction to IMS

Version 9

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”
- “Overview of the IMS Product” on page 4

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.

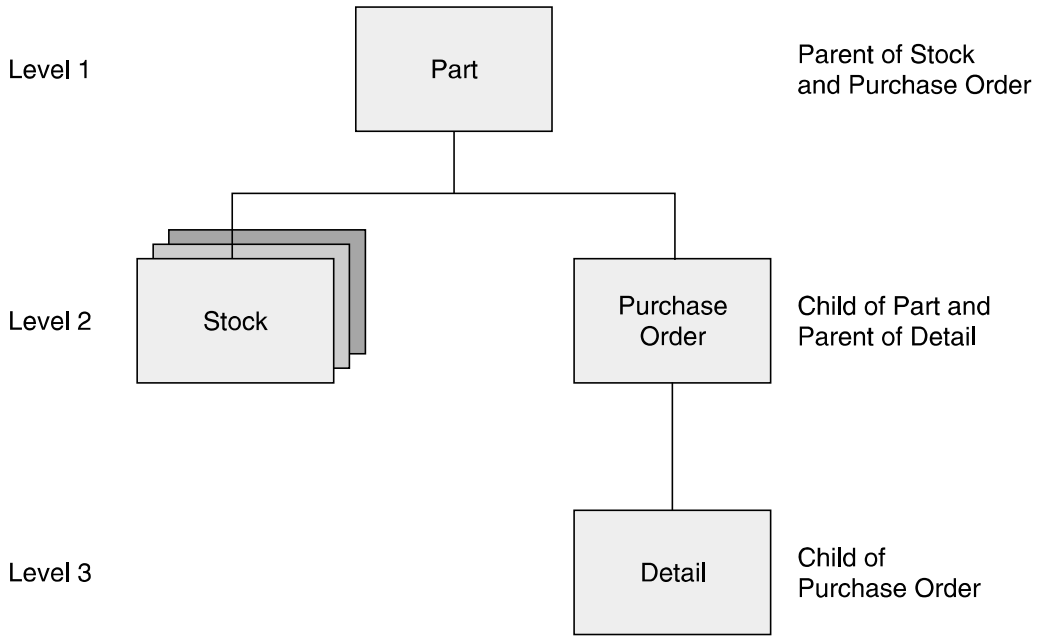

Figure 1. Example of a Hierarchical Data Model

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.

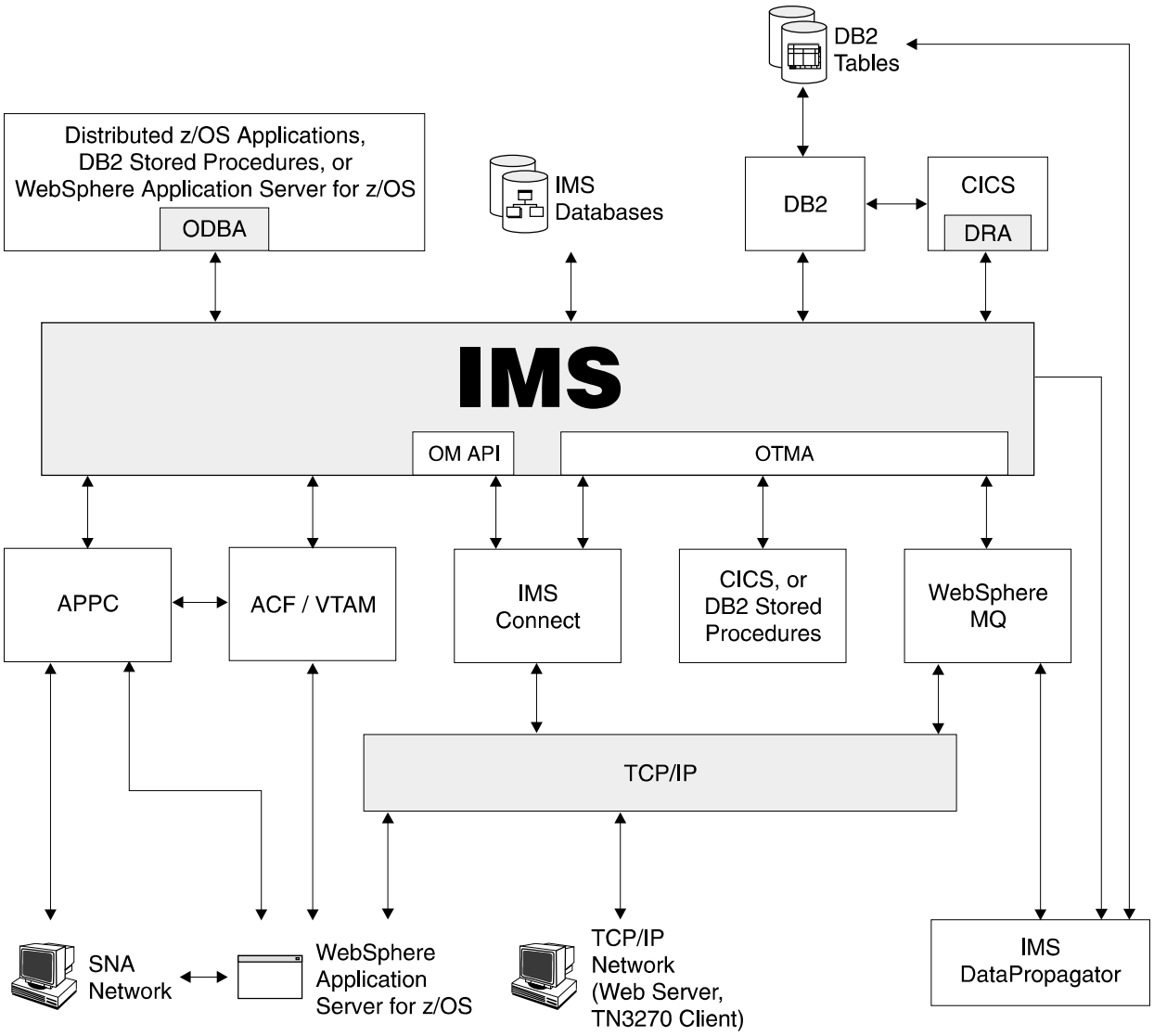

Figure 2. Interfaces to IMS

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.

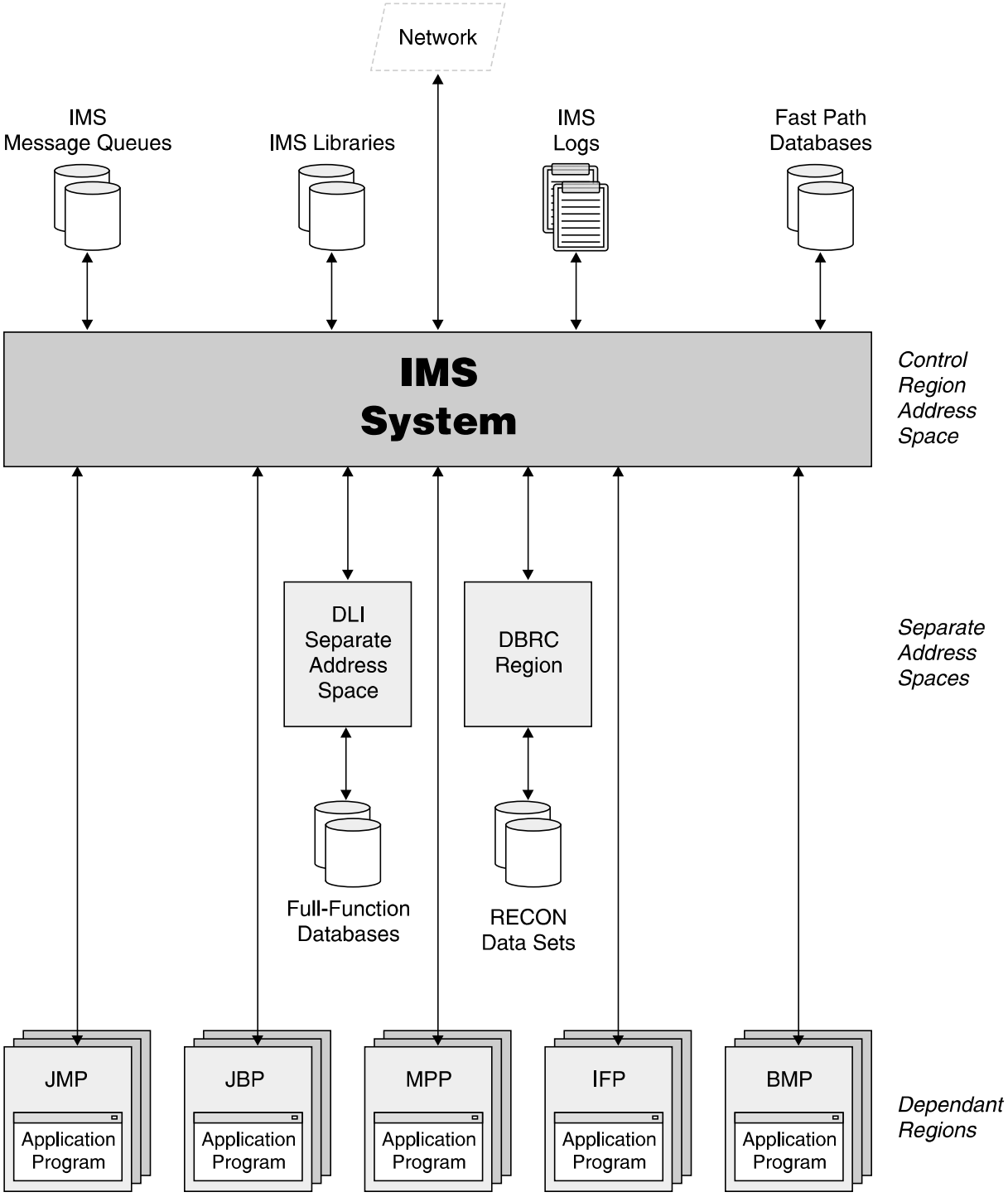

Figure 3. Structure of an IMS DB/DC Subsystem

- 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.

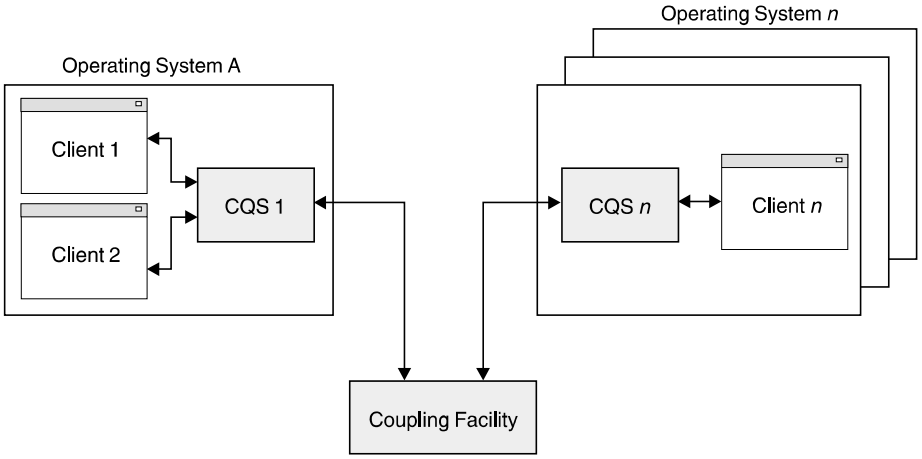

Figure 4. Client Systems, CQS, and a Coupling Facility

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.

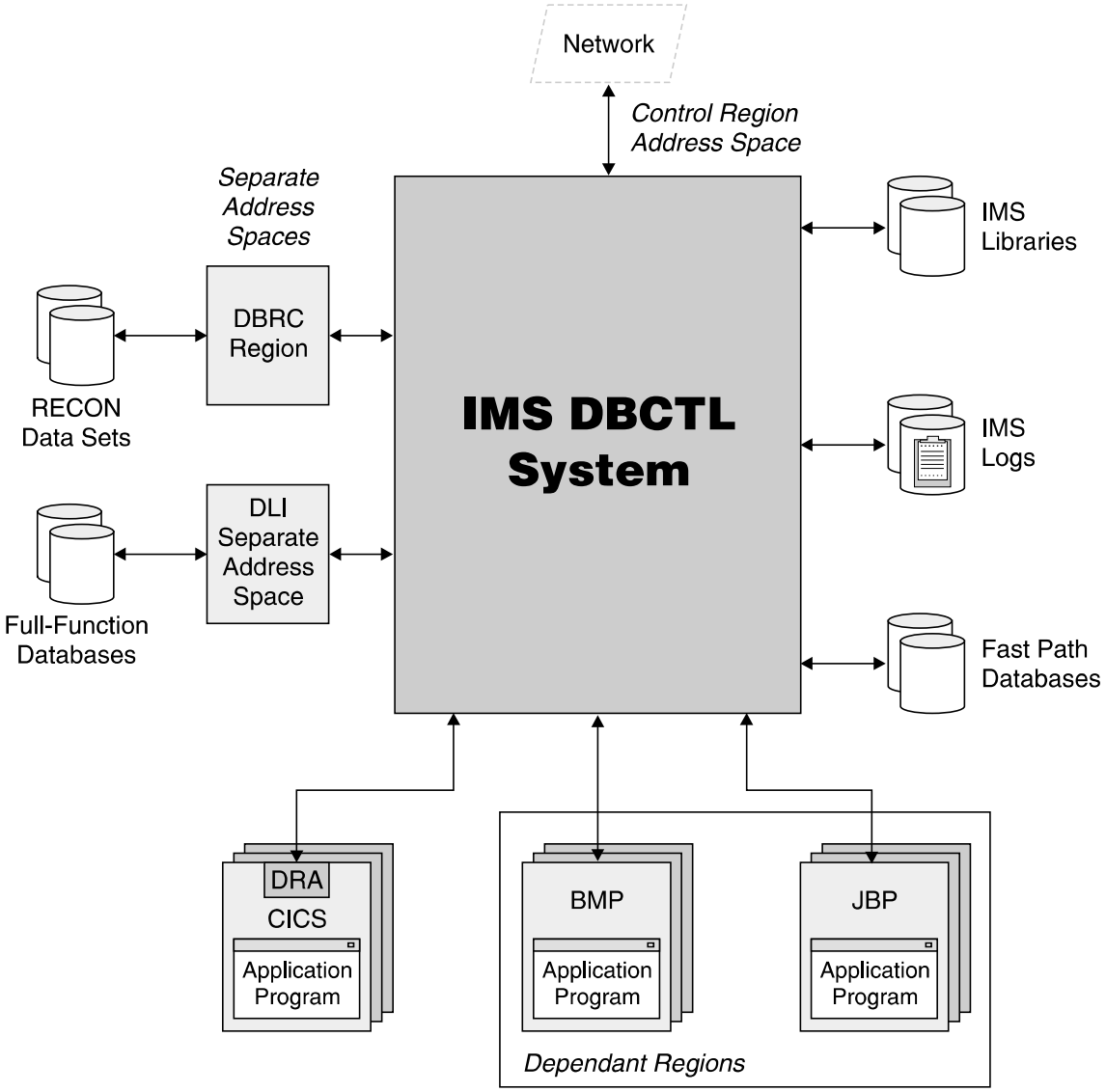

Figure 5. Structure of an IMS DBCTL System

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.

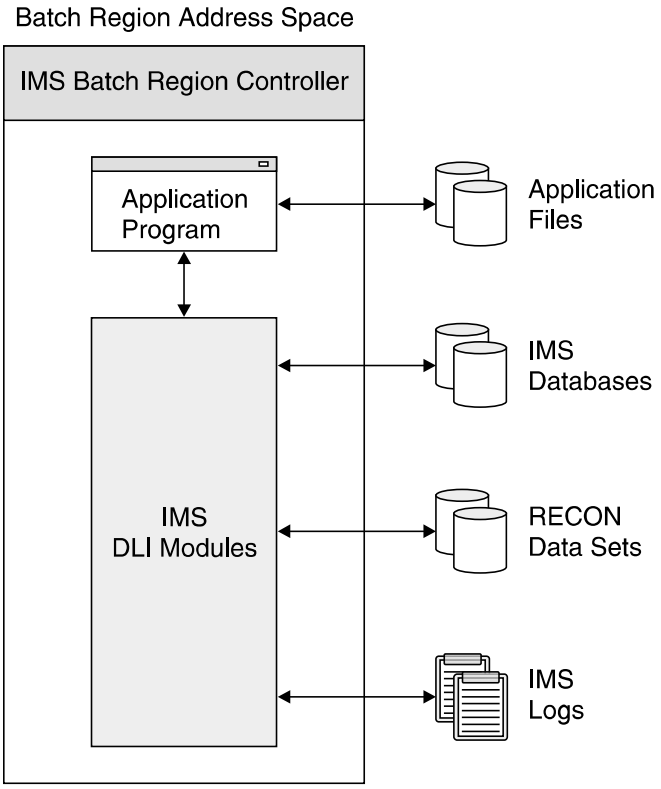

Figure 6. Structure of an IMS Batch Region

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 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.

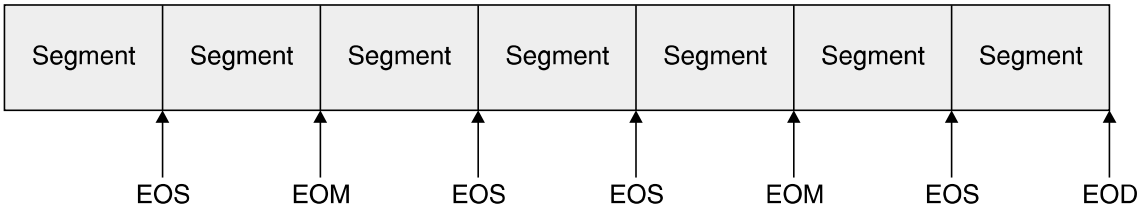

Figure 35. Transmission, Message, and Segment Relationships

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.

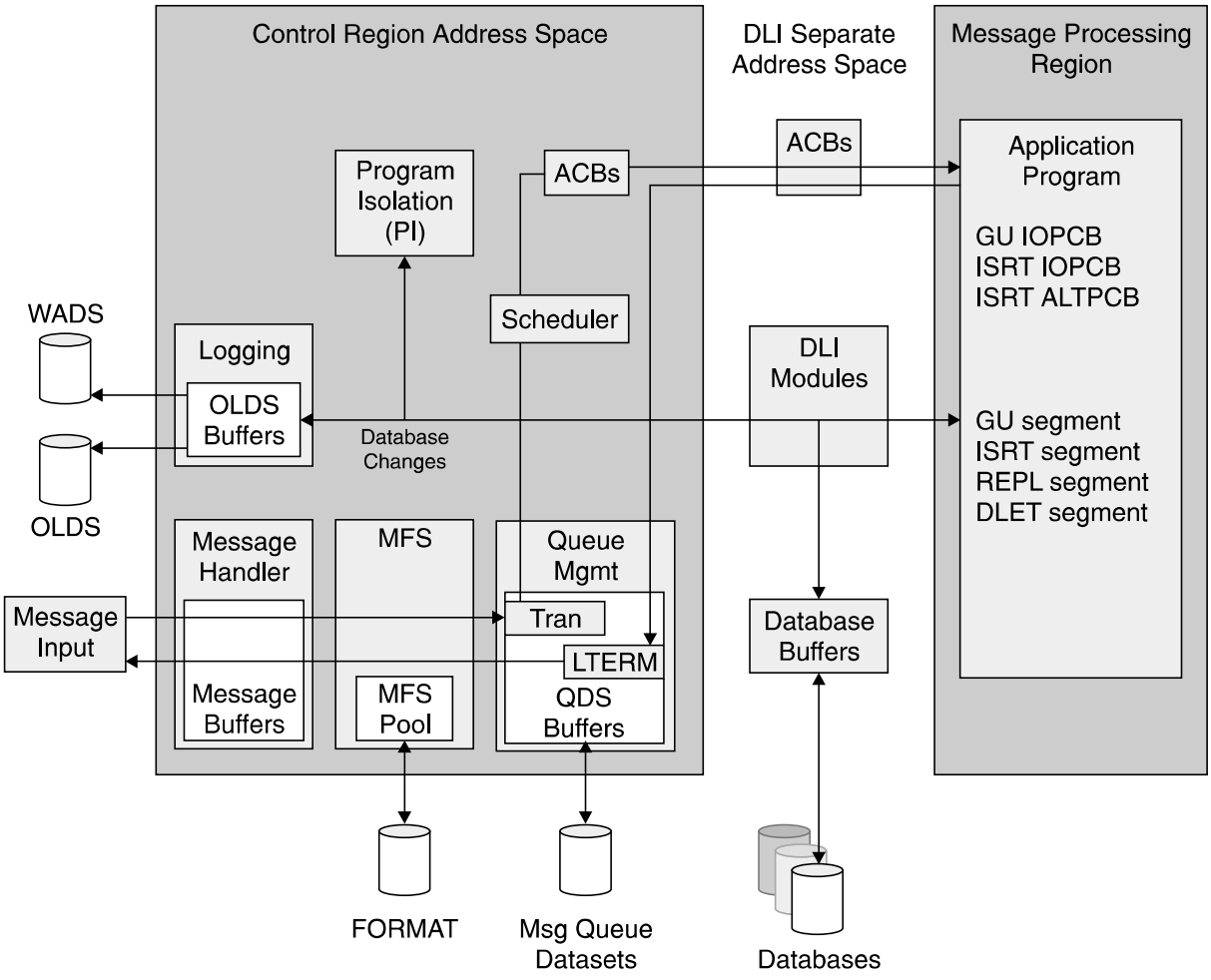

Figure 37. The IMS Control Region, Its Control, and Data (Message) Flow

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.

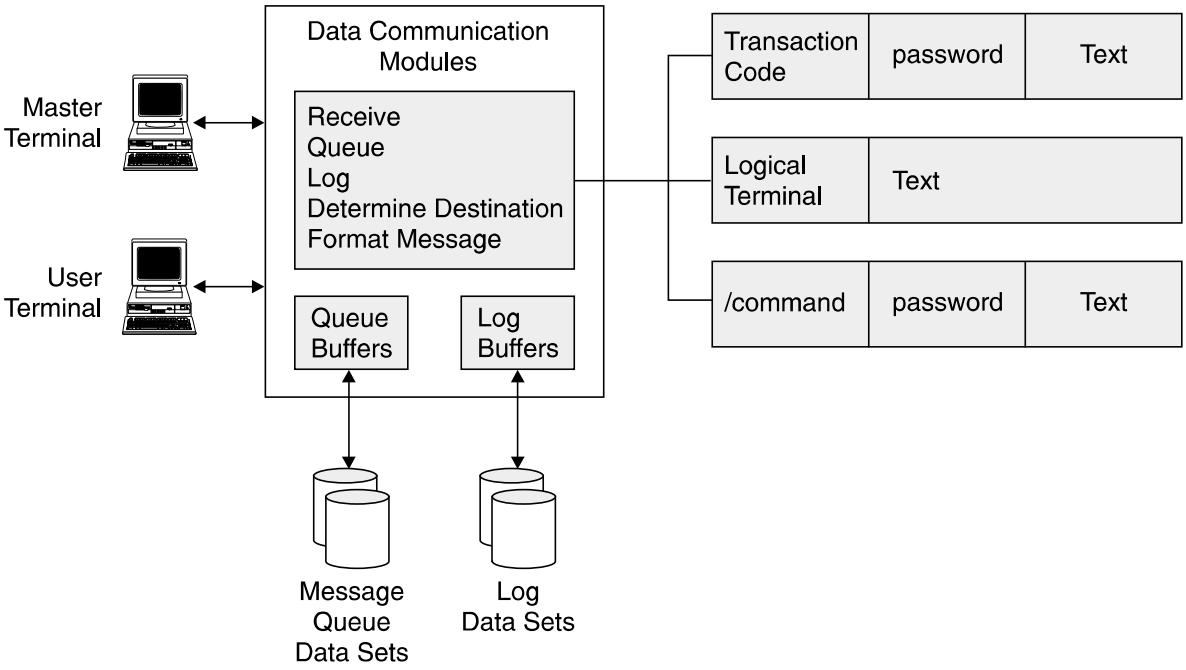

Figure 38. Input Message Processing

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.

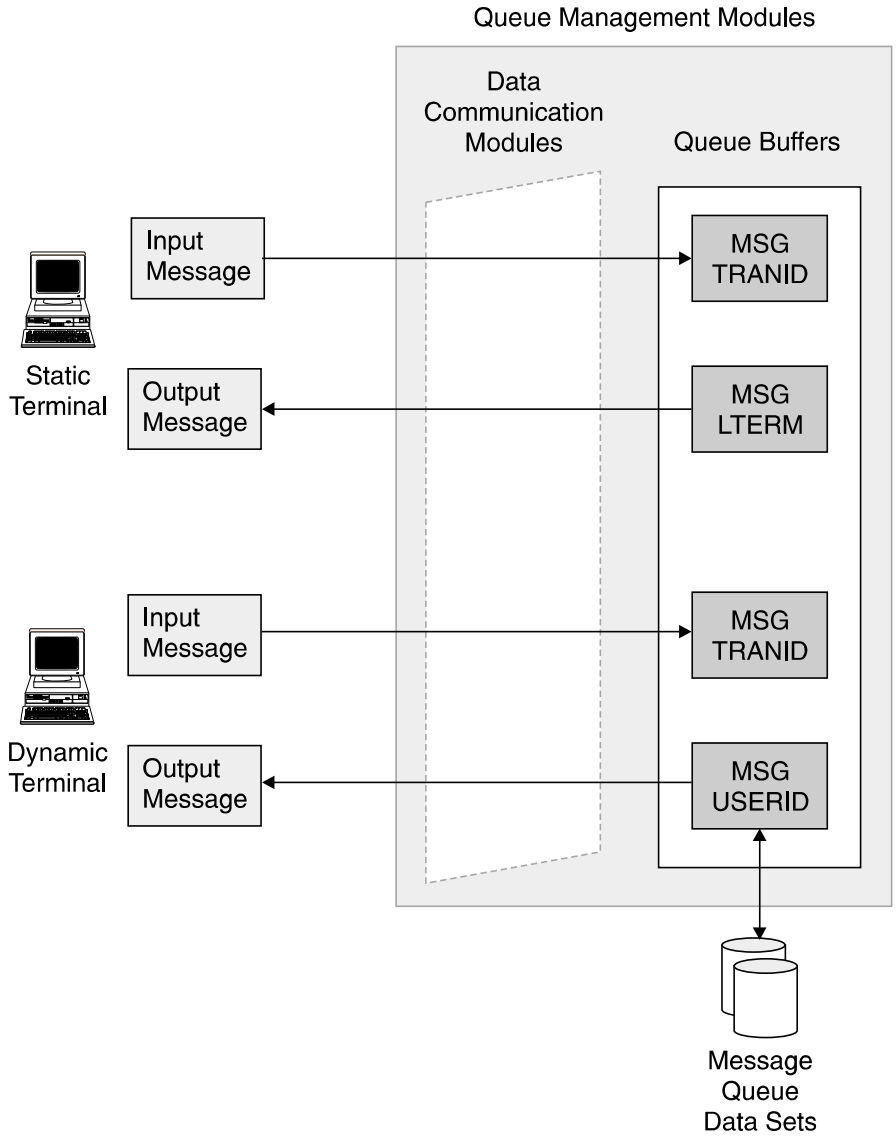

Figure 39. Overview of the Message Queuing Process

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.

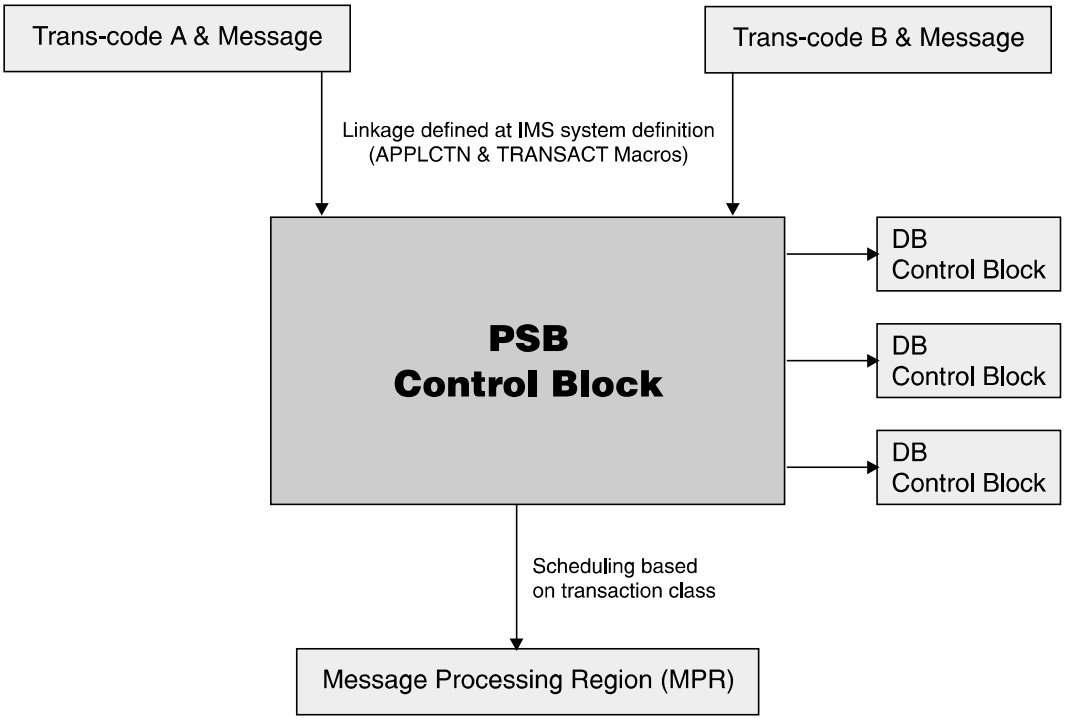

Figure 40. Message Scheduling

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"
- "The Secondary Master" on page 139
- "Using the z/OS Console as the Master Terminal" on page 139
- "Extended MCS/EMCS Console Support" on page 139

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.

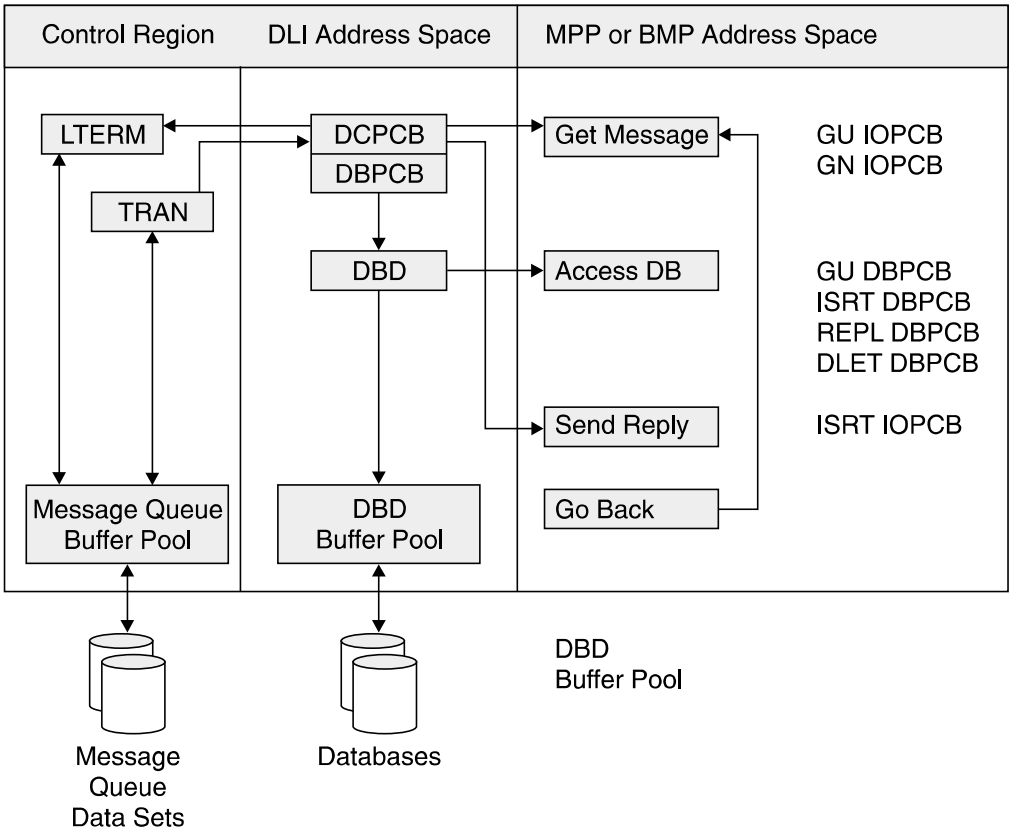

Figure 47. Overview of Basic Flow Through a MPP or BMP Address Space

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.

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.

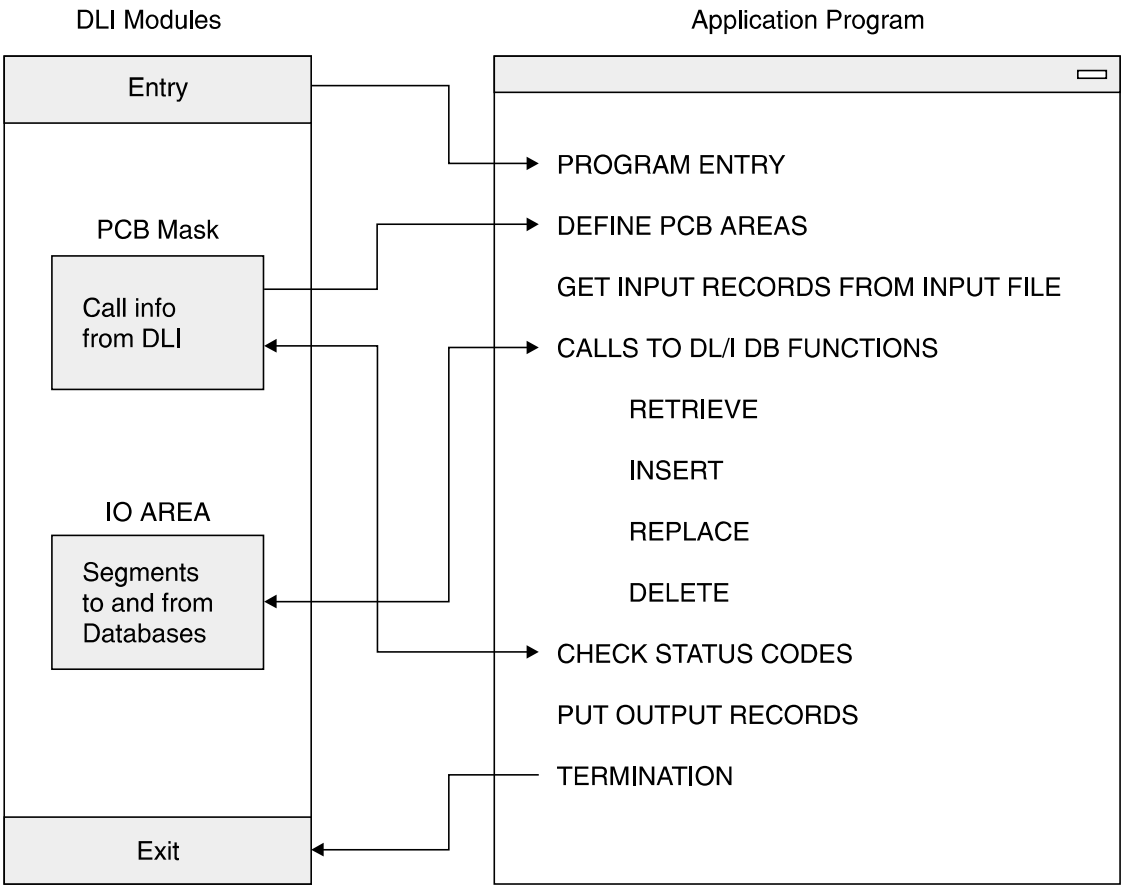

Figure 48. Structure of an IMS Application Program

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.

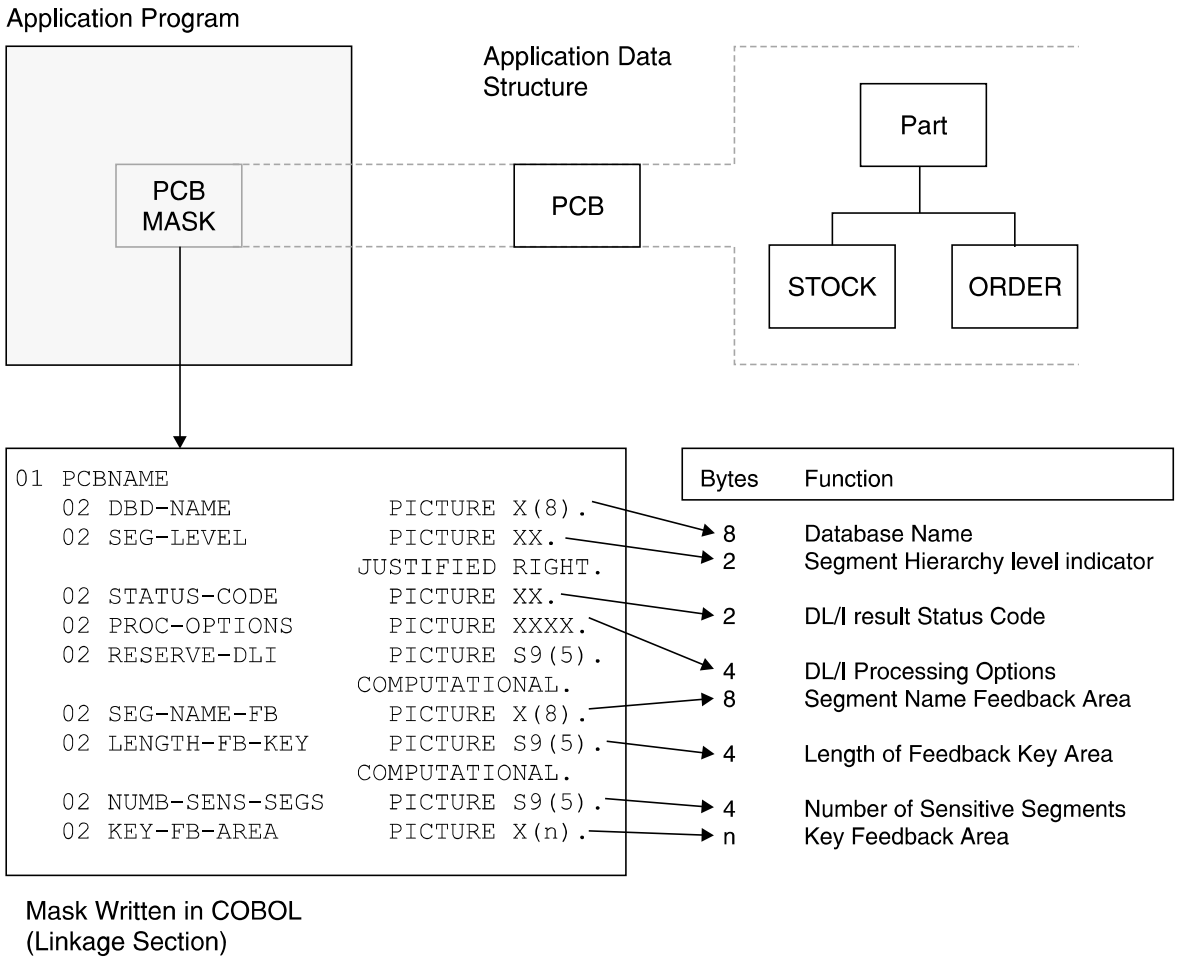

Figure 49. Application PCB Structure

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.

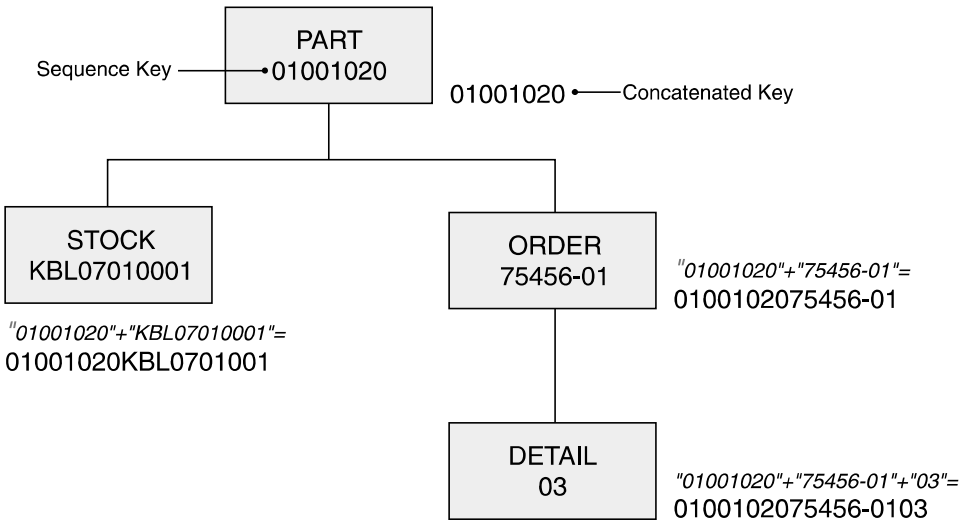

Figure 51. Examples of Concatenated Keys

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.

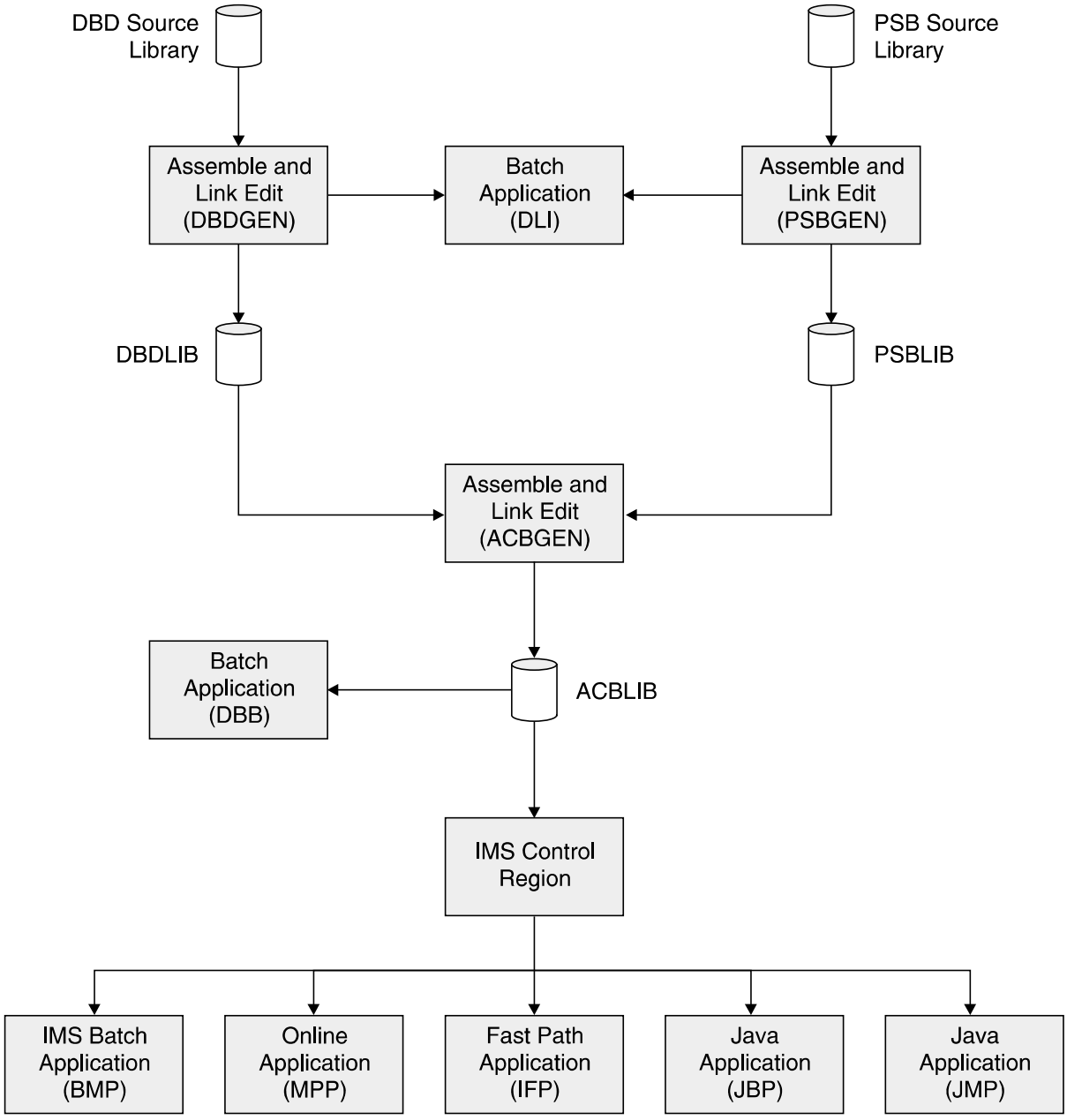

Figure 54. IMS Control Block Generation and Usage

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 CIPC BASED (DC_PTR),                      /* NOT USED IN       */ 0000018
     02 DUMMY CHAR (32),                       /* BATCH DL/I        */ 0000019
  01 DIPC BASED (DB_PTR),                      /* PHASE 2 ORDER DB  */ 0000021
     02 DIPCDBDN CHAR (8),                     /* DBD NAME          */ 0000022
     02 DIPCLEVL CHAR (2),                     /* SEGMENT LEVEL     */ 0000023
     02 DIPCSTAT CHAR (2),                     /* STATUS CODE       */ 0000024
     02 DIPCPROC CHAR (4),                     /* PROCESSING OPTN   */ 0000025
     02 OIPCRESV FIXED BINARY(31),             /* RESERVED          */ 0000026
     02 DIPCSEGN CHAR (8),                     /* SEGMENT NAME      */ 0000027
     02 DIPCKFBL FIXED BINARY(31),             /* KEY FEEOBACK LNG  */ 0000028
     02 DIPCNSSG FIXED BINARY(31),             /* N0. OF SENSEGS    */ 0000029
     02 DIPCKFBA 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 DIPCSTAT -- '  ' THEN                     /* CALL EROOR PRINT  */ 0000057
    CALL DFSOAER (DIFC,DERRID,IO_AREA,ERROPTI);                        0000058
    CALL DFSOAER (DIPC,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.

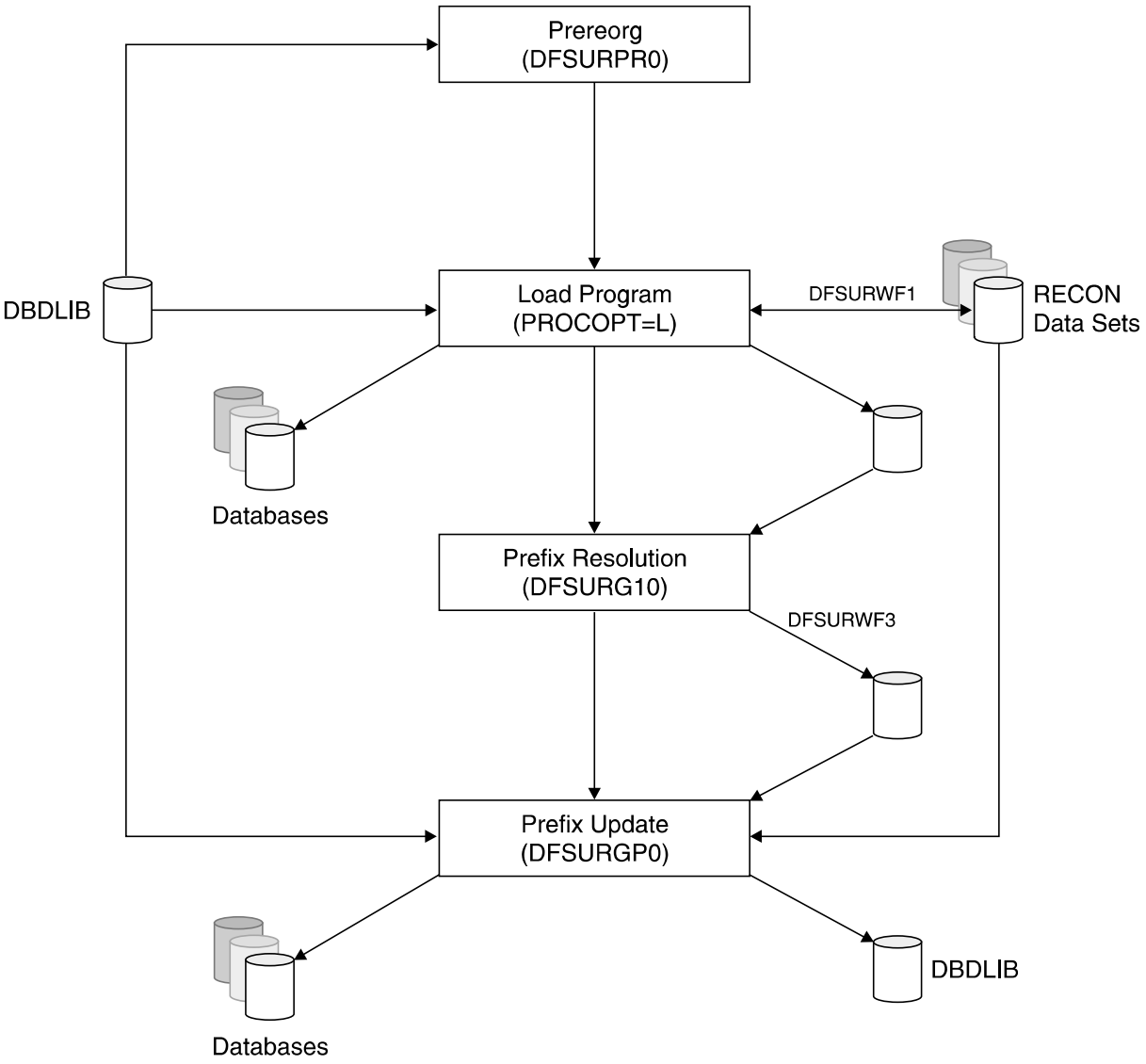

Figure 70. Overview of Loading a Database that has Logical Relationships

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).

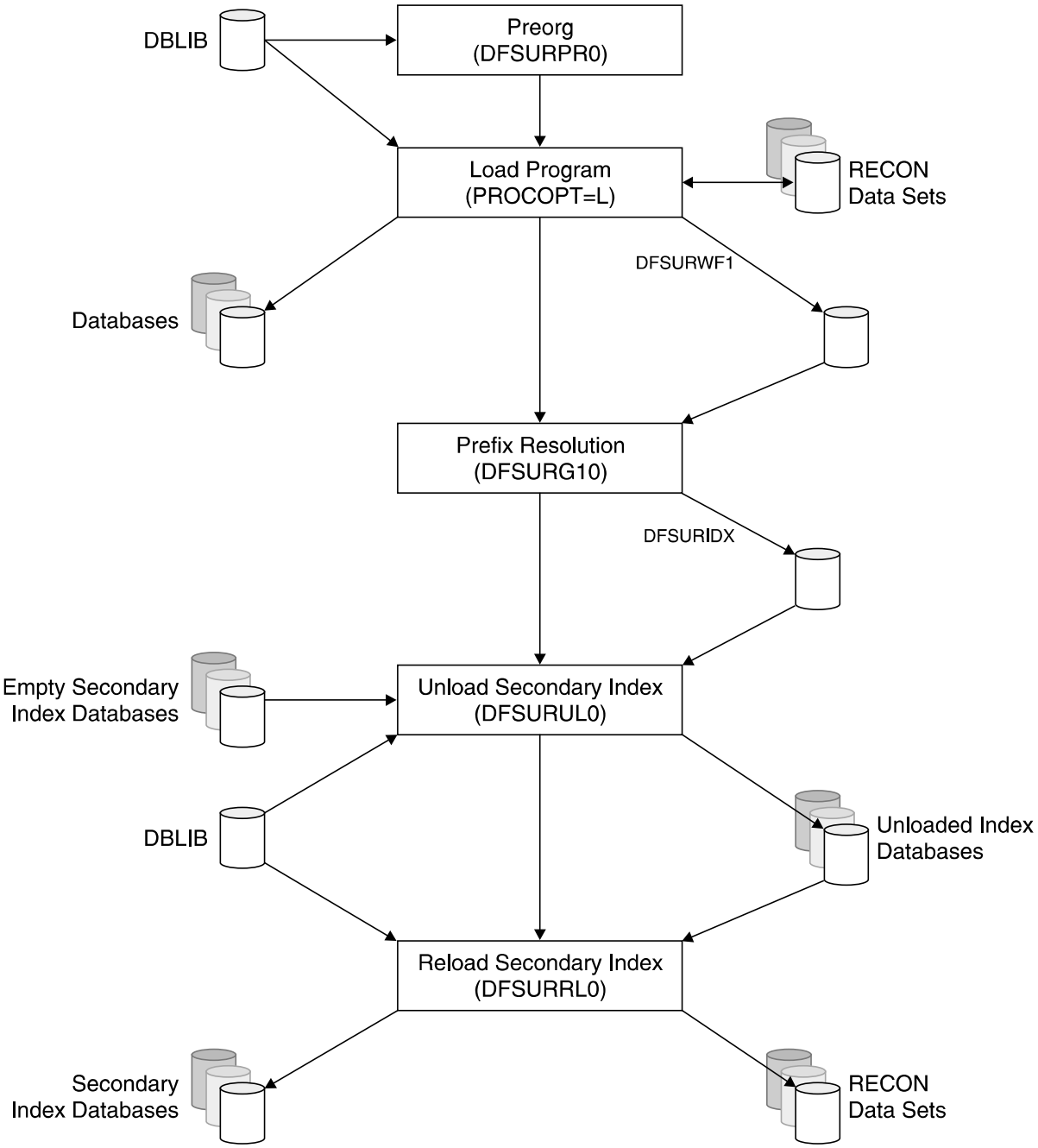

Figure 71. Overview of Loading a Database that has Secondary Indexes

Figure 72 on page 191 illustrates the process of loading a database that has logical relationships and secondary indexes.

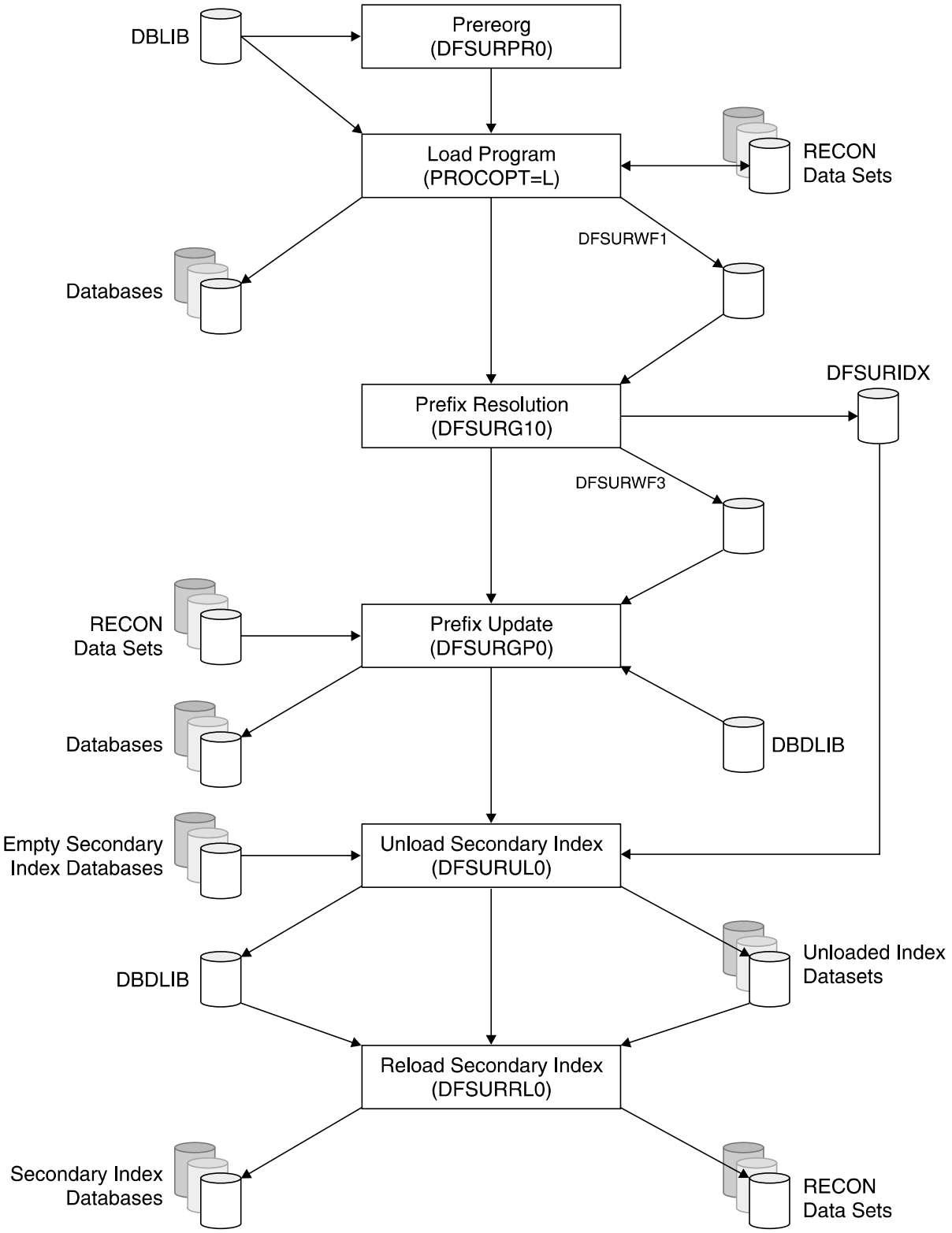

Figure 72. Overview of Loading a Database that has Logical Relationships and Secondary Indexes

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.

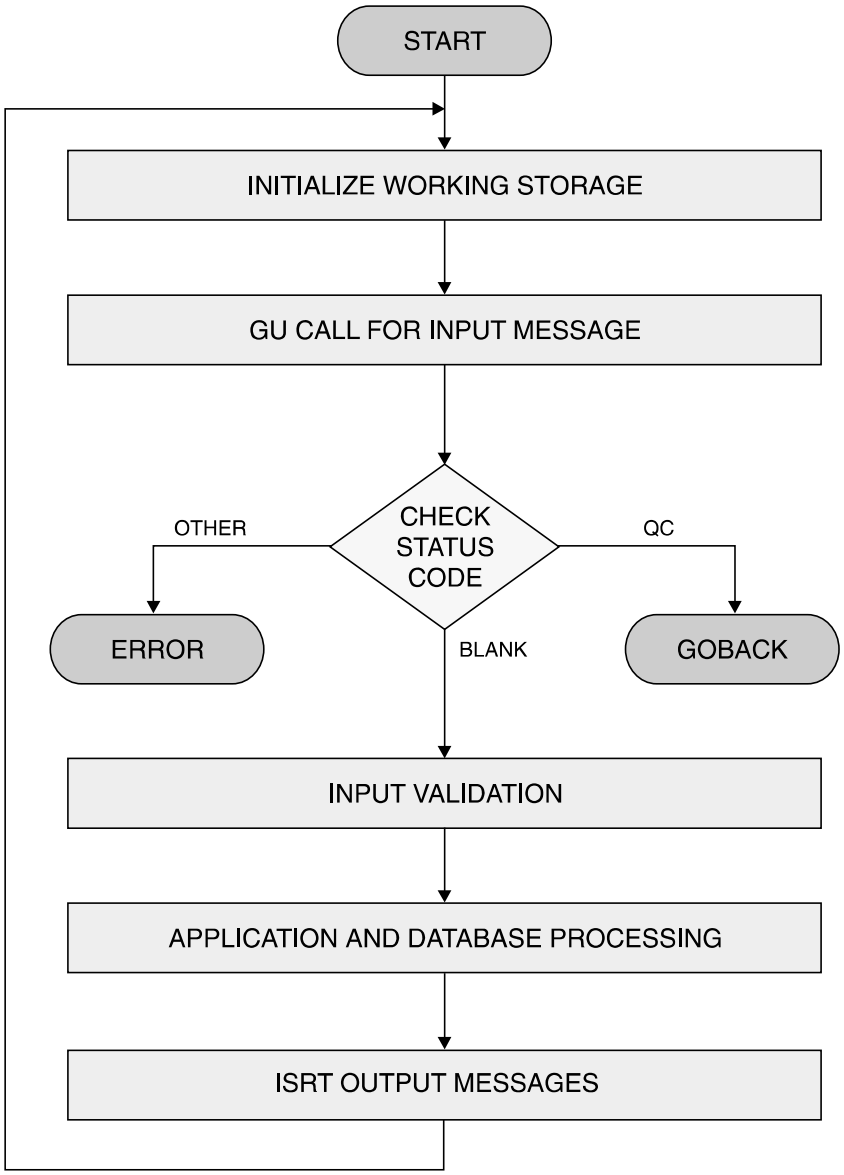

Figure 73. General MPP Structure and Flow

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.

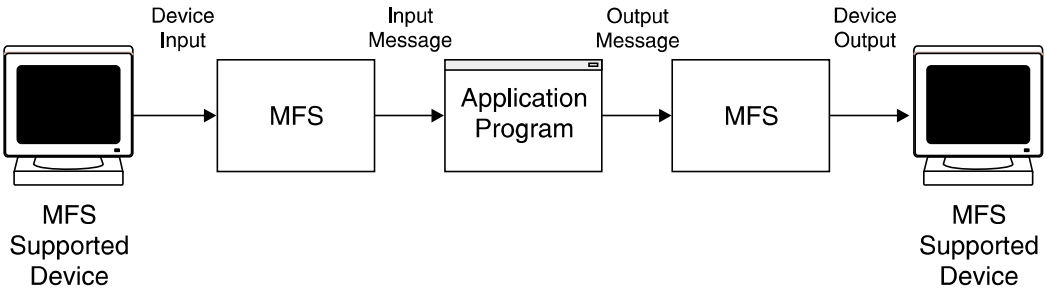

Figure 74. Message Formatting Using MFS

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.

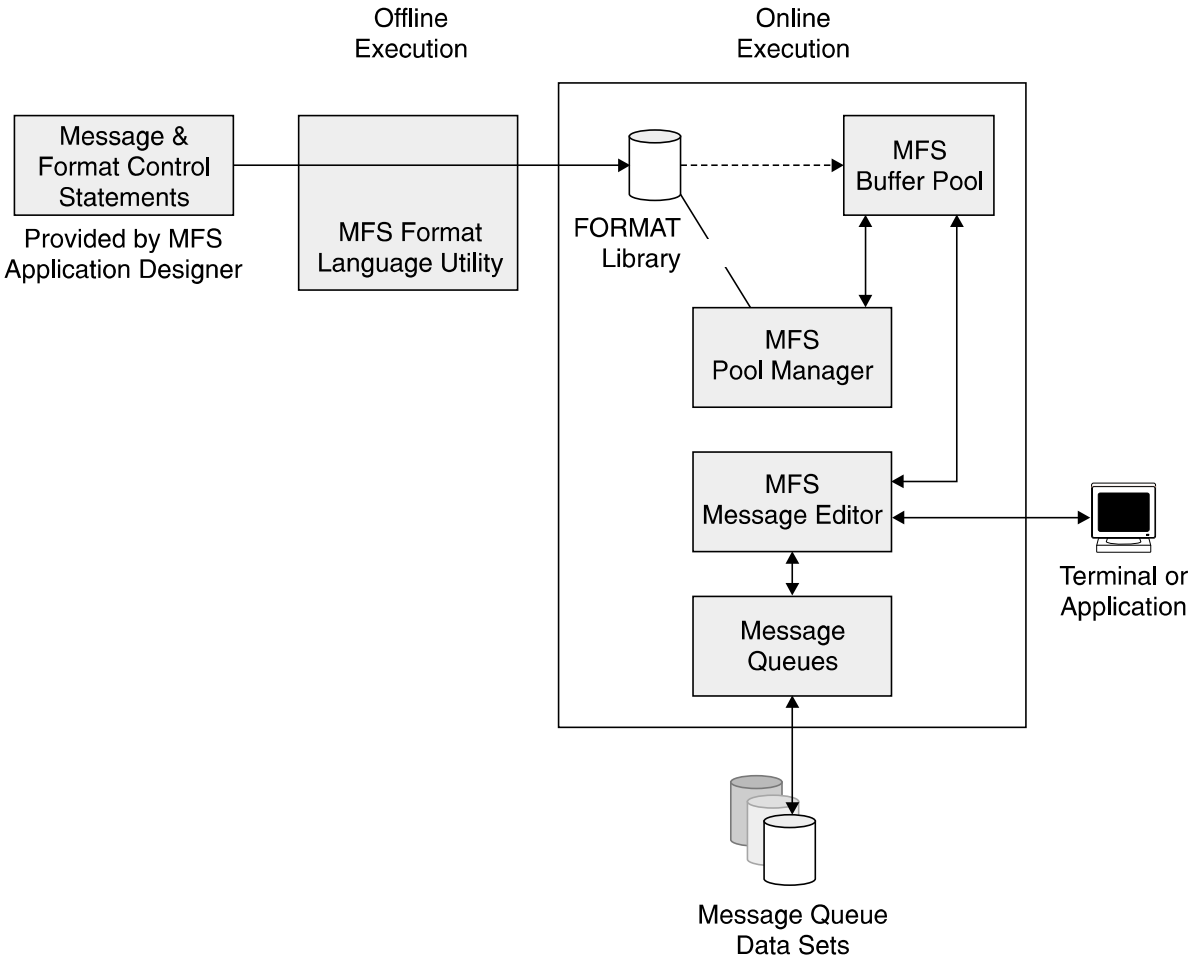

Figure 75. Overview of Message Format Service Functions

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” on page 210
- “Linkage Between Device Fields and Message Fields” on page 210
- “Linkage Between Logical Pages and Device Pages” on page 211
- “Message Description Linkage” on page 212
- “3270 Device Considerations Relative to Control Block Linkage” on page 212

MFS Control Block Chaining

Figure 76 shows the highest-level linkage, that of chained control blocks.

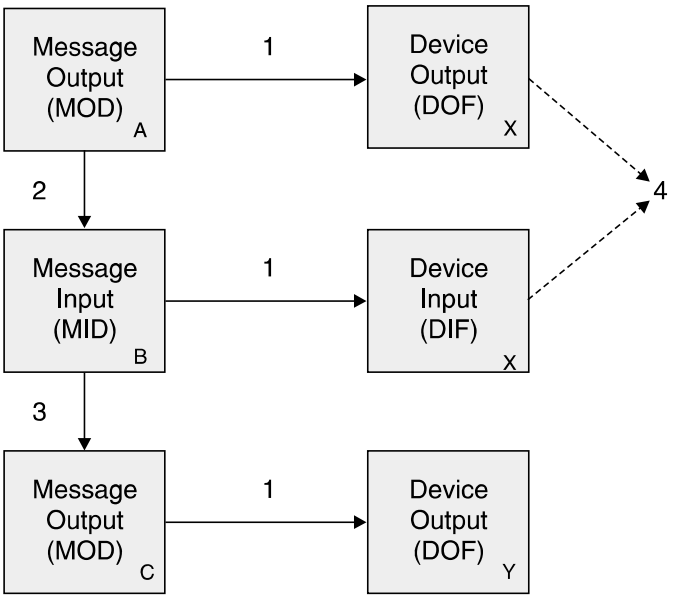

Figure 76. Chained Control Block Linkage

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.

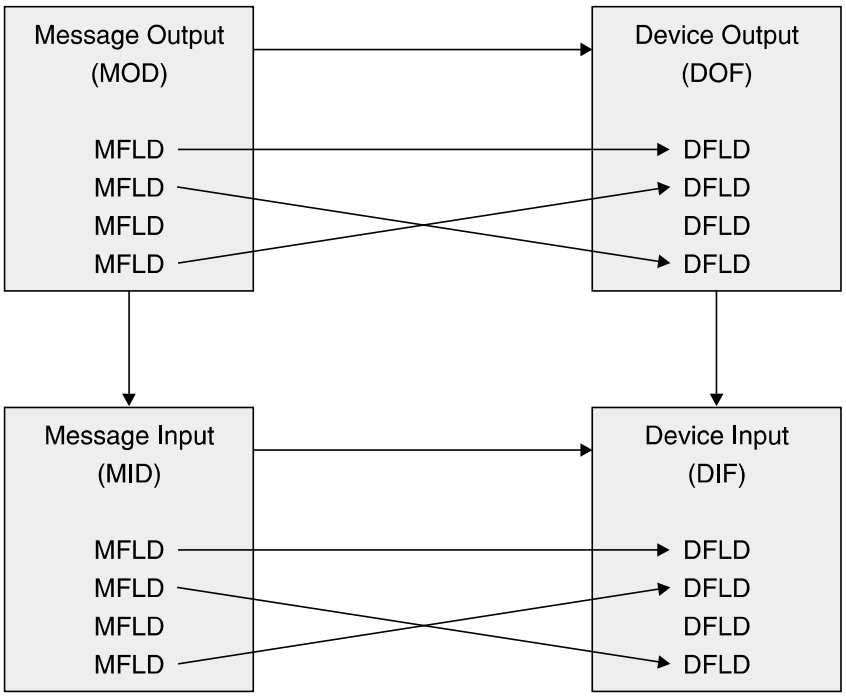

Figure 77. Linkage Between Message Fields and Device Fields

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.

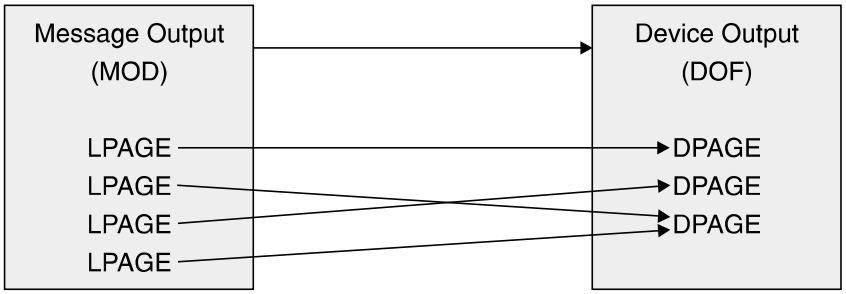

Figure 78. LPAGE - DPAGE Linkage

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.

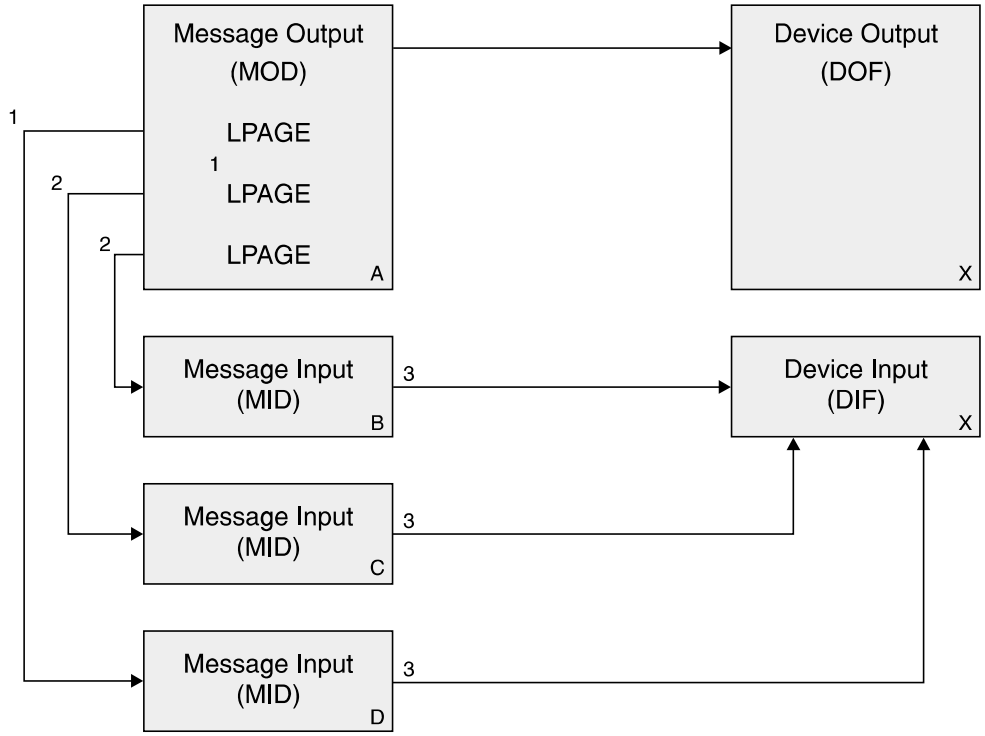

Figure 79. Optional Message Description Linkage

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.

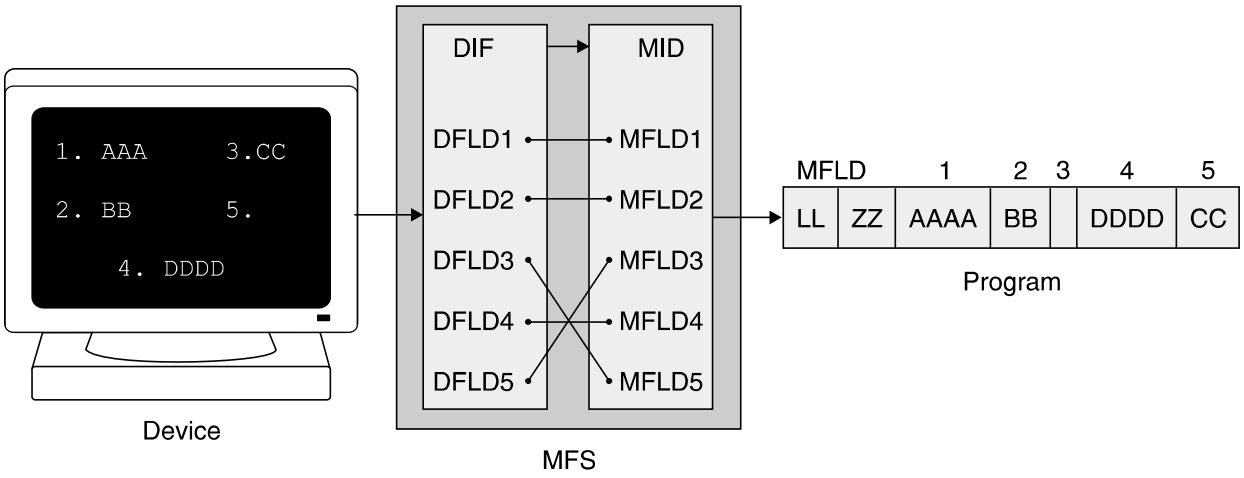

Figure 80. MFS Input Formatting

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.

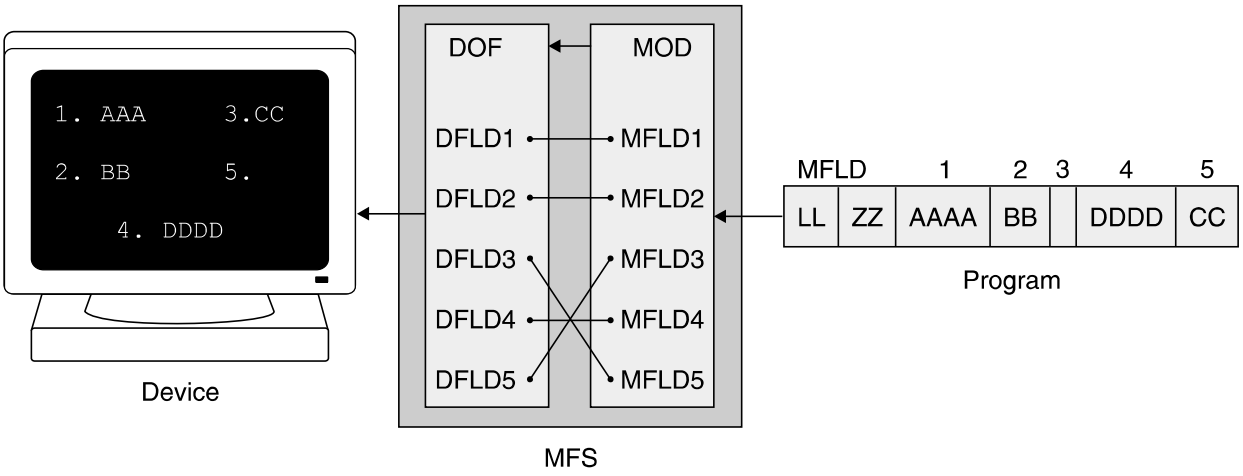

Figure 81. MFS Output Formatting

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.

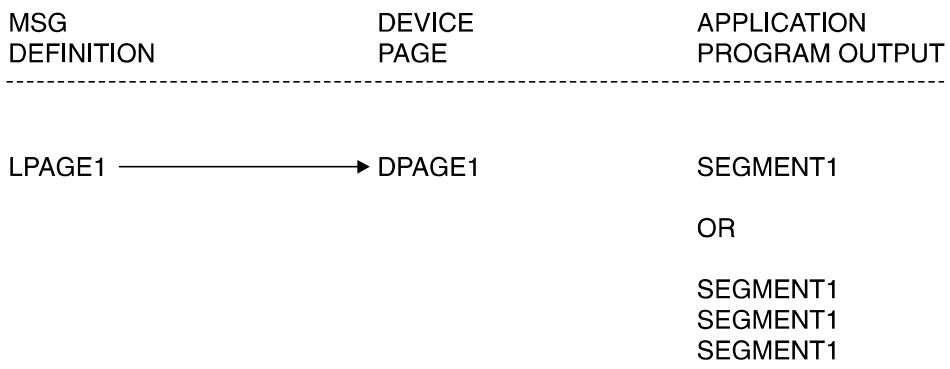

Figure 82. An Output Message Definition with One LPAGE

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.

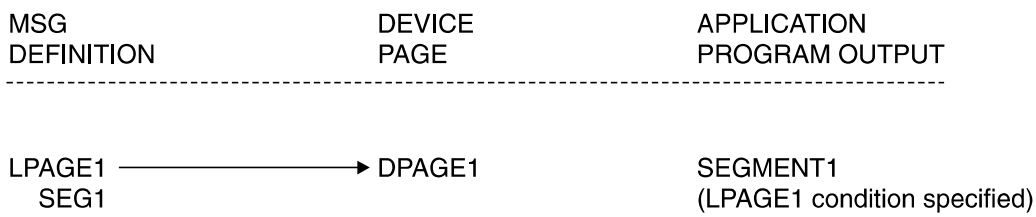

Figure 83. An Output Message Definition with Multiple Pages

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.

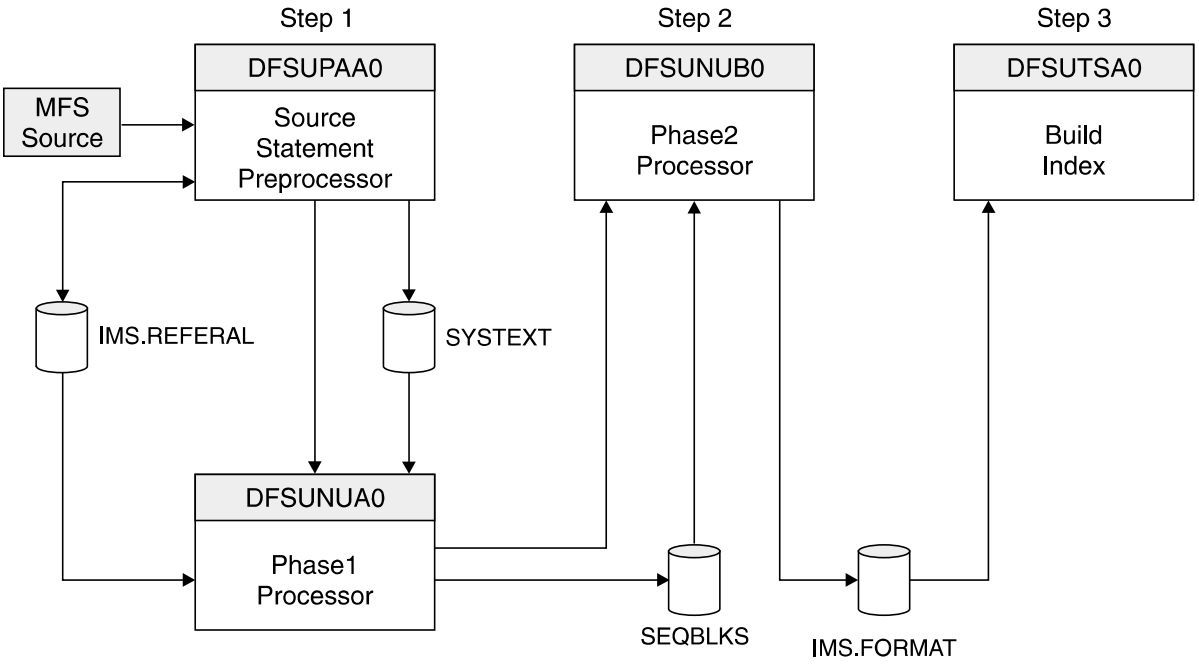

Figure 84. Overview of Process for Creating MFS Control Blocks

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.