SH20-9027-4

IMS/VS Version 1 System Programming Reference Manual

Program Product

Program Number 5740-XX2

Release 1.2

IBM

Fifth Edition (May 1976)

This edition replaces the previous edition (numbered SH20-9027-2), its technical newsletter (numbered SN20-9117), and the reprint (numbered SH20-9027-3), and makes them obsolete.

This edition applies to Version 1 Release 1.2 of IMS/VS, program number 5740-XX2, and to all subsequent releases unless otherwise indicated in new editions or technical newsletters. IMS/VS Version 1 Release 1.2 runs under VS1 Release 5. References to VS2 are for planning purposes only until Version 1 Release 1.3 of IMS/VS is available in August 1976.

Technical changes are summarized under "Summary of Amendments" following the list of figures. Each technical change is marked by a vertical line to the left of the change. In addition, miscellaneous editorial changes have been made throughout the publication.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM System/370 Bibliography,* GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P. O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

PREFACE

This is a reference manual for the person responsible for maintaining the IBM Information Management System/Virtual Storage (IMS/VS). Along with the IMS/VS Installation Guide, it provides the information necessary to install, tune, and maintain the IMS/VS system.

This manual assumes that the reader understands the basic concepts of IMS/VS, OS/VS, and the access methods that are part of the system under which IMS/VS will execute.

PREREQUISITE PUBLICATIONS

IMS/VS General Information Manual, GH20-1260
Provides a general description of IMS/VS. Describes IMS/VS system concepts and sample applications in the manufacturing, financial, medical, and process industries.

IMS/VS System/Application Design Guide, SH20-9025
Provides data base administrators, system designers, system programmers, and application programmers with information to design an IMS/VS system and the applications that operate under IMS/VS.

COREQUISITE PUBLICATIONS

IMS/VS Installation Guide, SH20-9081
This manual presents step-by-step details for the IMS/VS installation process.

HOW THIS MANUAL IS ORGANIZED

There are seven chapters and one appendix in this manual.

Chapter 1 -- contains information about jobs and procedures in the IMS/VS procedure library.

Chapter 2 -- describes the DL/I data base buffering facilities in IMS/VS.

Chapter 3 -- describes the DL/I user exit routines provided by IMS/VS.

Chapter 4 -- describes data communication functions that can be modified and how you can modify them.

Chapter 5 -- describes how to estimate storage requirements for DB and DB/DC systems.

Chapter 6 -- describes IMS/VS intelligent remote station support (System/3 and System/7).

Chapter 7 -- describes the Interactive Query Facility as it relates to IMS and provides data for estimating additional IMS/VS storage requirements when IQF is used.

Appendix A -- describes the organization of the IMS/VS Control Program.

ASSOCIATED PUBLICATIONS

<u>IMS/VS Application Programming Reference Manual</u>, SH20-9026
This document is a reference manual for the application programmer. It provides him with information about the coding techniques necessary to implement a designed application under the IMS/VS system.

<u>IMS/VS Utilities Reference Manual</u>, SH20-9029
This manual provides a description of the IMS/VS system utility programs. It describes how to execute these utilities under the operating system.

<u>IMS/VS Operator's Reference Manual</u>, SH20-9028
This manual provides the master terminal, remote terminal, and system console operators with the information associated with operating IMS/VS once the system has been established in a user environment.

<u>IMS/VS Messages and Codes Reference Manual</u>, SH20-9030
This manual lists, explains, and suggests appropriate responses to the completion codes and messages produced by all the IBM-supplied components of the IMS/VS system.

<u>IMS/VS Program Logic Manual, Volume 1 of 3</u>, LY20-8004
<u>IMS/VS Program Logic Manual, Volume 2 of 3</u>, LY20-8005
<u>IMS/VS Program Logic Manual, Volume 3 of 3</u>, LY20-8041
The internal program logic of IMS/VS is explained in the three volumes of this manual.

<u>IMS/VS Message Format Service User's Guide</u>, SH20-9053
This manual describes the use, definition, and implementation of the Message Format Service (MFS).

<u>IMS/VS Advanced Function for Communications</u>, SH20-9054
This manual explains the IMS/VS support for advanced function communications systems. It addresses the areas that programmers or analysts involved in communicating with IMS/VS must be familiar with.

<u>IMS/VS Low Level Code/Continuity Check in Data Language/I</u>: <u>Program Reference and Operation Manual</u>, SH20-9047
This manual is intended primarily for manufacturing industry DB/DC users whose programs maintain bills of material. It describes the purpose and use of the IMS/VS callable subroutine, Low-Level Code/Continuity check in Data Language/I.

<u>OS/VS1 Storage Estimates -- System Library</u>, GC24-5094
Provides instructions, formulas, and charts that can be used to estimate the real, virtual, and auxiliary storage requirements for VS1.

<u>OS/VS2 System Programming Library</u>: <u>Storage Estimates</u>, GC28-0604
Describes the real, virtual, and auxiliary storage areas of VS2 Release 2 and provides formulas for estimating the storage requirements of the system.

<u>OS/VS Linkage Editor and Loader</u>, GC26-3813
Provides the information necessary to use the linkage editor or loader program to prepare the output of a language translator for execution.

OS/VS Virtual Storage Access Method (VSAM) System Information, GC26-3835

Provides information on the release of OS/VS Virtual Storage Access Method as an independent component of OS/VS1, Release 2, and OS/VS2, Release 1.6. Describes the OS/VS VSAM distribution tape, provides detailed information on the installation of OS/VS VSAM, and provides information that temporarily supplements other OS/VS publications.

GUIDE TO USING IMS/VS SYSTEM PUBLICATIONS

Figure P-1 is a guide to using the IMS/VS system publications. This guide is divided into three parts, each dealing with a specific IMS/VS component -- Data Base System, Data Communication feature, and Interactive Query Facility (IQF) feature. For each component, one or more tasks are specified, and the IMS/VS manual or manuals that contain major information regarding this task are noted. The titles of the IMS/VS manuals are abbreviated as follows:

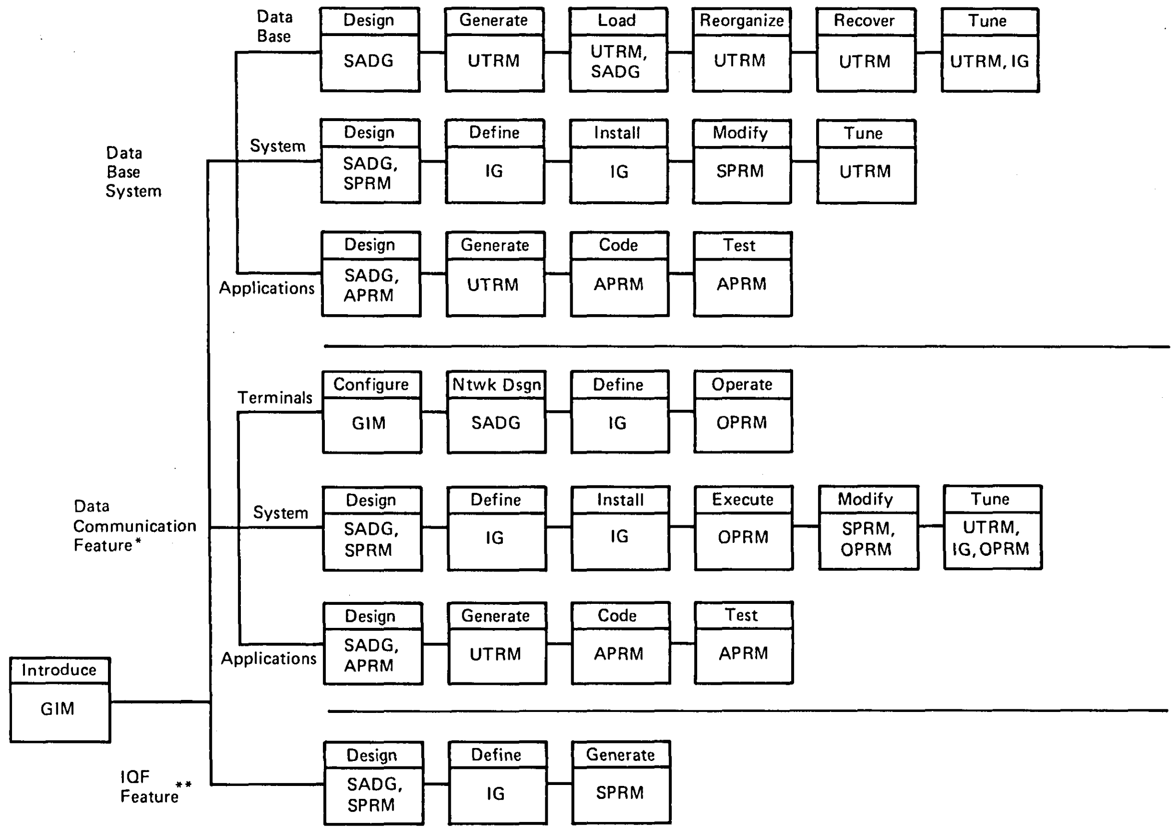
Abbreviation	Full Manual Title
GIM	IMS/VS General Information Manual
SADG	IMS/VS System/Application Design Guide
IG	IMS/VS Installation Guide
SPRM	IMS/VS System Programming Reference Manual
APRM	IMS/VS Application Programming Reference Manual
UTRM	IMS/VS Utilities Reference Manual
OPRM	IMS/VS Operator's Reference Manual

Four IMS/VS manuals are not referred to in Figure P-1:

- IMS/VS Messages and Codes Reference Manual: This manual supports essentially all tasks noted in Figure P-1.
- IMS/VS Low Level Code/Continuity Check in DL/I: Program Reference and Operation Manual: This manual supports the Data Base System when the LLC/CC function is used.
- IMS/VS Message Format Service User's Guide: This manual supports the Data Communication feature when MFS is used.
- IMS/VS Advanced Function for Communications: This manual supports the Data Communications feature when an AFC system is used.

The IQF section of Figure P-1 refers only to IMS/VS system library manuals that contain information on IQF. Additional IQF information can be found in:

- IQF General Information Manual, GH20-1074
- IQF Language Guide, GH20-1222
- IQF Terminal User's Reference Guide, GH20-1223



* References for the DC feature are in addition to those for the DB System.

** References for this feature are in addition to those for the DC feature.

Figure P-1. Guide to Using IMS/VS System Publications

CONTENTS

FIGURES

SUMMARY OF AMENDMENTS

VERSION 1, RELEASE 1.2

This publication has been revised to reflect technical and editorial changes made for Release 1.2.

IMS/VS SYSTEM LIBRARY REORGANIZATION

- IMS/VS system definition information moved to the IMS/VS Installation Guide, SH20-9081
- IMS/VS storage estimating information moved to this manual from the IMS/VS System/Application Design Guide, SH20-9025
- IMS/VS IQF information moved to this manual from the IMS/VS System/Application Design Guide, SH20-9025, and the IMS/VS Utilities Reference Manual, SH20-9029
- "IMS/VS Sample Problem" moved to the IMS/VS Installation Guide, SH20-9081, and renamed "IMS/VS Sample Application"
- Organization of the IMS/VS Control Program moved to this manual from the IMS/VS System/Application Design Guide, SH20-9025

ADDITIONAL DEVICE SUPPORT

- 3600 Acknowledge with Response Message facility incorporated into storage estimates and buffer sizes
- 3767, 3770 VTAM SDLC support incorporated into storage estimates and buffer sizes

OTHER TECHNICAL CHANGES

- Conversational Abnormal Termination Exit Routine modified
- Storage estimates updated

VERSION 1 MODIFICATION LEVEL 1 SERVICE UPDATE RELEASE 1

ADDITIONAL DEVICE SUPPORT

- Additional devices that may be defined for use with this release of IMS/VS are:
  - IBM 3740 Data Entry System
  - IBM System/7 attached on a nonswitched, binary synchronous contention or polled communication line

OTHER TECHNICAL CHANGES

- IMS/VS Data Base (DB) Monitor
- Utility Control Facility

VERSION 1 MODIFICATION LEVEL 1

ADDITIONAL DEVICE SUPPORT

- Additional devices that may be defined for use with this release of IMS/VS are:
  - IBM 3600 Finance Communication System
  - IBM 3790 Communication System
  - IBM 3275 Display Station attached through a switched communication line
- The 3600/3790 systems are supported through the Virtual Telecommunications Access Method (VTAM). VTAM is optional for the IBM 3270 Information Display System.
- Additional devices supported by the IMS/VS Message Format Service (MFS) with this release are:
  - IBM 2740/2741 Data Communications Terminals
  - IBM 3600 Finance Communication System

OTHER TECHNICAL CHANGES

- IMS/VS System definition has been modified to allow specification of the following new IMS/VS functions:
  - Additional device support (see above)
  - Parallel scheduling of application programs
  - Application program/transaction load balancing
  - Wait-for-input transactions
  - Unrecoverable inquiry transactions
  - Enforceable limits on the size and number of segments output by an application program
  - Optional MFS formatting support for the 3270 master terminal (requires a 3277-2)
  - MFS field and segment edit routines

- Fixed length scratchpad areas for conversation transaction processing
- Main storage resident PSBs and DMBs
- Response mode forced or negated by physical terminal definition
- User message tables
- Physical terminal input edit routine
- Message delete option

• Limits on system definition macro specifications have been extended.

CHAPTER 1. THE IMS/VS PROCEDURE LIBRARY

Various jobs and tasks associated with IMS/VS are supplied by IBM as procedures. The functions of these procedures are described in this chapter.

If PROCLIB=YES is specified when preparing the IMSGEN system definition macro statement, certain procedures and the jobs IMSMSG and IMSWTnnn are dynamically created and placed in IMSVS.PROCLIB. (Refer to "Perform IMS/VS System Definition" in the *IMS/VS Installation Guide* for instructions and recommendations for preparing the IMSGEN macro.) The created jobs and procedures should be examined carefully to determine if the JCL was generated as you require. These procedures may not apply to all applications, but can be used as guidelines for user-generated account oriented procedures.

If an online IMS/VS system has been defined, particular attention should be devoted to the terminal device allocation generated within the IMS procedure. A list of terminal addresses and logical and physical terminals is printed by Stage 1 of IMS/VS system definition. Examples of the procedure jobs in this chapter show the contents of the members as they are supplied by IBM. No card column image is intended. When coding your own procedures, follow JCL and VS Assembler language coding practices. Depending on the type of system being defined, your procedure library members may be a subset of the complete IMS/VS procedure library that is presented here.

PROCEDURE LIBRARY

Member Name	Description
ACBGEN	A one-step execution procedure for ACBLIB maintenance. Detailed information on ACBGEN can be found in the *IMS/VS Utilities Reference Manual*.
DBBBATCH	A one-step execution procedure for an offline Data Language/I batch processing region using IMSVS.ACBLIB.
DBDGEN	A two-step assemble and link edit procedure to produce data base definition blocks (DBDs). Detailed information on DBDGEN can be found in the *IMS/VS Utilities Reference Manual*.
DLIBATCH	A one-step execution procedure for an offline Data Language/I batch processing region using PSB and DBD libraries.
IMS	A procedure to execute an IMS/VS online control region.
IMSBATCH	A procedure to execute an IMS/VS online batch message processing region.
IMSCOBGO	A three-step compile, link edit, and go procedure combining the procedure IMSCOBOL with an exception step for a stand-alone Data Language/I batch processing region.

Member Name	Description
IMSCOBOL	A two-step compile and link edit procedure for IMS/VS applications written in COBOL.
IMSMSG	A job to execute an IMS/VS message processing region.
IMSPLI	A two-step compile and link edit procedure for IMS/VS applications written in PL/I.
IMSPLIGO	A three-step compile, link edit, and go procedure combining the procedure IMSPLI with an execution step for a stand-alone Data Language/I batch processing region.
IMSRDR	DASD read procedure to read IMSMSG job into the operating system job stream from direct access devices.
IMSWTnnn	These are jobs used to print data sets created by the SPOOL SYSOUT options.
IQFUT	This is a procedure for executing the Interactive Query Facility (IQF) Utility system. An EXEC statement to invoke the procedure is included in the Stage 2 OS/VS job stream by the IQF module DMGSI1 (Part 1 of IQF Stage 1). After system definition, this procedure is contained in IMSVS.PROCLIB. Refer to the "IQF with IMS/VS" chapter in this manual for information on IQF.
IQFFC	This procedure causes execution of the IQF System Data Base (Field File) C Utility program during the Stage 2 OS/VS job stream created by IQF Stage 1. An EXEC statement to invoke the procedure is included in the job stream by the DMGSI1 module. After system definition, this procedure is contained in the IMSVS.PROCLIB. Refer to the "IQF with IMS/VS" chapter in this manual for information on IQF.
IQFIU	This procedure causes execution of the IQF Index Creation/Update Utility program during the Stage 2 OS/VS job stream created by Stage 1. An EXEC statement to invoke the procedure is included in the job stream by the IQF DMGSI2 module (Part 2 of IQF Stage 1). After system definition, this procedure is contained in IMSVS.PROCLIB. Refer to the "IQF with IMS/VS" chapter in this manual for information on IQF.
MFDBDUMP	This is a procedure to dump the sample application data base onto a SYSOUT data set. Refer to "The IMS/VS Sample Application" in the IMS/VS Installation Guide for details about the sample application.

Member Name	Description
MFDBLOAD	A Data Language/I batch execution procedure used to load the sample application data base. Input data for the data base procedure is contained in the MFDFSYSN member of IMSVS.GENLIB. Refer to "The IMS/VS Sample Application" in the *IMS/VS Installation Guide* for details about the sample application.
MFSBACK	A two-step execution procedure to back up the MFS libraries. If the optional MFSTEST facility is used, MFSBACK contains an additional step. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSBTCH1	A one-step batch execution procedure for accumulating MFS online blocks. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSBTCH2	A one-step execution procedure for placing the MFS online blocks into IMSVS.FORMAT. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSREST	A two-step execution procedure to restore the MFS libraries. If the optional MFSTEST facility is used, MFSREST contains an additional step. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSSRVC	A one-step execution procedure for maintaining the MFS libraries. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSTEST	A two-step execution procedure for support of test mode operation of the message/format language utility. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
MFSUTL	A two-step execution procedure for defining message and format descriptions to the message/format language utility program. See the *IMS/VS Message Format Service User's Guide* for a listing of this procedure.
PSBGEN	A two-step assemble and link edit procedure to produce program specification blocks (PSBs). Detailed information on PSBGEN can be found in the *IMS/VS Utilities Reference Manual*.
SECURITY	A three-step execution, assembly, and link edit procedure for terminal and password security which invokes the security maintenance program.

In addition to the jobs and procedures placed in IMSVS.PROCLIB, two Data Language/I interfaces are also generated:

Member Name	Description
CBLTDLI	Control statements necessary to establish a COBOL to DL/I interface.
PLITDLI	Control statements necessary to establish a PL/I to DL/I interface.

The generated procedures accommodate either OS/VS1 or OS/VS2. The IMS/360 Version 1 language interface is not supported in IMS/VS.

All procedures should be placed into IMSVS.PROCLIB except the IMS and IMSRDR procedures. These two procedures should be placed into SYS1.PROCLIB.

EXECUTING JOBS USING PROCEDURES FROM IMSVS.PROCLIB

The OS/VS reader/interpreter requires that the reader procedure used to enter jobs into the OS/VS job stream specify the name of the procedure library containing the procedures used by those jobs. This name is specified on the reader procedure's IEFPDSI DD statement. IMS/VS system definition provides a reader procedure called IMSRDR which satisfies these requirements. This procedure is used, as generated, to start message regions for the online system. If entered from the operating system operator's console using the OS/VS START command (that is, S IMSRDR), it causes a message processing region to be started. If S IMSRDR,DDD, DCB=BLKSIZE=80D (where DDD is the device address of the card reader) is entered, it reads jobs into the operating system job stream from that card reader, allowing those jobs to use procedures from the IMSVS.PROCLIB data set. DCB BLKSIZE must be included with the OS/VS start command if DDD is included.

IMS/VS-SUPPLIED MEMBERS

The following procedure library members are supplied with IMS/VS by IBM.

Member Name ACBGEN

Detailed information on ACBGEN, and examples of the use of ACBGEN are in the IMS/VS Utilities Reference Manual.

```
//          PROC  SOUT=A,COMP=,RGN=100K
//G         EXEC  PGM=DFSRRC00,PARM='UPB,&COMP',REGION=&RGN
//SYSPRINT  DD    SYSOUT=&SOUT
//STEPLIB   DD    DSN=IMSVS.RESLIB,DISP=SHR
//IMS       DD    DSN=IMSVS.PSBLIB,DISP=SHR
//          DD    DSN=IMSVS.DBDLIB,DISP=SHR
//IMSACB    DD    DSN=IMSVS.ACBLIB,DISP=OLD
//SYSUT3    DD    UNIT=SYSDA,SPACE=(80,(100,100))
//SYSUT4    DD    UNIT=SYSDA,SPACE=(256,(100,100)),DCB=KEYLEN=8
//COMPCTL   DD    DSN=IMSVS.PROCLIB(DFSACBCP),DISP=SHR
```

- EXEC Statement Parameters for ACBGEN

SOUT=
: specifies the SYSOUT class. The default is A.

COMP=
: PRECOMP,POSTCOMP, in any combination, to cause the required in-place compression. The default is none.

RGN=
: specifies the region size for this execution. The default is 100K.

Member Name DBBBATCH

Assumes:

- User adds DD statements for data sets representing IMS/VS data bases.

- If VSAM data bases are used, see "Defining the IMS/VS VSAM Buffer Pool" in the IMS/VS Installation Guide.

```
//            PROC     MBR=TEMPNAME,SOUT=A,PSB=,BUF=8,
//            SPIE=0,TEST=0,EXCPVR=0,RST=0,
//            PRLD=,SRCH=0,CKPTID=,MON=N
//G EXEC      PGM=DFSRRC00,REGION=192K,
//            PARM=(DBB,&MBR,&PSB,&BUF,
//            &SPIE&TEST&EXCPVR&RST,&PRLD,&SRCH,&CKPTID,&MON)¹
//STEPLIB   DD   DSN=IMSVS.RESLIB,DISP=SHR
//          DD   DSN=IMSVS.PGMLIB,DISP=SHR
//IMSACB    DD   DSN=IMSVS.ACBLIB,DISP=SHR
//PROCLIB   DD   DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER   DD   DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),UNIT=(2400,,DEFER),
// DCB=(RECFM=VBS,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//IEFRDER2 DD   DSN=IMSLOG2,DISP=(,KEEP),VOL=(,,,99),²
// UNIT=(2400,,DEFER,SEP=IEFRDER),
// DCB=(RECFM=VBS,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//SYSUDUMP DD   SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
// SPACE=(605,(500,500),RLSE,,ROUND)
//IMSMON    DD   DUMMY³
```

[1] Parameters in parentheses are positional.

[2] This statement is included only when dual system log data sets are used.

[3] This statement describes the recording device to be used by the DB monitor. It is required only if MON=Y is specified in the PROC statement, and then only if a device other than the IMS/VS system log is to be used for monitor data. When a separate log device is used for DB monitor data, a //IMSMON DD statement must be included that specifies a sufficient BLKSIZE and LRECL (2048 and 2044 are suggested).

- EXEC Statement Parameters for DBBBATCH

MBR=
: specifies an application program name.

SOUT=
: specifies the class assigned to SYSOUT DD statements.

PSB=
: is an optional parameter specifying a PSB name when the PSB name and application program name are different.

BUF=
: specifies the data base buffer size. If not present, the default size specified at system definition will be used. Buffer size is specified in 1K multiples. Values may range from 1 through 999.

SPIE=
specifies the SPIE option:

0 - allow user SPIE, if any, to remain in effect while processing the application program call.

1 - negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before returning to the application program.

A value of 0 or 1 must appear in the generated JCL

TEST=
specifies whether (1) or not (0) the addresses in the user's call list should be checked for validity. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

EXCPVR=
specifies whether EXCP (0) or EXCPVR (1) is to be used for data sets processed by OSAM. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

RST=
specifies UCF restart: (0) no, (1) yes. Refer to the _IMS/VS Utilities Reference Manual_ for details. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

PRLD=
specifies a 2-character suffix for DFSMPLxx, the IMSVS.PROCLIB member that lists the modules to be preloaded in the region/partition. See the _IMS/VS Installation Guide_ for details.

SRCH=
is the module search indicator for directed load.

0 - standard search.
1 - search JPA and LPA before PDS (VS2 only).

CKPTID=
specifies the checkpoint at which the program is to be restarted; specified as either a 1- to 8-character extended checkpoint ID or a 12-character 'time-stamp' checkpoint ID.

MON=
specifies whether (Y) or not (N) the DB monitor is to be active for this execution.

Member Name DBDGEN

Detailed information on DBDGEN, and examples of the use of DBDGEN are in the *IMS/VS Utilities Reference Manual*.

```
// PROC     MBR=TEMPNAME,SOUT=A
//C         EXEC  PGM=IFOX00,REGION=128K,PARM='OBJ,NODECK'
//SYSLIB    DD    DSN=IMSVS.MACLIB,DISP=SHR
//SYSGO     DD    UNIT=SYSDA,DISP=(,PASS),SPACE=(80,(100,100),RLSE),
// DCB=(BLKSIZE=400,RECFM=FB,LRECL=80)
//SYSPRINT DD     SYSOUT=&SOUT,DCB=BLKSIZE=1089,
// SPACE=(121,(300,300),RLSE,,ROUND)
//SYSUT1    DD    UNIT=SYSDA,DISP=(,DELETE),SPACE=(1700,(100,50))
//SYSUT2    DD    UNIT=SYSDA,DISP=(,DELETE),SPACE=(1700,(100,50))
//SYSUT3    DD    UNIT=(SYSDA,SEP=(SYSLIB,SYSUT1,SYSUT2)),
// SPACE=(1700,(100,50))
//L EXEC PGM=DFSILNK0,PARM='XREF,LIST',COND=(4,LT,C),REGION=120K
//STEPLIB   DD    DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD    DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSPRINT DD     SYSOUT=&SOUT,DCB=BLKSIZE=1089,
// SPACE=(121,(90,90),RLSE)
//SYSLMOD   DD    DSN=IMSVS.DBDLIB(&MBR),DISP=SHR
//SYSUT1    DD    UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),DISP=(,DELETE),
// SPACE=(1024,(100,10),RLSE)
```

Member Name DLIBATCH

Assumes:

- User adds DD statements for data sets representing IMS/VS data bases.
- If VSAM data bases are used, see "Defining the IMS/VS VSAM Buffer Pool" in the IMS/VS Installation Guide.

```
//          PROC     MBR=TEMPNAME,SOUT=A,PSB=,BUF=,
//          SPIE=0,TEST=0,EXCPVR=0,RST=0,
//          PRLD=,SRCH=0,CKPTID=,MON=N
//G EXEC    PGM=DFSRRC00,REGION=192K,
//          PARM=(DLI,&MBR,&PSB,&BUF,
| //        &SPIE&TEST&EXCPVR&RST,&PRLD,&SRCH,&CKPTID,&MON) 1
//STEPLIB   DD       DSN=IMSVS.RESLIB,DISP=SHR
//          DD       DSN=IMSVS.PGMLIB,DISP=SHR
//IMS       DD       DSN=IMSVS.PSBLIB,DISP=SHR
//          DD       DSN=IMSVS.DBDLIB,DISP=SHR
//PROCLIB   DD       DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER   DD       DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),
// UNIT=(2400,,DEFER),
// DCB=(RECFM=VBS,BLKSIZE=1920,LRECL=1916,BUFNO=2) 2
//IEFRDER2 DD        DSN=IMSLOG2,DISP=(,KEEP),VOL=(,,,99),3
// UNIT=(2400,,DEFER,SEP=IEFRDER),
// DCB=(RECFM=VBS,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//SYSUDUMP DD        SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
//  SPACE=(605,(500,500),RLSE,,ROUND)
//IMSMON    DD       DUMMY4
```

| 1 Parameters in parentheses are positional.

2 The BLKSIZE and LRECL values shown are the default values. If the DCB parameters are changed, log initialization calculates the smallest value necessary for logical record length (the larger of 1008 or the longest message queue size plus 16). If the JCL logical record length value is larger than the calculated value, the JCL value is used; otherwise, log initialization uses the calculated value for logical record length and adds 4 for the block size.

Log initialization checks BUFNO. If BUFNO is less than 2, 2 is used. If the JCL BUFNO is greater than 2, the JCL value is used.

3 This statement is included only when dual system log data sets are used.

4 This statement describes the recording device to be used by the DB monitor. It is required only if MON=Y is specified in the PROC statement, and then only if a device other than the IMS/VS system log is to be used for monitor data. When a separate log device is used for DB monitor data, a //IMSMON DD statement must be included that specifies a sufficient BLKSIZE and LRECL (2048 and 2044 are suggested).

- EXEC Statement Parameters for DLIBATCH

MBR=
: specifies an application program name.

SOUT=
: specifies the class assigned to SYSOUT DD statements.

PSB=

is an optional parameter specifying a PSB name when the PSB name and application program name are different.

BUF=

specifies the data base buffer size. If not present, the default size specified at system definition will be used. Buffer size is specified in 1K multiples. Values may range from 1 through 999.

SPIE=

specifies the SPIE option:

0 - allow user SPIE, if any, to remain in effect while processing the application program call.

1 - negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before returning to the application program.

A value of 0 or 1 must appear in the generated JCL statement for this parameter.

TEST=

specifies whether (1) or not (0) the addresses in the user's call list should be checked for validity. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

EXCPVR=

specifies whether EXCP (0) or EXCPVR (1) is to be used for data sets processed by OSAM. EXCPVR is not valid in MVS systems. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

RST=

specifies UCF restart. Refer to the _IMS/VS Utilities Reference Manual_ for details. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

PRLD=

specifies a 2-character suffix for DFSMPLxx, the IMSVS.PROCLIB member that lists the modules to be preloaded in the region/partition. See the _IMS/VS Installation Guide_ for details.

SRCH=

is the module search indicator for directed load.

0 - standard search.

1 - search JPA and LPA before PDS (for VS2 only).

CKPTID=

specifies the checkpoint at which the program is to be restarted; specified as either a 1- to 8-character extended checkpoint ID or a 12-character 'time-stamp' checkpoint ID.

MON=

specifies whether (Y) or not (N) the DB monitor is to be active for this execution.

Member Name IMS

This procedure cannot be entered in the normal OS/VS job stream (through a card reader) unless modified as described in the IMS/VS Operator's Reference Manual.

Assumes:

- User adds DD statements for data sets representing IMS/VS data bases.
- If VSAM data bases are used, see "Defining the IMS/VS VSAM Buffer Pool" in the IMS/VS Installation Guide.

```
//       PROC  RGN=600K,SOUT=A,DPTY='(14,15)',
//             CTL=CTL1,RES=,FRE=,QBUF=,DYBN=,PST=,
//             SAV=,EXVR=,PRF=,SRCH=,FBP=,PSB=,DMB=,DBB=,
//             TPDP=,WKAP=,PSBW=,CWAP=,DBWP=,MFS=,
//             SUF=,FIX=,PRLD=,VSPEC=
//IEFPROC EXEC PGM=DFSRRC002,REGION=&RGN,DPRTY=&DPTY
// PARM=(&CTL,
// &RES,&FRE,&QBUF,&DYBN,&PST,&SAV,
// &EXVR,&PRF,&SRCH,&FBP,&PSB,&DMB,&DBB,
// &TPDP,&WKAP,&PSBW,&CWAP,&DBWP,&MFS,
// &SUF,&FIX,&PRLD,&VSPEC) 3
//*
//*
//* THE MEANING AND MAXIMUM SIZE OF EACH PARAMETER
//* IS AS FOLLOWS:
//*
//******** CONTROL REGION SPECIFICATIONS ********
//*   RES    X     BLOCK RESIDENT (N = NO, Y = YES)
//*   FRE    XXX   NUMBER OF FORMAT REQUEST ELEMENTS
//*   QBUF   XXX   NUMBER OF MESSAGE QUEUE BUFFERS
//*   DYBN   XXX   NUMBER OF DYNAMIC LOG BFFRS FOR PI
//*   PST    XX    NUMBER OF PST'S
//*   SAV    XXX   NUMBER OF DYNAMIC SAVE AREA SETS
//*   EXVR   X     EXCPVR INDICATOR (0 = NO OR EXCPVR=EXCP, 1 = EXCPVR)
//*   PRF    X     PREFETCH OPTION (Y = YES, N = NO)
//*   SRCH   X     MODULE SEARCH INDICATOR FOR DIRECTEDLOAD
//*                0 = STANDARD SEARCH
//*                1 = SEARCH JPA AND LPA BEFORE PDS
//*
//*
//******** STORAGE POOL SIZES IN 1K BLOCKS ******
//*
//*   FBP    XXX   MESSAGE BUFFER POOL
//*   PSB    XXX   PSB POOL
//*   DMB    XXX   DMB POOL
//*   DBB    XXX   DATA BASE BUFFER POOL
//*   TPDP   XXX   TP DEVICE I/O POOL
//*   WKAP   XXX   WORKING STORAGE BUFFER POOL
//*   PSBW   XXX   PSB WORK POOL
//*   CWAP   XXX   COMMUNICATIONS WORK AREA POOL
//*   DBWP   XXX   DATABASE WORK POOL
//*   MFS    XXX   MAXIMUM MFSTEST SPACE
//*
//******** MEMBER SUFFIXES *********************
//*
//*   SUF    X     LAST CHARACTER OF CTL PROGRAM LOAD MODULE MEMBER NAME
//*   FIX    XX    2 CHARACTER FIX PROCEDURE MODULE SUFFIX
//*   PRLD   XX    2 CHARACTER PROCLIB MEMBER SUFFIX FOR PRELOAD
//*   VSPEC  XX    2 CHARACTER VSAM BUFFER POOL SPEC MODULE SUFFIX
//*
```

```
//*************************************************
//*
//PROCLIB  DD  DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER  DD  DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),
// UNIT=(2400,,DEFER),
// DCB=(RECFM=VBS,BLKSIZE=3968,LRECL=3964,BUFNO=2)⁴
//IEFRDER2 DD  DSN=IMSLOG2,DISP=(,KEEP),VOL=(,,,99),⁵
// UNIT=(2400,,DEFER,SEP=IEFRDER),
// DCB=(RECFM=VBS,BLKSIZE=3968,LRECL=3964,BUFNO=2)
//IMSLOGR  DD  DSN=IMSLOG,DISP=(OLD,KEEP),
// VOL=SER=000000,UNIT=AFF=IEFRDER⁶
//IMSMON   DD  DSN=IMSMON,DISP=(,KEEP),⁷
// VOL=(,,,99),UNIT=(2400,,DEFER,SEP=IEFRDER)
//QBLKS    DD  DSN=IMSVS.QBLKS,DISP=OLD
//SHMSG    DD  DSN=IMSVS.SHMSG,DISP=OLD
//LGMSG    DD  DSN=IMSVS.LGMSG,DISP=OLD
//IMSACB   DD  DSN=IMSVS.ACBLIB,DISP=SHR
//IMSDILIB DD  DSN=IMSVS.FORMAT,DISP=OLD⁸
//IMSTFMT  DD  DSN=IMSVS.TFORMAT,DISP=SHR⁹
//         DD  DSN=IMSVS.FORMAT,DISP=OLD⁹
//IMSSPA   DD  DSN=IMSVS.SPA,DISP=OLD
//SYSUDUMP DD  SYSOUT=&SOUT,
// DCB=(LRECL=125,RECFM=FBA,BLKSIZE=3129),
// SPACE=(6050,300,,,ROUND)
//PRINTDD  DD  SYSOUT=&SOUT
//IMSDBL   DD  DSN=IMSVS.DBLLOG,DISP=SHR
//*
//* DD STATEMENTS FOR COMMUNICATIONS LINES
//* ARE INSERTED HERE BY IMS/VS SYSTEM
//* DEFINITION.
//*
//* USER MUST SUPPLY THE DD STATEMENTS
//* FOR THE ON-LINE DATA BASES TO BE
//* INSERTED HERE PRIOR TO ATTEMPTING
//* AN ON-LINE SYSTEM EXECUTION USING
//* THIS PROCEDURE.
```

[1] To execute the IMS/VS online system as a problem program instead of as a subtask of the master scheduler, the first parameter field of the execute card in the IMS procedure must be overridden. The JCL below accomplishes this, however, it is not recommended that IMS be run as a problem program in a production environment.

```
//IMSJOB JOB ACCT,MSGLEVEL=(1,1),PRTY=13
//IMS EXEC   IMS,PARM.IEFPROC=CTX,(include the remaining
             parameters generated for your system)
```

[2] The program name specified is DFSRRC00 for OS/VS1 and DFSMVRC0 for OS/VS2.

[3] Parameters in parentheses are positional.

[4] The BLKSIZE and LRECL values shown are the default values. If the DCB parameters are changed, log initialization calculates the smallest value necessary for LRECL (the larger of 1008 and the long message queue size plus 16). If the JCL LRECL value is larger, the JCL value is used; otherwise log initialization uses the calculated value for LRECL and adds 4 for the BLKSIZE.

The user must be concerned with the LRECL value required to perform an IMS/VS command that refers to all data communication lines and/or physical terminals (for example, /START LINE ALL). The following formula should be used as a guide when calculating the LRECL required to successfully execute such commands:

LRECL= (300+11*N)+(300+6*L)

where:

N is the number of defined VTAM node names.

L is the number of non-VTAM lines in the defined system.

The DCB BLKSIZE parameter need not be coded on the IEFRDER DD statement. If it is coded, it must not be made smaller nor omitted for subsequent executions of IMS unless a cold start is to be performed.

Log initialization checks BUFNO. If BUFNO is less than 2, 2 is used. If the JCL BUFNO is greater than 2, the JCL value is used.

5 This statement is included only when dual system log data sets are used.

6 The BLKSIZE parameter is ignored if coded on the IMSLOGR DD statement. IMSLOGR always uses the current blocksize from IEFRDER.

7 This DD statement is included only when the IMS/VS DC monitor is used.

8 This DD statement must specify DISP=OLD; it is included only when MFS is used. A DD DUMMY specification is not supported.

9 These statements are included only when MFSTEST is specified.

- EXEC Statement Parameters for IMS

RGN=
: specifies the size of the OS/VS region to be allocated to the IMS/VS control program. RGN= has no effect in an OS/VS1 system.

SOUT=
: specifies the class to be assigned to SYSOUT DD statements.

DPTY=
: specifies the OS/VS dispatching priority at which the IMS/VS control region should operate. See the OS/VS1 and OS/VS2 JCL documentation for details of DPRTY.

The IMS/VS control region must not be executed at priority zero or scheduled into a region whose priority falls within a JES2 APG, or a partition whose priority falls within JES1 DDG. The control region's priority must be higher than an OS/VS APG or DDG if IMS/VS message processing or batch message processing regions reside in the APG or DDG. A general rule to follow is: IMS CTL dispatching priority must always be higher than the dispatching priority of any IMS/VS dependent region.

CTL=CTL
: specifies that IMS/VS should execute as an OS/VS system task.

RES=

specifies whether (Y) or not (N) the PSBs and or DMBs defined as RESIDENT should be made resident at system initialization time.

FRE=

specifies the number of fetch request elements that are to be used for loading MFS blocks into the message format block pool.

QBUF=

specifies the number of message queue buffers in subpool 0 to be allocated to the queue pool.

DYBN=

specifies the number of dynamic log buffers.

PST=

specifies the number of PSTs (partition specification tables) to be allocated at system initialization time. The number specified indicates the maximum number of dependent regions that can be active concurrently.

SAV=

specifies the number of dynamic save area sets to be used for communication terminal I/O requests.

EXVR=

specifies whether (1) or not (0) EXCPVR is to be used in the online system for data sets processed by OSAM.

PRF=

specifies whether (Y) or not (N) the MFS prefetch option is to be used. Default value is Y.

SRCH=

specifies the module search indicator for directed load: 0= standard search and 1= search JPA and LPA before PDS.

FBP=

specifies the number of 1K blocks in subpool 0 to be allocated to the message format block pool. (Identified in a main storage dump as MFBP.) Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

PSB=

specifies the number of 1K blocks in subpool 0 to be allocated to the PSB pool. (Identified in a main storage dump as DLMP.) Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

DMB=

specifies the number of 1K blocks in subpool 0 to be allocated to the DMB pool. (Identified in a main storage dump as DLDP.) Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

DBB=

specifies the number of 1K blocks in subpool 0 to be allocated to the data base buffer pool. (Identified in a main storage dump as DBAS.) Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

TPDP=

specifies the number of 1K blocks in subpool 0 to be allocated to the communication line buffer pool. (Identified in a main storage dump as I/OP.) Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

WKAP=

specifies the number of 1K blocks in subpool 0 to be allocated to the control program working area. Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

PSBW=

specifies the number of 1K blocks in subpool 0 to be allocated to the PSB work area pool. Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

CWAP=

specifies the number of 1K blocks of subpool 0 to be allocated to the communications work area pool. Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

DBWP=

specifies the number of 1K blocks of subpool 0 to be allocated to the data base work area pool. Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

MFS=

specifies the maximum number of 1K blocks of the communication line buffer pool (TPDP) to be available for use by MFSTEST. The number specified must not exceed the TPDP size minus 5. Parameters for specifying pool sizes are rounded up to page size (OS/VS1=2K; OS/VS2=4K) if they are specified as less.

SUF=

specifies the suffix for the control program name. This allows multiple copies of the IMS/VS nucleus to reside on IMSVS.RESLIB.

FIX=

specifies the suffix for DFSFX. This indicates the IMSVS.PROCLIB member to be used to control page fixing of portions of the control program.

PRLD=

specifies a 2-character suffix for DFSMPLxx, the IMSVS.PROCLIB member that lists the modules to be preloaded in the region/partition. See the _IMS/VS Installation Guide_ for details.

VSPEC=

specifies the suffix of the VSAM buffer pool specification module.

Member Name IMSBATCH

```
//          PROC     MBR=TEMPNAME,SOUT=3,OPT=N,SPIE=0,TEST=0,
//          PSB=,PRLD=,CKPTID=,IN=,OUT=,DIRCA=000
//G         EXEC     PGM=DFSRRC00,REGION=52K,
//          PARM=(BMP,&MBR,&PSB,&IN,&OUT,
//          &OPT&SPIE&TEST&DIRCA,&PRLD,&STIMER,&CKPTID) 1
//STEPLIB   DD       DSN=IMSVS.RESLIB,DISP=SHR
//          DD       DSN=IMSVS.PGMLIB,DISP=SHR
//PROCLIB   DD       DSN=IMSVS.PROCLIB,DISP=SHR
//SYSUDUMP  DD SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=VBA,BLKSIZE=3129),
// SPACE=(125,(2500,100),RLSE,,ROUND)
```

[1] Parameters in parentheses are positional.

- EXEC Statement Parameters for IMSBATCH

MBR=

specifies an application program name.

SOUT=

specifies the class assigned to SYSOUT DD statements.

OPT=

specifies the action to be taken if the batch message region starts and no control program is active.

N - ask operator for decision. This is the default.
W - wait for a control program.
C - cancel the batch message region automatically.

A value of N or W or C must appear in the generated JCL statement for this parameter.

SPIE=

specifies the SPIE option:

0 - allow user SPIE, if any, to remain in effect while processing the application program call.

1 - negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before returning to the application program.

SPIE macros issued by the application program are only effective for program checks which occur within the batch message region. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

TEST=

specifies whether (1) or not (0) the addresses in the user's call list should be checked for validity. A value of 0 or 1 must appear in the generated JCL statement for this parameter.

PSB=

is an optional parameter specifying a PSB name when the PSB name and application program name are different.

PRLD=

specifies a 2-character suffix for DFSMPLxx, the IMSVS.PROCLIB member that lists the modules to be preloaded in the region/partition. See the _IMS/VS Installation Guide_ for details.

STIMER=

STIMER option:

0=none
1=no DL/I
2=with DL/I (default)

CKPTID=

specifies the checkpoint at which the program is to be restarted; specified as either a 1- to 8-character extended checkpoint ID or a 12-character 'time-stamp' checkpoint ID.

IN=

specifies an input transaction code. This parameter is necessary only when the application program intends to access the message queues. If this parameter is specified, the OUT= parameter is ignored.

OUT=

specifies the transaction code or logical terminal name to which an output message is to be sent. It is necessary when the application program desires to send output without accessing the input queues. This parameter is ignored if IN= is also specified.

DIRCA=

specifies the size of the dependent region interregion communication area; the size specified must be a three-digit number (for example, 001) representing the number of 1K blocks of subpool 253 to be reserved to hold a copy of the user's PCBs.

The size for DIRCA when DIRCA=000 equals the control words at the beginning of the DIRCA plus the sum of the PCBs in the largest PSB found by the block loader.

If dynamic PSBs are used, and the largest PSB is larger than the default size as calculated above, DIRCA must be specified on the EXEC statement in the PARM field. A three-digit number must appear in the generated JCL statement for this parameter.

Member Name IMSCOBGO

Assumes:

- User supplies source data from SYSIN.
- Output Class A.
- MBR=NAME, when NAME is load module name for program.
- SYSDA is a generic device name.
- User adds DD statements for data sets representing IMS/VS data bases.

- If VSAM data bases are used, see "Defining the IMS/VS VSAM Buffer Pool" in the IMS/VS Installation Guide.

- Execution time limit of 2 minutes specified.

```
//          PROC    MBR=,PAGES=60,
//          SOUT=A,PSB=,SPIE=0,TEST=0,EXCPVR=0,
//          RST=0,PRLD=,SRCH=0,CKPTID=,BUF=24
//C         EXEC    PGM=IKFCBL00,REGION=150K,
//          PARM='SIZE=130K,BUF=10K,LINECNT=50'
//SYSLIN    DD      DSN=&&LIN,DISP=(MOD,PASS),UNIT=SYSDA,
//          DCB=(LRECL=80,RECFM=FB,BLKSIZE=400),
//          SPACE=(3520,(40,10),RLSE,,ROUND)
//SYSPRINT DD      SYSOUT=&SOUT,DCB=(LRECL=121,BLKSIZE=605,RECFM=FBA),
//          SPACE=(605,(&PAGES.0,&PAGES),RLSE,,ROUND)
//SYSUT1    DD      UNIT=SYSDA,DISP=(,DELETE),
//          SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT2    DD      UNIT=SYSDA,DISP=(,DELETE),
//          SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT3    DD      UNIT=SYSDA,DISP=(,DELETE),
//          SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT4    DD      UNIT=SYSDA,DISP=(,DELETE),
//          SPACE=(3520,(100,10),RLSE,,ROUND)
//L         EXEC    PGM=DFSILNK0,REGION=120K,PARM='XREF,LET,LIST',
//          COND=(4,LT,C)
//STEPLIB   DD      DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIB    DD      DSN=SYS1.COBLIB,DISP=SHR
//RESLIB    DD      DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD      DSN=&&LIN,DISP=(OLD,DELETE),VOL=REF=*.C.SYSLIN
//          DD      DSN=IMSVS.PROCLIB(CBLTDLI),DISP=SHR
//          DD      DDNAME=SYSIN
//SYSLMOD   DD      DSN=IMSVS.PGMLIB(&MBR),DISP=SHR
//SYSPRINT DD      SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
//          SPACE=(605,(&PAGES.0,&PAGES),RLSE,,ROUND)
//SYSUT1    DD      UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),DISP=(,DELETE),
//          SPACE=(3520,(100,10),RLSE,,ROUND)
//G         EXEC    PGM=DFSRRC00,REGION=150K,TIME=2,COND=(4,LT),
// PARM='DLI,&MBR,&PSB,&BUF,&SPIE&TEST&EXCPVR&RST,&PRLD,&SRCH,&CKPTID'
//STEPLIB   DD      DSN=IMSVS.RESLIB,DISP=SHR
//          DD      DSN=IMSVS.PGMLIB,DISP=SHR
//IMS       DD      DSN=IMSVS.PSBLIB,DISP=SHR
//          DD      DSN=IMSVS.DBDLIB,DISP=SHR
//PROCLIB   DD      DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER   DD      DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),
// UNIT=(2400,,DEFER),
// DCB=(RECFM=VBS,BLKSIZE=1408,LRECL=1400,BUFNO=1)
//IEFRDER2 DD      DSN=IMSLOG2,DISP=(,KEEP),VOL=(,,,99),1
// UNIT=(2400,,DEFER,SEP=IEFRDER),
// DCB=(RECFM=VBS,BLKSIZE=1408,LRECL=1400,BUFNO=1)
//SYSOUT    DD      SYSOUT=&SOUT,SPACE=(CYL,(1,1)),DCB=(LRECL=133,RECFM=FA)
//SYSUDUMP DD      SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=FBA,BLKSIZE=3025),
//          SPACE=(3025,(200,100),RLSE,,ROUND)
```

1 This statement is included only when dual system log data sets are used.

Member Name IMSCOBOL

Assumes:

- User supplies source data from SYSIN.
- Output Class A.
- MBR=NAME, when NAME is load module name for program.
- SYSDA is a generic device name.
- RESLIB cataloged.

```
//           PROC    MBR=,PAGES=60,
//           SOUT=A
//C          EXEC    PGM=IKFCBL00,REGION=150K,
//           PARM='SIZE=130K,BUF=10K,LINECNT=50'
//SYSLIN     DD      DSN=&&LIN,DISP=(MOD,PASS),UNIT=SYSDA,
//           DCB=(LRECL=80,RECFM=FB,BLKSIZE=400),
//           SPACE=(3520,(40,10),RLSE,,ROUND)
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=121,BLKSIZE=605,RECFM=FBA),
//           SPACE=(605,(&PAGES.0,&PAGES),RLSE,,ROUND)
//SYSUT1     DD      UNIT=SYSDA,DISP=(,DELETE),
//           SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT2     DD      UNIT=SYSDA,DISP=(,DELETE),
//           SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT3     DD      UNIT=SYSDA,DISP=(,DELETE),
//           SPACE=(3520,(100,10),RLSE,,ROUND)
//SYSUT4     DD      UNIT=SYSDA,DISP=(,DELETE),
//           SPACE=(3520,(100,10),RLSE,,ROUND)
//L          EXEC    PGM=DFSILNK0,REGION=120K,PARM='XREF,LET,LIST',
//           COND=(4,LT,C)
//STEPLIB    DD      DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIB     DD      DSN=SYS1.COBLIB,DISP=SHR
//RESLIB     DD      DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN     DD      DSN=&&LIN,DISP=(OLD,DELETE),VOL=REF=*.C.SYSLIN
//           DD      DSN=IMSVS.PROCLIB(CBLTDLI),DISP=SHR
//           DD      DDNAME=SYSIN
//SYSLMOD    DD      DSN=IMSVS.PGMLIB(&MBR),DISP=SHR
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
//           SPACE=(605,(&PAGES.0,&PAGES),RLSE,,ROUND)
//SYSUT1     DD      UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),DISP=(,DELETE),
//           SPACE=(3520,(100,10),RLSE,,ROUND)
```

Member Name IMSMSG

```
//MESSAGE JOB 1,IMS,MSGLEVEL=1,PRTY=11,CLASS=A,MSGCLASS=A,REGION=52K
//REGION   EXEC    PGM=DFSRRC00,REGION=52K,TIME=1440,
// PARM='MSG,001000000000'
//STEPLIB  DD       DSN=IMSVS.RESLIB,DISP=SHR
//         DD       DSN=IMSVS.PGMLIB,DISP=SHR
//PROCLIB  DD       DSN=IMSVS.PROCLIB,DISP=SHR
//SYSUDUMP DD       SYSOUT=A,DCB=(LRECL=125,BLKSIZE=3219,RECFM=VBA),
// SPACE=(125,(2500,100),RLSE,,ROUND)
```

- EXEC Statement Parameters for IMSMSG

PARM=
: 'MSG,AAAAAAAAAAAA,BCDEFFGGG,HH,I'

MSG=
: is a required positional parameter indicating a message region is to be started.

AAAAAAAAAAAA=
: is a required positional parameter specifying 4 three-digit decimal numbers indicating which classes of messages will be handled by this message region. That is, if classes 1, 2, and 3 are to be processed by this region, the PARM field would be specified as PARM='MSG,001002003000'.

The sequence of specifying the classes determines relative class priority within the message region. In the above example, all Class 1 messages are selected for scheduling before any Class 2 messages would be considered. Class numbers cannot be greater than the maximum number of classes specified during system definition.

BCDEFFGGG is required if HH or I is specified.

B=
: specifies the action to be taken if the message region starts and no control region is active.

W - wait for control program to start.
N - ask operator for decision -- this is the default.
C - cancel message region automatically.

C=
: specifies the overlay supervisor option:

0 - allow OS/VS to load and delete the overlay supervisor for every overlay application program -- that is the default.

1 - load and retain a copy of the overlay supervisor when the message region is initialized.

D=

specifies the SPIE option:

0 - allow user SPIE, if any, to remain in effect while processing the application program call.

1 - negate the user's SPIE while processing the application program call. Negated SPIEs are reinstated before returning to the application program.

SPIE macros issued by the application program are only effective for program checks which occur within the message region.

E=

specifies the validity check option:

0 - no address validity checking will be made.
1 - validity check the addresses in the user's call list.

FF=

specifies the termination limit option. A decimal number between 1 and 99. The default is 1. When the number of application program abends reaches this limit, the message region is automatically terminated. This allows OS/VS to print the accumulated SYSOUT data sets.

GGG=

specifies the number of 1K blocks of subpool 253 to be reserved to hold a copy of the user's PCBs. This parameter must be a three-digit number (for example, 001). If this value is not specified, the system reserves an area which can hold the PCBs for any application program whose PSB is in IMSVS.ACBLIB. A U242 abend occurs if the application program PSB is not in IMSVS.ACBLIB (DOPT specified in APPLCTN macro) and is larger than any PSB in IMSVS.ACBLIB.

The output from the ACB generation utility program DFSUACB0 specifies application program PCB sizes.

HH=

specifies the 2-character suffix of the IMSVS.PROCLIB member that specifies preloaded program modules. If omitted, no modules are preloaded. See the IMS/VS Installation Guide for details.

I=

STIMER option:

0=none
1=no DL/I
2=with DL/I (default)

Member Name IMSPLI

Same assumptions as IMSCOBOL.

```
//          PROC     MBR=,PAGES=50,SOUT=A
//C         EXEC     PGM=IEMAA,REGION=114K,
// PARM='XREF,ATR,LOAD,NODECK,NOMACRO,,OPT=1'
//SYSUT1    DD       UNIT=SYSDA,SPACE=(1024,(60,60),RLSE,,ROUND),
// DCB=BLKSIZE=1024,DISP=(,DELETE)
//SYSUT3    DD       UNIT=SYSDA,SPACE=(1024,(60,60),RLSE,,ROUND),
// DCB=BLKSIZE=1024,DISP=(,DELETE)
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=125,BLKSIZE=629,RECFM=VBA),
// SPACE=(605,(&PAGES.0,&PAGES),RLSE)
//SYSLIN    DD       UNIT=SYSDA,SPACE=(80,(250,80),RLSE),DCB=BLKSIZE=80,
// DISP=(,PASS)
//L         EXEC     PGM=DFSILNK0,PARM='XREF,LIST,LET',COND=(4,LT,C),
// REGION=120K
//STEPLIB   DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIB    DD       DSN=SYS1.PL1LIB,DISP=SHR
//RESLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD       DSN=*.C.SYSLIN,DISP=(OLD,DELETE)
//          DD       DSN=IMSVS.PROCLIB(PLITDLI),DISP=SHR
//          DD       DDNAME=SYSIN
//SYSLMOD   DD       DSN=IMSVS.PGMLIB(&MBR),DISP=SHR
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=FBA,BLKSIZE=605),
// SPACE=(605,(&PAGES.0,&PAGES),RLSE)
//SYSUT1    DD       UNIT=SYSDA,DISP=(,DELETE),SPACE=(CYL,(5,1),RLSE)
```

Member Name IMSPLIGO

Same assumptions as IMSCOBGO, except an execution time of 5 minutes is specified.

```
//           PROC     MBR=,PAGES=50,
//           SOUT=A,PSB=,SPIE=0,TEST=0,EXCPVR=0,
//           RST=0,PRLD=,SRCH=0,CKPTID=,BUF=1000
//C         EXEC     PGM=IEMAA,REGION=114K,
// PARM='XREF,ATR,LOAD,NODECK,NOMACRO,,OPT=1'
//SYSUT1    DD       UNIT=SYSDA,SPACE=(1024,(60,60),RLSE,,ROUND),
// DCB=BLKSIZE=1024,DISP=(,DELETE)
//SYSUT3    DD       UNIT=SYSDA,SPACE=(1024,(60,60),RLSE,,ROUND),
// DCB=BLKSIZE=1024,DISP=(,DELETE)
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=125,BLKSIZE=629,RECFM=VBA),
// SPACE=(605,(&PAGES.0,&PAGES),RLSE)
//SYSLIN    DD       UNIT=SYSDA,SPACE=(80,(250,80),RLSE),DCB=BLKSIZE=80,
// DISP=(,PASS)
//L         EXEC     PGM=DFSILNK0,PARM='XREF,LIST,LET',COND=(4,LT,C),
// REGION=120K
//STEPLIB   DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIB    DD       DSN=SYS1.PL1LIB,DISP=SHR
//RESLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN    DD       DSN=*.C.SYSLIN,DISP=(OLD,DELETE)
//          DD       DSN=IMSVS.PROCLIB(PLITDLI),DISP=SHR
//          DD       DDNAME=SYSIN
//SYSLMOD   DD       DSN=IMSVS.PGMLIB(&MBR),DISP=SHR
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=FBA,BLKSIZE=605),
// SPACE=(605,(&PAGES.0,&PAGES),RLSE)
//SYSUT1    DD       UNIT=SYSDA,DISP=(,DELETE),SPACE=(CYL,(5,1),RLSE)
//G         EXEC     PGM=DFSRRC00,REGION=150K,TIME=5,COND=(4,LT),
// PARM='DLI,&MBR,&PSB,&BUF,&SPIE&TEST&EXCPVR,&RST,&PRLD,&SRCH,&CKPTID'
//STEPLIB   DD       DSN=IMSVS.RESLIB,DISP=SHR
//          DD       DSN=IMSVS.PGMLIB,DISP=SHR
//IMS       DD       DSN=IMSVS.PSBLIB,DISP=SHR
//          DD       DSN=IMSVS.DBDLIB,DISP=SHR
//PROCLIB   DD       DSN=IMSVS.PROCLIB,DISP=SHR
//IEFRDER   DD       DSN=IMSLOG,DISP=(,KEEP),VOL=(,,,99),
// UNIT=(2400,,DEFER),
// DCB=(RECFM=VBS,BLKSIZE=1408,LRECL=1400,BUFNO=1)
//IEFRDER2 DD        DSN=IMSLOG2,DISP=(,KEEP),VOL=(,,,99),[1]
// UNIT=2400,,DEFER,SEP=IEFRDER),
// DCB=(RECFM=VBS,BLKSIZE=1408,LRECL=1400,BUFNO=1)
//SYSPRINT DD        SYSOUT=&SOUT,DCB=(LRECL=121,BLKSIZE=605,RECFM=FBA),
// SPACE=(605,(500,500),RLSE,,ROUND)
//SYSUDUMP DD        SYSOUT=&SOUT,DCB=(LRECL=121,BLKSIZE=605,RECFM=FBA),
// SPACE=(605,(500,500),RLSE,,ROUND)
```

[1] This statement is included only when dual system log data sets are used.

Member Name IMSRDR

The IMSRDR procedure varies, based on the version of OS/VS that is used.

For OS/VS1:

```
//           PROC     MBR=IMSMSG
//IEFPROC    EXEC     PGM=IFFVMA,           READER FIRST LOAD
// PARM='00100300005210E00011A00'  DEFAULT OPTIONS
//*                   BPPTTTTSSCCCRLAAAAEFHXX   PARM FIELD
//*                   B               PROGRAMMER NAME AND ACCOUNT NUMBER NOT NEEDED
//*                   PP              PRIORITY=01
//*                   TTTTSS          JOB STEP INTERVAL=30 MINUTES
//*                   CCC             JOB STEP DEFAULT REGION=52K
//*                   R               DISPLAY AND EXECUTE COMMANDS=1
//*                   L               BYPASS LABEL=0
//*                   AAAA            COMMAND AUTHORITY FOR MCS=E000
//*                                   -  ALL COMMANDS MUST BE AUTHORIZED
//*                   E               JCL MESSAGE LEVEL DEFAULT=1 -ALL MESSAGES
//*                   F               ALLOC/TERM MESSAGE LEVEL DEFAULT=1 -ALL MESSAGES
//*                   H               DEFAULT MSGCLASS=A
//IEFRDER    DD       DSN=IMSVS.PROCLIB(&MBR),DISP=SHR,DCB=BUFNO=1
//IEFPDSI    DD       DSN=IMSVS.PROCLIB,DISP=SHR
//           DD       DSN=SYS1.PROCLIB,DISP=SHR
```

For OS/VS2:

```
//           PROC     MBR=IMSMSG,CLASS=A
//IEFPROC    EXEC     PGM=IEBEDIT
//SYSPRINT DD         SYSOUT=&CLASS
//SYSUT1     DD       DDNAME=IEFRDER
//SYSUT2     DD       SYSOUT=(&CLASS,INTRDR),DCB=BLKSIZE=80
//SYSIN      DD       DUMMY
//IEFRDER    DD       DSN=IMSVS.PROCLIB(&MBR),DISP=SHR
```

Member Name IMSWTnnn

IMSWTnnn member(s) job class and message class are determined by the MAXREGN keyword specified on the IMSCTRL macro statement during system definition.

```
//SPRTn      JOB      1,IMS,CLASS=A,MSGCLASS=3,MSGLEVEL=1
//PRINT      EXEC     PGM=DFSUPRT0,REGION=30K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD         SYSOUT=3,DCB=BLKSIZE=1410
//SYSUDUMP DD         SYSOUT=3
//SPOOLn     DD       DSN=IMSVS.SYSOn,DISP=SHR
```

Member Name IQFFC

Assumes:

* The DMGSI1 program (Stage 1, Part 1) provides JCL to allocate data set groups at initial creation time.

```
//IQFFC      PROC
//FC1        EXEC     PGM=DFSRRC00,PARM='DLI,DMGFC1,DMGFC1',REGION=300K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//IMS        DD       DSNAME=IMSVS.PSBLIB,DISP=SHR
//           DD       DSNAME=IMSVS.DBDLIB,DISP=SHR
//SYSPRINT   DD       SYSOUT=A
//SYSOUT     DD       SYSOUT=A
//UTPRINT    DD       SYSOUT=A
//UTDBD      DD   UNIT=SYSDA,DSN=UTDBD,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
//UTSPL      DD   UNIT=SYSDA,DSN=UTSPL,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
//SORTLIB    DD       DSN=SYS1.SORTLIB,DISP=SHR
//SSYNIN     DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=1040,LRECL=52,DSORG=PS,RECFM=FB),
// DSN=SSYNIN
//SSYNOUT    DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=1040,LRECL=52,DSORG=PS,RECFM=FB),
// DSN=SSYNOUT
//SPCBIN     DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=880,LRECL=44,DSORG=PS,RECFM=FB),
// DSN=SPCBIN
//SPCBOUT    DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=880,LRECL=44,DSORG=PS,RECFM=FB),
// DSN=SPCBOUT
//SWRKIN     DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=1920,LRECL=96,DSORG=PS,RECFM=FB),
// DSN=SWRKIN
//SWRKOUT    DD       DISP=(NEW,DELETE),SPACE=(CYL,(1,1)),UNIT=SYSDA,
// DCB=(BLKSIZE=1920,LRECL=96,DSORG=PS,RECFM=FB),
// DSN=SWRKOUT
//SPCBWK01 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SPCBWK02 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SPCBWK03 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SPCBWK04 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SPCBWK05 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SPCBWK06 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK01 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK02 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK03 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK04 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK05 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SSYNWK06 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK01 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK02 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK03 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK04 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK05 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SWRKWK06 DD         UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
```

Member Name IQFIU

Assumes:

Prior to executing the IQF Utility during IQF and IMS/VS installation, the user modifies this procedure to tailor it to his IQF indexing requirements.

The modification required is:

* Add DD statements to the IU1 step for the user's IMS/VS data bases to be indexed.

```
//IQFIU     PROC    SOUT=A,IMSREG=DLI,DISPS=OLD
//IU1       EXEC PGM=DFSRRC00,PARM='&IMSREG,DMGIU1,DMGIU1',REGION=300K
//STEPLIB   DD      DSN=IMSVS.RESLIB,DISP=SHR
//IMS       DD      DSN=*.QUS2X1.L.SYSLMOD,DISP=(OLD,PASS)¹
//          DD      DSN=IMSVS.PSBLIB,DISP=SHR
//          DD      DSN=IMSVS.DBDLIB,DISP=SHR
//QFF       DD      DSN=IQFIFFDB,DISP=SHR
//QFFOVF    DD      DSN=IQFOFFDB,DISP=SHR
//QXS1      DD      DSN=IQFXS1DB,DISP=&DISPS
//QXS1OV    DD      DSN=IQFXOVS1,DISP=&DISPS
//QXL1      DD      DSN=IQFXL1DB,DISP=&DISPS
//QXL1OV    DD      DSN=IQFXOVL1,DISP=&DISPS
//HOLDS     DD      UNIT=SYSDA,SPACE=(CYL,(4,1)),DISP=(,PASS)
//HOLDL     DD      UNIT=SYSDA,SPACE=(CYL,(4,1)),DISP=(,PASS)
//IEFRDER   DD      DUMMY
//SYSPRINT  DD      SYSOUT=&SOUT
//SYSOUT    DD      SYSOUT=&SOUT
//IU2   EXEC PGM=DFSRRC00,PARM='&IMSREG,DMGIU3,DMGIU1',REGION=300K,
// COND=(4,LT,IU1)
//STEPLIB   DD      DSN=IMSVS.RESLIB,DISP=SHR
//IMS       DD      DSN=*.QUS2X1.L.SYSLMOD,DISP=(OLD,PASS)¹
//          DD      DSN=IMSVS.PSBLIB,DISP=SHR
//          DD      DSN=IMSVS.DBDLIB,DISP=SHR
//QFF       DD      DSN=IQFIFFDB,DISP=SHR
//QFFOVF    DD      DSN=IQFOFFDB,DISP=SHR
//IEFRDER   DD      DUMMY
//SYSPRINT  DD      SYSOUT=&SOUT
//SYSOUT    DD      SYSOUT=&SOUT
//SORTLIB   DD      DSN=SYS1.SORTLIB,DISP=SHR
//SHRTIN    DD      DSN=*.IU1.HOLDS,DISP=(OLD,DELETE)
//SHRTOUT   DD      UNIT=SYSDA,SPACE=(CYL,(4,1)),DISP=(,PASS)
//SHRTWK01  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//SHRTWK02  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//SHRTWK03  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//LONGIN    DD      DSN=*.IU1.HOLDL,DISP=(OLD,DELETE)
//LONGOUT   DD      UNIT=SYSDA,SPACE=(CYL,(4,1)),DISP=(,PASS)
//LONGWK01  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//LONGWK02  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//LONGWK03  DD      UNIT=SYSDA,SPACE=(TRK,(10),,CONTIG)
//IU3   EXEC PGM=DFSRRC00,PARM='&IMSREG,DMGIU2,DMGIU1',REGION=300K,
// COND=((4,LT,IU1),(4,LT,IU2))
//STEPLIB   DD      DSN=IMSVS.RESLIB,DISP=SHR
//IMS       DD      DSN=*.QUS2X1.L.SYSLMOD,DISP=(OLD,DELETE)¹
//          DD      DSN=IMSVS.PSBLIB,DISP=SHR
//          DD      DSN=IMSVS.DBDLIB,DISP=SHR
//QFF       DD      DSN=IQFIFFDB,DISP=SHR
//QFFOVF    DD      DSN=IQFOFFDB,DISP=SHR
//QXS1      DD      DSN=IQFXS1DB,DISP=&DISPS
//QXS1OV    DD      DSN=IQFXOVS1,DISP=&DISPS
//QXL1      DD      DSN=IQFXL1DB,DISP=&DISPS
//QXL1OV    DD      DSN=IQFXOVL1,DISP=&DISPS
//HOLDS     DD      DSN=*.IU2.SHRTOUT,UNIT=SYSDA,DISP=(OLD,DELETE)
//HOLDL     DD      DSN=*.IU2.LONGOUT,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSPRINT  DD      SYSOUT=&SOUT
//SYSOUT    DD      SYSOUT=&SOUT
```

[1] The *.QUS2X1.L.SYSLMOD data set for the IMS DD statement refers back to the SYSLMOD card in the DMGIU1 PSBGEN step generated by DMGSI2.

Member Name IQFUT

Assumes:

- User supplies source data for SYSIN.
- SYSUT1 is a BSAM work data set.
- Output Class A is used for listing.
- Output Class B is used by DMGSI1 and DMGSI2 (Stage 1) to produce job steps in the Stage 2 OS/VS job stream.
- User defines IMS region type (batch or batch-message) in PARM field of EXEC statement for executing the procedure. (Not required at initial creation time.)

```
//           PROC     SOUT=A,SPCH=B,IMSREG=DLI
//SIA        EXEC     PGM=DMGSI1,REGION=300K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSUT1     DD       UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,(24,11))
//SYSPRINT DD         SYSOUT=&SOUT
//SYSPUNCH DD         SYSOUT=&SPCH
//SIB EXEC PGM=DFSRRC00,PARM='&IMSREG,DMGSI2,DMGSIB',REGION=200K,1
// COND=(0,LT)9
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//IMS        DD       DSN=IMSVS.PSBLIB,DISP=SHR
//           DD       DSN=IMSVS.DBDLIB,DISP=SHR
//SYSPRINT DD         SYSOUT=&SOUT
//SYSPUNCH DD         SYSOUT=&SPCH
//QFF        DD       DSN=IQFIFFDB,DISP=SHR
//QFFOVF     DD       DSN=IQFOFFDB,DISP=SHR
//SYSUT1     DD       DSN=*.SIA.SYSUT1,DISP=(OLD,DELETE)
```

1 The SIB step is bypassed when the IQFUT procedure is executed to create the System Data Base.

Member Name MFDBDUMP

```
//           PROC     SOUT=A
//DUMP       EXEC     PGM=DFSRRC00,PARM='DLI,DFSSAM08',REGION=130K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//           DD       DSN=IMSVS.PGMLIB,DISP=SHR
//IMS        DD       DSN=IMSVS.PSBLIB,DISP=SHR
//           DD       DSN=IMSVS.DBDLIB,DISP=SHR
//SYSUDUMP DD         SYSOUT=&SOUT
//DI21PART DD         DSN=IMSVS.DI21PART,DISP=SHR
//DI21PARO DD         DSN=IMSVS.DI21PARO,DISP=SHR
//OUTPUT     DD       SYSOUT=&SOUT
```

Member Name MFDBLOAD

```
//           PROC     SOUT=A
//LOAD       EXEC     PGM=DFSRRC00,PARM='DLI,DFSSAM01',REGION=130K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//           DD       DSN=IMSVS.PGMLIB,DISP=SHR
//IMS        DD       DSN=IMSVS.PSBLIB,DISP=SHR
//           DD       DSN=IMSVS.DBDLIB,DISP=SHR
//SYSUDUMP DD         SYSOUT=&SOUT
//DI21PART DD         DSN=IMSVS.DI21PART(PRIME),DISP=(,KEEP),DCB=DSORG=IS,
// SPACE=(CYL,3,,CONTIG),VOL=SER=&PSER,UNIT=&PUNIT
//DI21PARO DD    DSN=IMSVS.DI21PARO,DISP=(,KEEP),SPACE=(CYL,3,,CONTIG),
// VOL=SER=&OSER,UNIT=&OUNIT
//SYSOUT     DD       SYSOUT=&SOUT
//INPUT      DD       DSN=IMSVS.GENLIB(MFDFSYSN),DISP=SHR
```

Member Name PSBGEN

Detailed information on PSBGEN, and examples of the use of PSBGEN are in the IMS/VS Utilities Reference Manual.

```
//           PROC     MBR=TEMPNAME,SOUT=A
//C          EXEC     PGM=IFOX00,REGION=128K,PARM='OBJ,NODECK'
//SYSLIB     DD       DSN=IMSVS.MACLIB,DISP=SHR
//SYSGO      DD       UNIT=SYSDA,DISP=(,PASS),
// SPACE=(80,(100,100),RLSE),
// DCB=(BLKSIZE=400,RECFM=FB,LRECL=80)
//SYSPRINT DD         SYSOUT=&SOUT,DCB=BLKSIZE=1089,
// SPACE=(121,(300,300),RLSE,,ROUND)
//SYSUT1     DD       UNIT=SYSDA,DISP=(,DELETE),SPACE=(1700,(100,50))
//SYSUT2     DD       UNIT=SYSDA,DISP=(,DELETE),SPACE=(1700,(100,50))
//SYSUT3     DD       UNIT=(SYSDA,SEP=(SYSLIB,SYSUT1,SYSUT2)),
// SPACE=(1700,(100,50))
//L EXEC     PGM=DFSILNK0,PARM='XREF,LIST',COND=(0,LT,C),REGION=120K
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSLIN     DD       DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSPRINT DD         SYSOUT=&SOUT,DCB=(LRECL=121,RECFM=FBA,BLKSIZE=605),
// SPACE=(121,(100,100),RLSE)
//SYSLMOD    DD       DSN=IMSVS.PSBLIB(&MBR),DISP=SHR
//SYSUT1     DD       UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),DISP=(,DELETE),
// SPACE=(1024,(100,10),RLSE)
```

Member Name SECURITY

```
//           PROC     OPTN=UPDATE,IMS=',0',SOUT=A
//S          EXEC     PGM=DFSISMP0,PARM='&OPTN.&IMS.'
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD         SYSOUT=&SOUT,DCB=(RECFM=VBA,BLKSIZE=400,BUFL=404)
//SYSPUNCH DD         UNIT=SYSDA,SPACE=(80,(800,400),,,ROUND),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),DISP=(,PASS)
//SYSLIN     DD   UNIT=SYSDA,SPACE=(TRK,(1,1)),DCB=(RECFM=F,BLKSIZE=80),
// DISP=(,PASS)
//SYSUT1     DD       UNIT=SYSDA,SPACE=(100,(400,400),,,ROUND),
// DCB=(BLKSIZE=500,RECFM=FB)
//SYSUT2     DD       UNIT=(SYSDA,SEP=SYSUT1),SPACE=(100,(400,400),,,ROUND),
// DCB=*.S.SYSUT1
//SYSIN      DD       DSN=NO.SYSIN.DD.ASTERISK
//C EXEC     PGM=IFOX00,PARM='OBJ,NODECK',COND=(12,LT,S),REGION=128K
//SYSPRINT DD         SYSOUT=&SOUT,DCB=BLKSIZE=1089
//SYSGO      DD       UNIT=(SYSDA,SEP=SYSPRINT),DISP=(,PASS),
// DCB=*.S.SYSPUNCH,SPACE=(80,(400,400),,,ROUND)
//SYSUT1     DD       UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2     DD       UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT3     DD       UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(CYL,(5,1))
//SYSIN      DD       DSN=*.S.SYSPUNCH,DISP=(OLD,DELETE)
//L EXEC     PGM=DFSILNK0,PARM='LIST,NE,OL',REGION=110K,COND=(4,LT,S)
//STEPLIB    DD       DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD         SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605)
//SYSLMOD    DD       DSN=IMSVS.RESLIB,DISP=SHR
//INPUT      DD       DSN=*.C.SYSGO,DISP=(OLD,DELETE)
//SYSUT1     DD       UNIT=(SYSDA,SEP=INPUT),SPACE=(CYL,(5,1))
//SYSLIN     DD       DSN=*.S.SYSLIN,DISP=(OLD,DELETE)
```

DL/I INTERFACES

Member Name CBLTDLI

```
LIBRARY RESLIB (CBLTDLI) DL/I INTERFACE
ENTRY DLITCBL
```

Member Name PLITDLI

```
LIBRARY RESLIB (PLITDLI) DL/I LANGUAGE INTERFACE
ENTRY IHESAPD
```

CHAPTER 2. SYSTEM MAINTENANCE/TUNING FACILITIES

DL/I DATA BASE BUFFERING FACILITIES

The IMS/VS DL/I buffering services are controlled by three pools of control blocks and buffers; the ISAM/OSAM buffer pool, the VSAM shared resource pool, and the DL/I buffer handler pool. This section describes the structure, content, and use of these pools by DL/I.

The DL/I buffering services are the interface between the DL/I action modules (for example, Retrieve, Delete, Insert) and the data management access methods (VSAM, ISAM, and OSAM). Whenever an action module needs to inspect or change data in a data base, buffering services is called to perform whatever physical reading or writing is required. A separate pool of buffers is allocated for each type of data base; VSAM and ISAM/OSAM. Data bases that use the VSAM access method share the use of buffers in the VSAM shared resource pool. Data bases that use the ISAM or OSAM access methods share the use of buffers in the ISAM/OSAM buffer pool.

Implementing the concept of a buffer pool allows blocks of data to remain in main storage as long as possible to avoid secondary storage reads and writes. Data in a buffer pool can be accessed and updated without causing I/O as long as there is no need to reuse the buffer space the data occupies. A use chain determines the order in which the buffers are used. Empty buffers are placed at the bottom of the use chain and are always available for reuse. As buffers are accessed they are placed at the top of the use chain. When a retrieve request occurs, the buffer pool is searched using the use chain, to determine if the requested data is already in main storage. If the data is not found, the least recently used buffer (bottom of the use chain) is selected, the old data is written out if it has been changed, and the requested data is read into the selected buffer.

If an I/O error occurs while attempting to write a buffer of data, the buffer is marked as a permanent write error buffer and retained in the pool. No error indication is returned on the request that encountered the error, but an I/O error message is written to the operator, an error log record is recorded on the IMS/VS log data set, and the data base is stopped to prevent scheduling of additional transactions that use the data base. When all applications that use the data base have completed processing, the data buffer is marked as empty and made available for reuse. This error philosophy allows the application program to complete even though an I/O error has occurred. Whenever an I/O error occurs, the IMS/VS Data Base Recovery Utility program should be used to re-create the data base that was damaged.

IMS/VS maintains statistics on buffer pool utilization and access method requests. These statistics are of value for determining the optimum buffer pool definition for a given application. The DL/I statistics call (STAT) can be used to obtain these statistics in an application program (see _IMS/VS Application Programming Reference Manual_ for a description of the STAT call).

ISAM/OSAM BUFFER POOL

The ISAM/OSAM buffer pool is used to buffer data for data bases that use the ISAM or OSAM access methods. It is made up of a pool prefix (BFPL), which contains pool statistics and the use chain top and bottom pointers, and one or more variable length buffers. Each buffer is preceded by a buffer prefix (BFFR) which describes the size of the buffer, its status, and position on the use chain.

Buffer management and selection is controlled primarily by the use chain which logically orders the buffers. When space is needed in the pool to read in additional data or create a new block, the buffer at the bottom of the use chain is the prime candidate. If this buffer is not large enough to satisfy the request, then several buffers are selected and the remaining buffers are compressed to free enough contiguous space to accommodate the new buffer. The least recently used buffers, which when combined will satisfy the space requirement, are selected to be eliminated from the pool. If data has been changed in any of the selected buffers, they must be written back to external storage before they can be eliminated.

A buffer cannot be moved while it is busy with I/O. Therefore, the compression process may have to wait for I/O to complete before moving a buffer. The free space created by compressing the buffers is used to create a new buffer. If the free space is larger than the buffer space requested, the difference is compared to minimum buffer size, and an additional new buffer is created if the difference is greater than the minimum for one. Otherwise, the entire free space is used to create a single buffer to satisfy the request.

Fixed Length Buffers

In an environment where the block sizes of all DL/I data sets are approximately equal, it may be desirable to minimize the compression activity of the buffer handler. This can be accomplished by using the BFPLBFSZ parameter of the OPTIONS statement to specify the minimum buffer size to be allowed in the pool. See "Defining the IMS/VS VSAM Buffer Pool" in the *IMS/VS Installation Guide* for an explanation of the OPTIONS statement and the buffer pool initialization data set. Specifying a minimum buffer size of x causes all buffers in the pool to be either x or a multiple of x bytes long.

Note: The user is cautioned that the specification of a minimum buffer size other than the default can degrade performance if the value is inappropriate or if the environment does not lend itself to fixed size buffers.

VSAM SHARED RESOURCE POOL

The VSAM shared resource pool is used to buffer data for data bases that use the VSAM access method. It is constructed by VSAM based on parameters provided by the VSAM BLDVRP macro instruction issued by IMS/VS initialization. It contains buffers to be used for VSAM data sets (both index and data components) and the input/output-related control blocks necessary to perform VSAM requests. The buffers are combined in subpools. All buffers within a subpool are of equal length.

Buffer management and selection are controlled primarily by the use chain which logically orders the VSAM BUFC blocks. Since buffers within a subpool are fixed in length, no compression or movement of buffers is necessary.

VSAM Background Write

When the VSAM Buffer Manager needs space in a subpool to read a record or create a new block, it selects the buffer at the bottom of the use chain to satisfy the requirement. If the buffer selected contains data that has been modified, it must be written before the space can be used for the requested function. The purpose of Background Write (BGWRT) is to reduce the number of times the buffer manager selects a modified buffer.

Each time the buffer manager obtains space in a subpool it examines the next higher buffer on the use chain. If the contents of that buffer are modified, a return code is passed in the RPL to IMS/VS. This return code tells IMS/VS buffering services to activate (POST) the Background Write PST, and through normal IMS/VS scheduling BGWRT is dispatched. Background Write issues the VSAM WRTBFR TYPE=LRU macro which causes a percentage of the buffers at the bottom of the use chain in each subpool to be written out (if modified). In this manner, the data in the subpools which has not been used recently is written out before the buffer manager requires the space it occupies. This does not prevent reuse of data in the buffers. If a subsequent request requires the data before the buffer manager needs that space in the subpool, the data is used to satisfy the request, and the buffer is put on the top of the use chain.

The use of Background Write is determined by the OPTIONS statement in the IMS/VS VSAM buffer pool parameter data set (DFSVSAMP). See "Defining the IMS/VS VSAM Buffer Pool" in the _IMS/VS Installation Guide_ for an explanation of the OPTIONS statement.

DL/I BUFFER HANDLER POOL

The buffer handler pool is the focal point for recording buffering services activity. The pool prefix (BFSP) contains pointers to the other elements of the pool, indicator flags, and some statistics. If VSAM data bases are used, a subpool statistics block (BFUS) exists for each VSAM buffer subpool defined. The subpool statistics block contains statistics on buffering services and VSAM request activity relevant to the associated subpool.

A chain of RPL blocks (RPLI) is present if VSAM data bases are used. An RPL block is associated with each request made to VSAM. There is one RPL block for each PST and one for each sequential mode data base. An RPL block contains an error message area, an area to record RBA shift information, and a VSAM Request Parameter List (RPL) control block.

The last element of the buffer handler pool is the DL/I trace table. The trace table is a revolving trace of DL/I activity. It records calls to buffering services, open and close of data bases, and Program Isolation enqueues and dequeues.

The exact format of the control blocks and pools discussed in this section is described in the _IMS/VS Program Logic Manual, Volume 1 of 3_.

LOG TAPE WRITE-AHEAD

On systems in which power failure may cause main storage contents to be lost, the IMS/VS System Log Terminator utility cannot recover the data in the log buffers that were in main storage but had not been written at the time of failure. The log tape write-ahead option is provided to ensure that a data base log record for a data change is written to the log device before the changed data is written to the data base. This ensures that any change made to a data base is physically recorded on the log tape before the data base is changed.

Data bases in a batch (DLI or DBB) region which use one PCB only are accessed using QISAM instead of the normal BISAM. Since IMS/VS cannot predict when QISAM buffers are written, the log tape write-ahead option does not apply to these data bases. If log write-ahead is desired on a QISAM mode data base, an additional PCB for the data base may be added to the PSB to force BISAM mode.

Use of this option degrades system performance. The impact is system and application dependent. Some variables affecting the impact are log buffer size, number of log buffers, data base buffer pool size, and frequency of sync points.

The log tape write-ahead option is activated with the OPTIONS statement in the buffer pool initialization data set; see "Defining the IMS/VS VSAM Buffer Pool" in the _IMS/VS Installation Guide_ for a description of the OPTIONS statement.

IMS/VS COMMAND LANGUAGE MODIFICATION FACILITY

This section explains the modification of the command keyword table. Refer to the "IMS/VS Commands" chapter in the _IMS/VS Operator's Reference Manual_ for a complete explanation of the IMS/VS command language.

COMMAND KEYWORD TABLE

DFSCKWD0, a member of IMSVS.DCSOURCE, should be printed to obtain a listing of the command keyword table. It contains the IMS/VS keywords and synonyms described in the _IMS/VS Operator's Reference Manual_.

There can be several reasons for altering the keyword table. For example, an installation may want to tailor the keywords and synonyms to satisfy unique requirements. Or, a new keyword in a new IMS/VS release could conflict with a name already assigned by the installation to an LTERM or TRANSACTION.

CHANGING THE TABLE

Two of the macro statements that appear in the table, KEYWD and SYN, can be replaced to modify the keywords and synonyms. One way of modifying the table is:

1. Punch DFSCKWD0 into cards
2. Prepare new KEYWD and SYN macro statements
3. Replace the KEYWD and SYN statements to be changed
4. Reassemble the module
5. Relink the reassembled module into RESLIB
6. Relink the IMS/VS nucleus

KEYWD macro statements must be substituted one-for-one in the table. No new KEYWD macro statements can be added.

KEYWD Macro

```
KEYWD   keyword,LAST=NO|YES
```

Where 'keyword' is the new keyword desired. LAST=NO is the default and need not be supplied. LAST=YES must be specified if it is the last macro call in the module. A keyword cannot exceed 12 characters in length.

SYN Macro

```
        SYN     synonym,LAST=YES|NO
```

Where 'synonym' is the desired synonym. LAST=NO is the default and need not be specified. LAST=YES must be coded if this is the last macro call in the assembly. Synonyms cannot exceed 12 characters in length; they must be defined under the keyword to which they apply.

ERROR MESSAGES

Any error in a macro statement will terminate keyword table assembly and cause an error message. The remaining macro statments will be error checked but nothing will be generated. All macro assembly errors are severity code 16 errors.

KRYBL001 - SEQUENCE ERROR. XXX CANNOT FOLLOW IKEY
A macro was called which cannot immediately follow an IKEY macro call. XXX is either IKEY or SYN. IKEY calls cannot be modified.

KYTBL002 - XXX CALLED WITHOUT ANY PARAMETER
A macro was called without any parameter. XXX is either IKEY, KEYWD or SYN.

KYTBL003 - XXX IS NOT A VALID INTERNAL KEYWORD
The parameter specified on an IKEY call (XXX) is not known to the system. IKEY calls cannot be modified.

KYTBL004 - KEYWORD TABLE ASSEMBLY TERMINATED
This message appears as a comment after the first error message in a keyword table assembly. All following macro calls will only perform error checking. No code will be generated.

KYTBL005 - SEQUENCE ERROR. KEYWD MUST FOLLOW AN IKEY CALL
A KEYWD macro was called which does not immediately follow an IKEY call.

KYTBL006 - LENGTH ERROR. XXX TOO LONG
The parameter specified on a KEYWD or SYN macro is more than 12 characters in length.

KYTBL007 - INTERNAL KEYWORD 'XXX' HAS NOT BEEN USED
LAST=YES was specified on either a KEYWD or SYN macro call but not all internal keywords known to the system have been generated. IKEY calls cannot be modified. LAST=YES must appear only on the last macro call in the assembly.

KYTBL008 - XXX CANNOT BE SPECIFIED AGAIN
Internal keyword 'XXX' has already been used. IKEY macro calls cannot be modified.

Note: Message DFS058 COMMAND COMPLETED EXCEPT xxx y z... uses the keyword table to replace 'xxx' with the keyword associated with the command that caused the message. Therefore, keywords defined by KEYWD macro calls will appear in this message. Other messages, however, are pre-built, and keywords which may have changed will still appear in these.

CHAPTER 3. DL/I USER EXIT ROUTINES

This chapter describes the exits that IMS/VS provides to allow the use of internally generated data, or to allow users to incorporate processing extensions of their own. It provides some rules for writing exit routines and explains user generation of randomizing modules for use with HDAM file organizations. It also discusses user generation of segment edit/compression routines, user secondary index maintenance routines, and the IMS/VS log tape record format.

The IMSVS.DBSOURCE library contains the source for all sample and supplied exit routines described in this chapter, and should be referred to for the latest versions.

WRITING DL/I EXIT ROUTINES

Routines described in this chapter or written by users must be reenterable for the following reasons:

- IMS/VS loads the routine each time a request for it is encountered.
- The same edit/compression routine is used concurrently for different segment types (even if they are in the same data base).
- The same index maintenance routine is used concurrently for different segment types.
- The same randomizing routine is used concurrently for different data bases.

ACCESSING MAIN STORAGE

In the MVS online environment with parallel DL/I, all DL/I programs, control blocks, and work areas must be globally addressable. This includes user exits. To ensure this, IMS/VS manages the common service area (CSA) with the IMS/VS IMODULE function. All user written exit routines that load modules and/or access main storage in the MVS online environment must do so by using the IMS/VS IMODULE function.

ISWITCH Macro

All calls for CSA to IMDOULE must be issued from the IMS/VS control region. The ISWITCH macro switches IMS/VS execution from a dependent region to the control region. The exit routine that issues ISWITCH must be running under the IMS/VS dispatcher and must provide addressability to SCD and PST. The address of the SCD can be obtained from the PST field PSTSCDAD. An example of the use of the ISWITCH macro is:

```
MVI        PSTDECB,X'00'          Clear ECB.
ISWITCH    TO=CTL,ECB=PSTDECB
LTR        15,15                  Successful?
BNZ        NOSWT                  No, CTL region might be abending.
```

IMODULE Macro

The IMODULE macro provides functions equivalent to the OS LOAD, GETMAIN, FREEMAIN, and DELETE macros. For CSA, subpool 231 should be used. If the IMODULE macro is issued while IMS/VS is executing in a dependent region, subpool 251 (local space) is used in place of 231. IMODULE is a Type 4 SVC and so should be used only when necessary.

To Load a Module into CSA

To use IMODULE to load a module into CSA, the user exit routine must have issued ISWITCH and must provide addressability to SCD. The address of the SCD can be obtained from the PST field PSTSCDAD. An example of this use of IMODULE is:

```
    IMODULE    LOAD,EPLOC=NAME,SP=231
    LTR        15,15                             Okay?
    BNZ        LOADFAIL                          No.

  * Reg. 1 contains EP                        }
                                              } Load list.
  NAME         DC  CL8'module name'           }
```

Note: If a previously LOADed or GETMAINed module is not to be used, add the parameter USE=NO to the IMODULE macro.

To Get Storage from CSA

To use IMODULE to get storage from CSA, the user exit routine must have issued ISWITCH and must provide addressability to SCD. The address of the SCD can be obtained from the PST field PSTSCDAD. An example of this use of IMODULE is:

```
    IMODULE    GETMAIN,EPLOC=NAME,LV=(1),SP=231
    LTR        15,15                                     Okay?
    BNZ        GETFAILD                                  No.

  * Reg. 1 contains GETMAINed block address.  }
                                              }          Load list.
  NAME         DC  CL8'module name'           }
```

Note: LV= specifies the register containing the length for the GETMAIN.

To Delete a Module from CSA

To use IMODULE to delete a module from CSA, the user exit routine must have issued ISWITCH and must provide addressability to SCD. The address of the SCD can be obtained from the PST field PSTSCDAD. A module can be deleted either by name or by entry point. An example of each of these uses of IMODULE follows:

- By name

```
IMODULE     DELETE,EPLOC=NAME,SP=231
LTR         15,15                                 Okay?
BNZ         DELFAILD                              No.
```

- By entry point

```
IMODULE     DELETE,EPAD=(1),SP=231
LTR         15,15                                 Okay?
BNZ         DELFAILD                              No.
```

Note: EPAD= specifies the register containing the register 1 value returned by a previous IMODULE LOAD or IMODULE GETMAIN.

SEGMENT EDIT/COMPRESSION EXIT

The IMS/VS Edit/Compression Exit provides a facility for invoking user-written routines to edit a segment during its movement between the data base buffer pool and the input/output area of the application program. Design and implementation of this facility are also discussed in the *IMS/VS System/Application Design Guide* and the *IMS/VS Utilities Reference Manual*.

The exit provides the facility to encode and decode data for security purposes, invoking routines privately generated and controlled by the user.

Other ways to use the exit are for data validation purposes and for data formatting. One example of data formatting is compressing segments to save direct access space, and then to expand them to their original size when they are brought back to main storage for processing.

User installations that invoke the Edit/Compression Exit are given access to the IMS/VS buffer pool. The Edit/Compression routines should be implemented by those having overall systems and/or data base responsibility for an installation. They should be transparent to the application programs that access those data bases.

The following text provides a general description and overview, and then a specific discussion of the following:

- Types of segments that can be edited or compressed
- Types of compression that can be applied
- SEGM control statement requirements for DBD-generation, including a description of the Segment Edit/Compression Table appended to the DBD control block
- Interfaces presented by affected DL/I modules to the user edit/compression routine

These discussions are followed by detailed specifications of the following:

- Parameters passed by DL/I to the user routine
- Entry codes presented to the user routine
- Conversion of existing data bases

The section concludes with a discussion of performance considerations.

GENERAL DESCRIPTION AND OVERVIEW

The user edit/compression routine moves the segment, in either fixed- or variable-length format, from the source address to the destination address, performing the edit or the compression/expansion during the move operation. On a retrieve operation, the IMS/VS buffer pool is the source; on load, insert, or replace operations the application program I/O area is the source. For all operations, the destination address is an SWA (segment work area). This SWA is described in greater detail later in this section, and also in the discussions on the Variable Length Segment feature in the IMS/VS System/Application Design Guide and the IMS/VS Application Programming Reference Manual.

As a segment is requested by the user, its location in the buffer pool is obtained. If an edit/compression routine has been specified, the address of the data portion of the segment and the start of the SWA are supplied, and the routine is given control.

The edit/compression routine is responsible for moving the data from the buffer pool to the SWA, with the proper editing or expansion, and appropriate update to the segment length field. If no edit/compression routine is specified, this intermediate operation is not required.

For insert or replace operations, data is moved from the user work area to the SWA by the user edit/compression routine, then to the buffer pool by IMS/VS. These actions are summarized in Figure 3-1. A more detailed description is provided later in this section.

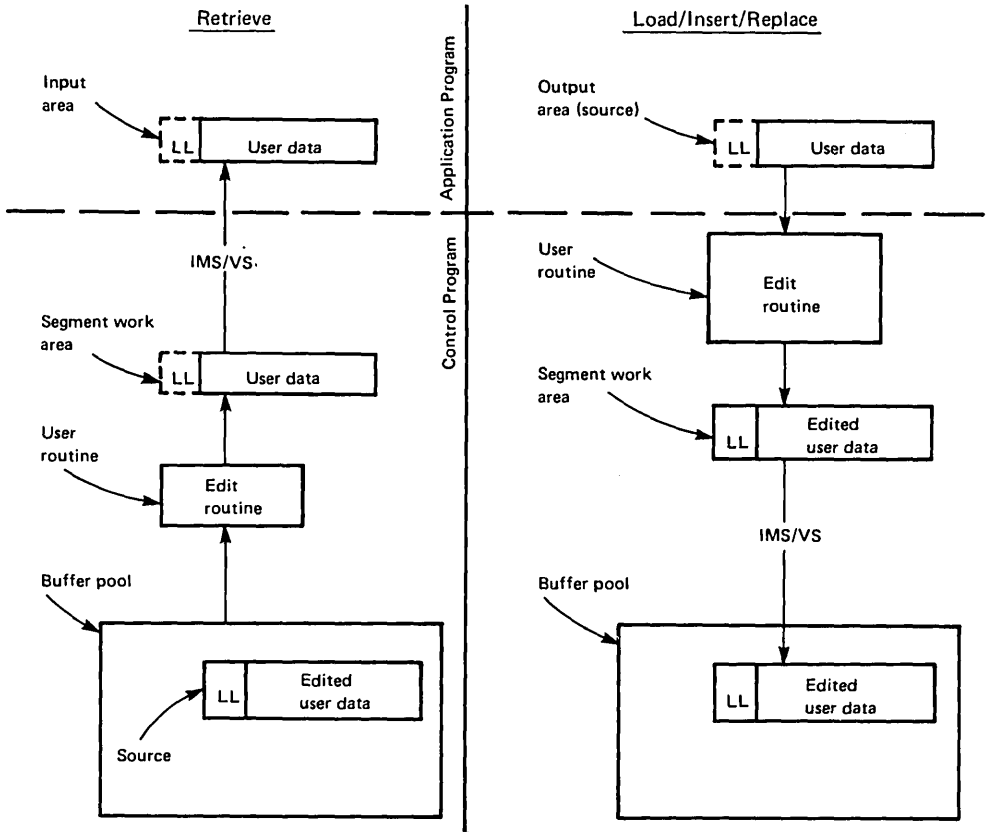


Figure 3-1. Segment Edit/Compression

Although the segments can be defined as fixed or variable length to the application program, the segments to be processed by the edit/compression routine must be variable length in the data base. The data length is contained in a field in the first two bytes of the segment. If the segment is defined as fixed-length to the application program, the length bytes must be stripped off by the edit/compression routine before the segment is presented to the application program. In addition, if the segment was compressed, it must be expanded by the edit routine to the fixed length expected by the application program. In reverse, if the application program presents a fixed length segment, the edit/compression routine must append the length bytes prior to the segment being written to the data base. If the edit/compression routine compresses the segment, the length field must be updated to reflect the correct length.

User Capabilities

The facility provided by DL/I permits the user-provided routine to do the following:

- Edit or compress both fixed- and variable-length segments.
- Accomplish either data edit/compression or key edit/compression.
- Apply the same routine to multiple segment types within the same or different data bases.

The logic for data encoding/decoding, or for other desired editing or formatting can be based on information contained within the user-written routine itself. It also can be based on information from an external source, such as data provided in the DBD block, or tables examined at execution time.

User Constraints

General constraints that apply to using the IMS/VS edit/compression facility are:

- Any segment specified as subject to editing or compression must reside in a VSAM data set.
- All editing or compression of segments takes place as the segments are described in a physical data base only. See "Types of Compression" later in this chapter for further specific restrictions.
- The user routine must reside in IMSVS.RESLIB, SYS1.LINKLIB, or any properly defined private library. When the routine is link-edited to one of these libraries, the user must specify one routine entry point.
- If the user routine is designed to edit or compress more than one segment type, in one or more physical data bases, the routine must be coded and link-edited as reenterable.
- Adequate storage for the edit/compression routine must be provided for both batch and on-line systems.
- Since this routine becomes a part of the IMS/VS control or batch region, any abnormal termination on its part terminates the entire IMS/VS region.
- The user routine cannot use the operating system macros LOAD, GETMAIN, SPIE, or STAE.

User Procedures

To take advantage of the IMS/VS edit/compression exit, the user must do two things:

- Expand the DBD control statement SEGM.
- Provide an edit/compression routine.

Details on the necessary procedures in each of these areas, and on the manner in which DL/I interfaces to the user routine follow.

TYPES OF SEGMENTS

Two types of segments can be presented to the edit/compression routine: fixed length segments, whose data length is static and is reflected in control blocks; and variable length segments, whose data length is contained within a field in the first two bytes of the segment itself. While a routine dealing with a single-segment type normally need not concern itself with the differences, a more general purpose module involved with multiple segment types can obtain sufficient information to differentiate between them. This is done by examining data provided in the segment compression control section.

TYPES OF EDIT/COMPRESSION

Two types of segment manipulation are possible through the DL/I edit/compression facility.

- Data compression -- movement or compression of data within a segment, in a manner that does not alter the content or position of the key field. Typically, this involves compression of data from the end of the key field to the end of the segment. Note that when a fixed length format segment is compressed, a two-byte size field must be added to the beginning of the data portion of the segment. This is done by the user data compression routine used by IMS/VS to determine secondary storage requirements. This is the only time that the location of the fields can be altered. The segment size field of a variable length segment cannot be compressed.

- Key compression -- movement or compression of any data within a segment, in a manner that can change the relative position, value, or length of the sequence field as well as any other fields.

Segments in a physical data base, except those types listed below, can be specified during DBDGEN as being compressible, with either the KEY or DATA option.

- Any segment which is defined as a logical child cannot be specified.

- Segments residing in an INDEX data base cannot be specified.

- Segments defined as root segments of a HISAM data base can be specified for DATA compression only.

Although the contents of the sequence field, or the data, can be modified by the edit/compression routine, the segment's position in the data base is determined by the original sequence field value. An example may help to explain this. If the defined sequence of a particular segment type is based on last names; and the data base contains segments for people named SMITH, JONES, and BROWN; the segments are maintained in alphabetical sequence -- BROWN, JONES, SMITH. Assume that an edit routine encodes these names as follows:

```
BROWN-------->29665

JONES-------->16552

SMITH-------->24938
```

The encoded value is placed in the sequence field. The segments are maintained in the original sequence (BROWN, JONES, SMITH), rather than in the numerical sequence implied by the encoded values (16552, 24938, 29665). The records are maintained in the originally defined sequence so that a GET NEXT request issued by the application program retrieves the correct segment.

To use the edit/compression facility, the user must extend the SEGM control statement in the following manner:

```
SEGM    NAME=seg-name
        [, COMPRTN=(routine-name [,{DATA} [,INIT]])]
                                   {KEY }
```

COMPRTN=
specifies that you want the segment edit/compression option. This operand must not be specified if the SOURCE operand is used. The COMPRTN operand is invalid in the DBDGEN operation for INDEX, and for simple HISAM DBDs. It must not change the sequence field offset for HISAM root segments. Segments specifying the COMPRTN parameter must reside in a VSAM data set.

routine-name
specifies the name of the user-supplied routine used to edit or compress this segment. This name must be a one- to eight-character alphameric value. It cannot be the same as any other name in IMSVS.RESLIB.

DATA
specifies that the indicated routine will edit or compress data fields only. Sequence fields are not modified; nor will data fields that change the position of the sequence field, in respect to the start of the segment, be modified. DATA is the default when an edit/compression routine is named but no option is selected.

KEY
specifies that the indicated edit/compression routine can condense or modify any or all fields within the named segment. This parameter is invalid for the root segment of a HISAM data base.

INIT
specifies that initialization and termination processing control is required by the segment edit/compression routine. If this parameter is present, the edit/compression routine is given control at open and close time for that data base.

Segment Edit/Compression

To assist the user in providing parameters to his edit/compression routine, the DBD control block has a table, in the form of assembly language control sections, appended to it. One control section is developed for each segment type to be edited or compressed. Each control section has a CSECT name equal to that of the segment name.

These control sections are placed at the end of the DBD module. They contain information such as the segment edit/compression routine name, the name of segment, and the total length of that control section. Each control section can be extended to contain any desired data or algorithm information. An example of a sample segment control section is shown in Figure 3-2.

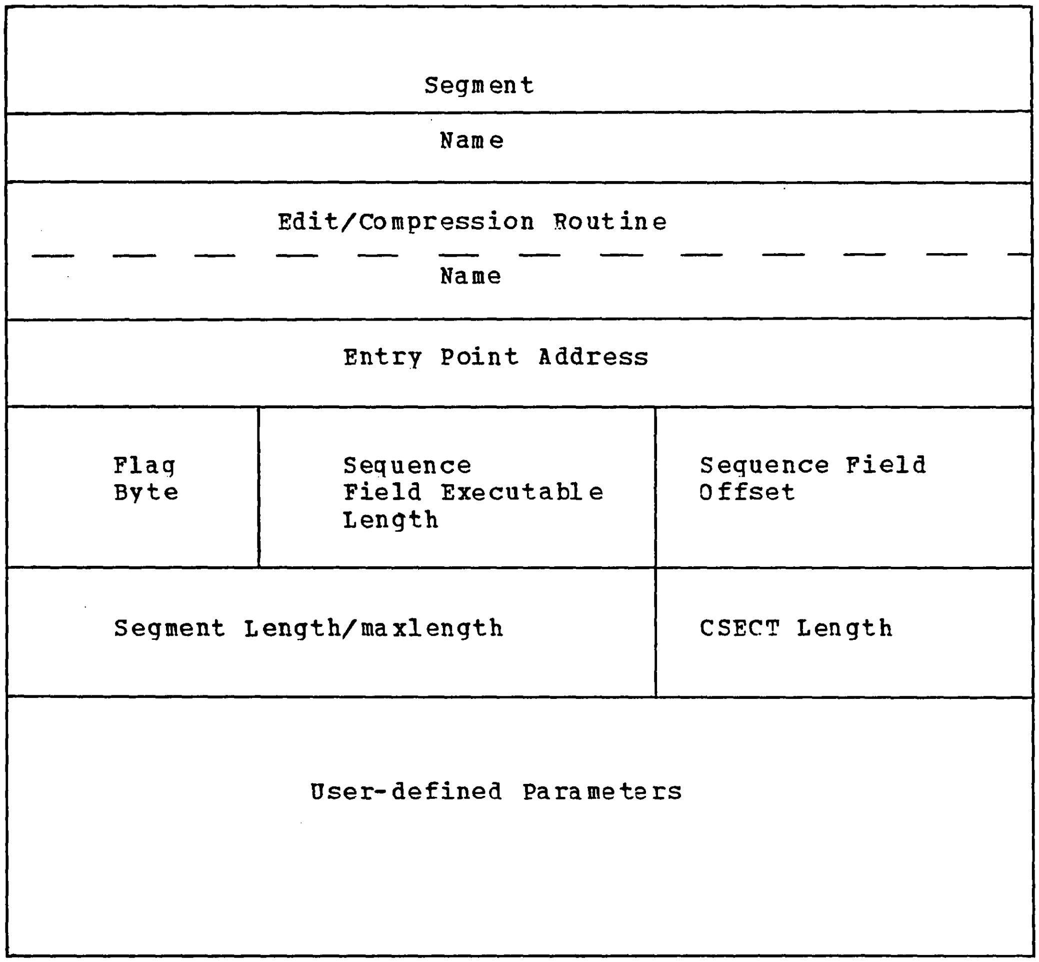


Figure 3-2. Segment Edit/Compression Control Section (SEGPAC)

Information in the various fields shown in Figure 3-2 is as follows:

DMBCPAC	DSECT		
DMBCPCNM	DS	CL8	Segment name
DMBCPCSG	DS	CL8	Edit/Compression routine name
DMBCPEP	DS	A	Entry point address
DMBCPFLG	DS	XL1	Flag byte
DMBCPKEY	EQU	X'02'	Segment has key compression option
DMBCPNIT	EQU	X'01'	Initialization processing is required
DMBCPVLR	EQU	X'04'	Segment is variable length
DMBCPSEQ	EQU	X'08'	Segment has key sequence field defined
DMBCPSQF	DS	XL1	Executable length of sequence field, if defined
DMBCPSQF	DS	H	Sequence field offset
DMBCPSGL	DS	H	For fixed length segments - segment length; for variable length segments - maximum length
DMBCPLNG	DS	H	Total length of CSECT; fixed length plus length of user-defined parameters (always a multiple of 8)
DMBCPUSR	DS	OD	Any quantity of user-defined data

The first 28 bytes are constants defined by DBDGEN. When the new table is defined to include additional parameters, these fields must be duplicated. The only exception to this rule is that the CSECT length field must be updated to reflect the new length. After an assembly of the new table, a link-edit is done to exchange the new table for the old one. User-added code should not contain address constants, because this CSECT is moved after it is loaded. Care must be taken to use an ENTRY statement specifying the name of the DBD when this operation takes place. See "Automatic CSECT Replacement" in _OS/370 Linkage Editor and Loader_ for additional details.

DL/I MODULE INTERFACES

Initialization

When the IMS/VS system is initialized prior to running an application, DL/I takes the following action.

- The IMS Block Builder module (DFSDLBL0) checks whether a user segment edit/compression routine has been specified for a data base. If it has, an SWA large enough to contain the largest expanded segment is constructed, and the address is placed in the PSB prefix.

- Each time the IMS/VS Open/Close module (DFSDLOC0) opens a physical data base, it examines each segment description to see if edit/compression has been specified for that segment type. If it has OPEN/CLOSE, it loads the user routine in the same manner that a HDAM randomizing module is loaded. The address of the user routine is placed in the appropriate segment edit/compression control section of the Data Management Block. If a user edit/compression routine is designed to handle more than one segment type, the routine must be link-edited as reenterable.

Processing

When the application program is activated and begins accessing segments, the DL/I action modules interface with the user edit/compression routine as described below. In all cases, the DL/I modules pass an entry code (described in "Parameters Passed by DL/I" and "Edit/Compression Routine Entry Codes" later in this chapter) to the edit/compression routine. The user's edit/compression routine must examine this entry code to determine the function to be performed.

Load/Insert (_DFSDDLE0_): As each segment is being processed for a load operation, the associated descriptive blocks (PSDBs) are checked to see if it is a candidate for edit/compression. If so, control is transferred to the associated user edit/compression routine. The following parameters are passed to this routine.

- Source address of the start of the segment in the user input/output area

- Destination address of the start of the segment work area (SWA)

- Information address of control blocks containing sufficient data for the edit/compression routine to properly perform its function

- Return address after edit or compression has been accomplished

The length of the segment to be moved is provided in one of two places. If the segment length was specified as fixed (relative to the user input/output area), but to be modified by an edit/compression routine, the source length is reflected in the segment descriptive block. If the segment is defined as variable in length and is to be modified by an edit/compression routine, the source length is provided as a binary value in the first two bytes at the source address. In either case, the move operation provided by the edit/compression routine must result in a two-byte length field, followed by the corresponding quantity of data in the segment work area. Load/Insert compares this two-byte length field with the min-value, if specified. The larger of these two values determines the direct access space requirements for this segment. Load/Insert also compares the two-byte length field with the max-value to verify that the segment does not exceed the maximum length. The length field for a fixed length compressed segment cannot exceed the defined segment length plus 10 bytes.

For a segment insert operation, the action is similar to that of segment load. Edit/Compression, if required, is performed with the segment work area (SWA) as the destination address. The length of the segment in this staging area, or the min-byte value, is used to determine the necessary secondary storage requirements.

<u>Delete/Replace</u> (<u>DFSDLD00</u>): If the segment length changes in an HS environment, the necessary shifting of segments to compensate for the new length occurs. If segment length changes in an HD environment, an effort is made to position the segment data as close as possible to the segment prefix. In both cases, the min-byte value must be properly observed.

<u>Retrieve</u> (<u>DFSDLR00</u>): Several alternatives exist for segment movement:

- If a segment is defined by the user as variable in length, and no edit/compression routine is specified, IMS/VS moves the segment directly from the buffer pool to the application program I/O area, by-passing the segment work area (SWA).

- If a segment is defined as variable in length, and an edit/compression routine is specified, the segment is moved from the buffer pool to the segment work area by the specified routine. The segment length is updated to reflect the expansion. The segment can now be moved on to the user.

- If a segment is defined as fixed in length, and an edit/compression routine is specified, the segment is moved from the buffer pool to the segment work area by the appropriate routine. However, since the two-byte segment length field is used only for the disk format, the user edit/compression routine must strip the two-byte length field while moving the segment to the SWA.

- All segment edit and compression takes place on a segment as it relates to its physical description. Therefore, any segment or segments involved in logical relationships must be properly expanded before Retrieve builds the logical image that is to be placed into the application program input/output area.

Segment movement out of the application program input/output area (IOA) follows one of two patterns. If the segment is eligible for edit/compression, it proceeds through an intermediate staging operation into the segment work area (SWA). If it is ineligible for edit/compression, staging to determine the edited or compressed length is not necessary. In this case, the length specified in the IOA is used to determine buffer space requirements. Segment movement during the retrieval operation is usually from the buffer, through the edit/compression routine to the SWA, and then on to the input/output area. However, if the user has requested a retrieval based upon the contents of a field in the compressed area of a segment, any segment that might qualify must first be expanded in the SWA for examination. Only the qualified segment is then moved into the I/O area.

The edit/compression routine obtains control from the appropriate action module. It is presented with both a source and destination address, as well as the address of the segment descriptive blocks. Its responsibility is to move the segment from the source area into the destination area, performing the desired operation, and updating the segment length field to reflect this operation.

The following summary represents the operation by module and function.

<u>Module</u>: <u>Load/Insert</u>

```
                                          Edit/Compression
   Function:  Load                           |
                                        +--|-----+  +--------+
                                        |        |  |        |
   Segment movement:                   IOA       SWA        Buffer Pool
                                        |                    |
                                        +--------------------+
                                             No Edit/Compression
```

Load/Insert uses the min-byte value (if provided), or specified length, whichever is greater, for segment length.

```
                                          Edit/Compression
   Function:  Insert                         |
                                        +--|-----+  +--------+
                                        |        |  |        |
   Segment movement:                   IOA       SWA        Buffer Pool
                                        |                    |
                                        +--------------------+
                                             No Edit/Compression
```

Load/Insert uses the min-byte value (if provided) or specified length, whichever is greater. In HS, Load/Insert moves all the following segments to the right, creating a new block if necessary.

Module: Delete/Replace

Function: Delete

Segment movement: None.

In HD, Delete/Replace frees the space the segment previously occupied.

Function: Replace

Segment movement:

```
         Edit/Compression
  ┌----|---┐ ┌--------┐
  |        | |        |
 IOA       SWA       Buf
  |                   |
  └-------------------┘
      No Edit/Compression
```

In HS,

- If the new segment is shorter than the old segment, Delete/Replace overlays the old data with new data, and moves the following segments, if any, to the left, observing the min-bytes parameter if specified.
- If the new data is of equal length to the old data, replace old data with new.
- If the new data is longer than the old data, Delete/Replace moves the following segments, if any, and inserts the new data. This operation requires a call to the Load/Insert module since the data shift may require the allocation of new OSAM blocks.

In HD,

- If the new data is shorter than the old data, and if the prefix and data are together, the new sequent is moved in and the excess space is freed, after checking the min-byte value. If the prefix and data are separate, space is obtained as close to the prefix as possible, the new data is moved in, and the previously occupied space is freed.
- If the new data is equal in length to the old data, the old data is replaced by the new data in a one-for-one manner.
- If the new data is longer than the old data, space is obtained as close to the prefix as possible. New data is inserted in the new space. The old data space is freed.

Module: Retrieve

Segment movement:

```
                     Expand
 ┌---------| |---|----┐
 |         | |        |
IOA         SWA        Buf
 |                    |
 └---- ---------------┘
         No Expand
```

For retrieval of segments, expansion occurs in the segment work area. If examination of compressed fields for segment qualification is required, a staging operation in the segment work area is necessary to analyze each candidate.

PARAMETERS PASSED BY DL/I

DL/I provides the following information to the user's edit/compression routine when a segment is to be processed:

- Register 1 contains the address of the Partition Specification Table (PST).
- Register 2 contains the address of the first byte of the segment to be processed (source address).
- Register 3 contains the address of the first byte of the work area into which the segment is to be moved (destination address).
- Register 4 contains the address of the Physical Segment Description Block (PSDB). From this block, the Field Description Blocks (FDB) can be located, as required.
- Register 5 contains the address of the segment edit/compression control section.
- Register 6 contains the entry code (described below).
- Register 13 contains the address of a save area into which the system's registers must be stored by the user.
- Register 14 contains the address used to return to DL/I when segment processing has been accomplished.
- Register 15 contains the user-specified entry point into the segment edit/compression routine.

All IMS/VS control blocks provided to the segment edit/compression routine are for reference only; no data can be changed, including the segment at the source area address. The only modification allowed is the alteration of the segment during the move operation from the source to the destination address. DSECT addressability to the above mentioned control blocks is provided by the IMS/VS IDLI macro, as shown in the examples provided earlier in this chapter.

EDIT/COMPRESSION ROUTINE ENTRY CODES

When the user segment edit/compression routine is placed into the IMSVS.RESLIB, or another valid library, by a linkage editor process, one entry point to it must be specified by the user. When the routine is entered, the entry code placed in register 6 can be used to determine the reason for invocation.

Entry code =

0 - segment edit/compression takes place. The source address points to a segment image as it appears in the application program input/output area.

4 - entire segment expansion takes place. The source address points to a segment that must be expanded into an image capable of being presented to the application program. Application program requests qualified on a data field require the use of entry code 4 for normal retrieval expansions.

The above two entries are the minimum required by the user for segment compression and expansion, and they are the two codes used when the DATA compression option is specified. To reduce the amount of processing overhead required with the movement of data, a third table entry is required when the KEY compression option is used.

8 - partial segment expansion for the key compression option. Expansion takes place from the start of the segment through the sequence field. This facility is required if the user elects to use key compression, or if he compresses any field that alters the starting position of the key field. All DL/I calls using sequence field qualification on key compressed segments require the use of this entry code.

To provide a data edit/compression routine with greater flexibility in the use of algorithms than is contained in the code itself, two additional options are provided to allow for tabled data information. The first is contained within the DBD module itself. For each segment defined during DBDGEN as being eligible for edit/compression, an entry is developed in an assembly language control section, described in a previous paragraph. This control section can be extended. This is done by an assembly and link-edit to contain any desired data or algorithm information. The second option allows the module to issue the IMS/VS IMODULE macro to provide functions equivalent to the OSLOAD or GETMAIN macro instructions. They bring additional information into storage in the form of modules from the IMSVS.RESLIB. An example is a table of substitution characters to be maintained separately from the executable code. This table could reflect different combinations for different segments, resulting in a general purpose, table-driven routine, capable of processing several segment types.

Since it is also possible that pre- and post-processing are required by the edit/compression routine (for example, to load and delete the compression algorithm table in the above case), two more entry codes are provided when the INIT parameter is specified in the SEGM control statement. With these codes, the OPEN/CLOSE module relinquishes control to the initialization/termination subroutines immediately after the data base is opened, and immediately prior to the data base being closed. Any processing required for the data base segments that cannot be directly related to any one segment can be done at this time.

Entry code =

12 - control is obtained for algorithm processing immediately after the data base is opened. Registers 2, 3, and 4 are unpredictable.

16 - control is obtained for algorithm post-processing immediately prior to the data base being closed. Registers 2, 3, and 4 are unpredictable.

For compression, regardless of the format at the source address, the segment at the destination address must be in variable length format. The first data field of the segment is a two-byte segment size field. DL/I processes the condensed segment through the buffer pool to secondary storage.

If a fixed length segment is to be compressed, and the data format is such that compression cannot take place, it is possible that the addition of control information by the user routine, indicating the segment could not be compressed, will lengthen the segment beyond its fixed length definition. To allow for this expansion, and to allow DL/I to validity check the results of the compression, an arbitrary value of 10 bytes is added to the defined length. This value is maintained in the Physical Segment Description Block and is used by DL/I as the maximum allowable segment length. No additional secondary storage is required due to this arbitrary value.

For segment expansion occurring during the segment retrieval process, the Retrieve module examines the application program request. If the request is to be satisfied by a compressed segment, a test is made to see which type of compression was used, either key or data. Then, depending upon the type of retrieval request, either entry code 4 or 8 is passed to the compression routine. The following criteria are used as a basis for the decision:

- If the segment can be accepted without analysis of either a key or data field, control is transferred using entry code 4. The segment is expanded to the form presented to the user.

- If the value of the segment sequence field requires examination prior to segment selection, an additional check is performed to determine data or key compression. Data compression requires no additional processing, while key compression requires activation of entry code 8. If, after key field validation, the segment is qualified for presentation, it is passed on to the user, after being properly formatted by entry code 4.

- If data field analysis is necessary to properly satisfy the DL/I call, proper expansion of the segment, via entry code 4, takes place. When the correct segment is found, it is passed on to the user.

The format of the segment presented through entry codes 4 and 8 of the compression routine is identical to that of a variable length segment; that is, a two-byte segment size field followed by the appropriate quantity of data. It is the responsibility of the called routine to properly expand the segment at the destination address in correct format, either fixed or variable length. In the case of key compression, expansion must take place from the start of the segment through the sequence field. For variable length segments, the segment data length field, after processing by the key expansion, must reflect the length of the expanded portion of the segment at the destination address.

CONVERTING EXISTING DATA BASES

To convert existing data bases to use this facility, do the following:

1. Unload the current data base using the reorganization/unload utility, and using the current DBD.

2. Define a new DBD which specifies VSAM as the access method, and specifies a COMPRTN for those segments that are to be converted. Reload the data with the reorganization/reload utility.

3. The named COMPRTN provided during reload should encode, compress, or edit the segment (as determined by the installation's requirements), and add the two-byte length field.

PERFORMANCE CONSIDERATIONS

The primary purpose of segment compression is to decrease the quantity of space required for segment storage. To accomplish this the user has two types of compression, DATA and KEY. However, the use of these options can have varying effects on performance that should be examined. For example, compressing or expanding each segment, on its way to or from the application program, involves additional processing. In addition, the search time required to locate the requested segment may be increased, depending on the options selected. In the case of full segment compression, using the KEY compression option, every segment type that is a candidate to satisfy either a fully qualified key or data field request must be expanded to allow examination of the appropriate field by the IMS/VS Retrieve module (DFSDLR00). For key field qualification, only those fields from the start of the segment through the sequence field are expanded. For data field qualification, the total segment is expanded. In the case of data compression and a key field request, little more processing is required to locate the segment than that of non-compressed segments, since the segment sequence field is used to determine if this segment occurrence satisifies the qualification.

Other considerations can impact total system performance, especially in an online teleprocessing environment. For example, being able to load an algorithm table into memory gives the compression routine a large amount of flexibility. However, this action can place the entire IMS/VS control region into a wait state until the requested member is present in main storage.

SEGMENT COMPRESSION/EXPANSION MODULE EXAMPLE: KMPEX

A compression/expansion example is provided as guidance to the IMS/VS system user. The example is not intended to be operational (for example it contains many unspecified series of routines), and no support by IBM for this routine is implied. The KMPEX program is a segment compression/expansion program coded according to the IMS/VS Program Functional Specifications. This program processes a particular segment for compression or expansion on the basis of the parameters and data passed by the IMS/VS Control Program.

When control is given to the KMPEX program, the program checks an entry code passed in register 6, finds out whether the code indicates a request for compression of a segment, or partial or entire expansion of a compressed segment. It then branches to an appropriate routine to perform the required task.

Upon normal completion of the task, it returns control to IMS/VS Control Program with a return code of 0.

Specific rules and restriction followed in compression and expansion of a segment are detailed in the following sections.

The Compression Routine

Compression of a segment requires different data handling according to the data organization of the segment. There are two data formats:

1. Fixed data format
2. Variable-length data format

A user may specify one of two options to either of the above segment formats. The options are KEY and DATA.

```
+-------------------------------------------------------------------+
|                            |                                      |
| Data before compression    |        Data after compression        |
|                            |                                      |
|-------------------------------------------------------------------|
|                            |                                      |
| Fixed length:              |-> KEY option  |LL'| P | D'| K'| D |  |
|        | D | K | D |<      |                                      |
|                            |-> DATA option |LL'| D | K | P | D'|  |
|                            |                                      |
|-------------------------------------------------------------------|
|                            |                                      |
| Variable-length:           | KEY option   |LL'|LL| P |D'|K'|D'|   |
|                            |                                      |
|        |LL | D | K | D |   | DATA option  |LL'| D| K |LL| P|D'|   |
|                            |                                      |
+-------------------------------------------------------------------+
```

D = data, K = key, P = pointer to the 1st CCB

LL' = new segment length, LL = original segment length

D' and K' = compressed data and key

Thus, compression of a segment results in one of the four formats listed above, depending upon the original record format, and the option specified.

Method of Compression

Compression of data is specified wherever any consecutively redundant characters of four bytes or more occur in a particular segment.

The Compression Control Block (CCB)

Compression is performed by replacing the repeated identical characters with a Compression Control Block (CCB). A CCB consists of 3 bytes containing the following information:

```
|CCB|PNCB|LRC|RC|
```

PNCB = a pointer to the next control block (CCB).
LRC = the length of the redundant character in bytes.
RC = the redundant character in hex.

- The PNCB is a 1-byte area whose value cannot exceed 255 (decimal). A block of four or more repeated characters is likely to occur within any span of 255 consecutive bytes in a normal data base. If two groups of repeated characters, however, are separated by more than 255 bytes, a dummy CCB must be constructed between them.

```
┌──────────────┐      ┌──────┐         ┌──────────────┐
|PNCB|LRC|RC|------|CCB-2|---------|PNCB|LRC|RC|
└──────────────┘      └──────┘         └──────────────┘
|<- CCB-1 <----------N>255------------->|   CCB-3
```

A dummy CCB is no different from a regular CCB except that its LRC field contains zero, meaning a redundancy of zero bytes in length.

- LRC represents the length of redundant characters in bytes. Like PNCB, the LRC's maximum value is 255. If the same character is repeated 256 times or more, therefore, there must be 1 CCB for every 255 bytes, plus 1 CCB for any residual characters.

```
┌-----------------------------------------------------------┐
|                                                           |
|   characters                     CCB                      |
|                                                           |
|-----------------------------------------------------------|
|                                                           |
|                              ┌--------┐                   |
|* 255 characters of "A"      |nn FF C1|                   |
|                              └--------┘                   |
|                                                           |
|                              ┌--------┐                   |
|* 258 characters of "B"      |nn FF C2|C2 C2 C2────┐      |
|                              └---┬----┘            ↓      |
|                                  |     3 residual chars-not|
|                                  |       compressed.      |
|                                  └────>1 CCB for the 1st   |
|                                          255 chars        |
|                                                           |
|                              ┌----------------┐           |
|* 259 characters of "C"      |03 FF C3|nn 04 03|────┐     |
|                              └---┬------------┘     ↓     |
|                                  |     2nd CCB for the last|
|                                  |       4 C's            |
|                                  └────>1st CCB for the 1st |
|                                          255 chars        |
|                                                           |
└-----------------------------------------------------------┘
```

The value in the LRC ranges from 0 through 255. The zero in LRC means that there is no character to be compressed. The CCB in this case plays a role of step-stone between two CCBs that are apart by more than 255 bytes.

- RC represents redundant character. It is a 1-byte area and can contain any value ranging from X'00' to X'FF'. A zero value here is of no special significance.

Pointer to the First Control Block (PFCB)

Regardless of the format of a segment, or the option for compression, the first byte of compressed data is allocated to the PFCB. It contains the offset to the first CCB, inclusive of the PFCB byte.

The location of the PFCB varies according to the data format.

```
┌─────────────────────────────────────────────────────────────────┐
|Data                                                             |
|format:   Option:         PFCB relative to other data            |
|                                                                 |
|-----------------------------------------------------------------|
|                                                                 |
|                        ┌──────────────────────┐                 |
|Fixed     key          |LL PFCB (D)  (K)  (D) |                 |
|                        └──────────────────────┘                 |
|                                    └────►compressed segment     |
|                                                                 |
|                        ┌──────────────────────┐                 |
|          data          |LL D   K   PFCB   (D) |                 |
|                        └──────────────────────┘                 |
|                                       └►compressed data         |
|                             └►data & key field--not compressed  |
|                                                                 |
|-----------------------------------------------------------------|
|                                                                 |
|                    ┌─────────────────────────┐                  |
|Variable  key       |LL LL PFCB  (D)  (K)  (D)|                  |
|                    └─────────────────────────┘                  |
|                                           └►compressed segment  |
|                        └──────────────►original segment length  |
|                    └──────────────────►new segment length       |
|                                                                 |
|                    ┌─────────────────────────┐                  |
|          data      |LL D   K  LL   PFCB (D)  |                  |
|                    └─────────────────────────┘                  |
|                                        └────►compressed data    |
|                              └►original segment length          |
|                        └─────►data & key field--not compressed  |
|                    └─────────►new segment length                |
|                                                                 |
└─────────────────────────────────────────────────────────────────┘
```

The Last Compression Control Block (LCCB)

After all data in a segment has been compressed, a one-byte area, which always contains zero, is assigned to the LCCB. When the PNCB of a CCB points to an area containing zero, it means that the CCB is the last CCB in the segment. The value in a PNCB of the last CCB varies, depending on how the segment ends.

End of Segment	After Compressible Characters X's	Last CCB to LCCB
XXXXXAAAAA	4 or more RCs	03 05 E7 | 03 05 C1 | 00 LCCB the last CCB CCB
XXXXXBCDE	no 4 or more RCs	07 05 E7 | C2 C3 C4 C5 00 LCCB no compressible data the last CCB

Length of New Compressed Segment

A segment size is not always reduced by the compression routine. It is increased when redundancy of a character occurs rarely, or a segment size is large, and the compression routine uses numerous dummy CCBs.

If the length of a compressed segment exceeds the size of the output buffer area passed by the IMS/VS Control Program (two bytes longer than the maximum segment length), the KMPEX program handles the situation as follows.

The compression routine maintains a counter containing the updated length of the processed compressed segment. If the segment length of a compressed data is equal to or greater than the original size of the segment, compression is regarded as unsuccessful, and the output area is replaced with a new length of segment (two-byte area), and the original segment.

The following new segment output by the compression routine indicates that the segment involved has not been compressed:

Segment Format	New Segment Length
Fixed	the 1st 2 bytes = a fixed segment length + 2
Variable	the 1st 2 bytes = an original segment length (saved in the second two bytes) + 2

The above segment is regarded as compressed data by the control program and treated as such. Differentiation is made only by the compression/expansion routine.

The Expansion Routine

The expansion routine receives control when a segment that has been compressed is retrieved from secondary storage. The method of expansion is the reverse of the compression process described above.

Special handling occurs when the following two conditions are found:

- The value in the length field in the first two bytes is 2. In this case:

segment format	actual segment data
fixed length	(none)
variable length	X'002'

* If any of the following conditions apply, the segment is interpreted as not compressed, and is not expanded:

Record Format	Length equal to	Current input data
Fixed	a fixed segment length + 2	not compressed -- ignore expansion
Variable-length	a value in the 2nd 2 bytes of input + 2	not compressed -- ignore expansion

In all other cases, the routine expands the segment by decoding the associated CCBs.

The Initialization Processing Routine

When so specified, IMS/VS gives control to the compression/expansion routine:

* Immediately after the data bases, have been opened
* Just before the data bases are closed

When a command code is given to branch to the post-OPEN routine or the pre-CLOSE routine in the KMPEX program, a WTO message, is issued stating that an entry to an appropriate routine has been made. No processing of particular data is attempted at this stage.

Program Messages and Codes

1. OPEN OF SEGMENT xxxxxxxx

   Control has been received by the compression/expansion routine after an OPEN of the data bases has been completed. Any preprocessing tasks of the named segment should be completed here.

2. CLOSE OF SEGMFNT xxxxxxxx

   Control has been received by the compression/expansion routine before the system closes data bases. Any post-processing tasks of the named segment should be completed here before close of the data base.

3. Abend codes (*All the abend instructions can be changed to a RETURN instruction to the system, with an abnormal return code).

   a. USER 2989 -- ABEND

      1. A segment data organization is variable length, but its length field is one of the following:

         2>N>32767 (decimal)

      2. A fixed length record, but the segment length in Compaction Control Table indicates:

         0>N>32767

   b. USER 2990 -- ABEND

      A command code passed by the control program is out of a valid range:

      0>N>16

   c. USER 2991 -- ABEND

      A command code is passed to compress after, or expand up to, a sequence field of a segment. No sequence field has been defined in the segment.

   d. USER 2992 -- ABEND

      Any of the following conditions results in an abend with the above code.

      Applicable to both fixed- and variable-length segments:

      1. A D/K length is greater than a SGL length of a segment.

      Applicable only to a variable-length segment:

      2. A D/K length is greater than an LL length.

      3. An LL length is greater than an SGL length.

      4. An LL length is less than 2.

      5. An SGL length is less than 2.

Applicable to a fixed segment:

6. An SGL length is a negative value.

D/K length	=	A sum of length from the beginning of a segment to the end of a key field (SEQUENCE FIELD).
SGL length	=	A length of a segment indicated in the segment length field of a Compression Control Table.
LL length	=	A length of a variable length record indicated in the first two bytes of a precompressed segment.

Program Assumptions

All parameters and data passed by the IMS/VS control program are assumed to be valid data; such as the address of the input segment data, the output data area address, and the length of an input segment.

The IMS/VS control program passes an address of an input segment data area in register 2, and an address of an output data area in register 3.

The size of output data area is:

- A segment length plus two bytes for a fixed length segment.
- The maximum segment length for a variable length segment.
- No segment length is greater than 32,767 bytes.

All segments processed by the compression routine are treated as variable length by the IMS system control program, regardless of their pre-compression format.

A listing of the KMPEX routine follows.

```
KMPX     TITLE 'KMPEX ROUTINE--USER DATA COMPRESSION PROGRAM'
*                                                                      *
************************************************************************
**                                                                    **
**                                                                    **
******   **********   'KMPEX' DATA COMPRESSION/EXPANSION PROGRAM ******
**                                                                    **
**          'KMPEX' PROGRAM IS A DATA COMPRESSION/EXPANSION ROUTI-    **
**       NE.  COMPRESSION OF DATA IS DONE TO ANY CONSECUTIVELY RE-    **
**       DUNDANT CHARACTERS OF 4 BYTES OR MORE IN THE DATA.  COMP-    **
**       RESSION USES A CONTROL BLOCK CONSISTING OF 3 BYTES, I.E.     **
**       1. PTR TO NEXT CONTRL BLK, 2. # OF REDUNDANCY, 3. THE CH-    **
**       ARACTER REDUNDANT.                                           **
**                                                                    **
**          COMPRESSION IS TERMINATED WHENEVER THE LENGTH OF PROC-    **
**       ESSED DATA BECOMES EQUAL TO OR LONGER THAN THE INITIAL       **
**       DATA LENGTH, AND THE PRE-PROCESSED DATA IS RETURNED TO       **
**       DL/I AS WAS.                                                 **
**          DETAILED FORMATS AND CONTROL BLOCKS OF COMPRESSION/       **
**       EXPANSION ARE DESCRIBED IN SPRM.                             **
**       *****  REGISTER USAGE IN THE 'KMPEX' PROGRAM ***              **
**                                                                    **
**       R1---WORK REGISTER                                           **
**       R2---PTR TO INPUT DATA                                       **
**       R3---PTR TO OUTPUT DATA                                      **
**       R4---PTR TO PSDB                                             **
**       R5---PTR TO 'SEGPAC' SEG COMP CSECT                          **
**       R6---CTR FOR CURRENT INPUT PROCESSING                        **
**       R7---CTR FOR OUTPUT DATA                                     **
**       R8---CTR FOR INPUT PROCESSED                                 **
**       R9---PTR TO THE CURRENT INPUT                                **
**       R10---WORK REGISTER                                          **
**       R11---WORK REGISTER                                          **
**       R12---KMPEX BASE REGISTER                                    **
**       R13---REGISTER SAVE AREA                                     **
**       R14---RETURN ADDR TO DL/I                                    **
**       R15---KMPEX ENTRY POINT                                      **
**                                                                    **
**                                                                    **
************************************************************************
**                                                                    **
         CNOP  0,8
KMPEX    CSECT
         SAVE  (14,12)
         BALR  12,0                 ESTABLISH THE ADDRESSABILITY
         USING *,12
         LA    R10,KSAV1
         ST    R13,4(R10)           SAVE PASSED SAVE AREA
         ST    R10,8(R10)
         LR    R13,R10
         USING KCCB,R5
INIT     MVC   KNITA(KNITL),KFO+3   INITIALIZE FLAGS
         STC   R6,KCMCD             SAVE COMMAND CODE
         CLI   KCMCD,KQINIT
         BNL   KA350                BRANCH IF  INIT PROCESSING RTN
         BAL   R11,KA3600         ## BR TO SYSTEM DATA CHK RTN
         ST    R2,KASN1             SAVE IN-BUFFER ADDR
         ST    R3,KASN2
         TM    KFLG,KVLN            CHK IF V-LENG SEGMT
```

```
         BZ     KA300                  BR IF FIXED SEGMT
         TM     KFLG,KKEY
         BO     KA200
         OI     KFLGX,KVLDT            SET V-LENG, DATA OPTION FLG
KA200    EQU    *
         LH     R1,0(R2)               GET ORG SEGMT LENG
         STH    R1,KTLL1                    SAVE IPT LINE LENGTH
         LH     R9,KSGL                GET SEGMT MAX LENGTH
         SH     R9,KH3
         CR     R1,R9                  CHK V-LEN SEGMT LENGTH
         BL     KA310                  BR IF NOT  LST 4 BYTES
         OI     KFLGX,KNPRSW           SET NON PROCESS SW ON
         LR     R1,R9                  GET SEGMT MAX LENGTH
         B      KA310
KA300    LH     R1,KSGL                CLEAR OUTPUT BUFFER
         STH    R1,KTLL1                    SAVE IPT LINE LENGTH
KA310    EQU    *
         STH    R1,KMAXL               SAVE MAXIMUM BUFF LENGTH
         EX     R1,KEXBF
KA350    EQU    *
         SR     R6,R6
         LR     R7,R6
         LR     R8,R6                  CLEAR REGS
         LR     R9,R6                  CLEAR REGS
         LR     R10,R6
         IC     R10,KCMCD              GET CMD CODE
         B      *+4(R10)
         B      KA400                  BR TO COMPACT RTN
         B      KA2200                 BR TO TOTAL EXPANSION RTN
         B      KA2200                 BR TO PARTL EXPANSION RTN
         B      KA1600                 BR TO POST-OPEN RTN
         B      KA1700                 BR TO PRE-CLOSE RTN
KAB2990  EQU    *
         LH     R1,KABCX90             GET ABEND CODE
         B      KA4500
KA400    EQU    *
         TM     KFLG,KVLN                   CHK IF VL REC
         BZ     KA420                       BR IF FIX REC
**                                                                    **
************************************************************************
**                                                                    **
*     *****   VARIABLE-LENGTH SEG COMPRESSION CHECK RTN   *****       **
**                                                                    **
************************************************************************
         LH     R1,0(R2)               GET VLEN LENGTH
         CH     R1,KH2                 CHK IF MIN LENGTH
         BL     KAB2989                BR IF LESS THAN MIN
         BH     KA450                       BR IF MORE THAN MAX
         B      KA430
KA420    EQU    *                      FIX LENGTH RECORD
**                                                                    **
************************************************************************
**                                                                    **
**    *****   FIXED-LENGTH SEG COMPRESSION CHECK RTN   *****          **
**                                                                    **
************************************************************************
         CH     R1,KH0                 CHK IF 0 LENGTH
         BE     KA430                  BR IF SO
         BH     KA450                  BR IF MORE THAN 0 BYTE
```

```
KAB2989  EQU    *
         LH     R1,KABCX89          GET ABEND CODE
         B      KA4500
KA430    MVC    0(2,R3),KH2              MOVE REC LENG
         B      KA1800
KA450    EQU    *
         TM     KFLG,KKEY                 CHK IF KEY OPTION
         BZ     KA1300                    BR IF DATA OPTION
KA500    TM     KFLG,KVLN                 CHK IF VLN REC-FORM
         BO     KA700                     BR IF VLN REC
KA600    EQU    *                   FIX-KEY OPTION
         LA     R3,3(R3)
         LA     R7,3(R7)
         B      KA750
KA700    MVC    2(2,R3),0(R2)       VLEN-KEY OPTION
         LA     R2,2(R2)
         LA     R6,2(R6)
         LA     R3,5(R3)
         LA     R7,5(R7)
KA750    LR     R1,R3
         CH     R7,KMAXL                 CHK IF MS LENGTH EXCEEDED
         BNL    KA3500                   BR  TO MOVE ORIGINAL SEG
         BCTR   R1,0
         ST     R1,KFCCB            SAVE PTR TO COB IN AREA
***********************************************************************
**                                                                   **
***********************************************************************
**                                                                   **
KA800    BAL    R11,KMPSR           BRANCH TO COMPRESSION RTN
**                                                                   **
***********************************************************************
**                                                                   **
***********************************************************************
         B      KA1800              BR TO END RTN
KA1300   EQU    *                   FIXED/VLN DATA OPTION
         TM     KFLG,KVLN           CHK IF V-LENG SEGMT
         BO     KA1320              BR IF SO
         LA     R3,2(R3)
         LA     R7,2(R7)
KA1320   EQU    *
         SR     R1,R1
         IC     R1,KSQL
         AH     R1,KSQA
         AR     R7,R1
         CH     R7,KMAXL                 CHK IF MS LENGTH EXCEEDED
         BNL    KA3500                   BR  TO MOVE ORIGINAL SEG
         BAL    R11,KEXR1           BR TO MOVE DATA
         AR     R3,R1               UPDATE KSN2 TO LL+D1+K
         TM     KFLG,KVLN           CHK IF V-LEN SEGMT
         BZ     KA1350              BR IF FIXED SEGMT
         MVC    0(2,R3),0(R2)       MOVE SEGMT LENGTH
         LA     R3,2(R3)
         LA     R7,2(R7)
KA1350   AR     R2,R1               UPDATE KSN1 TO D1+K
         AR     R6,R1
KA1360   EQU    *
         ST     R3,KFCCB            GIVE 1ST CCB PRT ADDR
         LA     R3,1(R3)            UPDATE KSN2 PTR
         LA     R7,1(R7)            UPDATE KN2 CTR
```

```
         CH    R7,KMAXL                    CHK IF MS LENGTH EXCEEDED
         BNL   KA3500                      BR  TO MOVE ORIGINAL SEG
         B     KA800
KA1600   EQU   *                      POST-OPEN PROC RTN
         TM    KFLG,KNIT              CHK IF INIT PROC SPECIFIED
         BZ    KAB2990                BR IF NOT SO
*********************************************************************
**                                                                 **
**       *****   POST-OPEN ROUTINE   *****                         **
**          THIS ROUTINE IS BRANCHED WHEN A COMMAND CODE OF X'0C'  **
**       IS PASSED IN R6 BY DL/I.  PRE-PROCESSING TASKS ARE TO BE  **
**       DONE HERE.  A MESSAGE OF ENTRY AFTER 'OPEN' IS ISSUED BY  **
**       KMPEX.                                                    **
**                                                                 **
*********************************************************************
         MVC   KA1650(8),KSGN         MOVE SEG NAME
         CNOP  0,4
         BAL   1,KA1760
         DC    AL2(28)                TEST LENGTH
         DC    2X'00'                 MCS FLAGS
         DC    CL16'OPEN OF SEGMENT '
KA1650   DC    CL8'XXXXXXXX'
*        B     KA1800
         DS    0H
KA1700   TM    KFLG,KNIT              CHK IF INIT PROC SPECIFIED
         BZ    KAB2990                BR IF INVALID
**                                                                 **
*********************************************************************
**                                                                 **
**       *****   PRE-CLOSE ROUTINE   *****                         **
**          THIS ROUTINE IS BRANCHED WHEN A COMMAND CODE X'10'     **
**       IS PASSED IN R6 BY DL/I.  POST-PROCESSING TASKS ARE TO    **
**       BE DONE HERE.  A MESSAGE OF ENTRY BEFORE 'CLOSE' IS ISSU- **
**       ED BY KMPEX.                                              **
**                                                                 **
**                                                                 **
*********************************************************************
         MVC   KA1750(8),KSGN         MOVE SEG NAME
         CNOP  0,4
         BAL   1,KA1760
         DC    AL2(29)                TEXT LENGTH
         DC    2X'00'                 MCS FLAGS
         DC    CL17'CLOSE OF SEGMENT '
KA1750   DC    CL8'YYYYYYYY'
KA1760   DS    0H
         SVC   35
*        B     KA1800                      BR TO END OF ROUTINE
*********************************************
         SPACE 3
*********************************************************************
**                                                                 **
**       *****   RETURN TO DL/I   *****                            **
**                                                                 **
*********************************************************************
KA1800   EQU   *                                                   #
         L     13,4(13)                                            #
         RETURN (14,12),RC=0          RETURN TO CNTRL PGM
KA1900   EQU   *
         TM    KFLG,KVLN              CHK IF VARIABLE LEN-REC
```

```
         BO      KA1910              BR IF V-LENG REC SEGMT
         LH      R1,KSGL             FIX REC LENGTH
         B       KA2010
KA1910   EQU     *
         LH      R1,0(R2)            GET SEGMT LENGTH
         LA      R1,1(R1)
         CH      R1,KSGL             CHK IF FINAL 2 BYTES
         BL      KA1950              BR IF NOT SO
         L       R2,KASN1
         L       R3,KASN2
         LH      R1,0(R2)            GET LENGTH OF SEGMT
         CLI     KCMCD,KTLSQ         CHK IF KEY EXPANSION
         BNE     KA1930              BR IF ALL EXPANSION
         SR      R1,R1
         IC      R1,KSQL
         AH      R1,KSQA             GET LENGTH THRU KEY
KA1930   EQU     *
         BAL     R11,KEXR1           MOVE TO OUT AREA
         B       KA1800
KA1950   EQU     *
         TM      KFLG,KKEY           CHK IF KEY OPTN
         BO      KA2000              BR IF KEY OPTION
         SR      R1,R1               VLEN + DATA OPTION, EXPANSION
         IC      R1,KSQL
         AH      R1,KSQA
         AR      R1,R2
         LH      R1,0(R1)            GET ORIGINAL SEGMT LENGTH
         B       KA2010
KA2000   LH      R1,2(R2)
KA2010   LA      R1,2(R1)            ADD NEW LENGTH FIELD
         CH      R1,0(R2)            CHK IF NO COMPACTN/EXPNSN
         BNE     KA2250              BR IF NOT SO
         TM      KFLG,KVLN           CHK IF V-LENG SEGMT
         BZ      KA2050              BR IF FIXED SEGMT
         TM      KFLG,KKEY           CHK IF KEY OPTION
         BO      KA2050              BR IF SO
         CLI     KCMCD,KALL          CHK IF ALL EXPANSION
         BNE     KA2050              BR IF NOT SO
         BAL     R11,KMVORGXV        BR TO MOVE ORG SEGMT
         B       KA2070
KA2050   LH      R6,0(R2)            GET LENGTH OF IN-DATA
         SH      R6,KH2
         BAL     R11,KMVORGX         BR TO MOVE ORG SEG RTN
KA2070   EQU     *
         B       KA1800
         SPACE 5
KA2200   EQU     *
         CLC     0(2,R2),KH2               CHK IF REC LENG = ZERO
         BNE     KA1900
         TM      KFLG,KVLN                 CHK IF VL REC
         BZ      KA1800                    BR IF FIX REC
         MVC     0(2,R3),KH2
         B       KA1800
KA2250   EQU     *
         TM      KFLG,KKEY                  CHK IF KEY OPTION
         BZ      KA3000                     BR IF DATA OPTION
KA2300   TM      KFLG,KVLN           EXPANSION OF FIX REC KEY OPTN
         BO      KA2500
         LA      R2,2(R2)
```

```
         LA     R6,2(R6)
         B      KA2600
KA2500   MVC    0(2,R3),2(R2)         EXPANSION OF VAR LENG KEY OPTN
         LA     R2,4(R2)
         LA     R6,4(R6)
         LA     R3,2(R3)
         LA     R7,2(R7)
KA2550   CLI    KCMCD,KTLSQ           CHK IF THRU SEQ FIELD
         BNE    KA2600                BR IF NOT SO
         TM     KFLG,KSEQ             CHK SEQ FLD DEFINED
         BO     KA2560
KAB2991  EQU    *
         LH     R1,KABCX91            GET ABEND CODE
         B      KA4500
KA2560   SR     R1,R1
         IC     R1,KSQL
         AH     R1,KSQA               GET LENG OF D1 + KEY
         TM     KFLG,KVLN                   CHK IF VLN REC
         BZ     KA2580                      BR IF NOT SO
         SH     R1,KH2
KA2580   STH    R1,KEXPLH             SAVE EXPANSION LENGTH
         B      KA2650                DEFORE BR TO EXPNSN RTN
KA2600   MVC    KEXPLH(2),KHM1        INIT EXPNSN LEN TO KEY ID OF REC
***********************************************************************
**                                                                   **
KA2650   BAL    R11,KEXSR             BRANCH TO EXPANSION RTN
**                                                                   **
***********************************************************************
         B      KA1800                BR IF NORMAL END
*
KA3000   EQU    *                     VLEN/FIXED REC, DATA OPTION
         TM     KFLG,KVLN             CHK IF V-LENG SEGMT
         BO     KA3050                BR IF V-LENG SEGMT
         LA     R2,2(R2)
         LA     R6,2(R6)
KA3050   EQU    *
         SR     R1,R1                 PRE-EXPNSN REC LL1 SAVED
         IC     R1,KSQL
         AH     R1,KSQA
         LTR    R1,R1
         BZ     KAB2991               BR TO ABEND
         BAL    R11,KEXR1             BR TO MOVE DATA
         AR     R2,R1                 UPDATE PTRS TO
         AR     R6,R1
         TM     KFLG,KVLN             CHK IF V-LENG SEGMT
         BZ     KA3100                BR IF FIXED SEGMT
         MVC    KTLL2(2),0(R3)        LENG OF COMPCTED SEGMT
         MVC    0(2,R3),0(R2)         MOVE ORG SEGMT LENG
         LA     R2,2(R2)
         LA     R6,2(R6)
KA3100   EQU    *
         AR     R3,R1                 INPUT/OUTPUT AREAS
         AR     R7,R1
KA3250   CLI    KCMCD,KTLSQ           CHK IF EXPANSION IS TO D1 + KEY
         BNE    KA2600                BR IF NOT SO
         B      KA1800
KA3500   EQU    *
         TM     KFLG,KVLN             CHK IF V-LENG SEGMT
         BZ     KA3550                BR IF FIXED SEGMT
```

```
         TM    KFLGX,KNPRSW          CHK IF NON-PROC SW ON
         BZ    KA3530                BR IF NOT SO
         BAL   R11,KNPSMV            BR TO MOVE NON-PROC SEGMT
         B     KA1800
         SPACE 3
KA3530   EQU   *
         TM    KFLG,KKEY CHK IF KEY OPTION
         BO    KA3550                BR IF KEY OPTION
         BAL   R11,KMVORGXV          MOVE DATA OF V-LENG, DATA OPTION
         B     KA1800
KA3550   EQU   *                     ALL BUT V-LENG, DATA OPTN CMPCTN
         BAL   R11,KMVORG            BR TO MOVE ORG SEG ROUTINE
         B     KA1800
KA3600   EQU   *                  ## CCT SYSTEM DATA CHK RTN
         SR    R1,R1
         IC    R1,KSQL
         AH    R1,KSQA               SQA + SQL
         CH    R1,KSGL               IF SQA+SQL MORE THAN SGL, ERROR
         BNH   KA3900
KA3800   EQU   *
         LH    R1,KABCX92            GET ABEND CODE
         B     KA4500
KA3900   TM    KFLG,KVLN             CHK IF VLEN REC
         BO    KA4400
         LH    R1,KSGL               GET SEGMT LENG
         LTR   R1,R1
         BM    KA3800                ERR IF FIX SGL IS NEGATIVE
KA4400   BR    R11                   RET TO CALLER
KA4500   EQU   *
         STH   R1,KABX
         L     R1,KABCD              GET ABEND CODE
         SVC   13
KMVORGXV EQU   *
         ST    R11,KVRB              SAVE REGS
         L     R2,KASN1              GET IN WORK AREA ADDR
         L     R3,KASN2              GET OUT WORK AREA ADDR
         SR    R1,R1
         IC    R1,KSQL
         AH    R1,KSQA
         BAL   R11,KEXR1             MOVE LL, D, K DATA
KA4650   CLI   KCMCD,KALL            CHK IF ALL EXPANSION
         BE    KA4700                BR IF SO
         LH    R9,0(R2)              GET ORIGNL SEG LENG
         LA    R9,2(R9)              GET NEW SEG LENG AFT COMPRESS
         STH   R9,0(R3)              SAVE  NEW LENG
         AR    R3,R1                 COMPRESSION RTN
         AR    R7,R1
         MVC   0(2,R3),0(R2)         MOVE ORG SEGMT LENGTH
         AR    R2,R1
         AR    R6,R1
         LA    R3,2(R3)              UPDATE OUT PTR + CTR
         LA    R7,2(R7)
         B     KA4800
KA4700   AR    R2,R1
         AR    R6,R1
         MVC   0(2,R3),0(R2)         MOVE ORG SEGMT LENG
         AR    R3,R1                 UPDATE OUT PTR + CTR
         AR    R7,R1
         LA    R2,2(R2)
```

```
         LA    R6,2(R6)
KA4800   STH   R1,KTLLX                  SAVE LL,D,K LENGTH
         L     R1,KASN1                  GET IN AREA
         LH    R1,0(R1)                  GET V-LENG SEGMT LENGTH
         CLI   KCMCD,KALL                CHK IF ALL EXPANSION
         BNE   KA4850                    BR IF COMPRESS
         SH    R1,KH2
KA4850   EQU   *
         SH    R1,KTLLX                  GET D2 LENGTH
         BAL   R11,KEXR1                 MOVE D2 DATA
         AR    R2,R1                     UPDATE IN PTR + CTR
         AR    R6,R1
         AR    R3,R1                     UPDATE OUT PTR + CTR
         AR    R7,R1
         L     R11,KVRB
         BR    R11
KEXR1    EQU   *                         RTN TO MOVE REG 1 DATA TO OUT AREA
         STM   R1,R9,12(13)              SAVE REGS
         LR    R6,R1                     SAVE DATA LENGTH
         LH    R9,KH256
         LH    R1,KH255
KA5000   EQU   *
         CR    R6,R9
         BNH   KA5100                    BR DATA MOVABLE IN 1 EXECUTE
         SR    R6,R9
         EX    R1,KEXMVC                 MOVE PARTIAL DATA
         AR    R2,R9
         AR    R3,R9
         B     KA5000                    BR BACK TO LOOP
KA5100   LR    R1,R6
         BCTR  R1,0
         EX    R1,KEXMVC                 MOVE ALL THE DATA
         LM    R1,R9,12(13)              RESTORE REGS
         BR    R11
         EJECT
*************************************************************************
*                                                                       *
*************************************************************************
**                                                                     **
**       *****   DATA COMPRESSION ROUTINE   *****                      **
**          IN COMPRESSION ROUTINE, DATA REDUNDANT IN 4 BYTES OR       **
**       MORE ARE COMPRESSED IN 3 BYTE CONTROL BLOCK ACCORDING TO      **
**       THE SPECIFICATIONS DESCRIBED IN SPRM.  R2 POINTS TO THE       **
**       BEGINNING OF DATA TO BE COMPRESSED UPON ENTRY.                **
**          REGISTER USAGES ARE LISTED IN THE HEADING SECTION.         **
**                                                                     **
*************************************************************************
KMPSR    DS    0H                        COMPACTION RTN
         SAVE  (14,12)                   SAVE REGS
         ST    R13,KSAV2+4
         LA    R13,KSAV2
         LR    R9,R2                     SET INPUT DATA PTR1
         LA    R8,1                      INCLUDE PTR TO FCCB BYTE
         CH    R6,KTLL1                      CHK ALREADY EOD REACHED
         BL    KB300                         BR IF NOT SO
         L     R1,KFCCB
         MVC   0(2,R1),KX0100                SET FCCB + LCCB
         LA    R3,1(R3)                      UPDATE PTR
         LA    R7,1(R7)                      UPDATE CTR
```

```
         CH    R7,KMAXL                CHK IF MS LENGTH EXCEEDED
         BH    KB4300                  BR  TO MOVE ORIGINAL SEG
         B     KB1700
KB300    CLC   1(3,R9),0(R9)       CHK IF QUALIFIES TO COMPACT
         BE    KB2000              BR IF SO
KB500    LA    R6,1(R6)            UPDATE PTRS, KSN1 PTR
         LA    R8,1(R8)            KN1 CTR
         LA    R9,1(R9)            SPTR
         CH    R6,KTLL1               CHK ALREADY EOD REACHED
         BNL   KB4100              BR IF SO
         CH    R8,KHCMX            CHK CTR IF REACHED MAX NO.
         BL    KB300               BR IF NOT MAX
KB700    L     R1,KFCCB            GE CCB ADDR
         STC   R8,0(R1)            FILL NEXT CCB ADDR
         LR    R1,R8
         TM    KFSW,X'01'          CHK IF 1ST DONE INDICATED
         BZ    KB800               BR IF UNDONE
         SH    R1,KH3                  SUBTRACT CCB LENG
         B     KB900
*
KB800    BCTR  R1,0
         OI    KFSW,X'01'          SET 1ST DONE SW ON
KB900    AR    R7,R1               KN2 CTR
         CH    R7,KMAXL            CHK IF MAX BUF LENGTH USED
         BNL   KB4300              BR IF ALREADY SO
         BCTR  R1,0                    GET MOVE DATA LENG
         EX    R1,KEXMVC           MOVE NON-COMPRESS CHARS
         LA    R1,1(R1)
         AR    R2,R1               UPDATE DATA PTRS/CTRS, KSN1 DATA
         LR    R9,R2               UPDATE IN DATA PTR1
         AR    R3,R1               KSN2 DATA PTR
         ST    R3,KFCCB            REPLACE NEW CCB ADDR
         TM    KEOD,X'01'          CHK IF EOD SW IS ON
         BO    KB4000
KB1500   LA    R7,3(R7)
         CH    R7,KMAXL            CHK IF MAX BUF LENGTH USED
         BNL   KB4300              BR IF ALREADY SO
         MVC   0(3,R3),KF0         ZERO OUT  CCB
         LA    R3,3(R3)
         LA    R8,3
KB1600   TM    KEOD,X'01'          CHK IF EOD REACHED
         BO    KB1700              BR IF SO
         B     KB300
KB1700   L     R1,KASN2            GET KSN2 ORIGINAL ADDR
         STH   R7,KTLL2            SAV KN2 CTR
         CH    R7,KMAXL                CHK IF MS LENGTH EXCEEDED
         BH    KB4300                  BR  TO MOVE ORIGINAL SEG
         MVC   0(2,R1),KTLL2
KB1800   EQU   *
         L     R13,4(R13)
         LM    R14,R12,12(R13)     RESTORE REGS
         SR    R1,R1
         BR    R11                 BR BACK TO CALLER
KB2000   L     R1,KFCCB            GET PTR TO NEXT CCB ADDR
         STC   R8,0(R1)            FILL PTR TO NEXT CCB
         LR    R1,R8
         TM    KFSW,X'01'          CHK IF 1ST SW TO BE SET
         BZ    KB2020              BR IF SO
         SH    R1,KH3
```

```
         B      KB2050
KB2020   BCTR   R1,0
         OI     KFSW,X'01'          SET FIRST-DONE SW
KB2050   LTR    R1,R1               CHK FOR NON-COMPACT CHAR'S
         BNH    KB2300                   BR IF 0 OR NEGATIVE
         AR     R7,R1
         CH     R7,KMAXL            CHK IF MAX BUF LENGTH USED
         BNL    KB4300              BR IF ALREADY SO
         BCTR   R1,0
         EX     R1,KEXMVC           MOVE NON-COMPACT CHARS
         LA     R1,1(R1)
         AR     R2,R1               UPDATE PTRS & CTRS
         AR     R3,R1
         SR     R8,R8
KB2300   MVC    0(3,R3),KF0         MOVE CCB PRE-CMPACTION
         LA     R1,0(R3)
         ST     R1,KFCCB
         LA     R3,3(R3)            UPDATE KSN2 PTR
         LA     R7,3(R7)            UPDATE KN2 CTR
         CH     R7,KMAXL            CHK IF MAX BUF LENGTH USED
         BNL    KB4300              BR IF ALREADY SO
         LA     R8,3                RESET NEXT CCB CTR
KB2700   LA     R8,4(R8)            INCREMENT OF CTR FOR 4 CHARS
         LA     R10,3(R9)
         LA     R9,4(R9)            AND PTR
         LA     R6,4(R6)
         LR     R1,R6
         CH     R1,KTLL1            CHK IF EXCEED SEGMENT LENGTH
         BL     KB3000
KB2900   EQU    *
         SH     R6,KH4              RESET KSN1 PTR
         SH     R8,KH4              RESET CTR TO NEXT CCB
         SH     R9,KH4              RESET CURRENT DATA PTR
KB3000   CLC    0(1,R9),0(R10)      COMPARE CHARS BEYOND 4 CHARS
         BE     KB3400
KB3100   L      R1,KFCCB
         MVC    2(1,R1),0(R2)       SAVE REDUNDANT CHARS
         SH     R8,KH3
         STC    R8,1(R1)
         LR     R2,R9
         LA     R8,3
         CH     R6,KTLL1                CHK ALREADY EOD REACHED
         BL     KB300
         B      KB4000
KB3400   LA     R8,1(R8)            UPDATE CCB PTR
         LA     R9,1(R9)            UPDATE CUR DATA PTR
         LA     R6,1(R6)            UPDATE KN1 CTR
         CH     R6,KTLL1                CHK ALREADY EOD REACHED
         BNL    KB3600              BR IF SO
         CH     R8,KH258            CHK IF CTR MAX VAL REACHED
         BL     KB3000
KB3520   L      R1,KFCCB            GET CCB PTR
         MVC    2(1,R1),0(R2)       MOVE REDUNDANT CHAR
         SH     R8,KH3
         STC    R8,1(R1)            FILL LENGTH OF REDUNDANT CHARS
         LA     R8,3
         LR     R2,R9
         B      KB300
KB3600   OI     KEOD,X'01'          SET EOD SW ON
```

```
         L      R1,KFCCB              GET ADDR OF CCB PTR
         MVI    0(R1),X'03'           SAVE CTR TO CCB
         MVC    2(1,R1),0(R2)         SAVE REPEAT CHAR
         SH     R8,KH3
         STC    R8,1(R1)
KB4000   MVI    0(R3),X'00'           INSERT EOD CCB 0
         LA     R7,1(R7)              UPDATE PTR/CTR OF OUTPUT
         LA     R3,1(R3)
         B      KB1600
KB4100   OI     KEOD,X'01'            SET EOD SW ON
         B      KB700
KB4300   EQU    *
         TM     KFLG,KVLN             CHK IF V-LENG SEGMT
         BZ     KB4350                BR IF FIXED LENG SEGMT
         TM     KFLGX,KNPRSW          CHK IF NON-PROCESS SEGMT
         BZ     KB4330                BR IF NOT SO
         BAL    R11,KNPSMV            BR TO NON-PROC SEGMT
         B      KB1800
KB4330   EQU    *
         TM     KFLG,KKEY             CHK IF KEY OPTN
         BO     KB4350                BR IF SO
         BAL    R11,KMVORGXV          MOVE DATA OF V-LENG DATA OPTN
         B      KB1800
KB4350   EQU    *
         BAL    R11,KMVORG            BR TO MOVE ORG SGMENT RTN
         B      KB1800                BR TO END OF RTN
*
         SPACE 3
KMVORG   EQU    *                     MOVE ORG SEGMENT RTN
         L      R2,KASN1              GET START ADDR OF IN-DATA
         L      R3,KASN2              GET START ADDR OF OUT-DATA
         LH     R1,KMAXL              GET MAX LENGTH
         LA     R1,2(R1)
KB4400   EQU    *
         STH    R1,0(R3)
         LA     R3,2(R3)
         LH     R6,KMAXL              GET MAX LENG OF RECORD
         LH     R9,KH256
         LH     R1,KH255
KB4600   CR     R6,R9                 CHK IF REC IS MORE THAN 1 MOVE
         BNH    KB4900                BR IF NOT SO
         SR     R6,R9
         EX     R1,KEXMVC             MOVE 1 GROUP DATA
         AR     R2,R9                 UPDATE IN-BUFF PTR
         AR     R3,R9                 UPDATE OUT-BUFF PTR
         B      KB4600                BR BACK TO EOD
KB4900   LR     R1,R6                 GET LAST DATA
         BCTR   R1,0
         EX     R1,KEXMVC             MOVE LAST DATA
         BR     R11                   RETURN TO CALLER
         SPACE 3
KNPSMV   EQU    *
         ST     R11,KVRBY             SAVE RET ADDR
         L      R2,KASN1              GET IN AREA ADDR
         LH     R1,0(R2)              GET LENGTH
         LH     R3,KSGL               GET SEGMT LENGTH
         SH     R3,KH2
         CR     R1,R3                 CHK IF LENGTH FALLS IN LAST 2 BYTES
         BH     KB5500                BR IF SO
```

```
         LR    R3,R1                    GET SEG LENGTH
         AR    R3,R2                    PTR TO EOS
         MVC   0(2,R3),KH0              MOVE PADDING CHARS
         LH    R1,KSGL
         BCTR  R1,0
KB5500   STH   R1,KTLLX                 SAVE NEW SEGMT LENGTH
         L     R3,KASN2                 GET OUT AREA ADDR
         BAL   R11,KEXR1                BR TO MOVE DATA
         MVC   0(2,R3),KTLLX            MOVE NEW LENGTH
         L     R11,KVRBY
         BR    R11                      BR BACK TO CALLER
         EJECT
*********************************************************************
*********************************************************************
**                                                                 **
**       *****   DATA EXPANSION ROUTINE   *****                    **
**           EXPANSION ROUTINE REVERSES THE COMPRESSION PROCESS.   **
**       ALL COMPRESSION CONTROL BLOCKS ARE DE-CODED AND RESTORED  **
**       TO A NORMAL DATA STREAM.  REGISTER USAGES ARE AS LISTED   **
**       IN THE HEADING SECTION OF THIS PROGRAM.                   **
**                                                                 **
**                                                                 **
*********************************************************************
*                                                                   *
KEXSR    DS    0H                       EXPANSION SUB ROUTINE
         SAVE  (14,12)                  SAVE REGS
         ST    R13,KSAV2+4
         LA    R13,KSAV2
KC200    SR    R1,R1
         IC    R1,0(R2)                 CHK NXT CCB OFFSET
         CH    R1,KH1
         BE    KC600
         BL    KC300
         LA    R2,1(R2)                 UPDATE INPUT PTR
         LA    R6,1(R6)
         SH    R1,KH2
         EX    R1,KEXMVC                MOVE CHARS TILL NXT CCB
         LA    R1,1(R1)
         AR    R3,R1                    UPDATE OUT PTR
         AR    R7,R1                    UPDATE OUT CTR
         AR    R2,R1                    UPDATE IN PTR
         AR    R6,R1                    UPDATE IN CTR
         B     KC700
KC300    EQU   *
         SR    R7,R7
         B     KC1000
KC600    LA    R2,1(R2)                 UPDATE INPUT PTR
         LA    R6,1(R6)                 UPDATE INPUT CTR
KC700    EQU   *
         LH    R1,KEXPLH                CHK IF ALL SEG OR KEY ONLY
         LTR   R1,R1
         BM    KC900
         LA    R7,0(R7)
         CH    R7,KEXPLH                CHK IF ALL SEG OR KEY ONLY
         BNL   KC1000
KC900    CLI   0(R2),KQXOF              CHK IF EOD CCB REACHED
         BNE   KC1200
KC1000   EQU   *
         TM    KFLG,KVLN
```

```
         BZ     KC1100
         L      R1,KASN2
         L      R8,KASN1               GET INPUT ADDR
         TM     KFLG,KKEY                   CHK IF KEY OPTION
         BZ     KC1100                      BR IF DATA OPTION
         MVC    0(2,R1),2(R8)          INSERT SEGMENT LENGTH
KC1100   EQU    *
         L      R13,4(R13)             RESTORE REGS
         LM     R14,R12,12(R13)
         SR     R1,R1
         BR     R11                    BR BACK TO CALLER
KC1200   CLI    1(R2),KQXOF            CHK IF SKIP CCB
         BE     KC1700                 BR IF SO
         SR     R1,R1
         IC     R1,1(R2)               GET LRC LENGTH
         SH     R1,KH2                 GET EX MOVE LENGTH
         MVC    0(1,R3),2(R2)          MOVE 1 CHAR TO OUT AREA
         EX     R1,KEXEXP              EXPAND CHARS
KC1500   LA     R1,2(R1)               UPDATE CTR/PTR
         AR     R3,R1                  OUTPUT PTR
         AR     R7,R1                  OUTPUT CTR
KC1700   SR     R1,R1
         IC     R1,0(R2)               GET NXT CCB PTR OFFSET
KC1800   CH     R1,KH3                 CHK IF ITS BACK TO BACK
         BH     KC2000
         LA     R2,3(R2)               UPDATE IN PTR
         LA     R6,3(R6)               UPDATE IN CTR
         B      KC700
KC2000   SH     R1,KH4
         LA     R2,3(R2)
         LA     R6,3(R6)
         EX     R1,KEXMVC              MOVE CHARS TO OUTAREA
KC2100   LA     R1,1(R1)               UPDATE PTR/CTR
         AR     R3,R1                  UPDATE OUTPUT DATA PTR/CTR
         AR     R7,R1
         AR     R2,R1                  UPDATE INPUT DATA PTR/CTR
         AR     R6,R1
         B      KC700
         SPACE 3
KMVORGX  ST     R11,KVRB               SAVE RET ADDR
         L      R2,KASN1               GET INPUT ADDR
         L      R3,KASN2               GET OUTPUT ADDR
         TM     KFLGX,KVLDT        CHK IF V-LEN, DATA OPTION
         BO     KC2400                 BR IF SO
         LA     R2,2(R2)
KC2400   EQU    *
         TM     KCMCD,KALL             CHK IF ALL EXPAND
         BO     KC2600                      BR IF ALL
         SR     R6,R6
         IC     R6,KSQL                     GET KEY LEN
         AH     R6,KSQA                     AND OFF-SET
         B      KC2900
KC2600   EQU    *
         TM     KFLG,KVLN              CHK IF V-LEN
         BO     KC2700                 BR IF SO
         LH     R6,KSGL                     GET SEG LENG
         B      KC2900
KC2700   EQU    *
         LH     R6,0(R2)               GET V-LEN
```

```
KC2900   LH     R9,KH256
         LH     R1,KH255
         BAL    R11,KB4600           BR TO MOVE DATA
         TM     KFLGX,KVLDT          CHK IF V-LEN, DATA OPTION
         BZ     KC3000               BR IF NOT SO
         CLI    KCMCD,KTLSQ          CHK IF PARTIAL EXPANSION
         BNE    KC3000               BR IF NOT SO
         L      R1,KASN1
         L      R9,KASN2
         LH     R1,0(R1)             GET IN DATA LENGTH
         SH     R1,KH2               GET ORIGINAL LENGTH
         STH    R1,0(R9)             SAVE IN OUT AREA
KC3000   EQU    *
         L      R11,KVRB
         BR     R11                RET TO CALLER
         SPACE 3
         EJECT
***********************************************************************
*                                                                     *
R0       EQU    0
R1       EQU    1
R2       EQU    2
R3       EQU    3
R4       EQU    4
R5       EQU    5
R6       EQU    6
R7       EQU    7
R8       EQU    8
R9       EQU    9
R10      EQU    10
R11      EQU    11
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
KOCMP    EQU    X'00'                                                 #
KOEXP    EQU    X'04'                                                 #
KX0100   DC     X'0100'                  CONSTANT
KH0      DC     H'0'
KH2      DC     H'2'
KH3      DC     H'3'
KHM1     DC     H'-1'
KHCMX    DC     H'255'
KH1      DC     H'1'
KH4      DC     H'4'
KOXON    EQU    X'01'
KOXOF    EQU    X'00'
KSAV1    DS     18F
KSAV2    DS     18F
KAB2992  EQU    KA3800
         DS     0F
KABCX89  DC     H'2989'              ABEND CODE 2989
KABCX90  DC     H'2990'              ABEND CODE 2990
KABCX91  DC     H'2991'              ABEND CODE 2991
KABCX92  DC     H'2992'              ABEND CODE 2992
KABCD    DC     X'8000'              ABEND CODE 1
KABX     DC     H'0'                 ABEND CODE 2
KVRB     DS     F                    RET ADDR SAVE AREA
KVRB1    DS     F                    REG SAVE AREA
```

```
KVRBY    DS    F                         REG SAVE AREA
KFO      DC    F'0'
***********************************************************************
*                                                                     *
*                                                                     *
KNITA    DS    0F                        INIT AREA
KMAXL    DS    H                         MAX LENGTH OF OUTPUT BUFFER
KTLLX    DS    H                         WORK AREA
KTLL1    DS    H                         LENGTH OF THE INPUT SEG
KTLL2    DS    H                         NEW SEGL AFTER CMPCT/EXPNSION
KEOD     DS    X
KFSW     DS    X
KEXPLH   DS    H
KFCCB    DS    A                         PTR TO 1ST CCB
KASN1    DS    A                         INPUT BUFFER ADDR
KASN2    DS    A                         ADDR OF OUTPUT AREA
KCMCD    DS    X                         PASSED CMND CODE
KFLGX    DS    X                         FLAG AREA
KVLDT    EQU   X'80'                     V-LEN SEG, DATA OPTION
KNPRSW   EQU   X'40'                     NO COMPRESSION SEGMT
KWNEG    EQU   X'20'                     FORCED ERR - V-LENG=NEGATIVE NO.  #
KNITZ    EQU   *
*                                                                     *
*                                                                     *
***********************************************************************
KNITL    EQU   KNITZ-KNITA
KALL     EQU   X'04'                     CMD CODE=EXPND ALL
KTLSQ    EQU   X'08'                     CMD CODE=EXPND TILL KEY
KQINIT   EQU   X'0C'                     COMMAND CODE  FOR AFTER OPEN
KH255    DC    H'255'                    CONSTANT
KH256    DC    H'256'                    CONSTANT
KH258    DC    H'258'                    CONSTANT
KNEGNO   EQU   X'80'                     NEGATIVE SIGN                     #
KEXMVC   MVC   0(0,R3),0(R2)
KEXEXP   MVC   1(0,R3),0(R3)
KEXBF    XC    0(0,R3),0(R3)             CLEAR KSN2 BUFF
KZZ      EQU   *
         LTORG
**********************************************************************
**                                                                  **
**   ***** DSECT OF SEGMENT COMPRESSION CONTROL SECTION(SEGPAC) ***** **
**                                                                  **
**********************************************************************
* CMPACT TAB
KCCB     DSECT
KSGN     DS    CL8                       SEGMENT NAME
KRTN     DS    CL8                       CMPRS RTN NAME
KEP      DS    A                         ENTRY POINT
KFLG     DS    X                         FLAG
KSQL     DS    X                         KEY LENGTH
KSQA     DS    H                         OFFSET TO KEY
KSGL     DS    H                         SEGMENT LENGTH
KTBL     DS    H                         TAB LENGTH
KSEQ     EQU   X'08'                           SEQ FLD DEFINED
KVLN     EQU   X'04'                           RECFM= VAR LENGTH
KKEY     EQU   X'02'                           KEY OPTION SPECIFIED
KNIT     EQU   X'01'                           INIT OPTION SPECIFIED
         END
/*
```

HDAM RANDOMIZING MODULES

The DL/I access method called HDAM requires the IMS/VS user to supply a module for placing root segments in, or retrieving them from, an HDAM data base. One or more such modules, called randomizing modules, can be used within the IMS/VS system. Any one data base has only one randomizing module associated with it.

A randomizing module is a module that uses a mathematical technique to convert a key into an address. The same key will always convert to the same address. The randomizing module required by IMS/VS must convert an SSA (segment search argument) key field value into a relative block number and anchor point number. The SSA key field value is supplied by an application program for root segment placement in, or retrieval from, an HDAM data base.

A generalized module, which uses DBD generation-supplied parameters to perform randomizing for a particular data base, can be written to service several data bases.

After a randomizing module has been compiled and tested, and before its use by the IMS/VS system, it must be placed into the IMSVS.RESLIB data set. Each randomizing module must have a unique name. The name must not conflict with the existing members of the IMSVS.RESLIB data set. Alternative locations for randomizing module storage are SYS1.LINKLIB, or any operating system partitioned data set to which access is provided with a JOBLIB or STEPLIB JCL statement.

The name given to the load module used for randomizing functions with a specific data base should also appear in the DBD generation associated with the data base. The load module name must be the value of the "mod" parameter of the RMNAME= operand on the DBD statement in the HDAM DBD generation.

The necessary randomizing module associated with a specific data base is brought into main storage in either the IMS/VS online control program region, or batch processing region, at the time the associated data base is opened. If a single randomizing module is used for more than one HDAM data base, it must be written, compiled, and link edited as reenterable (RENT). It can also be placed in the LPA (linkpack area). This allows one copy of the module to service several data bases that are concurrently open.

When an HDAM data base is to be used in either the IMS/VS online control region, or a DL/I batch processing region, and the randomizing module does not exist in OS/VS LPA, space must be provided for it. Space must be provided in the IMS/VS control region to accommodate all randomizing modules that can be used for online HDAM data bases.

All randomizing modules are loaded from their resident library by the IMS/VS OPEN module, DFSDLOC0. The IMS/VS OPEN module obtains the name of the randomizing module from the RDMVTAB control block. This block is constructed by the utility block builder program and placed in IMSVS.ACBLIB from parameters specified in the associated DBD. If the IMSVS.ACBLIB data set is not being used, the block is constructed in main storage and passed to the IMS/VS OPEN module. The IMS IMODULE macro instruction is used.

When an application program issues a Get Unique, Get Next with Qualification, or Insert call which operates on a root segment of an HDAM data base, the user-supplied randomizing module is invoked. The SSA and the segment I/O work area, in the data base call relating to the sequence field of a root segment, provide the primary input parameter to the randomizing module. The following illustrates the format of an SSA:

ROOT SEGMENT NAME (SEQUENCE FIELD NAME-OPERATOR-value)

The root segment and sequence field names are eight-character alphameric values. The operator is a two-character arithmetic value. A description is provided in the IMS/VS Application Programming Reference Manual. Other operators at the root level give unpredictable results. The value parameter is a term whose length equals the length of a root segment sequence field in the data bases and whose content defines an already existent root segment to be retrieved. If the data base call consists of a root segment insert, the SSA consists only of the segment name. In this case, the field value is obtained from the segment I/O area provided in the insert call.

This field value parameter is supplied to the randomizing module for conversion to a relative block number and anchor point number within the data base. In addition to the field value parameter supplied by an application program, parameters from the DBD generation associated with the data base being used are available to the randomizing module.

When a randomizing module is invoked for the purposes of conversion, control is passed from the IMS/VS data base logical retrieve function module, DFSDLR00.

The parameters from DBD generation are available to a randomizing module in a CSECT named RDMVTAB. The address of this CSECT is passed to the module each time a conversion is requested.

This control section is placed at the end of the DBD module and contains information such as the randomizing routine name, anchor point information, and the total length of that control section. Each control section can be extended by the user to contain any desired data or algorithm information by an assembly and link edit process.

The first 32 bytes are constants defined by DBDGEN. When the new table is defined by the user to include additional parameters, these fields must be duplicated. The only exception to this rule is that the CSECT length field must be properly updated to reflect the new length. After an assembly of the new table, a link edit can be done to exchange the new table for the old one. Care must be taken to use an ENTRY statement specifying the name of the DBD when this operation takes place. See "Automatic CSECT Replacement" in OS/370 Linkage Editor and Loader for additional details.

The following DSECT defines the format of this CSECT:

```
DMBDACS    DSECT
DMBDANME   DS     CL8    NAME OF ADDR ALGORITHM LOAD MODULE
DMBDAKL    DS     OCL1   EXECUTABLE KEY LENGTH OF ROOT
DMBDAEP    DS     A      EP OF ADDR LOAD MODULE
DMBDASZE   DS     H      SIZE OF THIS CSECT
DMBDARAP   DS     H      NUMBER OF ROOT ANCHOR POINTS/BLOCK
DMBDABLK   DS     F      NUMBER OF HIGHEST BLOCK DIRECTLY ADDRSD
DMBDABYM   DS     F      MAX NUMBER OF BYTES BEFORE OFLOW TO 2NDARY
DMBDABYC   DS     F      CUR NUM OF BYTES INSERTED UNDER ROOT
DMBDACP    DS     F      RESULT OF LAST ADDRESS CONVERSION
```

RANDOMIZING MODULE INTERFACES

Upon entry to any randomizing module, registers must be saved. Upon return to IMS/VS, registers must be restored. A save area address is provided in register 13 upon entry for the purpose of saving the registers.

The following registers, on entry to a randomizing module, have the indicated meanings:

Register	Meaning or Content
0	Data Management Block address (DMB).
1	DMBDACS CSECT address.
7	Partition Specification Table address (PST).
9	Address of first byte of key field value supplied by an application program.
13	Save area address. The first three words in the save area must not be changed.
14	Return to IMS/VS address.
15	Entry point address of randomizing module.

Notes:

1. If an HDAM data base does not have a sequence field defined, the values supplied to the randomizing module are as follows:

   a. The executable key length field in the CSECT named RDMVTAB is not initialized and should not be used.

   b. The value in register 9 at entry to the randomizing module contains the address of the first byte of the user I/O area.

2. If an HDAM data base does not have a sequence field defined at the root level, the randomizing module is given control only on an insert call. All retrieval type calls result in a scan mode operation to satisfy the root level qualification. On GU type calls, the scan starts at the beginning of the data base. On GN type calls, the scan starts at the current root level position within the data base.

Internal IMS/VS control blocks that are of value to a randomizing routine are: the Partition Specification Table (PST), the Data Management Block (DMB), the Physical Segment Description Block (PSDB) for the root segment, and the first Field Description Block (FDB). The FDB is the root segment key field format description. DSECTs of these blocks are provided in the examples shown later in this chapter.

The result of a randomizing module conversion must be in the form:

BBBR

where:

BBB
: is a three-byte binary number of the block into which a root segment is to be inserted, or from which it is to be retrieved.

R
: is a one-byte binary number of the appropriate anchor point, within a relative block, within an OSAM data set of the data base.

This result must be placed in the CSECT addressed by register 1 in the four-byte fixed name DMBDACP. If the result exceeds the content of the field DMBDABLK, the result is changed to the highest block and last anchor point of that block.

HDAM RANDOMIZING MODULE EXAMPLES

Four randomizing module examples are provided as guidance to the IMS/VS system user. The four modules (DFSHDC10, DFSHDC20, DFSHDC30, and DFSHDC40) are linked into the IMSVS.RESLIB data set during system definition. The examples use the following techniques:

- Modulo or division method
- Binary halving method
- Hashing method

The intent of a randomizing module is to convert a root segment key field value to a relative block number and anchor point number in an HDAM data base. The relative block number may range from 1 to $2^{24}-1$. The anchor point number may range from 1 to 255.

Modulo or Division Method Example (DFSHDC10)

This randomizing module uses the principle that the remainder of a division can only range from zero to the divisor minus one. Thus, any number divided by four can only yield a remainder of 0, 1, 2, or 3. To determine the base location for a root segment, multiply the number of blocks in the root segment addressable area by the number of anchor points per block. This is effectively the number of base locations for root segments in the root segment addressable area. Then, divide the root segment key field value by the result of the multiplication. The remainder indicates the appropriate base location.

To convert the base location to relative block and anchor point numbers, divide the base location by the number of anchor points per block. This last division leaves the relative block number as the quotient and the anchor point number as the remainder. Since both numbers are relative to zero, both must be incremented by one to yield the correct block and anchor point.

Example:

Assume a) Root segment addressable area is 50 blocks.

b) 2 anchor points per block.

c) Root segment key value is 23.

Result a) Number of base locations = 50 x 2 = 100.

b) Appropriate base location = 23/100 = 23 remainder.

c) Appropriate block = 23/2 = 11 (the quotient), appropriate anchor point = 1 (the remainder).

d) Adjust both numbers by one; thus, relative block = 12 and anchor point = 2.

Notice that external keys 123, 223, 323, etc. will be synonyms. As the number of base locations is increased, the distance between root segments increases. This may waste direct access space. However, the number of synonyms decreases as the number of base locations approaches or exceeds the largest key value. When the root segment key field value is numeric, and the number of base locations equals or exceeds the largest key value, no synonyms are produced.

```
STMT   SOURCE STATEMENT

 HDCNVRT1 CSECT
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                       *
 *                 S A M P L E   C O N V E R S I O N   P R O G R A M    *
 *                                                                       *
 *              THIS CSECT CONVERTS AN EBCDIC NUMERIC KEY TO A RELATIVE*
 *        BLOCK AND ROOT ANCHOR POINT. THIS RESULT IS OBTAINED AS      *
 *        FOLLOWS  RECNO= MOD(KEY,DMBDABLK*DMBDARAP)                   *
 *                 BLOCK= RECNO/DMBDARAP+1                              *
 *                 RAP  = MOD(RECNO,DMBDARAP)+1                         *
 *                                                                       *
 *              THE CSECT ASSUMES THAT THE EXTERNAL KEY IS 15 BYTES OR *
 *        LESS.  NON-NUMERIC CHARACTERS ARE VALID, HOWEVER ONLY THE    *
 *        FOUR LOW ORDER BITS WILL BE USED.                             *
 *                                                                       *
 *        CALLING SEQUENCE                                               *
 *              R0 - DMB                                                 *
 *              R1 - DMBDACS                                             *
 *              R7 - PST                                                 *
 *              R9 - KEY ADDRESS                                         *
 *        ON RETURN                                                      *
 *              DMBDACP - BBBR                                           *
 *                                                                       *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          STM   14,12,12(13)        SAVE
          USING PST,R7
          USING DMBDACS,R1
          USING HDCNVRT1,R15
          XC    PSTDECB(8),PSTDECB INIT FOR CVB
          IC    R5,DMBDAKL          GET EXECUTABLE KEY FLD LENGTH
          EX    R5,PACK
          SR    R4,R4
          OI    PSTDECB+7,X'0F'     FORCE SIGN
          SR    R8,R8
 COMPARE  EQU   *
          CP    PSTDECB(8),MAXP(6)  IS NUMBER TOO LARGE FOR CVB
          BH    DECR                YES, BRANCH
          CVB   R5,PSTDECB
          B     ALMOST              FINISH UP
 DECR     EQU   *
          SP    PSTDECB(8),MAXP(6)  DECR NUMBER BY 2147483647
          AL    R8,MAXB             INCR REG 8 BY SAME AMOUNT
          BC    CARRY,CARRY1        BR IF CARRY OUT OF REG
          B     COMPARE             OTHERWISE COMPARE AGAIN
 CARRY1   EQU   *
          LA    R4,1(,R4)           TAKE CARE OF CARRY
          B     COMPARE             GO COMPARE
 ALMOST   EQU   *
          ALR   R5,R8               PUT IF ALL TOGETHER
          BC    NOCARRY,DONE        IF NO CARRY, WE ARE DONE
          LA    R4,1(,R4)           ELSE, TAKE CARE OF CARRY
 *                                  EVEN-ODD PAIR 4,5 HAVE
 *                                  CONVERTED NUMBER
 DONE     EQU   *
          L     R6,DMBDABLK         HIGEST BLOCK NUMBER DIRECTLY ADDR
```

```
STMT   SOURCE STATEMENT

  57            MH    R6,DMBDARAP          HIGHEST RECORD NUMBER
  58            DR    R4,R6
  59            LR    R5,R4                RECNUM
  60            SR    R4,R4
  61            LH    R6,DMBDARAP
  62            DR    R4,R6
  63            LA    R4,1(,R4)            ROOT ANCHOR POINT
  64            LA    R5,1(,R5)            BLOCK
  65            SLL   R5,8
  66            OR    R4,R5                BBBR
  67            ST    R4,DMBDACP           RESULT
  68            LM    14,12,12(13)         RESTORE
  69            BR    R14                  RETURN
  70 PACK       PACK  PSTDECB(8),0(0,R9)
  71            REQUATE
  72+**********************************************************************
  73+*                                                                    *
  74+*                    REGISTER EQUATES                                *
  75+*                                                                    *
  76+**********************************************************************


  78+R0         EQU   0
  79+R1         EQU   1
  80+R2         EQU   2
  81+R3         EQU   3
  82+R4         EQU   4
  83+R5         EQU   5
  84+R6         EQU   6
  85+R7         EQU   7
  86+R8         EQU   8
  87+R9         EQU   9
  88+R10        EQU   10
  89+R11        EQU   11
  90+R12        EQU   12
  91+R13        EQU   13
  92+R14        EQU   14
  93+R15        EQU   15
  95 CARRY      EQU   3
  96 NOCARRY    EQU   12
  97 *
  98 MAXP       DC    P'2147483647'
  99 MAXB       DC    F'2147483647'          MAX SIGNED 32-BIT NUMBER
 100            IDLI  PSTBASE=0,DMBBASE=0
                END
```

Binary Halving Method Example (DFSHDC20)

This module attempts to distribute root segments across the root segment addressable area, according to the bit pattern of a root segment key field value after it has been converted to a binary value. This distribution is performed as follows:

A result register is set to zero. After a key field value has been converted to binary, the number of base locations (number of blocks in the root segment addressable area times number of anchor points per block) is computed and divided by two. The low-order bit of the converted key field value is tested for one. If equal to one, the current number of base locations is added to the result register. If the low-order bit is zero, no addition to the result register is performed.

The number of remaining base locations is again divided by two and the quotient tested for zero. If other than zero, the next higher bit position in the converted key field is tested for a one or zero and the appropriate action taken. This process continues until the number of remaining base locations divided by two yields a quotient of zero. At this point, the appropriate base location is in the result register. In order to produce the proper relative block number and anchor point number, divide by the number of anchor points per block. The division yields a quotient of relative block number, and remainder of anchor point number. As in the module method, the results are relative to zero and must be incremented by one to yield the appropriate values.

Example:

Assume a) 10 blocks in root segment addressable area.

b) 2 anchor points per block.

c) Root segment key field value of 29.

After initialization:

Converted Key Field	No. of Remaining Base Locations	Result Register
1 1 1 0 1	(10x2)/2 = 10	0
After bit tested		
. . . . x	10	10
. . . x .	5	10
. . x . .	2	12
. x . . .	1	13

At this point, the number of remaining base locations is reduced to zero. Hence, the appropriate base location is 13. To get the actual relative block number and root anchor point, divide 13 by 2 and add 1 to both the quotient and the remainder. This results in a relative block number of 7 and an anchor point number of 2.

Notice that the number of base locations determines when testing ceases. Hence, in this example, all key field values ending in the same four bits will be synonyms. Additional bits of the key are tested when the number of base locations exceeds another power of two. If the number of base locations is not a power of two, some of the base locations are never used.

The major advantage of this method is that the relative order of root segment placement is disturbed very little when the number of base locations is changed.

```
STMT    SOURCE STATEMENT                                              F15OCT70

   3 *
   4 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   5 *                                                                       *
   6 *                  B I N A R Y   H A L V I N G   C O N V E R T          *
   7 *                                                                       *
   8 *                 THIS CSECT DETERMINES THE RELATIVE BLOCK AND ROOT     *
   9 *           ANCHOR POINT BY A BINARY HALVING TECHNIQUE. THIS APPROACH   *
  10 *           IS SLOWER THAN THE MODULO SCHEMES, BUT IT DOES TEND TO KEEP *
  11 *           THE SAME PHYSICAL SEQUENCE WHEN THE NUMBER OF ADDRESSABLE   *
  12 *           BLOCKS IS CHANGED. SINCE THE ROUTINE USES SHIFTS ON INTEGER *
  13 *           NUMBERS, SOME RECORD NUMBERS WILL BE INACCESSABLE IF THE    *
  14 *           TOTAL NUMBER OF DIRECTLY ADDRESSABLE RECORDS (BLOCKS*ROOT   *
  15 *           ANCHOR POINTS) IS NOT A POWER OF 2                          *
  16 *                                                                       *
  17 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  18           STM   14,12,12(13)
  19           USING PST,R7
  20           USING DMBDACS,R1
  21           USING DFSHDC20,R15
  22           XC    PSTDECB(8),PSTDECB  INIT FOR CVB
  23           IC    R5,DMBDAKL         GET EX KEY LENGTH
  24           EX    R5,PACK
  25           OI    PSTDECB+7,X'0F'    FORCE VALID SIGN
  26           CVB   R2,PSTDECB
  27           L     R4,DMBDABLK
  28           MH    R4,DMBDARAP        HIGHEST RECORD IN RANGE
  29           SR    R5,R5              CLEAR RESULT REG
  30 CVTLP     SRL   R4,1               CUT RANGE IN HALF
  31           LTR   R4,R4              RANGE EXHAUSTED
  32           BZ    XIT                YES
  33           SR    R3,R3              NO
  34           SRDL  R2,1               TEST MASK FOR 1
  35           LTR   R3,R3
  36           BZ    CVTLP              NO ONE
  37           BXH   R5,R4,CVTLP        ONE - ADD IN RANGE
  38 XIT       DS    0H
  39           LH    R6,DMBDARAP
  40           DR    R4,R6
  41           LA    R4,1(,R4)          ROOT ANCHOR POINT
  42           LA    R5,1(,R5)          BLOCK
  43           SLL   R5,8
  44           OR    R4,R5
  45           ST    R4,DMBDACP         RESULT
  46           LM    14,12,12(13)
  47           BR    R14
  48 PACK      PACK  PSTDECB(8),0(0,R9)
  49           PRINT NOGEN
  50           IDLI  PSTBASE=C,DMBBASE=0
  95           REQUATE
  20           END
```

Hashing Method Example (DFSHDC30)

This method uses a shift and add technique to develop a 31-bit binary number which has a fairly even distribution from 0 to 2^{31}. The number is developed as follows:

The result register is initialized to zero. The first character of a key field value is added to the result register and the register is shifted left three hexadecimal digits. The bits of the register shifted left and off the register are then added back to the register containing the previous shift result. This partial result is tested for odd or even. If odd, the contents of the register are complemented. The original character is then added to the register. This process is repeated for each character in the key field value. Instead of starting off with a zero content in the result register, the result of the previous content is used. When the key field value characters are exhausted, the result is adjusted to guarantee a 31-bit positive result.

Example:

Assume a) Key field value = ABCD

Key Character	Result Register	
A	0C100000	After test for complement
	0C10C100	After completion of A
B	1C20C10C	After test for complement
	1CE1C20C	After completion of B
C	2CF1CE1C	After test for complement
	EDF2CF1C	After completion of C
D	FE0EDF2C	After test for complement
	FF0FE0ED	After completion of D
	7F0FE0ED	Positive number

The result can then be used as input to the modulo or binary halving technique. The latter technique is used in this example.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    S A M P L E    H A S H I N G    T E C H N I Q U E*
*                                                                        *
*                 THIS CSECT IS A ONE METHOD OF HASHING AN EXTERNAL KEY  *
*           INTO A 31 BIT BINARY NUMBER WHICH CAN THEN BE USED AS INPUT  *
*           TO THE BINARY HALVING ADDRESSES RESOLUTION OR A MODULO SCHEME*
*           TO DETERMINE THE BLOCK AND ROOT ANCHOR POINT.                *
*                 THIS ROUTINE PLACES FEW RESTRICTIONS ON THE EXTERNAL   *
*           KEY E.G. IT CAN BE 156 BYTES LONG, IT CAN CONTAIN ANY BIT    *
*           PATTERN. THE KEY SHOULD BE LONGER THAN 3 CHARACTERS TO INSURE*
*           SOME SPREADING, HOWEVER IT WILL WORK ON SHORTER KEYS.        *
*                                                                        *
*           CALLING SEQUENCE                                             *
*                 R0 - DMB                                               *
*                  1 - DMBDACS                                           *
*                  7 - PST                                               *
*                  9 - KEY ADDRESS                                       *
*           ON RETURN                                                    *
*                 DMBDACP - BBBR                                         *
*                                                                        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
DFSHDC30 CSECT
         STM   R14,R12,12(R13)
         USING DFSHDC30,R15
         USING DMBDACS,R1
         SR    R12,R12
         BCTR  R12,0                 SET TO ALL FF S
         SR    R11,R11
         LA    R9,0(,R9)             CLEAR ANY HIGH ORDER BITS
         SR    R7,R7                 INIT
         IC    R7,DMBDAKL                FOR
         AR    R7,R9                        LATER
         LA    R6,1                              BXLE
         SR    R2,R2
LOOP     DS    0H
         IC    R11,0(,R9)            GET GROUP OF 8 BITS
         ALR   R2,R11                ADD TO HASH
         SR    R3,R3
         SRDL  R2,12                 BREAK UP CHAR PATTERNS
         OR    R2,R3                 ADD INTO HIGH PORTION
         STC   R2,DMBDACP            COMPLEMENT
         TM    DMBDACP,X'01'                   ON
         BZ    PASS                              MODERATELY
         XR    R2,R12                            CHANGING
PASS     SR    R3,R3                           BIT
         ALR   R2,R11                DO SECOND PASS
         SRDL  R2,12                 WITHOUT
         OR    R2,R3                 COMPLIMENT
         BXLE  R9,R6,LOOP            EXHAUST KEY
         N     R2,NOSIGN             FORCE POSITIVE 31 BIT RESULT
*        USE R2 AS INPUT TO HALVING OR MODULO SCHEME - HALVING SHOWN
         L     R4,DMBDABLK
         MH    R4,DMBDARAP           HIGHEST RECORD IN RANGE
         SR    R5,R5                 RESULT REG
CVTLP    SRL   R4,1                  CUT RANGE IN HALF
```

```
STMT    SOURCE STATEMENT                                                    F15OCT70

  57             LTR    R4,R4                    RANGE EXHAUSTED
  58             BZ     XIT                      YES
  59             SR     R3,R3                    NO
  60             SRDL   R2,1                     TEST MASK FOR ONE
  61             LTR    R3,R3
  62             BZ     CVTLP                    NO ONE
  63             BXH    R5,R4,CVTLP              ONE - ADD IN RANGE
  64 XIT         LH     R6,DMBDARAP
  65             DR     R4,R6
  66             LA     R4,1(,R4)                ROOT ANCHOR POINT
  67             LA     R5,1(,R5)                BLOCK
  68             SLL    R5,8
  69             OR     R4,R5
  70             ST     R4,DMBDACP               RESULT
  71             LM     R14,R12,12(R13)
  72             BR     R14                      RETURN
  73             DS     0F
  74 NOSIGN      DC     X'7FFFFFFF'
  75             PRINT  NOGEN
  76             IDLI   DMBBASE=C
 313             REQUATE
 338             END
```

Generalized Randomizing Routine Example (DFSHDC40)

If root keys are unique and totally random storage is desired, this routine can be used for any HDAM data base without performing an analysis of key distributions.

This randomizing routine works with a maximum of 16 bytes of a key at a time. It contains fewer than 70 instructions and requires less than 600 bytes of storage. Its characteristics are:

- It is reentrant.
- Keys can contain any of the 256 System/370 characters and key length can be from 1 to 256 bytes.
- It converts any key distribution (with unique key values) to a totally random address distribution.
- It never returns an address in block 1, which is always a bit map block in HDAM. The user can specify any number of blocks and RAPs.
- It allows the insertion of a dummy root at the highest block-RAP to ensure the formatting of the entire root addressable area at load time.

The basic logic of the routine is:

1. Perform the first conversion. For example:

   ```
   123456-------->436152
   123457-------->437152
   ```

2. Translate against a table whose zero point is selected by an encipherment using every bit of the 16 bytes. For example:

   ```
   436152-------->X'AC7E2D241F39'
   437152-------->X'221949EA3F76'
   ```

3. Repeat 2 (on the result of 2) using XC instead of TR, and with a different bit encipherment.
4. Repeat 1 through 3 for the next 16 bytes (or less). XC results onto the result of the previous 16 bytes. Continue until key is accumulated.
5. Fold 16 bytes to 8 bytes and treat as a binary number.
6. Subtract 1 from the number of blocks, divide the binary number by the new block count, and add 2 to the remainder. This gives the block number.
7. Encipher the binary number, divide by the number of RAPs, and add 1 to the remainder. This gives RAP.

Routine Listing

```
              MACRO
     &N       RANDT &P,&S
              LCLA  &A,&C
     &A       SETA  &P
     &C       SETA  &S
     &N       DS    0F
     .L       ANOP
     &C       SETA  &C-1
              AIF   (&C EQ 0).END
     &A       SETA  &A*29+&C*47
     .S       AIF   (&A LT 1001111).OK
     &A       SETA  &A-1001111
              AGO   .S
     .OK      DC    AL2(&A,&A*23,&A*297,&A*191)
              AGO   .L
     .END     MEXIT
              MEND

     DFSHDC40 CSECT
              SAVE  (14,12),,DFSHDC40   SAVE REGISTERS
              B     14(0,15)                              BRANCH AROUND ID
              DC    ALI(8)                                LENGTH OF IDENTIFIER
              DC    CL8'DFSHDC40'                         IDENTIFIER
              STM   14,12,12(13)                          SAVE REGISTERS
              USING DFSHDC40,R15        ESTABLISH BASE REGISTER FOR PGM
              USING DMBDACS,R1          ESTABLISH BASE REG FOR PARMLIST



     * IF KEY STARTS X'FF' RETURN HIGHEST BLOCK-RAP                             *
     ****************************************************************************
              CLI   0(R9),X'FF'         IS FIRST BYTE OF KEY X'FF'?
              BNE   NORMKEY             NO...GO PROCESS NORMAL KEY
              MVC   DMBDACP(3),DMBDABLK+1  STORE HIGHEST BLOCK NO
              MVC   DMBDACP+3(1),DMBDARAP+1  STORE HIGHEST ANCHOR PT NO.
              B     GOBACK              RETURN TO CALLING MODULE

     ****************************************************************************
     *              I N I T   F O R   W H O L E   K E Y                         *
     ****************************************************************************
     NORMKEY  DS    0H
              SR    R5,R5               CLEAR WORK REGISTER
              SR    R3,R3               CLEAR WORK REGISTER
              IC    R3,DMBDAKL          LOAD EXECUTABLE KEY LENGTH

     ****************************************************************************
     *        WORK WITH NEXT 16 BYTES OF KEY

     XC       DS    0H
              CH    R3,ZERO             ANY MORE KEY LEFT?
              BL    END                 NO...GO CALCULATE BLOCK AND RAP
              CH    R3,=H'15'           YES...ARE 16 OR MORE CHARS LEFT?
              BL    EX                  NO...GO DO FINAL MOVE
              MVC   0(16,R7),0(R9)      YES...MOVE NEXT 16 BYTES.
              LA    R9,16(,R9)          UPDATE KEY ADDRESS TO NEXT 16 BYTES
              B     SCRAMBLE            GO TRANSPOSE THIS PART OF KEY
     EX       DS    0H
              XC    0(16,R7),0(R7)
              EX    R3,MVE              MOVE REMAINING AMOUNT OF BYTES
```

```
*************************************************************************
*         THIS SECTION TRANSLATES THE CURRENT SECTION OF KEY            *
*         IN PREPARATION FOR CALCULATING BLOCK AND RAP.                 *
*************************************************************************
SCRAMBLE DS     0H
         SH     R3,=H'16'             CALC AMT OF KEY REMAINING.
*                                     NEGATIVE VALUE SHOWS END OF KEY.  *

         MVC    16(16,R7),TRAN        FIRST TRANSPOSE THE KEY. THIS IS
         TR     16(16,R7),0(R7)       STAGE1 OF CLUSTER BREAKING.

         XC     1(15,R7),0(R7)        STAGE2 OF CLUSTER BREAKING IS TO
         NI     15(R7),X'1F'          TRANSLATE KEY AGAINST A TABLE THAT
         SR     R6,R6                 USES POLY-ALPHA CODE KEYED ON THE
         IC     R6,15(,R7)            TOTAL KEY VALUE.
         LA     R6,TRANTAB(R6)        UPDATE BY TABLE LENGTH
         TR     16(16,R7),0(R6)       TRANSLATE KEY
         NC     12(2,R7),=X'007F'     ENCIPHER ENCODED KEY TO PREVENT
         AH     R6,12(,R7)            REPETITION OR HI-ORDER BIT EFFECTS
         XC     16(16,R7),0(R6)

         XC     20(12,R7),16(R7)      ROLL 16 BYTES INTO 4
         NI     28(R7),X'3F'          ACCUMULATE IN REG. MAKING SURE
         A      R5,28(,R7)            THAT OVERFLOW CANNOT OCCUR
         N      R5,LOWBITS
         B      SC                    GO CHECK FOR MORE KEY

*************************************************************************
*         DEVELOP BLOCK AND RAP USING RANGE RATIO METHOD                *
*************************************************************************
END      DS     0H
         L      R2,DMBDABLK           STORE NO. OF BLOCKS
         BCTR   R2,0                  SUBTRACT 1 FROM COUNT
         MH     R2,DMBDARAP           NUMBER RAPS X (BLOCKS-1)
         MR     R4,R2                 R5 DEFINED AS 0-1 RANGE WITH POINT
         SRDL   R4,30                 BEFORE BIT2. AFTER MULT EXTRACT
         LH     R3,DMBDARAP           THE NUMERIC PART,
         DR     R4,R3                 THEN DIVIDE BY RAPS
         SLL    R5,8                  GIVING RAP IN R4 AND BLOCK IN R5.
         LA     R4,513(,R4)           ADD 2 TO BLOCK AND 1 TO RAP
         ALR    R5,R4                 STORE RAP AND BLOCK IN R5.
         ST     R5,DMBDACP            STORE BLOCK-RAP NO

GOBACK   DS     0H
         RETURN (14,12)               RETURN TO CALLING MODULE
         LM     14,12,12(13)                    RESTORE THE REGISTERS
         BR     14                              RETURN

         MVC    0(1,R7),0(R9)         EXECUTE INSTRUCTION
```

```
188 ***********************************************************************
189 *                                                                     *
190 *                 T A B L E S   A N D   C O N S T A N T S             *
191 *                                                                     *
192 ***********************************************************************
193 TRANTAB  RANDT 77777,36
194+TRANTAB  DS    0F
195+         DC    AL2(254956,254956*23,254956*297,254956*191)
196+         DC    AL2(387545,387545*23,387545*297,387545*191)
197+         DC    AL2(228135,228135*23,228135*297,228135*191)
198+         DC    AL2(610753,610753*23,610753*297,610753*191)
199+         DC    AL2(694407,694407*23,694407*297,694407*191)
200+         DC    AL2(116993,116993*23,116993*297,116993*191)
201+         DC    AL2(390827,390827*23,390827*297,390827*191)
202+         DC    AL2(323078,323078*23,323078*297,323078*191)
203+         DC    AL2(360532,360532*23,360532*297,360532*191)
204+         DC    AL2(445540,445540*23,445540*297,445540*191)
205+         DC    AL2(908503,908503*23,908503*297,908503*191)
206+         DC    AL2(318829,318829*23,318829*297,318829*191)
207+         DC    AL2(237123,237123*23,237123*297,237123*191)
208+         DC    AL2(870935,870935*23,870935*297,870935*191)
209+         DC    AL2(230327,230327*23,230327*297,230327*191)
210+         DC    AL2(673757,673757*23,673757*297,673757*191)
211+         DC    AL2(518737,518737*23,518737*297,518737*191)
212+         DC    AL2(27554,27554*23,27554*297,27554*191)
213+         DC    AL2(799865,799865*23,799865*297,799865*191)
214+         DC    AL2(171284,171284*23,171284*297,171284*191)
215+         DC    AL2(963497,963497*23,963497*297,963497*191)
216+         DC    AL2(912074,912074*23,912074*297,912074*191)
217+         DC    AL2(421871,421871*23,421871*297,421871*191)
218+         DC    AL2(221491,221491*23,221491*297,221491*191)
219+         DC    AL2(417090,417090*23,417090*297,417090*191)
220+         DC    AL2(82748,82748*23,82748*297,82748*191)
221+         DC    AL2(397893,397893*23,397893*297,397893*191)
222+         DC    AL2(527052,527052*23,527052*297,527052*191)
223+         DC    AL2(268172,268172*23,268172*297,268172*191)
224+         DC    AL2(769493,769493*23,769493*297,769493*191)
225+         DC    AL2(291090,291090*23,291090*297,291090*191)
226+         DC    AL2(432910,432910*23,432910*297,432910*191)
227+         DC    AL2(541199,541199*23,541199*297,541199*191)
228+         DC    AL2(678200,678200*23,678200*297,678200*191)
229+         DC    AL2(646738,646738*23,646738*297,646738*191)

231 TRAN     DC    X'0D0A0E0C090F0B060502070103080004'
232 ZERO     DC    F'0'
233 LOWBITS  DC    X'3FFFFFFF'
234          LTORG
235                =H'15'
236                =H'16'
237                =X'007F'

239          PRINT NOGEN
240          IDLI  DMBBASE=0             CREATE DSECTS
685          REQUATE                     CREATE REGISTER EQUATES
709          END
106 ***********************************************************************
```

SECONDARY INDEX DATA BASE MAINTENANCE EXIT ROUTINE INTERFACE

Two options are available to the data base manager to control the volume of entries in secondary index data bases -- the NULLVAL operand and the index maintenance exit routine. The process of withholding a prospective index pointer segment from the index is called suppression of indexing. This is the process by which a sparse index is built and maintained.

The NULLVAL operand can be used to suppress indexing when the entire indexed field contains one specified character or value. For example, NULLVAL might be used to suppress indexing when the indexed field contains only blanks. A different NULLVAL can be specified for each indexed segment.

Alternatively, secondary indexing allows specification of a user-supplied exit routine that can selectively cause suppression of secondary indexing. The user can thereby control the density of a secondary index. One exit routine is allowed for every secondary index; however, one generalized routine can be written to serve several index relationships.

After an exit routine has been compiled and tested, it can be placed into the IMSVS.RESLIB data set, from which it is loaded by IMS/VS. It can also be placed in SYS1.LINKLIB, or any operating system partitioned data set to which access is provided with a JOBLIB or STEPLIB JCL statement. Each exit routine must have a name unique with respect to all IMS/VS module names and to any other user routines in the IMS/VS libraries. The name corresponds to the name specified in the EXTRTN subparameter, in the XDFLD statement, for the DBD generation. Before any segment which is an index source segment in a data base can be loaded or updated, its EXTRTN routine, if one was specified, must be in the system library. This prevents abnormal termination.

The exit routine associated with the specific data base is loaded into storage in either the IMS/VS online control program region or batch processing region when the associated data base is opened. If a single exit routine is used for several data bases, the module must be written, compiled, and link edited as reenterable (RENT). This allows one copy of the module to service several data bases that are open concurrently.

When an index maintenance exit routine is used in either the IMS/VS online control region or a DL/I batch processing region, and the exit routine does not exist in LINKPACK, space must be provided in the IMS/VS control region to accommodate the exit routines that can be used for online data bases.

All exit routines are loaded from their resident library by the IMS/VS Open/Close module (DFSDLOC0). Open obtains the name of the exit routine to be loaded from the name specified in the associated DBD. The IMS/VS IMODULE macro instruction is used.

The user should be aware of the way in which the index exit routine is applied to the index maintenance process. When an application program issues a REPL, ISRT, or DLET call of a segment serving as an index source segment for one or more indexing relationships, the DL/I index maintenance routine is invoked.

In the case of DLET, an indexing segment is built corresponding to the existing index source segment. If it passes the null value test, the index exit routine is invoked. This routine indicates whether this indexing segment should appear in the index or not. If it should appear, the actual indexing segment is retrieved and deleted; otherwise, no delete is attempted.

In the case of ISRT, the indexing segment is built to correspond to the segment to be inserted, and the null value test and the user exit routine tests are performed. If no suppression of indexing is indicated by either, it is inserted into the index.

A REPL call can be a combination of the above, a simple replace, or a NOP, depending on the fields changed in the replace. If a field in the Index Source Segment (ISS) is changed by a REPL call that changes the indexed data or sub-sequence data, the existing indexing segment is deleted and a new one inserted. The index edit routine is invoked for each operation. If the change in the ISS affects a source data field, a replace operation on the indexing segment is executed, unless the index exit routine indicated that indexing was suppressed. If the ISS replace made no changes in the indexing segment, no action is taken.

The supression of indexing by the exit routine must be consistent. The same indexing segment cannot be examined at two different times and have suppression indicated only once. User data cannot be used to evaluate suppression, since the actual indexing segment is seen by the exit routine just before the insertion of a new one. In the cases of replace and delete, only a prototype is passed. The prototype contains the constant, indexed data, sub-sequence data, source data, and any symbolic pointer that may have been added. Therefore, index suppression must not be based on any user data.

Parameters to be passed to the index routine are indicated later in this discussion. The exit routine indicates, with a return code, whether the present index pointer segment belongs in the index or should be suppressed. The exit routine should not change any IMS/VS control blocks, or any fields in the indexing segment.

The user can include additional information about the segment in the exit routine CSECT. This CSECT is part of the DBD, and as such can be replaced by a link edit. It is of variable length and contains a fixed format header. A separate CSECT is provided for each XDFLD in the DBD for which an exit routine is specified. The availability of this CSECT is described in the exit routine interface specifications. This control section can be replaced by the user in the same manner as the segment compression control section. See the "Segment Edit/Compression" discussion earlier in this chapter, for additional information.

INTERFACE TO THE INDEX MAINTENANCE EXIT ROUTINE

At entry to the index maintenance exit routine, registers must be saved. A save area address is provided in register 13 for this purpose. The first three words of this save area must remain unchanged.

These are the register contents upon entry to the exit routine:

Register	Meaning or Content
1	Partition Specification Table (PST) address.
2	Address of (proposed or existing) index segment.
3	Address of Index Maintenance Routine Parms CSECT.
4	Address of Index Source Segment.
13	Save area address.
14	Return to IMS address.
15	Entry point address of the exit routine.

Upon return to IMS/VS, registers 1 through 14 must be restored. Register 15 must contain a return code of either 0 or 4. A return code of 4 indicates that indexing should be suppressed in this case; a return code of 0 indicates that the indexing segment should appear in the index for this data base segment.

INDEX MAINTENANCE EXIT ROUTINE PARAMETER CSECT

```
┌──────────────────────────────────────────────┐
| 0                                            |
|            INDEXED SEGMENT NAME              |
|                                              |
|----------------------------------------------|
| 8                                            |
|         INDEXED FIELD (XDFLD) NAME           |
|                                              |
|----------------------------------------------|
| 16                                           |
|            INDEX MAINTENANCE                 |
|                                              |
|              EXIT ROUTINE                    |
|                                              |
|                  NAME                        |
|                                              |
|----------------------------------------------|
| 24                                           |
|            ENTRY POINT ADDRESS               |
|                                              |
|----------------------------------------------|
| 28                    |                      |
|      CSECT            |        RSVD          |
|     LENGTH            |                      |
|                       |                      |
|----------------------------------------------|
| 32                                           |
|            USER                              |
|                                              |
|            DATA                              |
|                                              |
|                                              |
|                                              |
└──────────────────────────────────────────────┘
```

```
DMBXMPRM     DSECT
DMBXMSGN     DS        CL8       Name of indexed segment
DMBXMXDN     DS        CL8       Name of indexed field
DMBXMXNM     DS        CL8       Name of user exit routine
DMBXMXEP     DS        A         Entry point addr
DMBXMPLN     DS        H         Total length of CSECT
             DS        H         Not Used
```

DATA BASE LOG TAPE RECORD FORMAT

The following DSECT provides an image of the log tape record format for all data base modifications. This log tape record format is provided to facilitate the writing of any user-written statistics, recovery analysis, or batch checkpoint/restart programs.

```
DBLOG       DSECT
DLENGTH     DS     H         LENGTH OF LOG RECORD
DSPACE      DS     H         ZEROS
DLOGCODE    DS     CL1       LOG RECORD I.D.
DLOGFLG1    DS     CL1
*                       BITS 0-3 =  REGION PROTECT KEY
*                       BITS 4-7 =  COUNT OF FSE'S IN LOG RECORD

DLOGFLG2    DS     CL1
DNDXC       EQU    X'80'     INDEX MAINTENANCE RECORD
DCMC        EQU    X'00'     BITS 1-3 =  000 CHAIN MAINTENANCE RECORD
DPHYI       EQU    X'40'     PHYSICAL INSERT
DPHYD       EQU    X'20'     PHYSICAL DELETE
DNCTR       EQU    X'70'     COUNTER MAINTENANCE CALL

DPHYR       EQU    X'10'     PHYSICAL REPLACE
DLASTREC    EQU    X'08'     LAST RECORD FOR THIS USER CALL
DOSAM       EQU    X'00'     BIT 5=0 OSAM DATA SET
DISAM       EQU    X'04'     BIT 5=1 ISAM DATA SET
DHS         EQU    X'00'     BIT 6=0 HS ORGANIZATION
DHD         EQU    X'02'     BIT 6=1 HD ORGANIZATION
DNEWBLK     EQU    X'01'      NEW BLOCK CALL

DLOGFLG3    DS     CL1
DRCALL      EQU    X'80'     REPL CALL
DDCALL      EQU    X'40'     DLET CALL
DICALL      EQU    X'20'     ISRT CALL
DLGDLET     EQU    X'60'     LOGICAL DELETE
DREG0       EQU    X'00'     BITS 3-4 = 00 MOD BY TYPE 0 REGION
DREG3       EQU    X'10'     MOD BY TYPE 3 REGION
DREG12      EQU    X'08'     MOD BY TYPE 1/2 REGION
DINITGU     EQU    X'04'     BUFFERS WASHED WITH EACH MSG GU CALL
DFIRSTSG    EQU    X'02'     FIRST LOG RECORD OF A SEGMENT
DLASTSEG    EQU    X'01'     LAST LOG RECORD OF A SEGMENT

DIDLN       DS     CL2       LENGTH OF DDATAID FIELD
DOFFSET     DS     CL2       DATA OFFSET FROM BEGINNING OF A BLOCK
DDATALN     DS     CL2       LENGTH OF DDATA FIELD
DCCODE      DS     CL2       DL/I COMPLETION CODE
DPGMNAME    DS     CL8       PROGRAM NAME
DDBDNAME    DS     CL8       DATA BASE NAME
DDSID       DS     CL1       DATA SET I.D.
DDATE       DS     CL3       DATE
DTIME       DS     CL4       TIME
DSEQ        DS     CL2       SEQUENCE NUMBER
DDATAID     DS     0CL1      ISAM PRIME KEY OR OSAM RBN
DDATA       DS     0CL1      SEGMENT DATA
DFSEOFF     DS     0CL2      FREE SPACE ELEMENT OFFSET
DFSE        DS     0CL4      FREE SPACE ELEMENT
```

CHAPTER 4. DC USER EXIT AND EDIT ROUTINES

This chapter describes the data communication functions that can be modified by the IMS/VS user and the procedure required to make these modifications. Alterations to IMS/VS data base functions are described in the preceding chapters.

Modifications to the data communication facility relate primarily to the addition of user-written message edit routines. Several basic edit functions are provided to all users. 274X, 3270, 3600, 3767, and 3770 users can select to use the IMS/VS message format service (MFS) instead of the basic edit. MFS provides edit functions similar to the basic edit but, in addition, allows you to select and describe alternative message formats.

All users have the option of writing and including additional routines to edit transaction code input, message switching input, and/or physical terminal input and output. Transaction code, message switch, and physical terminal input edit routines may return messages to inputting terminals through use of a user message table. Users of conversational processing who want to provide a clean-up program for prematurely-terminated conversations must include an exit routine to the clean-up program. IMS/VS provides an exit routine, or you can write your own.

Further, user-routine exits are provided for the users of 7770, 2980, 3741, and 3614 devices. Default routines are provided for these exits. Any default routine that does not meet your needs can be replaced by a user-written routine. These routines are described in this chapter, with the exception of the 3614 edit routine (described in _IMS/VS Advanced Function for Communications_) and the MFS edit routines (described in the _IMS/VS Message Format Service User's Guide_).

All user-written routines are incorporated into the IMS/VS control program nucleus during stage 2 of IMS/VS system definition.

The IMSVS.DCSOURCE library contains the source for all sample and supplied exit routines described in this chapter, and should be referred to for the latest versions.

BASIC IMS/VS EDIT FUNCTIONS

The IMS/VS-supplied basic edit routine performs the following functions for input messages:

- Removes leading control characters from the first segment of each message, whether the message type is a transaction, a command, or a message switch. Leading blanks are also removed from the first segment if the message is not the continuation of a conversation or a message from a terminal in Preset Mode.

- Removes leading control characters from all subsequent message segments, whether the message type is a transaction or a command (except /BROADCAST).

- Removes line control characters from all segments.

- Removes trailing carriage return characters from all segments of a transaction.

- Eliminates backspaces, on a one-for-one basis, from all segments, when the entering or transmission of backspaces is a normal correction procedure on the entering terminal.

- Removes the password and replaces it with a blank when necessary to provide separation between the transaction code, logical terminal, or command verb, and following data.

- Inserts, in front of data entered in the first segment of a message, the transaction code or logical terminal name defined by the prior /SET command. A blank is inserted following the transaction code, if necessary to obtain separation between the inserted transaction code and the entered data.

The IMS/VS-supplied edit for output messages inserts any necessary idle characters after new line, line feed, and tab characters. Line control characters are added for operation of the communication line.

If the input is processed by MFS, the editing performed is dependent on the descriptions provided through the message format language utility. Since input segments from the device may have no relationship to input message segments after MFS editing, the input segment from the device is not available to user-written edit routines.

USER EDIT ROUTINE INCLUSION DURING SYSTEM DEFINITION

All user-supplied edit routines should be placed in the operating system partitioned data set defined by the USERLIB= operand of the IMSGEN macro instruction of IMS/VS system definition. This must be performed prior to execution of IMS/VS system definition Stage 2. If you do not specify a value for the USERLIB= operand, IMS/VS system definition assumes the IMSVS.RESLIB data set contains any user-defined edit routines. System definition attempts to obtain any user-specified edit routines from the specified library during Stage 2 of execution, and link edits them as part of the IMS/VS control program nucleus. The names of the edit routines specified to IMS/VS system definition should be the same as the CSECT and load module names for the edit routine modules in the library specified by USERLIB=. The message switch edit routine must have a CSECT and load module name of DFSCNTE0.

COMMON DC ROUTINES

PHYSICAL TERMINAL (INPUT) EDIT ROUTINE

A sample input edit routine is shown in Figure 4-1. This user-written edit routine gains control before the IMS/VS basic edit routine. If the input message is processed by MFS, the physical terminal (input) edit routine is not called.

Message segments are passed one at a time to the physical terminal input edit routine, and the edit routine can handle them in one of the following ways:

- Accept the segment and release it for further editing by the IMS/VS basic edit routine.

- Modify the segment and release it for further editing by the IMS/VS basic edit routine. Examples of segment modifications that could be made are changing the transaction code, adding a password, and reformatting the message text.

Any required modifications can be made, since IMS/VS has not yet performed destination or security checking.

- Cancel the segment.
- Cancel the message and request that the terminal operator be notified accordingly.
- Cancel the message and request that a specific message from the User Message Table be sent to the terminal operator.

The physical terminal input edit routine requests the above actions by specifying different return codes that are interpreted and acted upon by IMS/VS.

Interface

The CSECT name for this edit routine is the name specified in the TYPE or LINEGRP macro for which this edit routine applies. Registers on entry and exit are discussed below.

- Registers on Entry

Upon entry to this edit routine, all registers to be used must be saved. The following interface applies:

R1 — Address of the input message segment buffer. IMS/VS editing has not been performed. The first two bytes of the buffer contain the segment length (binary length includes the four-byte overhead). The third and fourth bytes of the buffer are binary zeros. The message text begins in the fifth byte of the buffer.

If the device was defined with MFS support, but this message is not being processed by MFS, the first segment of the message has backspace error correction performed before entry to this edit routine. If escape (//) was entered, the first two data bytes have been changed to binary zeros.

R7 — CTB address for the physical terminal from which the message was entered.

R9 — CLB address for the physical terminal from which the message was entered.

R13 — Save area address for use by an edit routine. The first three words in the save area must not be modified.

R14 — Return address to IMS/VS.

R15 — Entry point address to the invoked edit routine.

The user-supplied edit routine must edit the message segment in the buffer addressed by register 1.

The user can reduce the length of the message segment to any desired size by replacing the length in the buffer with the appropriate value. The length field must appear in the same place at exit as at entry, and bytes 3 and 4 must not be changed.

• Registers on Exit

Upon return to IMS/VS, all registers must be restored except register 15.

R1	Message number if register 15 contains a value of 12; otherwise ignored.
R15	Return codes:

00 - Segment is processed normally.

04 - Segment is canceled.

08 - Message is canceled and the terminal operator is notified.

12 - Message is canceled, and the message identified by register 1 is sent to the terminal.

Any other return code causes the message to be canceled and the terminal operator to be notified.

Example of a Physical Terminal Input Edit Routine

The sample routine in Figure 4-1 does the following:

- Scans the input message segment for an expected format -- TESTEXIT
- Generates return codes (XX) based on the input request (TESTEXIT,XX)
- Verifies the user message number (YYY) if specified (TESTEXIT,XX,YYY)
- Replaces TESTEXIT with ERROR if return code or message number is invalid and passes the segment to IMS/VS (return code 0)

```
PIXT     TITLE ' PHYSICAL TERMINAL INPUT EDIT ROUTINE SAMPLE.   I590  '
DFSPIXTO CSECT
********
*
*        PHYSICAL TERMINAL INPUT EDIT ROUTINE
*
*              FOR TEST PURPOSES
*
*
*        REQUIREMENTS: DEFINITICNS FOR THIS EXIT ROUTINE IN AT
*                      LEAST ONE SET CF TYPE/TERMINAL OR
*                      LINEGRP/TERMINAL MACROS
*                      DURING IMS SYSTEM DEFINITION.
*
*        AT ENTRY:  REGISTER 1 POINTS TC LENGTH FIELD OF UNEDITED
*                                MESSAGE
*
*        AT RETURN: REGISTER 15: RETURN CODE
*                                0 GO CN EDITING THE MESSAGE
*                                4 CANCEL SEGMENT
*                                8 CANCEL MESSAGE
*                                12 CANCEL MSG AND SEND USER ERROR
*                                   MESSAGE
*                   REGISTER 1: IF RETURN CODE = 12, USER ERROR MESSAGE
*                                                   NUMBER
*                               REQUIREMENT:
*                                USER EUILT USER MESSAGE TABLE,
*                                CONTAINING THE REQUESTED MESSAGE
*                                LINKECITED INTO  THE IMS NUCLEUS.
*                                REQUIREC NAME: DFSCMTU0.
*
*        INTERNAL REGISTER USAGE:
*                                R1: UNCHANGED SEGMENT POINTER
*                                R2: WORK REGISTER
*                                R3: SEGMENT SCAN POINTER
*                                R12: BASE REGISTER
*                                R15: RETURN CODE
*
*        IF AN ERROR IN THE TESTEXIT-MESSAGE IS FOUND, THEN
*        THE CONSTANT 'TESTEXIT' IN THE SEGMENT IS REPLACED BY
*        ' ERROR  ' AND THE SEGMENT RETURNED WITH A RETURN
*        CODE OF 0.
*
*        EXPECTED MESSAGE FORMAT:  (FIXED FORMAT)
*                                TESTEXIT,XX,YYY
*                                           ,YYY ONLY FCR XX=12
*                                               USER MESSAGE NUMBER
*                                         XX RETURN CODE TO EE GEN'D
*
*******
         SAVE  (14,12),,PIX8084     SAVE REGS
         LR    R12,R15              SETUP PGM BASE REGISTER
         USING DFSPIXTO,R12
         LR    R3,R1                SEGMENT POINTER
         LA    R3,4(R3)             SKIP LENGTH FIELD
```

Figure 4-1 (Part 1 of 3). Sample Physical Terminal Input Edit Routine

```
         LH    R2,0(R1)           GET SEGMENT LENGTH
         AR    R2,R1              CALCULATE END OF SEGMENT
         S     R2,=F'10'           MINUS KONSTANT TESTEXIT AND MIN PARM
         CR    R2,R3
         BNH   RET0
*
LOOP     EQU   *
         CLC   0(8,R3),KEXITU     IS IT UPPERCASE TESTEXIT MSG?
         BE    GENUMSG            YES
         CLC   0(8,R3),KEXITL     IS IT LOWERCASE TESTEXIT MSG?
         BE    GENUMSG            YES
         LA    R3,1(R3)           INCR SCANPOINTER
         CR    R3,R2              END OF SEGMENT
         BNH   LOOP               NO
         SR    R15,R15            YES- SET RC = 0
         B     RET                AND EXIT
*
GENUMSG  EQU   *                  DETERMINE REQUESTED RETURNCODE
         CLC    9(2,R3),=C'00'
         BE    RET0
         CLC    9(2,R3),=C'04'
         BE    RET4
         CLC    9(2,R3),=C'08'
         BE    RET8
         CLC    9(2,R3),=C'12'
         BE    RET12
         B     RETERR
*
RET0     EQU   *                  SET RC=0
         SR    R15,R15
         B     RET
*
RET4     EQU   *                  SET RC=4
         LA    R15,4
         B     RET
*
RET8     EQU   *                  SET RC=8
         LA    R15,8
         B     RET
*
RET12    EQU   *                  RC=12 REQUESTED
         CLI   12(R3),C'0'        IS USER MESSAGE NUMBER
         BL    RETERR
         CLI   12(R3),C'9'        *
         BH    RETERR
         CLI   13(R3),C'0'        *
         BL    RETERR
         CLI   13(R3),C'9'        *
         BH    RETERR
         CLI   14(R3),C'0'        *
         BL    RETERR
         CLI   14(R3),C'9'        VALID?
         BH    RETERR             - NO: SET RC=0 AND ' ERROR  '
         XC    WORK1,WORK1        - YES
         PACK  WORK1+6(2),12(3,R3) CONVERT IT
```

Figure 4-1 (Part 2 of 3). Sample Physical Terminal Input Edit Routine

```
         CVB   R1,WORK1           TO BINARY, PASS IT IN REG 1
         LA    R15,12             SET RC=12
         B     RET
*
RETERR   EQU   *
         SR    R15,R15            SET RC=0
         MVC   4(8,R1),=C' ERROR  ' REPLACE 'TESTEXIT'
         B     RET
*
RET      EQU   *
         ST    R15,SAVERC         SAVE RETURN CODE
         LM    R2,R12,28(R13)     RESTORE REGISTERS
         L     R0,20(R13)
         L     R14,12(R13)
         BR    R14                RETURN
*
KEXITU   DC    CL8'TESTEXIT'      UPPERCASE
KEXITL   DC    XL8'A385A2A385A789A3' LOWERCASE 'TESTEXIT'
WORK1    DC    D'0'
SAVERC   DC    F'0'
         REQUATE
         END
```

Figure 4-1 (Part 3 of 3). Sample Physical Terminal Input Edit Routine

PHYSICAL TERMINAL (OUTPUT) EDIT ROUTINE

You can specify, during system definition, a physical terminal output edit routine to edit output messages just before they are sent to a terminal. One physical terminal output routine can be specified for each BTAM telecommunication line group. During system definition, you specify which physical terminals or set of VTAM nodes use the defined edit routine for output editing. These edit routines can be used to provide special user editing needs by communciation terminal types. An output message can be processed by a physical terminal output edit routine and the basic IMS/VS edit routine or MFS (message format service). Output editing is performed in this sequence. Therefore, if MFS is used, the output provided by the edit routine must be the format defined to MFS instead of the format created by the application program.

Interface

- Registers on Entry

Upon entry to an edit routine, all registers to be used must be saved. The following interface applies:

R1 The address of a buffer containing the output message segment to be edited. The first two bytes are a binary count of the message segment length. The second two bytes are control information provided by the application program which constructed the message. The text of the output message starts in byte five. The count includes the first four bytes in length.

R7 CTB address for the destination terminal.

R9 CLB address. This block starts with a DECB. The content of DECAREA field in the DECB is equivalent to register 1 content.

R11 SCD address.

R13 The address of a save area for use by the edit routine. The first three words in the save area must not be changed.

R14 Return address to IMS/VS.

R15 The entry point address to the invoked edit routine.

The resultant output message segment returned to IMS/VS from the user's edit routine must be pointed to by the content of the DECB, DECAREA field. The first four bytes must be in a format as received at input with the binary count updated to the edited message segment length inclusive of the four bytes of prefix.

- Registers on Exit

Upon return to IMS/VS, all registers must be restored. If the message is to be edited in place, the length must not be increased by more than ten bytes.

When the last segment of a message has been edited, IMS/VS returns control to the user's edit routine once more. The edit routine can do some housekeeping activities at this time. Upon entry to the user's edit routine, registers 7, 9, 11, 12, 13, 14, and 15 are as described above.

Whenever a physical terminal output edit is invoked, the CTB is addressed by register 7. A one byte field in the block, CTBACTL will contain a one in the second bit position if this entry to the user's edit routine is for end of message. The _IMS/VS Program Logic Manual, Volume 1 of 3_ defines the IMS/VS control blocks.

Example of a Physical Terminal Output Edit Routine

The example in Figure 4-2 shows how an output message can be extended in length and a prefix attached. Two capabilities within IMS/VS are used. One allows the edit routine to obtain a buffer area. This is called ICREATE. When ICREATE is used, an identifier of four bytes is provided in register 2. The length of the requested area is placed in register 3. The address of the buffer area is returned to the edit routine in register 3. This area is used to build the output message, prefixed with the PTERM output message count, the LTERM name, and the LTERM output message count. The edited output message is addressed by DECAREA. When the second segment or the end of message entry to the edit routine is made (CTBAEOM=1), the buffer area obtained by the edit routine is returned to IMS/VS. This is performed by the second IMS/VS facility called IDESTROY. Register 2 is used to symbolically identify the area to IMS/VS. This example applies to single segment or multisegment messages, and to as many devices as the edit routine's table was assembled to handle. The default table size allows for five devices, but can be changed by modifying the label NUMENTS. If the table capacity is exceeded, a user 555 ABEND results. If the prefix had not increased the message length by more than ten bytes, it could have been performed in place without the creation of an additional buffer area.

Figure 4-2 is an example of a physical terminal output edit routine.

```
CTTO     TITLE 'DFSCTTOO  SAMPLE PTERM (OUTPUT) EDIT ROUTINE'
***********************************************************************
*                                                                     *
*             THIS MODULE WILL PREFIX EACH MESSAGE WITH THE PTERM     *
*        OUTPUT MESSAGE COUNT, THE LTERM NAME AND OUTPUT MESSAGE      *
*         COUNT IN THE FORM: PTFRMOUT-XXXX LTERMNAM-XXXX ..MSG TXT..  *
*                                                                     *
*             A TEN CHARACTER PAD IS PROVIDED BY IMS IN EACH          *
*        BUFFER.  THE PREFIX HERE WAS SELECTED TO EXCEED THE PAD      *
*        THUS PROVIDING A REQUIREMENT FOR AN ICREATE AND IDESTROY.    *
*                                                                     *
*             THE CODE IS RE-ENTRANT.  'TABLE' MUST CONTAIN AS        *
*        MANY ENTRIES AS THERE ARE PTERMS USING THIS ROUTINE.  THE    *
*        TABLE SIZE CAN BE CHANGED BY MODIFYING THE VALUE 'NUMENTS'   *
*        IF THE TABLE CAPACITY IS EXCEEDED A U555 ABEND WILL RESULT   *
*                                                                     *
*                                                                     *
*        REGISTERS ON ENTRY          ON EXIT                          *
*        R7     CTB                                                   *
*        R9     CLB                                                   *
*        R11    SCD                  ALL REGISTERS RESTORED           *
*        R13    SAVE AREA                                             *
*        R14    RETURN ADDRESS                                        *
*        R15    ENTRY POINT ADDRESS                                   *
***********************************************************************
         EJECT
DFSCTTOO CSECT
         PRINT NOGEN
         SAVE  (14,12),,*              SAVE REGISTERS
         L     R13,8(,R13)             R13= NEXT SAVE AREA
         LR    R12,R15                 R12 WILL BE MY BASE REG
         USING DFSCTTOO,R12
         USING ENTRY,R5                R5=  'TABLE' POINTER
         USING CTB,R7                  R7=  CTB ADDR
         USING IECTDECB,R9             R9=  CLB (DECB) ADDR
         USING SCD,R11                 R11= SCD ADDR
         L     R15,DECAREA             R15 = ADDRESS OF SEGMENT DL FIELD
*
         S     R7,SCDCTB               CREATE CTB OFFSET
         BAL   R14,GETNTRY             FIND/OBTAIN AN ENTRY IN DEV TABLE
         A     R7,SCDCTB               RETURN ONLY IF I DO & ADJ R7
*
         TM    CTBACTL,CTBAEOM         END-OF-MSG CALL?
         BO    EOMSG                   YES, BRANCH TO EOM HANDLER
         CLI   ENTSTAT,0               NO, FIRST SEGMENT?
         BE    FIRSTSEG                 YES, BRANCH TO 1ST SEG HANDLER
*                                       NO, NOT FIRST SEG & NOT EOM --
*                                      DELETE BUFFER POOL IF NECESSARY
         CLI   ENTSTAT,X'FF'           DOES THE BUFFER STILL EXIST?
         BE    RETURN                  NO,  RETURN  (NO MORE TO DO NOW)
*                                      YES, GIVE UP THE SPACE
         BAL   R14,DESTROY             DO AN 'IDESTROY'
         MVI   ENTSTAT,X'FF'           INDICATE 'BUFFER-FREED'
         B     RETURN
         SPACE 1
```

Figure 4-2 (Part 1 of 4). Sample Physical Terminal Output Edit Routine

```
************************************************************************
FIRSTSEG DS    0H                                                      *
*        FIRST SEGMENT -                                               *
*              SINCE THE PREFIX EXCEEDS THE DEFAULT 10                 *
*              CHARACTER PAD SPACE WE MUST                             *
*              GET SOME BUFFER POOL SPACE (ICREATE)                    *
************************************************************************
*                                                                      *
         MVI   ENTSTAT,C'X'           COMPLETE THE ID FOR ICREATE
         L     R2,ENTRY               LOAD THE ID
         LR    R0,R11                 SCD ADDR
         LR    R1,R9                  CLE ADDR
         SR    R4,R4                  'BUFFERS-IN-POOL-ARE-VARIABLE-LEN'
         LR    R3,R4                  ZERO REG 3 TOO
         IC    R3,0(,R15)
         SLL   R3,8
         IC    R3,1(,R15)             R3= LEN OF SEG PASSED TO ME
         LR    R6,R3                  SAVE ORIGINAL LENGTH,
         LR    R10,R15                 AND ADDR FOR LATER USE
         LA    R3,PREFIX(,R3)         INCR BY LENGTH OF PREFIX
*                                     R3= POOL SIZE FOR ICREATE
         LR    R8,R3                  SAVE NEW LENGTH FOR LATER USE
         L     R15,SCDSMMCP
         BALR  R14,R15                DO AN ICREATE
         EJECT
************************************************************************
*              EDIT MSG INTO NEW AREA                                  *
************************************************************************
*                                                                      *
         STH   R8,0(,R3)              NEW DL FIELD
         MVC   2(2,R3),2(R10)         ZZ FIELDS MUST REMAIN THE SAME
         L     R8,4(,R13)             GET CNT POINTER FROM
         L     R8,60(,R8)                SAVE AREA
         USING CNT,R8
         MVI   4(R3),CR               CARRIAGE RETURN
         MVC   5(9,R3),PTERM          'PTERMOUT-'
         SR    R4,R4
         LH    R4,CTBOUTCT            PICK UP COUNT
         BAL   R14,CONVRT
         MVC   14(6,R3),0(R5)         MOVE PTERM OUTPUT COUNT INTO MSG
         XC    20(17,R3),20(R3)       CLEAR AREA
         TM    CTBACTL,CTBAINC        IS THIS INCORE MSG
         BO    SKIPLT                 IF SC ..MSG IS NOT QUEUED
         MVI   20(R3),BLANK
         MVC   21(8,R3),CNTNAME           LTERM NAME
         MVI   29(R3),DASH            SEPERATE
         LH    R4,CNTDQCT             PICK UP COUNT
         BAL   R14,CONVRT             CONVERT TO EBCDIC
         MVC   30(6,R3),0(R5)         MOVE LTERM DEQ COUNT INTO MSG
         CLI   4(R10),CR              CK FOR AT LEAST ONE CR
         BE    SKIPLT
         MVI   36(R3),CR              IF NOT BUT ONE IN
SKIPLT   EQU   *
         DROP  R8
         SH    R6,=H'5'               DECR ORIG LEN BY (LLZZ+1)
         EX    R6,MOVEMSG             APPEND MSG TEXT TO PREFIX
         ST    R3,DECAREA             PASS NEW ADDR BACK TO IMS
         B     RETURN
         EJECT
```

Figure 4-2 (Part 2 of 4). Sample Physical Terminal Output Edit Routine

```
***********************************************************************
EOMSG    DS    0H                    COME HERE WHEN CTBAECM IS ON       *
*                                                                       *
*         WHEN I GET HERE ALL SEGS OF A MSG HAVE BEEN PROCESSED         *
*        IF MSG IS SINGLE-SEG, THE BUFFER POOL STILL EXISTS.            *
*                                                                       *
***********************************************************************
*                                                                       *
         CLI   ENTSTAT,X'E7'         POOL BEEN FREED YET?
         BNE   ZAPTBL                YES, BRANCH
         LA    R14,ZAPTBL            NO, RELEASE IT-- SET RTRN ADDR HERE
DESTROY  L     R2,ENTRY              BUFFER POOL ID
         LR    R0,R11                SCD ADDR
         LR    R1,R9                 CLB ADDR
         L     R15,SCDSMMDP
         BR    R15                   TO DESTROY ROUTINE IN IMS
*
ZAPTBL   EQU   *
         MVI   ENTSTAT,0             OPEN UP THE SLOT
         SPACE 2
RETURN   EQU   *
         L     R13,4(,R13)           RESTORE R13
         RETURN (14,12)
         SPACE 1
MOVEMSG  MVC   4+PREFIX(R6-R6,R3),4(R10)
         EJECT
***********************************************************************
GETNTRY  DS    0H                    FIND THE CALLING DEVICE'S ENTRY    *
*                                    IN 'TABLE'.  IF NOT PRESENT TRY    *
*                                    TO FIND AN EMPTY SLOT.             *
***********************************************************************
*                                                                       *
         DROP  R5
         USING ENTRY,R2              NOW REG 2 POINTS TO 'TABLE'
*
         LM    R0,R2,=A(L'TABLE,LASTENT,TABLE)
LOOP1    CH    R7,ENTCTB             IS THIS RIGHT PLACE?
         BE    GOTDEV                YES, BRANCH  (ALL SEEMS WELL)
         BXLE  R2,R0,LOOP1           NO, KEEP TRYING
*                                    OOPS, NOT HERE-- FIND AN EMPTY ONE
         LA    R2,TABLE              R2 AGAIN= START OF TABLE
LOOP2    CLI   ENTSTAT,0             ZERO IF AVAILABLE
         BZ    FILLIN                YES, BRANCH  (I'LL USE IT)
         BXLE  R2,R0,LOOP2           NO, TRY AGAIN
***********************************************************************
*           THE CALLING DEVICE IS NOT IN THE TABLE, AND THE TABLE       *
*           IS FULL.  THE NUMBER OF DEVS IS MORE THAN I CAN HANDLE.     *
*           RE-ASSEMBLY IS IN ORDER TO EXPAND THE TABLE.                *
*                                                                       *
***********************************************************************
         ABEND 555,DUMP
FILLIN   EQU   *
         STH   R7,ENTCTB             SAVE CTB OFFSET AS PLLO ID
GOTDEV   EQU   *
         LR    R5,R2                 SET R5 TO TABLE ENTRY
         BR    R14                   RETURN
         EJECT
```

Figure 4-2 (Part 3 of 4). Sample Physical Terminal Output Edit Routine

```
**********************************************************************
*              SUBROUTINE TO CONVERT CCUNT TO EBCDIC
*              PRIOR TO MOVING INTO NEW PREFIX
*
**********************************************************************
         SPACE 1
CONVRT   LA    R4,1(R4)
         CVD   R4,CNVFIELD
         UNPK  SAVEFLD(8),CNVFIELD(8)    BUILD EBCDIC NUMBER
         OI    SAVEFLD+7,X'F0'      STRIP SIGN
         LA    R1,7                 SET COUNT
         LA    R5,SAVEFLD+2         SET START
STRIP    CLI   0(R5),X'F0'          STRIP HI ORDER ZEROS
         BNE   MOVEIT
         MVI   0(R5),BLANK          BLANK IT
         LA    R5,1(R5)
         BCT   R1,STRIP
MOVEIT   LA    R5,SAVEFLD+2         CALC WHERE TO MOVE FROM
         BR    R14
         EJECT
         LTORG
         SPACE 2
**********************************************************************
*                                                                    *
*        EACH TABLE ENTRY CONSISTS OF   XX00DDDD                     *
*                                                                    *
*              00 MEANS ENTRY INACTIVE  (INDICATES 1-ST SEG)         *
*        XX=   E7 (C'X') MEANS BUFFER POOL EXISTS (THIS IS ITS ID)   *
*              FF MEANS POOL DELETED (FOR MULTI-SEG MSGS)            *
*                                                                    *
*              SECOND BYTE IS X'00'  (NOT ASSIGNED FOR NOW)          *
*                                                                    *
*              DDDD= DEV'S CTB OFFSET  (FOR A UNIQUE POOL ID)        *
*                                                                    *
*                                                                    *
NUMENTS  EQU   5                    NUMBER DEVICES USING THIS ROUTINE *
*                                                                    *
TABLE    DC    (NUMENTS)F'0'        LIST OF BUFR POOL ID'S           *
LASTENT  EQU   *-4                  ADDR OF LAST ENTRY IN LIST       *
*                                                                    *
**********************************************************************
         SPACE 3
CR       EQU   X'15'                LINE FEED
BLANK    EQU   C' '                 BLANK
DASH     EQU   C'-'                 DASH
PREFIX   EQU   33                   00020303
PTERM    DC    C'PTERMOUT-'
CNVFIELD DC    D'0'
SAVEFLD  DC    CL8'0'
         SPACE 4
ENTRY    DSECT ,                    LAYOUT OF 'TABLE ENTRYS
ENTSTAT  DS    X                    ENTRY'S STATUS
ENTSPARE DS    X'00'
ENTCTB   DS    XL2                  CTB OFFSET
         REQUATE
         ICLI  CLBBASE=0,CTBBASE=0,CNTBASE=0
         ISCD  SCDBASE=0
         END
```

Figure 4-2 (Part 4 of 4). Sample Physical Terminal Output Edit Routine

TRANSACTION CODE (INPUT) EDIT ROUTINE

IMS/VS gives you the ability to specify, during system definition, the inclusion in the IMS/VS control program nucleus of one or more user-supplied input edit routines. This allows the user to edit input messages before they are enqueued for scheduling. When IMS/VS is executed, this user edit function is performed in addition to the basic IMS/VS edit function or MFS (message format service) editing and subsequent to this function. The user can specify, to system definition, up to 255 editing routines, and also which edit routine is to be used, by transaction type.

The user should know the contents and meaning of the various fields in the IMS/VS control blocks (defined in the IMS/VS Program Logic Manual, Volume 1 of 3. He can test them in an edit routine. Under no circumstances should an edit routine modify any of these blocks.

If specified, a user-supplied input edit routine gains control after each message data segment is processed by the IMS/VS basic input edit or MFS. Transaction code validity and security will have already been checked. A user edit routine is not entered if the transaction code is the only data in the message segment, and the transaction is a conversational transaction.

Interface

- Registers on Entry

Upon entry to a user-supplied transaction code edit routine, all registers to be used must be saved. The following interface applies:

R1 — The buffer location of the input message segment after translation to EBCDIC but prior to the IMS/VS basic editing. The first two bytes of the buffer contain a binary count of the message length. The third and fourth bytes of the buffer are binary zeros. The fifth byte contains the first byte of message text. The binary count includes the four-byte prefix. Because the input buffer has no relationship to the segment after it has been processed by MFS, this register will point to the resultant segment (same as DECAREA) if the message was processed by this service instead of the basic input edit service. The fourth byte of the message segment (Z2) is X'00' if the basic edit service was used. It contains a X'01', X'02', or X'03' if MFS was used, signifying that option 1, 2, or 3 respectively was selected for the message by the format designer.

R7 — CTB address for the physical terminal from which the message segment was entered.

R9 CLB address for the communication line from which the message was entered. This control block starts with a BTAM DECB. The DECAREA field in the DECB contains the address of a buffer. This buffer contains the input message segment after IMS/VS editing. The first four bytes are two bytes of binary count (length of this message segment) and two bytes of binary zeros as above. The length of this buffer is equivalent to the binary count pointed to by register 1 plus 10 if basic editing was performed.

If the input was processed by MFS, the length of this buffer is given by the first two bytes of the buffer (length of this message segment). No extra space is provided in this buffer for user-written edit routines.

R13 Save area address for use by an edit routine. The first three words in the save area must not be modified.

R14 Return address to IMS/VS.

R15 Entry point address to the invoked edit routine. The entry point name and load module name for an edit routine must be the same and equivalent to the name used for the edit routine in system definition.

If the input was processed by the IMS/VS basic edit, you can use either the message segment in the buffer addressed by register 1, or that addressed by the DECAREA field as input to edit. If the input was processed by MFS, you can use only the message segment addressed by the DECAREA field as input to edit.

The user-supplied edit routine must place the text of the user-edited message segment to be returned to IMS/VS in the buffer addressed by the DECAREA field. If the input was processed by the IMS/VS basic edit, this buffer is always 10 bytes greater than the two-byte binary count at the beginning of the message segment, and you can expand the length of the message segment. Alternatively, you can reduce the length of the message segment to any desired size. The format of the user-edited message segment in the buffer upon return to IMS/VS must be two bytes of binary count, two bytes of binary zeros (except when input was processed by MFS -- the second two bytes should not be changed), and edited text.

- Registers on Exit

Upon return to IMS/VS, all registers must be restored except register 15.

R1 Message number if register 15 contains a value of 12; otherwise ignored.

R15 Return codes:

00 - Segment is processed normally.
04 - Segment is canceled.
08 - Message is canceled, and the terminal operator is notified.
12 - Message is canceled, and the user message identified by register 1 is sent to the terminal.

Any other value causes the message to be canceled and the terminal operator to be notified.

Example of a Transaction Code Edit Routine

Assume a multisegment transaction named ICS. Normally, the first segment of this message contains ICS GN, meaning to get the next segment of a given message, or ICS CAN, meaning to cancel this message. A user-supplied edit routine allows further input flexibility, as shown in the following decision table.

	MSG AS REC'D AND EDITED BY IMS/VS	MSG AS REEDITED BY USER EDIT ROUTINE
First Segments	ICS GN ICS ICS CAN Any other	As received ICS GN msg canceled msg canceled
Other Segment	GN CAN Any other	As received msg canceled segment canceled

The transaction code edit routine allows the input of a shortened format for the ICS GN message segment.

Figure 4-3 is an example of a transaction code edit routine.

```
CSMB     TITLE 'DFSCSMBO, SAMPLE SMB DESTINATION EDIT ROUTINE.'
*****************************************************************
*****************************************************************
*        SAMPLE TRANSACTION CODE EDIT RCUTINE                   *
*****************************************************************
*        * MESSAGE RECEIVED        * RETURN CODE  RETURNED MSG  *
*****************************************************************
*        * 'ICS GN'                *  RC = 0      'ICS GN'      *
* FIRST  * 'ICS   '                *  RC = 0      'ICS GN'      *
* SEGMENT* 'ICS CAN'               *  RC = 8      'ICS CAN'     *
*        * 'ICS MSGXXX'            *  RC =12      'ICS MSGXXX'  *
*        * XXX=MSG NUMBER 000-999  *                            *
*        * ANY OTHER               *  RC = 4      AS RECEIVED   *
*****************************************************************
*        *                         *                            *
* OTHER  * 'GN'                    *  RC = 0      'GN'          *
* SEGS   * 'CAN'                   *  RC = 8      'CAN'         *
*        * ANY OTHER               *  RC = 4      AS RECEIVED   *
*****************************************************************
* RETURN CODE                      MEANING                      *
*****************************************************************
*        0                         PROCESS MESSAGE SEGMENT      *
*        4                         CANCEL MESSAGE SEGMENT       *
*        8                         CANCEL MESSAGE               *
*       12                         CANCEL MESSAGE AND USE R1 FOR *
*                                  PTR IN USER MESSAGE TABLE    *
*****************************************************************
*****************************************************************
         SPACE
DFSCSMBO CSECT
         PRINT NOGEN
         SAVE  (14,12),,*
         LR    R12,R15             COPY FCR BASE
         USING DFSCSMBO,R12
         USING CTB,R7
         USING IECTDECB,R9
         L     R8,DECAREA          A(MESSAGE SEGMENT)
         CLC   0(2,R8),MAXLTH      CHECK FOR MAXIMUM SEG LENGTH
         BH    CANSEG              IF TOO BIG - CANCEL SEGMENT
         IC    R2,1(R8)            GET CNE BYTE CF LENGTH
         SH    R2,H5               GET EXECUTE LTH OF DATA
         EX    R2,MAKUPER          GET UPPER CASE
         TM    CTBFLAG3,CTB3SEG1   IS THIS THE FIRST SEGMENT?
         BZ    MULTSEG             NO - BRANCH
*                                  ******************************
*                                  * FIRST SEGMENT PROCESSING   *
*                                  ******************************
         CLI   1(R8),10            MUST BE AT LEAST 10
         BL    CANSEG              IF NC - CANCEL SEGMENT
         CLC   4(3,R8),=CL3'ICS'   MUST HAVE ICS
         BNE   CANSEG              IF NCT - CANCEL SEGMENT
         CLC   7(3,R8),BLANKS      3 BLANKS AFTER ICS?
         BNE   CKOPER              NO
         MVC   8(2,R8),=CL2'GN'    YES - SET GN DEFAULT
         B     RETURNO             AND EXIT
CKOPER   EQU   *                   CHECK OPERATOR
         CLC   7(3,R8),=CL3' GN'   GN REQUEST?
         BE    RETURNO             YES
         CLI   1(R8),11            MUST BE AT LEAST 11
         BL    CANSEG              NC - CANCEL SEGMENT
```

Figure 4-3 (Part 1 of 2). Sample Transaction Code Edit Routine

```
         CLC   7(4,R8),=CL4' CAN'  CANCEL REQUEST?
         BE    CANMSG              YES
         CLC   7(4,R8),=CL4' MSG'  CANCEL WITH USER MSG
         BE    USERMSG             YES
         B     CANSEG              OTHER - CANCEL SEGMENT
*                                  ********************************
MULTSEG  EQU   *                   * OTHER THAN FIRST SEGMENT     *
*                                  ********************************
         CLI   1(R8),6             MUST BE AT LEAST 6
         BL    CANSEG              IF NOT - CANCEL SEGMENT
         CLC   4(2,R8),=CL2'GN'    GN?
         BE    RETURN0             YES - OK
         CLI   1(R8),7             MUST BE AT LEAST 7
         BL    CANSEG              NO - CANCEL SEGMENT
         CLC   4(3,R8),=CL3'CAN'   CANCEL REQUEST?
         BE    CANMSG              YES
***      B     CANSEG              OTHER - CANCEL SEGMENT
         SPACE 2
CANSEG   EQU   *                   * CANCEL SEGMENT               *
         LA    R15,4               RC
         B     RETURN
CANMSG   EQU   *                   * CANCEL MESSAGE               *
         LA    R15,8               RC
         B     RETURN
USERMSG  CLI   11(R8),X'F0'        IS MSG NUMBER VALID?
         BL    CANSEG              NO - TREAT AS OTHER
         CLI   11(R8),X'F9'        IS MSG NUMBER VALID?
         BH    CANSEG              NO
         CLI   12(R8),X'F0'        VALID
         BL    CANSEG              RANGE
         CLI   12(R8),X'F9'        IS
         BH    CANSEG              FROM
         CLI   13(R8),X'F0'        000
         BL    CANSEG              TO
         CLI   13(R8),X'F9'        999
         BH    CANSEG              NUMERIC ONLY
         PACK  WORK1,11(3,R8)      CONVERT IT
         CVB   R1,WORK1            TO BINARY, PASS IT IN REG 1
         LA    R15,12              SET RC=12
         B     RETURNM             LEAVE R1 AS MSG NUMBER
RETURN0  EQU   *                   * RETURN CODE 0                *
         SR    R15,R15             RC
RETURN   EQU   *                   RETURN TO IMS
*                                  R13 POINTS TO CALLERS SAVEAREA
         L     R1,24(R13)          RESTORE R1 IF NOT MSG NUMBER
RETURNM  LM    R2,R12,28(R13)      RESTORE REGISTERS
         L     R0,20(R13)
         L     R14,12(R13)
         BR    R14
         SPACE 2
H5       DC    H'5'
MAXLTH   DC    H'84'               MAX SEG LENGTH - 80 DATA BYTES
MAKUPER  OC    4(1,R8),BLANKS      EXECUTED
BLANKS   DC    CL80' '             BLANKS
WORK1    DC    D'0'                USED FOR MESSAGE NUMBER
         SPACE 1
         REQUATE
         ICLI  CTBBASE=0,CLBBASE=0
         END
```

Figure 4-3 (Part 2 of 2). Sample Transaction Code Edit Routine

MESSAGE SWITCHING (INPUT) EDIT ROUTINE

A facility similar to the transaction code (input) edit is provided for message switching. The optionally supplied, user-written routine, whose CSECT and load module name must be DFSCNTE0, is included in the user's system at IMS/VS system definition time. Only one message switching edit routine can be specified for an IMS/VS online control program. This routine is specified (in the NAME macro) for inclusion with the online control program during system definition.

Interface

The interface between the IMS/VS control program and the user-supplied message switching edit routine is the same as previously defined for the transaction code edit routine.

Example of a Message Switching Edit Routine

The user-supplied edit routine might be used to identify, in the text of the output message to the output terminal, the logical terminal name and message number from which the message was entered.

Figure 4-4 is an example of a Message Switching Edit Routine.

Assume the following message is being entered from a logical terminal named 'XSYSI' and is input message number one.

    ABC SEND ALL XYZ MSGS TO THIS TERMINAL

The message as received at the output terminal associated with logical terminal name ABC has the input logical terminal name and input message number appended to it by the user's edit routine.

    ABC SEND ALL XYZ MSGS TO THIS TERMINAL XSYSI 1

In this example, the logical terminal input name is used. This name exists within the IMS/VS control block for the input logical terminal, the Communication Name Table (CNT). The CNT is addressed by a field in the Communication Line Block called CLBCNTPT. The field in the CNT containing the logical terminal name is called CNTNAME. Control blocks are defined in the <u>IMS/VS Program Logic Manual, Volume 1 of 3</u>.

```
CNTE     TITLE 'DFSCNTE0, SAMPLE CNT DESTINATION EDIT ROUTINE.'
***********************************************************************
*        SAMPLE MESSAGE SWITCHING EDIT                                *
*---------------------------------------------------------------------*
*        FUNCTION:                                                    *
*        THE LOGICAL TERMINAL NAME OF THE INPUTTING TERMINAL AND      *
*        THE MESSAGE NUMBER ARE ADDED TO THE END OF THE MESSAGE       *
*                                                                     *
*        REGISTERS ON ENTRY        ON EXIT                            *
*        R6     CNT                R6     CNT                         *
*        R7     CTB                R7     CTB                         *
*        R9     CLB                R9     CLB                         *
*        R14    RETURN ADDRESS     R15    RETURN CODE                 *
***********************************************************************
         SPACE 2
DFSCNTE0 CSECT
         PRINT NOGEN
         SAVE  (14,12),,*
         USING DFSCNTE0,R15
         USING CNT,R6
         USING CTB,R7
         USING IECTDECB,R9               CLB PCINTER
         SPACE
*************  FINC THE END OF THE PRE-ECITED MESSAGE  ****************
*                                                                     *
         L     R5,CECAREA                POINT TO MESSAGE
         LH    R4,0(,R5)                 LCAD OF 'DL'
         AR    R5,R4                     R5= ENC OF MESSAGE
         SPACE
*************  GET LOGICAL TERMINAL NAME, ANC ACD IT TO MSG  **********
*                                                                     *
         LH    R6,CTBCNTPT               ACCRESS OF CNT
         MVC   1(5,R5),CNTNAME           INSERT 5 CHARS OF NAME
         SPACE
*************  NOW FIND AND INSERT MESSAGE NUMBER  *******************
*                                                                     *
         LH    R3,CTBINCT                LCAD MSG NUMBER
         CVD   R3,MSGNUMP
         UNPK  MSGNUM(4),MSGNUMP+4(4)    * CCNVERT TO
         OI    MSGNUM+3,240              *   CHARACTERS
         MVC   7(3,R5),MSGNUM+1          SLIDE NUMBER NEXT TO NAME
         MVI   6(R5),C' '                BLANK SEPARATOR
         SPACE
*************  CHANGE 'DL' TO REFLECT NEW MSG LENGTH  ****************
*                                                                     *
         SR    R5,R4                     R5= START OF MSG (DL)
         LA    R4,9(,R4)                 NEW LENGTH IS 9 MORE
         STH   R4,0(,R5)                 REPLACE 'DL'
         RETURN (14,12),RC=0
*************  CONSTANTS  *********************************************
*                                                                     *
MSGNUM   DS    F
MSGNUMP  DS    D
         REQUATE
         ICLI  CNTBASE=0,CTBBASE=0,CLBBASE=0
         END
```

Figure 4-4.   Sample Message Switching Edit Routine

CONVERSATION ABNORMAL TERMINATION EXIT ROUTINE

A conversational process terminates abnormally when:

- A conversation is ended by an /EXIT or /START command.
- A conversational application program terminates abnormally during a conversation.
- A conversational program does not insert to a response PCB or to an alternate PCB that represents another conversational program.
- An uncorrectable IMS/VS conversational error occurs, such as an I/O error while reading or writing the scratchpad area.

The IMS/VS user can provide an application program to "clean-up," if required, when a conversation is prematurely terminated. Upon entry, this program's I/O PCB contains the name of the terminal that had its conversation abended. An exit routine to schedule the application program is required. IMS/VS provides an exit routine named DFSCONE0, or you can write your own. To use the IMS/VS-provided routine, you must:

- Define a transaction code named DFSCONE.
- Write a non-conversational application program to be invoked by DFSCONE.

When the exit routine (DFSCONE0) is finished, the IMS/VS conversational processor determines whether the transaction DFSCONE has been defined. If DFSCONE is not defined, conversation termination completes and the SPA is discarded. If DFSCONE is included, the conversational processor schedules the transaction DFSCONE with the SPA of the terminated conversation as a non-conversational single segment message.

As an alternative to the above, you can provide a more tailored exit routine. For example, you might want to interrogate the CCB to determine which transaction was in process when the conversation terminated, or to inspect the SPA to find out what had occurred before the conversation terminated. No DL/I calls can be issued. A message processing program should be scheduled to handle data base inquiries and updates, or extensive analysis of the conversation. The application program can output messages to the terminal associated with the terminated conversation.

To cause an application program to be scheduled, the exit routine should:

- Place the 8-byte name of the non-conversational transaction in the SPA (offset 6 bytes into the SPA).
- Set the desired length of the SPA.
- Insert information to be communicated to the scheduled program into the SPA.
- Set a return code of 04 in register 15.

The transaction code inserted into the SPA must be a valid, non-conversational transaction. If it is not, no action will take place.

Inclusion During System Definition

To include a user-written exit routine, you must replace the default DFSCONE0 in IMSVS.RESLIB with your own DFSCONE0 before link-editing the IMS/VS nucleus. To use the default DFSCONE0, you need only define the transaction DFSCONE.

Interface

- Registers on Entry

R0 — Cause of conversation termination.

Byte 0:

- 00 - A conversational application program has abended or IMS/VS has abended the conversation.
- 01 - The /EXIT command was issued by the terminal in conversation causing the conversation to be terminated. There is no pointer to the CCB or CTB.
- 02 - The /EXIT or /START command was issued by a terminal other than the one in conversation causing the conversation to be terminated.
- 04 - The input CNT could not be found. The master terminal is set as the input terminal.

Byte 1: Reserved

Byte 2: Reserved

Byte 3: Vector describing the calling reason.

- 00 - Conversational application program abended.
- 04 - While processing the conversation, an error occurred when reading or writing a disk SPA.
- 08 - /EXIT command for input or other (remote) terminal processed.
- 0C - /START LINE command processed for terminal in conversation.
- 10 - SPA received for an inactive conversation.

R1	Address of the SPA; if the SPA length field (first halfword) is binary zero, the SPA is unavailable (I/O error retrieving from disk).
	If R0=X'08', X'0C': the SPA was obtained from SPA data set. If R0=X'00', X'04', X'10', X'14': the SPA was obtained from the message.
R6	CCB address of the terminal in conversation if the conversation is still active. Zero if the conversation is already terminated.
R7	CTB address of the terminal in conversation if the conversation is still active. Zero if the conversation is already terminated.
R11	SCD address.
R13	Save area address. The first three words in the save area must not be changed.
R14	Return address to IMS/VS.
R15	Entry point to the user routine.

Note: The SPA length equals zero if a read SPA from a disk SPA-data set is unsuccessful due to an I/O error.

- Registers on Exit

Upon return to IMS/VS, all registers must be restored except R15.

R15 Return codes:

00 - Exit routine has completed all clean-up required; no further action is necessary. Terminate the conversation.

04 - Cause the transaction indicated in the name field of the SPA to be scheduled with the SPA (length indicated) to be used as the message and terminate the conversation.

Program Listing

The source listing of the default DFSCONE0 is shown in Figure 4-5. It is for reference only.

```
STMT   SOURCE STATEMENT

   1 ***********************************************************************
   2 *  DFSCONE0 WHICH MAY BE REPLACED BY A CUSTOMER WRITTEN ROUTINE.      *
   3 *                                                                     *
   4 *  IF A TRANSACTION IS DEFINED BY THE CUSTOMER (DFSCONE) TO BE SCHED  *
   5 *  UPON NON-PROGRAM CONTROLLED CONVERSATION TERMINATION, THIS ROUTINE *
   6 *  CAUSES THE LAST SPA TO BE ENQUEUED ON DFSCONE FOR CLEANUP PROCESS. *
   7 ***********************************************************************
   8 DFSCONE0  CSECT
   9           USING   DFSCONE0,R15
  10           MVC     6(8,R1),PROGNAME    MOVE IN SMB NAME FOR SCHEDULING
  11           LA      R15,4               SET SCHD RETURN CODE
  12           BR      R14
  13 *
  14 PROGNAME  DC      C'DFSCONE '
```

Figure 4-5. IBM-Supplied Conversation Abend Exit Routine

USER MESSAGE TABLE

IMS/VS users can create a message table for use by the following types of user edit routines:

- Physical terminal input edit routine
- Transaction code input edit routine
- Message switching input edit routine
- Message Format Service (MFS) segment edit routine (described in the *Message Format Service User's Guide*)

All user messages invoked by these routines should be generated in the user message table.

Definition Requirements

The following steps are required to use a user message table, and must occur prior to stage 2 of IMS/VS system definition:

- OPTIONS=(...USERMSGS,...) must be specified in the COMM macro during IMS/VS system definition.
- The user message table module must be named DFSCMTU0.
- Once assembled, DFSCMTU0 should be placed in the operating system partitioned data set defined by the USERLIB= operand of the IMSGEN macro during IMS/VS system definition.

User Message Table Format

The format of the user message table (DFSCMTU0) must be similar to IMS/VS system message tables such as DFSCMT00:

- The table must start with the instruction BALR 15,14.
- Message numbers range from 1 to (and including) 999, in ascending sequence.
- The maximum size for the text of each message is 100 characters; the length must be an even value. If the message text exceeds 78 characters, it may be truncated if sent to a 3270 terminal.
- It is recommended that device control characters not be included in message text. IMS/VS always adds NEW LINE control characters to the beginning and end of the message.
- Each message entry must start on a halfword boundary. The entry format is:

```
label    DC  H'message number'
         DC  AL2 (entry length including number and length
             fields)
         DC  C'message text of even length'
```

- An entry with message number X'7FFF' signals the end of the message table.

Example

```
DFSCMTU0     CSECT
             BALR     15,14
M1           DC       H'001'
             DC       AL2(M5-M1)
             DC       C'text'
M5           DC       H'005'
             DC       AL2(M6-M5)
             DC       C'text'
M6           DS       0H
MEND         DC       X'7FFF'
             END
```

HARDWARE REQUIRED ROUTINES

7770-3 SIGN-ON EXIT ROUTINE -- DFSS7770

Since the 7770-3 is a switched device and the calling terminal may not be able to generate the alphameric characters required to form an /IAM command to sign on for an LTERM, IMS/VS requires that a sign-on routine be defined at system definition time for the 7770-3 lines in the system. This routine is invoked by the 7770-3 device-dependent module any time an input message or message segment is received from the line and a logical connection does not exist. Only one routine can be defined, and it applies to all 7770-3 lines in the system. A minimum user routine should validity check the input data received from the line, and use the data to develop an /IAM command to be passed on to IMS/VS. The user routine gains control before any IMS/VS security checking, validity checking, or editing functions are performed. The message text is in EBCDIC.

The sign-on routine can build an /IAM command in the input buffer, or can place a response message in the input buffer. Any response to be sent back to the caller must be in 7770-3 output vocabulary drum address form.

Through return codes to IMS/VS, the sign-on routine can cause the contents of the input buffer to be passed on into the system (/IAM command in buffer), or cause the contents of the buffer to be sent to the caller followed by a READ to allow retry. This routine can also cause the contents of the input buffer to be sent to the caller with a reset to the line to disconnect the caller after the response is sent.

Interface

- Registers on Entry

R1	Address of input data/buffer area received from the line.
R2	Length of the input data/buffer area.
R7	CTB address.
R8	CTT address.
R9	CLB address.
R11	SCD address.
R13	Save area address. The first three words in the save area must not be changed.
R14	Return address to IMS/VS.
R15	Entry point to the user routine.

- Data Format on Entry

The data format at entry and the relationship of registers 1 and 2 to the data are shown in Figure 4-6.

• Registers on Exit

All registers must be restored except registers 0, 1, 2, and 15. The contents of registers 0 and 1 are ignored by IMS/VS.

R2 The length of the data now in the input buffer area that was pointed to by R1 on entry.

R15 Return codes:

- 00 - Continue input processing with the contents of the input buffer.
- 04 - Send the contents of the input buffer to the caller, followed by a read. Allows retry of sign on operation.
- 08 - Send the contents of the input buffer to the caller, followed by a disable to disconnect the caller.

For return codes 04 and 08, the contents of the input buffer to be sent to the caller must be in drum address form, because no translation is performed before the data is sent to the caller. It is also your responsibility to determine when a sequence of sign-on attempts should be terminated with a reset operation.

Error Conditions

IMS/VS stops the line and generates a message to the master terminal for either of the following sign-on routine error conditions:

- The return code from the sign-on routine exceeds 8.
- The count value returned in R2 is greater than the available space in the buffer.

After the line has been stopped, system messages can still be transmitted to the 7770-3. The sign-on exit routine is not invoked.

Inclusion During System Definition

A usable sign-on routine is supplied with the system in IMSVS.LOAD. This routine automatically signs the caller on for the INQUIRY LTERM whenever the 7770-3 answers a call and receives data. As supplied, this routine is transparent to the caller. If the supplied module is to be used, it is your responsibility to move the module from IMSVS.LOAD to the user library specified in the IMSGEN statement before Stage 2 of system definition is executed.

Program Listing

For further information on the IMS/VS-supplied sign-on routine, see the _IMS/VS Program Logic Manual, Volume 1 of 3_. The source listing of the sign-on routine, DFSS7770, is shown in Figure 4-6. It is for reference only.

```
S777     TITLE 'COMM, SIGN ON MODULE FOR 7770 MODEL 3'
***********************************************************************
*                                                                     *
********************** 7770 AUTOMATIC INQUIRY ************************
**********************     SIGN-ON MODULE      ***********************
*                                                                     *
*  THIS MODULE RECEIVES CONTROL FROM THE 7770 DEVICE DEPENDENT MODULE *
*  WHEN A READ HAS COMPLETED BUT A LOGICAL CONNECTION HAS NOT BEEN    *
*  ESTABLISHED.                                                       *
*                                                                     *
*  THIS MODULE SETS THE PROPER FLAGS AND FIELDS TO INDICATE THAT THE  *
*  TERMINAL IS SIGNED ON FOR THE INQUIRY LOGICAL TERMINAL.            *
*                                                                     *
*  BLOCKS AND TABLES:                                                 *
*                                                                     *
*                  THIS MODULE USES THE CLB, CTB, AND THE CNT         *
*                  THIS MODULE RECEIVES ACCESS TO THE INPUT DATA,     *
*                  CTB, CTT, CLB, CNT, AND THE SCD.                   *
*  SIZE OF MODULE:                                                    *
*                  THIS MODULE CONTAINS APPROXIMATELY 54 BYTES OF     *
*                  CODE.                                              *
*  INTERFACE                                                          *
*     REGISTERS                                                       *
*       ON ENTRY:                                                     *
*                  R1       ADDRESS OF INPUT DATA                     *
*                  R2       LENGTH OF INPUT DATA                      *
*                  R7       HAS DIAL CTB ADDRESS                      *
*                  R8       HAS CTT ADDRESS                           *
*                  R9       HAS CLB ADDRESS                           *
*                  R11      HAS SCD ADDRESS                           *
*                  R13-R15  STANDARD O.S. LINKAGE REGISTERS           *
*                                                                     *
*                  THE INPUT DATA AREA HAS THE FOLLOWING FORMAT:      *
*                                                                     *
*                  *---------------*                                  *
*                  | REG 2 COUNT   |----*--------------------------* *
*                  *---------------*    |                          | *
*                                       V      |        9         NV *
*                                       *------+-----------+---------* *
*                                       | CTB  |           |  DATA   | *
*                                     *->| LINE | 9 BLANKS  |   IN    | *
*                                     | | NO.  |           | EBCDIC  | *
*                                     | *------+-----------+---------* *
*                  *---------------*  |                               *
*                  | REG 1 ADDR.   |-*                                *
*                  *---------------*                                  *
*                                                                     *
*                                                                     *
*       ON EXIT:                                                      *
*                  R2       LENGTH OF DATA IN BUFFER                  *
*                  R15      RETURN CODE                               *
*                  ALL OTHER REGISTERS ARE RESTORED.                  *
*                                                                     *
*     RETURN CODES:                                                   *
*                                                                     *
*        0 - CONTINUE INPUT PROCESSING WITH CONTENTS OF THE BUFFER    *
*        4 - SEND CONTENTS OF BUFFER TO CALLER FOLLOWED BY READ TO    *
*            ALLOW RETRY.  OUTPUT MUST BE IN DRUM ADDRESS FORM.       *
*        8 - SEND CONTENTS OF BUFFER TO CALLER FOLLOWED BY A DISABLE *
*            TO DISCONNECT THE CALLER.                                *
```

Figure 4-6 (Part 1 of 2). IBM-Supplied 7770-3 Sign-On Exit Routine

```
*                                                                      *
*  ABENDS:                                                             *
*                                                                      *
*     NOT APPLICABLE                                                   *
*                                                                      *
************************************************************************
         EJECT
DFSS7770 CSECT
*
         USING *,R12                        BASE REGISTER
         USING SCD,R11                      --> SCD
         USING CNT,R10                      --> CNT (INQUIRY)
         USING IECTDECB,R9                  --> CLB
         USING CTT,R8                       --> CTT
         USING CTB,R7                       --> CTB
*
         SAVE  (14,12),,S777023                                    V1.1
*
         LR    R12,R15
         L     R10,CTBCNT               GET INQUIRY CNT OFFSET     V1.1
         ST    R10,60(,R13)                 SET CNT ADDRESS TO BE PASSED BAC
         NI    CTBFLAG1,NIMASK-CTB1PRES  RESET PRESET FLAGS        V1.1
         NI    CTBFLAG2,NIMASK-CTB2LOCK-CTB2TEST-CTB2EXCL  + OTHERS
         OI    CTBFLAG1,CTB1SIGN            DIAL CTB IS LOGICALLY CONNECTED
         OI    CNTFLAG1,CNT1SIGN            SIGN ON LTERM ONLY SPECIFICATION
         MVC   CLBPSCTB(4),CNTCTBPT POINT CLB TO CTB               V1.1
*
         RETURN (14,12),RC=0                RESTORE AND RETURN TO DEVICE MOD
         EJECT
*            *
*** EQUATES ***
*            *
         SPACE 3
NIMASK   EQU   255                          ALL BITS
R0       EQU   00                           R
R1       EQU   01                           E
R2       EQU   02                           G
R3       EQU   03                           I
R4       EQU   04                           S
R5       EQU   05                           T
R6       EQU   06                           E
R7       EQU   07                           R
R8       EQU   08                           S
R9       EQU   09
R10      EQU   10                           E
R11      EQU   11                           Q
R12      EQU   12                           U
R13      EQU   13                           A
R14      EQU   14                           T
R15      EQU   15                           E
         EJECT
*                   *
*** DUMMY SECTIONS ***
*                   *
         SPACE 3
         ISCD  SCDBASE=0
         EJECT
         ICLI  CLBBASE=0,CNTBASE=0,CTBBASE=0,CTTBASE=0
         END
```

Figure 4-6 (Part 2 of 2). IBM-Supplied 7770-3 Sign-On Exit Routine

7770-3 INPUT EDIT ROUTINE -- DFSI7770

For the 7770-3, a user input edit exit has been implemented at the line level (from device module DFSDS030). This exit is primarily provided for a user edit routine to operate conversationally with the line (caller). It does basic (no data base reference) validity checking of input fields. (The 7770-3 has limited error detection.) It must also build a transaction, field by field, until enough data has been received and validity checked that the message (transaction) can be scheduled by IMS/VS. Message text has been translated to EBCDIC before the user routine is invoked.

Note: IMS/VS checkpoint/restart and recovery capabilities are not effective until the message has been scheduled into the system (see return codes 0 and 4 below).

In conjunction with the above concept of input editing, several additional entries and actions have been provided for the user input edit routine to allow the user edit to be continually aware of the line status from operation to operation.

Interface

- Registers on Entry

R0 — Entry vector value:

- 00 - Entry is for normal segment read completion from the line (caller).
- 04 - Re-entry for next segment of message after input edit has indicated that it has more segments to send to IMS/VS.
- 08 - The calling party on the line has hung up.
- 12 - The line is being stopped or the system is shutting down.

R1 — Address of the input data/buffer area. If the entry vector is 12, R1 is not used.

R2 — Length of the input data/buffer area. If the entry vector is 12, R2 is not used.

R7 — CTB address.

R8 — CTT address.

R9 — CLB address.

R10 — CNT address.

R11 — SCD address.

R13 — Save area address. The first three words in the save area must not be changed.

R14 — Return address to IMS/VS.

R15 — Entry point to the user routine.

• Data Format on Entry

The data format on entry is the same as for the 7770 sign-on exit routine. It is shown in Part 1 of Figure 4-6.

• Registers on Exit

All registers must be restored except registers 0, 1, 2, and 15. The contents of registers 0 and 1 are ignored by IMS/VS.

R2 — The length of the data now in the input buffer area that was pointed to by R1 on entry.

R15 — Return codes:

- 00 - The message segment in the input buffer is to be sent to IMS/VS and is the last segment of the message.
- 04 - The message segment in the input buffer is to be sent to IMS/VS and is not the last segment of the message. The next time the device module is entered for a READ, it enters the edit module with R1 pointing to a buffer area, and R2 containing the amount of available area contained in the buffer. R0 contains the value of 04.
- 08 - The message in the input buffer is to be sent to the caller followed by a READ. R2 must contain the count for the message to be sent to the caller. The message must be in drum address form.
- 12 - Repeat the last output message for the caller.
- 16 - The contents of the input buffer should be sent to the caller with a reset to hang up the caller.

<u>Error Conditions</u>

IMS/VS stops the line and generates a message to the master terminal for any one of the following input edit module error conditions:

• The return code from the input edit module exceeds 16.

• The count value returned in register 2 is greater than the available space in the buffer (buffer overrun).

• The input-edit module sent a single segment message to IMS/VS after the caller has hung up and indicated that it had more segments to send to IMS/VS.

• The return code from the routine exceeds 8 after entered for disconnect indication.

After the line has been stopped, system messages can still be transmitted to the 7770-3. The input edit routine is not invoked.

Special Conditions

After the edit module has been entered with the 08 entry vector value indicating that the caller has hung up, the edit routine can use return codes 00 and 04 to continue sending data to IMS/VS before IMS/VS is notified of the line drop condition. During this mode of processing, return code 00 indicates the end of input edit control, and that the message should be enqueued for processing. Alternatively, a return code of 08 during this mode causes the message to be canceled, and terminates input edit control for this sequence.

Note: IMS/VS does not accept input for conversational transactions if the disconnect occurred during a WRITE operation. The response message from the conversational program is still in the queue, and therefore negates input operations.

No IMS/VS action can be specified if the edit module was entered with input vector 12. Returned parameters, if any, are not used, as the entry with entry vector 12 is an information-only entry. The return code value of 12 or 16 can only be returned after the user routine was entered for a normal READ completion.

Data Special Characters

The input data may contain one or more of the following special characters:

X'00'	For Invalid Input Line Codes
X'16'	For 2721 Cancel Key
X'26'	For EOB (on 2721 also '000' key and '#' Key as EOIs)
X'B0'	For 2721 Verify Key
X'B1'	For 2721 Repeat Key
X'B2'	For 2721 Function 1 (F1) Key
X'B3'	For 2721 Function 2 (F2) Key
X'B4'	For 2721 Function 3 (F3) Key
X'B5'	For 2721 Function 4 (F4) Key
X'B6'	For 2721 Function 5 (F5) Key
X'B7'	For 2721 ID X'19' Code
X'B8'	For 2721 ID X'59' Code
X'B9'	For 2721 ID X'21' Code
X'BA'	For 2721 ID X'61' Code
X'FA'	For 2721 00 Key and for TOUCH-TONE (or equivalent) Phone '*' Key when working on the ABB' Code Line Interface

Inclusion During System Definition

IMS/VS supplies a basic input edit routine for the 7770-3 as module DFSI7770 in IMSVS.LOAD. If you want to use the supplied module, it is your responsibility to move the supplied module from IMSVS.LOAD to the user library specified in the IMSGEN statement. If you have written your own input edit routine, that module must be placed into the user library specified in the IMSGEN statement prior to system definition. The module must be named and have an entry point with the name DFSI7770.

Program Listing

For more information on the IMS/VS-supplied input edit routine, see the description of module DFSI7770 in the *IMS/VS Program Logic Manual, Volume 1 of 3*. The source listing of the IMS/VS-supplied module is shown in Figure 4-7. It is for reference only.

```
I777     TITLE 'COMM, INPUT EDIT FOR 7770 MODEL 3'
DFSI7770 CSECT
***********************************************************************
*                                                                     *
*    7770 USER INPUT EDIT MODULE SUPPLIED BY IMS                      *
*                                                                     *
*   . THIS MODULE ASSUMES NO RESPONSIBILITY FCR TRANSMISSION ERROR    *
*     DETECTION OR CORRECTION.                                        *
*                                                                     *
*   . A MESSAGE IS ASSUMED COMPLETE AND NO ATTEMPT WILL BE MADE TC    *
*     SEGMENTIZE INPUT DATA                                           *
*                                                                     *
*   . THE FIRST TWO CHARACTERS CF THE DATA IS ASSUMED TC CONTAIN A    *
*     DEFINED TRANSACTION CODE CR LOGICAL TERMINAL NAME               *
*                                                                     *
*   . INPUT PASSED BY THIS MODULE WILL BE 1 BYTE LCNGER THAN THE DATA *
*     INPUT FROM THE TERMINAL WITH A BLANK INSERTED AFTER THE SECOND  *
*     CHARACTER                                                       *
*                                                                     *
*   . EOI ONLY INPUT WILL BE SENT TO THE SYSTEM AS A NO TEXT MESSAGE  *
*                                                                     *
*   . ANY CHARACTER FOLLOWED BY EOI WILL BE SENT AS A REPEAT REQUEST  *
*                                                                     *
*   . AN INPUT OF 99+EOI WILL BE USED AS NORMAL SIGN/OFF; THE EDIT    *
*     ROUTIN E WILL RETURN TO THE DDM WITH A DISCONNECT REQUEST.      *
*                                                                     *
***********************************************************************
         EJECT
         SAVE  (14,12),,I779090
         USING DFSI7770,R12
         LR    R12,R15          SET BASE REGISTER
         CH    R0,TWLVE         VALIDITY CHECK ENTRY VECTCR
         BH    BADVECT          BRANCH IF TOO HIGH
         LR    R15,R0           COPY THE ENTY VECTOR
         B     ENTRY(R15)       GC TO PROPER ROUTINE
ENTRY    EQU   *
         B     ENTRY1           0C READ CCMPLETION FROM LINE
         B     BADVECT          04 GET NEXT SHOULD NOT CCCUR FOR THIS
         B     ENTRY2           08 LINE DISCCNNECT ENTRY
         B     RETURN           12 NO ACTION ON LINE STOP CR SHUTDCWN
         EJECT
ENTRY1   EQU   *
         CH    R2,TWLVE         CHECK NC. DATA CHARS REC'D
         BNH   SPECIAL              LESS THAN 3 CHAR IS FUNCTION REQUEST
         CH    R2,THIRTEEN           TWO DATA CHAR + ECI ?
         BNE   MOVER                 BR IF NOT
         CLC   10(2,R1),=C'99'       IS IT 99 + EOI ?
         BE    SIGNOFF               BR IF YES
```

Figure 4-7 (Part 1 of 2). IBM-Supplied 7770-3 Input Edit Routine

```
MOVER    EQU   *
         MVC   0(2,R1),10(R1)        SET TRANSACTION CODE
         SH    R2,TWLVE         REMOVE OVERHEAD COUNT
         EX    R2,MOVTXT             MOVE REMAINDER OF DATA TEXT
         MVI   2(R1),X'40'           TRANSACTION SEPERATOR
         AH    R2,THREE              SET DATA LENGTH
         SR    R15,R15          SCHEDULE SEGMENT WITH EOT R.C.
         B     RETURN                RETURN MESSAGE TO ANALYZER
MOVTXT   MVC   3(1,R1),12(R1)
SIGNOFF  EQU   *
         SR    R2,R2                 NO MESSAGE FOR CALLER
         LA    R15,16                SET DISCONNECT REQUEST RC
         B     RETURN                AND GO HANG UP THE LINE
         EJECT
ENTRY2   EQU   *
         LA    R15,8                 CANCEL ANY MESSAGE IN PROCESS
RETURN   EQU   *
         L     R14,12(,R13)     GET RETURN ADDRESS
         LM    R3,R12,32(R13)   R0,R1 NOT RESTORED. R15,R2 PRESET
         BR    R14              RETURN TO DEVICE MODULE
*
SPECIAL  EQU   *   THIS SECTION DEPENDENT ON COMPARE IN ENTRY1 CODE..
         LA    R15,12                    SET REPEAT VECTOR
         BE    RETURN                    AND DO REPEAT IF 2 CHARS REC'D
         MVI   0(R1),EOT                 SET EOT ONLY FOR NO TEXT MESG
         LA    R2,1                      AND SET DATA COUNT
         LA    R15,0                     AND SET FOR EOT RETURN
         B     RETURN
         EJECT
BADVECT  EQU   *
         SR    R15,R15          IF BAD INPUT VECTOR SET EOT R.C.
         B     RETURN                    AND TRY TO CONTINUE
*              CONSTANTS AND DSECTS FOR INPUT EDIT
         SPACE 3
THIRTEEN DC    H'13'
TWLVE    DC    H'12'
THREE    DC    H'3'
EOT      EQU   055
         REQUATE
         END
```

Figure 4-7 (Part 2 of 2). IBM-Supplied 7770-3 Input Edit Routine

The IMS/VS user has the ability to install a 7770-3 with an installation-tailored vocabulary. IMS/VS cannot, of course, predict this vocabulary. For this reason, an output edit exit is implemented to allow a user-written module to inspect system messages and terminal-to-terminal message switch messages and convert them, at the user's discretion, to a message that is compatible with his vocabulary.

Interface

The output edit module receives control on system messages and message switches. It does not receive control for a message from an application program that is a response to an input transaction.

- Registers on Entry

R1	Address of the output message segment.
R2	Length of the output message segment.
R7	CTB address.
R8	CTT address.
R9	CLB address.
R10	CNT address.
R11	SCD address.
R13	Save area address. The first three words in the save area must not be changed.
R14	Return address to IMS/VS.
R15	Entry point to the user routine.

- Data Format on Entry

Before control is given to the output edit module, IMS/VS edits the output message into the output buffer until the end of message is reached or the buffer is full. The buffer contains only output message data in EBCDIC.

- Registers on Exit

All registers must be restored except registers 0, 1, 2, and 15. The contents of registers 0 and 1 are ignored by IMS/VS.

R2 The length of the data now in the output buffer area that was pointed to by R1 on entry.

R15 Return codes:

00 - No action taken by the output edit. IMS/VS is to continue sending the message and any further segments without routing control to the output edit module.

04 - IMS/VS is to send the current contents of the buffer to the line, and the output edit module desires to gain control for any further segments of this message.

08 - The contents of the buffer have been changed. IMS/VS is to send what is now in the buffer and ignore (dequeue and not send) any further segments of the message.

Error Conditions

IMS/VS stops the line and generates a message to the master terminal for any one of the following output edit module error conditions:

- The return code from the output edit module exceeds 8.
- The count returned in register 2 is negative or zero.
- The count returned in register 2 is greater than the available buffer space (buffer overrun).

After the line has been stopped, system messages can still be transmitted to the 7770-3. The output edit routine is not invoked.

Special Conditions

The supplied output edit module makes the following assumptions:

- The vocabulary of the 7770-3 contains the phonetic equivalents for the numbers 0 through 9 and that the translate table supplied by the user converts the EBCDIC numbers to their vocabulary equivalents.
- The prefix phrase (in drum address form) to be sent for system messages follows the user translate table, and the orientation phrase and has the form nppp, where n is a single byte containing the count of the number of drum address bytes (p) following. The orientation phrase has the format nppp.
- Because of the variable nature of the 7770-3 vocabulary, the system definition utility requires that you supply the output translate table for the 7770-3. It is also your responsibility to provide the required orientation phrase also to be used for system message conversion.

Inclusion During System Definition

If the IMS/VS-provided output edit routine is to be used, it is your responsibility to move the module, DFSO7770, from IMSVS.LOAD to the user library specified in the IMSGEN statement prior to system definition.

If you are providing your own output edit routine, the module must be placed into the user library prior to system definition.

Program Listing

For more information on the IMS/VS-supplied output edit module, see the description of module DFSO7770 in the *IMS/VS Program Logic Manual, Volume 1 of 3*.

The edit routine program listing is shown in Figure 4-8. It is for reference only.

```
0777     TITLE 'COMM, OUTPUT EDIT FOR 7770 MODEL 3'
DFSO7770 CSECT
************************************************************************
*                                                                      *
*        7770  SYSTEM MESSAGE EDIT ROUTINE SUPPLIED BY IMS             *
*                                                                      *
*    . ANY MESSAGE SWITCHED TO THIS TERMINAL IS SENT AS IS WITH NO     *
*      MODIFICATION BY THIS PROGRAM                                    *
*                                                                      *
*    . SYSTEM 'COMMAND COMPLETED' MESSAGES ARE CONVERTED TO THE USER   *
*      SUPPLIED ORIENTATION PHRASE                                     *
*                                                                      *
*    . SYSTEM ERROR MESSAGES ARE REPLACED BY THE USER SUPPLIED ERROR   *
*      PHRASE PLUS THE IMS ERROR MESSAGE NUMBER                        *
*                                                                      *
************************************************************************
         EJECT
         SAVE  (14,12),,0779090
         USING DFSO7770,R12
         LR    R12,R15
         CH    R2,SEVEN         TOO SHORT FOR SYSTEM USE
         BL    MSGSW            YES
         CLC   1(3,R1),DFS      IS IT A SYSTEM MSG?
         BNE   MSGSW
         TM    4(R1),X'F0'
         BNO   MSGSW            NC
         TM    5(R1),X'F0'
         BNO   MSGSW            NO
         TM    6(R1),X'F0'
         BNO   MSGSW            AND NO
         USING CTT,R8
         L     R3,CTTSEND
         LA    R3,256(R3)       GET ACK PHRASE
         SR    R4,R4
         IC    R4,0(R3)         LENGTH OF PHRASE
         CLC   4(3,R1),059      COMMAND COMPLETE PHRASE
         BH    ERRMSG           NO - ERROR MSG
         EX    R4,MOVFRAZE
         LR    R2,R4            SET NEW TEXT LENGTH
         LA    R15,8            SET SKIP REST RETURN CODE
RETURN   L     R14,12(13)
         LM    3,12,32(13)
         BR    R14
         EJECT
```

Figure 4-8 (Part 1 of 2). IBM-Supplied 7770-3 Output Edit Routine

```
ERRMSG   EQU   *
         LA    R3,1(R3,R4)       POINT TO ERROR PHRASE
         IC    R4,0(R3)          GET LENGTH
         LA    R5,7(R1,R4)       STEP PAST POSSIBLE SELF DESTRUCTION
         MVC   0(3,R5),4(R1)     SAVE ERROR NUMBER OF MESSAGE
         EX    R4,MOVFRAZE       MOVE USER ERROR PHRASE
         LA    R3,0(R4,R1)
         MVC   0(3,R3),0(R5)     SET ERROR NUMBER
         LA    R2,3(R4)          SET NEW LENGTH
         LA    R15,8             SET SKIP REST RETURN CODE
         B     RETURN
*
         EJECT
MSGSW    EQU   *
         SR    R15,R15
         B     RETURN
         EJECT
*              CONSTANTS  AND DSECTS USED BY DFS07770
         SPACE 3
SEVEN    DC    H'7'
DFS      DC    C'DFS'
059      DC    C'059'
MOVFRAZE MVC   0(1,R1),1(R3)
         REQUATE
         ICLI  CTTBASE=0
         END
```

Figure 4-8 (Part 2 of 2). IBM-Supplied 7770-3 Output Edit Routine

7770-3 User Output Translate Table

Refer to the paragraph "Special Conditions" in the section of this chapter under "7770-3 Output Edit Routine -- DFS07770" for a description of the requirements for the user output translate table. Refer also to the user output translate table listing that follows in this chapter.

The orientation phrase is used by the device-dependent module. Before and after each READ, the phrase is sent to the terminal operator to indicate that a READ is pending on the line, and that he can now enter his data.

The prefix phrase is optional. It is used only by the supplied output edit routine DFS07770. See the description of module DFS07770 functions in this chapter.

Inclusion During System Definition: Before executing Stage 2 of IMS/VS system definition, the user-supplied translate table must be placed in the user library specified in the IMSGEN statement. The table must be a load module with the name specified in the LINEGRP statement.

<u>Sample Output Translate Table Listing</u>: The following is an example of a listing which might be produced for a user-supplied output translate table. See also "7770 User Input Edit Routine" and "7770 User Output Edit Routine" in this chapter.

```
OUT7770 CSECT
**********************************************************************
*                                                                    *
* 7770 OUTPUT TRANSLATE TABLE                                        *
*                                                                    *
*         THIS TABLE IS DEPENDENT UPON THE VOCABULARY PRESENT        *
*         ON THE 7770 DRUM TRACKS.                                   *
*                                                                    *
**********************************************************************

*                        0 1 2 3 4 5 6 7 8 9 A B C D E F
          DC      X'000102030405060708090A0B0C0D0E0F' 0 PRE
          DC      X'101112131415161718191A1B1C1D1E1F' 1 FORMATTED
          DC      X'202122232425262728292A2B2C2D2E2F' 2   MESSAGES
          DC      X'303132333435363738393A3B3C3D3E3F' 3
          DC      X'00000000000000000000000000000000' 4
          DC      X'00000000000000000000000000000000' 5
          DC      X'00000000000000000000000000000000' 6
          DC      X'00000000000000000000000000000000' 7
          DC      X'00010203040506070809000000000000' 8 LOWER
          DC      X'00111213141516171819000000000000' 9  CASE
          DC      X'00022232425262728290000000000000' A   ALPHA
          DC      X'00000000000000000000000000000000' B
          DC      X'00010203040506070809000000000000' C UPPER
          DC      X'00111213141516171819000000000000' D  CASE
          DC      X'00022232425262728290000000000000' E   ALPHA
          DC      X'16313233343536373839000000000000' F NUMERIC
*                        0 1 2 3 4 5 6 7 8 9 A B C D E F

****************************************
*                                      *
* 7770-3 IMS/VS ORIENTATION PHRASE     *
*                                      *
****************************************

          DC      AL1(ORIEND-*-1)      PHRASE LENGTH
          DC      X'2B0E'              PHRASE IS 'DIAL RELEASED'
ORIEND    EQU     *


****************************************
*                                      *
* 7770-3 IMS/VS OUTPUT PREFIX PHRASE   *
*                                      *
****************************************

          DC      AL1(OPREND-*-1)      PHRASE LENGTH
          DC      X'061919161900'      PHRASE IS 'E R R O R'
OPREND    EQU     *
          END
```

2972/2980 INPUT EDIT ROUTINE

An input edit routine is required to perform terminal-related functions inherent in the design of the 2972/2980 General Banking Terminal system. Usage and value of these functional characteristics are installation-oriented, and are therefore not performed by normal IMS/VS procedures. Control is passed to the 2972/2980 input edit routine to process each entered message segment after that message segment has been translated by IMS/VS.

Required Function

The 2972/2980 input edit routine must perform the following functions:

1. Determine the IMS/VS destination (SMB or CNT) of messages entered from a 2980 teller or administrative station.

2. Determine end-of-message of multisegment messages (by setting DECCSWST bit 7 to indicate EOM).

3. Reposition the entered data to the beginning of the input buffer for IMS/VS processing (the entered segment must be in standard IMS/VS input message format after edit processing).

In addition to performing the above required functions, the 2972/2980 input edit routine may add input terminal status information to the entered segment, such as the presence or absence of a passbook or auditor key on the input terminal. The input edit routine can initiate re-transmission of the last successfully transmitted message to a 2980 logical terminal by a return code to the calling routine.

IQF Considerations

If IQF is incorporated into the IMS/VS system and is to receive input from the 2980, the following additional steps must be taken by the input edit routine:

1. The input terminal status information must be separated from IQF elements by at least one blank.

2. If the input terminal status information is appended to the end of a segment, any preceding carriage return must be removed (replaced with a blank).

3. The input terminal status information must be defined to IQF as a null word.

4. In the edited segment, the input terminal status information must not be the initial characters of the segment.

Interface

Familiarity with IMS/VS terminal handling procedures and control blocks is required for a user to write an input edit routine to interface with IMS/VS routines in the IMS/VS control region. Examination of these control blocks may be required, but modification of IMS/VS control blocks by a user-written routine seriously endangers the integrity of the entire system.

• Registers on Entry

R0	Input buffer length.
R1	Address of the input area.
R2	Input data length. (The length of the area pointed to in register 1.)
R7	CTB address.
R9	CLB address.
R11	SCD base.
R13	Save area address. The first three words in the save area must not be modified.
R14	Return address to IMS/VS.
R15	Entry point to the user routine.

• Data Format on Entry

The format of the data contained in the buffer pointed to by register 1 at entry to the 2972/2980 input edit routine is shown below.

9 BLANKS	TERMINAL ADDRESS	ENTERED TEXT*

* If entry is from a 2980-4, the first byte of the entered text is the teller identification number.

• Registers on Exit

R2	Data length after edit (a zero length signifies a no data segment).
R10	The inputting CNT address if a retransmission of the last successfully outputted message is required.
R15	Return codes:
	0 - Process the entered segment.
	4 - Resend the last message to the CNT in register 10.

Inclusion During System Definition

The entry name (CSECT) of the 2972/2980 input edit routine must be DFS29800. Because it will be called directly by the IMS/VS 2972/2980 device dependent module (DFSDN110), the input edit routine must be link-edited with the IMS/VS control region nucleus.

IMS/VS provides a default input edit routine for the 2972/2980. The listing of the default routine is shown in Figure 4-9.

Program Listing

```
2980     TITLE 'COMM, SAMPLE 2980 INPUT EDIT EXIT ROUTINE'
DFS29800 CSECT
***********************************************************************
*                                                                     *
*        THIS IS A SAMPLE OF THE 2980 INPUT EDIT ROUTINE REQUIRED BY  *
* IMS/VS 2972/2980 DEVICE SUPPORT.  THE INPUT EDIT ROUTINE MUST PER-  *
* FORM THE FOLLOWING FUNCTIONS:                                       *
*                                                                     *
*     1. DETERMINE THE IMS/VS DESTINATION (SMB OR CNT) OF MESSAGES    *
*        ENTERED FROM A 2980 TELLER OR ADMINISTRATIVE STATION.        *
*                                                                     *
*     2. DETERMINE END-OF-MESSAGE OF MULTI-SEGMENT MESSAGES AND SET   *
*        DECCSWST BIT 7 AT END-OF-MESSAGE.                            *
*                                                                     *
*     3. REPOSITION THE ENTERED DATA TO THE BEGINING OF THE INPUT     *
*        BUFFER FOR IMS/VS PROCESSING.  THE ADDRESS OF THE INPUT      *
*        BUFFER IS PASSED TO THE EDIT ROUTINE IN REGISTER 1.          *
*                                                                     *
*        IN ADDITION TO PERFORMING THE ABOVE FUNCTIONS THIS SAMPLE    *
* ROUTINE ALSO DOES THE FOLLOWING:                                    *
*                                                                     *
*     1. DETERMINES THE INPUTING LOGICAL TERMINAL (CNT) FOR MESSAGES  *
*        ENTERED FROM A 2980-4 TO BE USED FOR SECURITY VALIDATION AND *
*        AS THE I/O PCB FOR THE APPLICATION PROGRAM.                  *
*                                                                     *
*     2. INITIATES RE-TRANSMISSION OF THE LAST SUCCESSFULLY OUTPUTED  *
*        MESSAGE TO ANY PHYSICAL TERMINAL.                            *
*                                                                     *
*        DETERMINATION OF INPUT DESTINATION IS NOT PERFORMED ON DATA  *
* ENTERED FROM A 2980-2 ADMINISTRATIVE STATION AS THIS TERMINAL CAN   *
* READILY USE THE STANDARD IMS/VS MESSAGE FORMAT.  DATA ENTRY FROM A  *
* 2980-1 OR 2980-4 TELLER STATION REQUIRE THE ENTRY OF A TRANSACTION  *
* CODE SEQUENCE IN THE FIRST SEGMENT OF ALL ENTERED MESSAGES (IMS/VS  *
* COMMANDS MUST BE ENTERED IN STANDARD IMS/VS FORMAT).  THE TRANSACT- *
* ION SEQUENCE MAY OCCUR ANYWHERE IN THE FIRST SEGMENT AND CONSIST OF *
* A DESIGNATED BEGIN CHARACTER, FOLLOWED BY A VALID IMS/VS TRANSACT-  *
* ION CODE TERMINATED BY ANY CHARACTER WHICH WHEN TRANSLATED BY IMS   *
* HAS A HEXADECIMAL VALUE LESS THAN X'C1', OR END OF MESSAGE SEGMENT. *
* IF A SCAN OF THE FIRST MESSAGE SEGMENT DOES NOT ENCOUNTER A VALID   *
* TRANSACTION SEQUENCE (IE: A BEGIN CHARACTER FOLLOWED BY NO MORE     *
* THAN EIGHT (8) CHARACTERS BEFORE THE TERMINATION CHARACTER),  THIS  *
* ROUTINE ASSUMES THE MESSAGE WAS ENTERED IN STANDARD IMS/VS INPUT    *
* MESSAGE FORMAT AND BYPASSES THE DESTINATION EDIT FUNCTION.  THE     *
* DESIGNATED BEGIN CHARACTERS SCANNED FOR ARE:                        *
*                                                                     *
*        X'41' NUMERIC ENTRY OF KEY 0 (MSGACK) FROM A 2980-1.         *
*        X'59' NUMERIC ENTRY OF KEY 15 (CODE) FROM A 2980-4.          *
*                                                                     *
*        END-OF-MESSAGE IS DETERMINED BY THE ENTRY OF A PERIOD(.) AS  *
* THE LAST CHARACTER OF THE LAST SEGMENT OF A MULTI-SEGMENT MESSAGE,  *
* OR AS THE LAST CHARACTER OF A SINGLE SEGMENT MESSAGE.               *
*                                                                     *
*        INPUTING TERMINAL STATUS INFORMATION IS APPENDED TO EACH MSG *
* SEGMENT IN THE FOLLOWING FORMAT:                                    *
*                                                                     *
```

Figure 4-9 (Part 1 of 6). IBM-Supplied 2972/2980 Input Edit Routine

```
*                                                                      *
*       AABC                                                           *
*                                                                      *
*       WHERE:   AA- IS A TWO (2) BYTE HEXADECIMAL FIELD CONTAINING    *
*                    TWO NINES (X'F9F9')                               *
*                 B- IS A 'P' (X'D7') TO INDICATE A PASSBOOK WAS       *
*                    PRESENT AT SEGMENT ENTRY (OR THE AUDITOR'S        *
*                    KEY WAS INSERTED ON A 2980-2);  OTHERWISE         *
*                    THIS CHARACTER IS AN 'N' (X'D5').                 *
*                 C- IS THE TELLER IDENTIFICATION CHARACTER FOR A      *
*                    2980-4.                                           *
*                     A - TELLER A WITHOUT SUPERVISOR KEY              *
*                     B - TELLER B WITHOUT SUPERVISOR KEY              *
*                     J - TELLER A WITH SUPERVISOR KEY                 *
*                     K - TELLER B WITH SUPERVISOR KEY                 *
*                    IF ENTRY WAS NOT FROM A 2980-4 THIS CHARACTER     *
*                    IS BLANK (X'40').  THE TELLER IDENTIFICATION      *
*                    CHARACTER IS REMOVED FROM THE INPUT TEXT.         *
*                                                                      *
*       DETERMINATION OF THE INPUTING LOGICAL TERMINAL (CNT) IS MADE *
* BY EXAMINATION OF THE NAMES OF THE CNTS ASSIGNED TO THE INPUTING     *
* PHYSICAL TERMINAL.  EACH CNT IS EXAMINED TO FIND ONE WITH A NAME     *
* WHOSE FIRST CHARACTER MATCHES THE TELLER IDENTIFICATION CHARACTER;  *
* IF ONE IS FOUND THE CNT CHAIN IS ALTERED TO MAKE THAT CNT THE FIRST *
* CNT IN THE CHAIN OF CNTS.  THE CNT CHAIN REMAINS UNALTERED IF NO CNT*
* IS FOUND.                                                            *
*                                                                      *
*       ENTRY OF THE CHARACTERS '&RESEND' AS THE ONLY CHARACTERS OF  *
* A MESSAGE WILL CAUSE THE LAST SUCCESSFULLY OUTPUTED MESSAGE TO BE    *
* RE-TRANSMITTED TO THE INPUTING TERMINAL.                             *
*                                                                      *
*   REGISTERS AT ENTRY:                                                *
*                                                                      *
*       R0     INPUT BUFFER LENGTH                                     *
*       R1     POINTS TO THE INPUT MESSAGE SEGMENT;  PREFIXED BY       *
*              NINE BLANKS, THE TERMINAL ADDRESS CHARACTER, THE        *
*              TELLER IDENTIFICATION CHARACTER(IF ENTERED FROM         *
*              A 2980-4), AND THE ENTERED TEXT.                        *
*       R2     DATA LENGTH                                             *
*       R7     CTB BASE                                                *
*       R9     CLB BASE                                                *
*       R11    SCD BASE                                                *
*       R13    CALLER'S SAVE AREA (MY SAVE AREA IS PRE-CHAINED)        *
*       R14    RETURN ADDRESS                                          *
*       R15    ENTRY POINT ADDRESS                                     *
*                                                                      *
*   RETURN REGISTERS:                                                  *
*                                                                      *
*       R2     DATA LENGTH AFTER EDIT                                  *
*       R10    CNT BASE                                                *
*       R15    RETURN CODE                                             *
*                                                                      *
************************************************************************
```

Figure 4-9 (Part 2 of 6). IBM-Supplied 2972/2980 Input Edit Routine

```
         REQUATE
         USING CTB,R7
         USING IECTDECB,R9
         USING CNT,R10
         USING SCD,R11
         USING DFS29800,R12
         SAVE  (14,12),,EDIT2980.5295    SAVE REGISTERS
         B     18(0,15)                          BRANCH AROUND ID
         DC    AL1(13)                           LENGTH OF IDENTIFIER
         DC    CL8'EDIT2980'                     IDENTIFIER
         DC    CL5'.5295'                        IDENTIFIER
         STM   14,12,12(13)                      SAVE REGISTERS
         LR    R12,R15               SET PROGRAM BASE
         L     R13,8(,R13)           STEP TO NEXT SAVE AREA
         SR    R15,R15               CLEAR RETURN CODE
         LR    R5,R1                 SAVE MESSAGE POINTER
         SH    R2,=H'10'             REMOVE BLANKS FROM LENGTH
         LTR   R6,R2                 SET LENGTH REG
         BNP   ZEROLNG               BRANCH IF NO DATA
         MVI   DESTLNG,0             SET DESTINATION LENGTH TO ZERO
         MVI   TELLERID,C' '         CLEAR TELLER ID
         LA    R14,10(,R5)           SET BEGIN OF TEXT
         TM    CTBFEAT,CTBFMOD4      2980-4?
         BZ    CKRESEND              NO
         MVC   TELLERID(1),0(R14)    SAVE TELLER ID
         LA    R14,1(,R14)           STEP TO TEXT
         BCTR  R6,0                  DECREMENT DATA LENGTH
         LTR   R6,R6                 NO DATA?
         BNP   ZEROLNG               YES
CKRESEND EQU   *
         CLI   0(R14),C'&&'          POSSIBLE RESEND REQUEST?
         BE    RESEND                YES
SETSCAN1 EQU   *
         TM    CTBFEAT,CTBFMOD2      2980-2?
         BO    SETSTAT               YES
         BCTR  R6,0                  REDUCE LENGTH FOR SCAN
         EX    R6,SCAN1              FIND BEGIN CHARACTER
         LA    R6,1(,R6)             RE-ADJUST LENGTH
         BC    10,SETSTAT            BRANCH IF NOT FOUND
         LA    R4,1(,R1)             1ST CHAR OF DESTINATION
         LA    R3,0(R6,R14)          POINT TO END OF SEGMENT
         LR    R1,R3                 SET END OF SECOND SCAN
         SR    R3,R4                 SCAN LENGTH
         BCTR  R3,0
         EX    R3,SCAN2              SCAN FOR SECOND DELIMITER
         BC    6,FOUNDIT             BRANCH IF FOUND
         BCTR  R1,0                  LAST CHARACTER WAS DELIMITER
FOUNDIT  EQU   *
         SR    R1,R4                 DESTINATION LENGTH
         CH    R1,=H'8'              VALID LENGTH?
         BH    SETSTAT               NO
         STC   R1,DESTLNG            STORE LENGTH
         MVC   DEST,0(R4)            AND DESTINATION
SETSTAT  EQU   *
         MVI   PASSBOOK,C'N'         INDICATE NO PASSBOOK
         CLC   9(1,R5),CTBTERM+1     NORMAL ADDRESS?
```

Figure 4-9 (Part 3 of 6). IBM-Supplied 2972/2980 Input Edit Routine

```
         BE     CKEOM                   YES
         MVI    PASSBOOK,C'P'           INDICATE PASSBOOK PRESENT
CKEOM    EQU    *
         LA     R4,0(R6,R14)            R4 = END OF SEGMENT
         LR     R8,R4
         BCTR   R8,0                    BACK UP TO LAST MSG CHARACTER
         CLI    0(R8),X'15'             ENDS WITH CARRIAGE RETURN?
         BNE    *+6                     NO
         BCTR   R8,0
         BCTR   R8,0
         CLC    0(2,R8),=C'**'          SEGMENT TO BE CANCELLED?
         BE     TESTMOD4                YES,  DON'T ADD STATUS INFO
         CLI    1(R8),COMMA             MORE SEGMENTS COMING?
         BE     NOTEOM                  YES
         CLI    1(R8),PERIOD            END-OF-MESSAGE?
         BNE    ADDSTAT                 NO
         OI     DECCSWST,X'01'          INDICATE END-OF-MESSAGE
NOTEOM   EQU    *
         LA     R8,1(,R8)
         LR     R4,R8                   R4 = END-OF-SEGMENT POINTER
         SR     R8,R14                  RE-CALCULATE SEGMENT LENGTH
         LTR    R6,R8                   AND TEST FOR NO-DATA SEGMENT
         BP     ADDSTAT                 BRANCH IF DATA SEGMENT
ZEROLNG  EQU    *
         SR     R6,R6                   SET ZERO LENGTH
RETURN   EQU    *
         L      R13,4(,R13)             GET CALLER'S SAVE AREA
         ST     R6,28(,R13)             STORE LENGTH IN R2 OF CALLER
         L      R14,12(,R13)            GET RETURN ADDRESS
         RETURN (0,12)                  AND RETURN,  RC IN R15
         LM     0,12,20(13)                          RESTORE THE REGISTERS
         BR     14                                   RETURN
ADDSTAT  EQU    *
         LA     R6,L'STATUS(,R6)        ADD STATUS LENGTH TO SEG LENGTH
         MVC    0(L'STATUS,R4),STATUS  ADD STATUS INFO TO SEGMENT
         CLI    DESTLNG,0               DESTINATION LENGTH ZERO?
         BE     MOVESEG                 YES
         MVC    0(8,R5),DEST            PUT DESTINATION IN SEGMENT
         AH     R5,DESTL                UPDATE TEXT POINTER
         LA     R5,1(,R5)               INSURE 1 BLANK AFTER DESTINATION
MOVESEG  EQU    *
         BCTR   R6,0                    REDUCE LENGTH FOR MOVE
         EX     R6,MOVE                 MOVE SEGMENT TO FRONT OF BUFFER
         LA     R6,1(,R6)               RE-ADJUST LENGTH
         CLI    DESTLNG,0               DESTINATION LENGTH ZERO?
         BE     TESTMOD4                YES
         LA     R6,1(,R6)               ADD 1 FOR BLANK AFTER TRAN CODE
         AH     R6,DESTL                ADD DEST LENGTH TO DATA LENGTH
TESTMOD4 EQU    *
         TM     CTBFEAT,CTBFMOD4        2980-4?
         BZ     RETURN                  NO
         BAL    R4,FINDCNT              FIND INPUTING CNT
         B      RETURN                  AND RETURN
```

Figure 4-9 (Part 4 of 6). IBM-Supplied 2972/2980 Input Edit Routine

```
RESEND   EQU   *
         CH    R6,=H'8'              VALID MESSAGE LENGTH?
         BNE   SETSCAN1              NO
         CLC   RESENDSQ,0(R14)       RESEND REQUEST?
         BNE   SETSCAN1              NO
         BAL   R4,FINDCNT            GET CNT ADDRESS
         L     R4,4(,R13)            GET CALLER'S SAVE AREA ADDRESS
         ST    R10,60(,R4)           STORE CNT ADDRESS IN CALLERS R10
         LA    R15,4                 SET RETURN CODE
         B     ZEROLNG               ZERO DATA LENGTH AND RETURN

FINDCNT  EQU   *
         L     R10,CTBCNT            FIRST CNT ON CTB
         LR    R3,R10
NEXTCNT  EQU   *
         CLC   TELLERID(1),CNTNAME NAME MATCH TELLER ID?
         BE    CNTFOUND              YES
         LR    R5,R10                POINTER TO PREVIOUS CNT
         L     R10,CNTCNTPT          FIRST CNT IN CHAIN
         LTR   R10,R10               LAST CNT?
         BNZ   NEXTCNT               NO, BRANCH
CNTRET   EQU   *
         LR    R10,R3                USE 1ST CNT IN CHAIN
         BR    R4                    RETURN

CNTFOUND EQU   *
         CR    R10,R3                1ST CNT?
         BE    CNTRET                YES
         MVC   CNTCNTPT-CNT(,R5),CNTCNTPT   SET PREVIOUS=NEXT
         ST    R3,CNTCNTPT           MAKE IT NEXT AFTER THIS CNT
         ST    R10,CTBCNT            MAKE THIS CNT FIRST IN CHAIN
         BR    R4                                   AND RETURN

***                                                             ***
*              CONSTANTS,  DSECTS,  AND EQUATES                   *
***                                                             ***

DESTL    DC    H'0'
DEST     DS    CL8
STATUS   DC    X'F9F90000'
TABLE1   DC    256XL1'00'
         ORG   TABLE1+65
         DC    X'41'
         ORG   TABLE1+89
         DC    X'59'
         ORG
TABLE2   DC    192XL1'FF',64XL1'00'
RESENDSQ DC    C'&&RESEND '
         ORG   *-1
         DC    X'15'
         ORG
```

Figure 4-9 (Part 5 of 6). IBM-Supplied 2972/2980 Input Edit Routine

```
SCAN1    TRT   0(0,R14),TABLE1
SCAN2    TRT   0(0,R4),TABLE2
MOVE     MVC   0(0,R5),0(R14)

DESTLNG  EQU   DESTL+1
PASSBOOK EQU   STATUS+2
TELLERID EQU   STATUS+3
PERIOD   EQU   X'4B'
COMMA    EQU   X'6B'
CTBFMOD4 EQU   X'02'               CTBFEAT SETTING IDENTIFYING A 2980-4
CTBFMOD2 EQU   X'01'               CTBFEAT SETTING IDENTIFYING A 2980-2

         LTORG
               =H'10'
               =H'8'
               =C'**'

         EJECT
         ICLI  CLBBASE=0,CTBBASE=0,CNTBASE=0
         PRINT NOGEN
         ISCD  SCDBASE=0
         END
```

Figure 4-9 (Part 6 of 6). IBM-Supplied 2972/2980 Input Edit Routine

3741 SIGN-ON EXIT ROUTINE -- DFSS3741

IMS/VS requires a sign-on exit routine to provide the /IAM command and /SET command values required to complete the logical connection between IMS/VS and the 3741. This routine is invoked by the 3741 device-dependent module after the physical connection occurs, and before any IMS/VS security checking, validity checking, or editing functions are performed. If the 3741 terminal identification feature is installed, IMS/VS passes the ID to the sign-on routine.

The 3741 sign-on exit routine must provide the names of the input logical terminal and the destination transaction code or logical terminal. If /IAM or /SET command passwords are required, they must also be provided by the sign-on routine. The sign-on routine may request disconnection from IMS/VS.

IMS/VS provides a default 3741 sign-on exit routine that may be modified by the user. The default routine provides names based on line identity, but does not provide passwords. It is capable of receiving 3741 terminal IDs but does nothing if one is received. A listing of the IMS/VS-provided routine is shown in Figure 4-10.

Interface

- Registers on Entry

R1	Address of the 4-byte terminal identification; if none is received, R1 contains zeros.
R2	Address of the 3741 Name Table into which the required names should be entered.
R6	Line buffer address.
R7	CTB address.
R8	CTT address.
R9	CLB address.
R11	SCD address.
R13	Save area address. The first three words in the save area must not be changed.
R14	Return address to IMS/VS.
R15	Entry point to the user routine.

- Registers on Exit

All registers must be restored except register 15.

R15 Return codes:

00 - Generate an /IAM PTERM LTERM command.

04 - Generate an /IAM LTERM command.

08 - Request disconnection from IMS/VS.

3741 Name Table Format

This table contains six 8-byte entries:

1. Password for /IAM PTERM command
2. Logical terminal name for /IAM LTERM command
3. Password for /IAM LTERM command
4. Transaction code name for /SET TRAN command
5. Logical terminal name for /SET LTERM command
6. Password for /SET command

Entries for which no data is provided are left blank.

Inclusion During System Definition

The entry name (CSECT) of the 3741 sign-on exit routine must be DFSS3741.

The IMS/VS-provided routine is supplied as DFSS3741 in IMSVS.LOAD. If you want to use the supplied module, you must move it from IMSVS.LOAD to the user library specified in the IMSGEN macro during IMS/VS system definition.

If you have written your own sign-on routine, you must place it into the user library specified in the IMSGEN macro prior to system definition. The module must be named and have an entry point DFSS3741.

Program Listing

For further information on the IMS/VS-supplied sign-on routine, see the IMS/VS Program Logic Manual, Volume 1 of 3. The source listing of the sign-on routine is shown in Figure 4-10.

```
         TITLE 'S374      DFSS3741 SWITCHED BATCH BSC USER SIGNON EXIT
               @BI65A'
************************************************************
*
*
*
*    MODULE NAME : DFSS3741
*
*
*    TITLE : SWITCHED BATCH BSC USER SIGNON EXIT
*
*
*    ENTRY POINT(S) : DFSS3741
*
*
*    FUNCTION :
*
*       + THE MODULE RECEIVES CONTROL FROM THE BATCH BSC SWITCHED
*         DDM AFTER PHYSICAL CONNECTION HAS BEEN MADE.
*       + THE FUNCTION OF THE MODULE IS TO CREATE THE NAMES REQUIRED
*         BY THE USER FOR THE DDM TO CONSTRUCT INTERNAL /IAM AND /SET
*         COMMANDS TO ACHIEVE LOGICAL CONNECTION AND SET THE INPUT
*         MESSAGE DESTINATION RESPECTIVELY.
*       + IF THE DEVICE TRANSMITS A 4 BYTE HARDWARE ID, THIS IS PASSED
*         TO THE EXIT.
*       + THE USER MAY OPTIONALLY DECIDE TO DISCONNECT THE TERMINAL
*         BY SETTING A RETURN CODE.
*
*
*    ENTRY INTERFACES :
*
*               REGISTERS AT ENTRY :
*
*                   R1      ADDRESS OF 4 BYTE H/W ID, 0 IF ABSENT
*                   R2      ADDRESS OF AREA TO RECEIVE CREATED NAMES
*                   R6      LINE BUFFER ADDRESS
*                   R7      CTB ADDRESS
*                   R8      CTT ADDRESS
*                   R9      CLB ADDRESS
*                   R11     SCD ADDRESS
*                   R13-R15 STANDARD OS/VS LINKAGE REGISTERS
*
*
*               DATA/OTHER :
*
*                   R2 POINTS TO A 6 BY 8 BYTE BLANK TABLE WHICH WILL
*                   RECEIVE THE USER REQUIRED NAMES.  THE FORMAT IS
*                   DESCRIBED UNDER EXIT INTERFACES.
*
*
         EJECT
*    EXIT INTERFACES :
*
*               REGISTERS AT EXIT (IF DIFFERENT FROM ENTRY) :
*
*                   R15     RETURN CODE
*
```

Figure 4-10 (Part 1 of 3). IBM-Supplied 3741 Sign-On Exit Routine

```
*
*              RETURN CODES :
*
GENPTERM EQU   0               CREATE /IAM PTERM LTERM COMMAND FORMAT
GENLTERM EQU   4               CREATE /IAM LTERM COMMAND FORMAT
DISCLINE EQU   8               DISCONNECT THE TERMINAL
*
*
*              DATA/OTHER :
*
*                  R2 POINTS TO THE COMPLETED 6 BY 8 BYTE TABLE
*                  CONTAINING THE USER REQUIRED NAMES. A BLANK NAME
*                  INDICATES WHERE AN ENTRY IS NOT REQUIRED.
*                  1       /IAM PTERM PASSWORD
*                  2       /IAM LTERM NAME
*                  3       /IAM LTERM PASSWORD
*                  4       /SET TRAN DESTINATION NAME
*                        OR
*                  5       /SET LTERM DESTINATION NAME
*                  6       /SET DESTINATION PASSWORD
*
*
*    EXTERNAL ROUTINES CALLED : NONE
*
*
*    MESSAGE NUMBERS : NONE
*
*
*    ABEND CODES : NONE
*
*
*
*
*
*************************************************************
         EJECT
DFSS3741 CSECT
         ISAVE (14,12),,S373015,TYPE=CHAIN INITIALIZATION
         USING SCD,R11              ADDRESS SCD
         USING IECTDECB,R9          ADDRESS CLB
         USING CTT,R8               ADDRESS CTT
         USING CTB,R7               ADDRESS CTB
         USING BUFBTAM,R6           ADDRESS LINE BUFFER
         USING NAMELIST,R2          ADDRESS PARAMETER AREA
         SPACE 2
         LH    R4,CTBLINNO          LINE NUMBER
         CVD   R4,CVDDWORD          CONVERT TO DECIMAL
         OI    CVDDWORD+7,X'0F'     ENSURE UNPACKABLE TO EBCIDIC
         UNPK  IAMLTERM+5(3),CVDDWORD+6(2) STORE IN IAM LTERM NAME
         MVC   SETTRAN+4(3),IAMLTERM+5 AND IN SET TRAN NAME
         SPACE
         MVC   IAMPTPWD,BLANK8      NULLIFY IAM PTERM PASSWORD
         MVC   IAMLTNME,IAMLTERM    INSERT IAM LTERM NAME
         MVC   IAMLTPWD,BLANK8      NULLIFY IAM LTERM PASSWORD
         SPACE
         MVC   SETTXNME,SETTRAN     INSERT SET TRANSACTION NAME
         MVC   SETLTNME,BLANK8      NULLIFY SET LTERM NAME
         MVC   SETPWD,BLANK8        NULLIFY SET PASSWORD
         SPACE
         LA    R15,GENPTERM         RC=0 TO REQUEST /IAM PTERM LTERM
         SPACE
```

Figure 4-10 (Part 2 of 3). IBM-Supplied 3741 Sign-On Exit Routine

```
RETURN   EQU   *
         L     R13,SAVELAST        POINT TO ENTRY SAVE AREA
         RETURN (14,12),T,RC=(15)  RESTORE REGISTERS
         SPACE 3
CVDDWORD DS    D
IAMLTERM DC    CL8'LTERMXXX'
SETTRAN  DC    CL8'TRANXXX '
BLANK*   DC    CL8'        '
         EJECT
*
*        BUFFER DSECT USED FOR 3741 INPUT AND OUTPUT
BUFBTAM  DSECT
BUFLNGTH DS    H                BUFFER LENGTH PASSED BY ANALYZER
BUFCURR  DS    H                OFFSET TO CURRENT POSITION IN BUFFER
BUFRESID DS    H                OFFSET TO LAST BYTE IN BUFFER
BUFDECTY DS    H                DECB DECTYPE FOR LAST I/O
BUFSAVRC DS    X                FIELD TO SAVE I/O CHECKER RC
BUFDL    DS    2X               FIELD TO HOLD DL OF FIRST SEGMENT
BUFZZ    DS    X                FIELD TO HOLD ZZ OF FIRST SEGMENT
BUFDATA  EQU   *                DATA READ/WRITTEN FROM/TO TERMINAL
         SPACE 2
NAMELIST DSECT                      PARAMETER AREA
IAMPTPWD DS    CL8              /IAM PTERM PASSWORD
IAMLTNME DS    CL8              /IAM LTERM NAME
IAMLTPWD DS    CL8              /IAM LTERM PASSWORD
SETTXNME DS    CL8              /SET TRAN DESTINATION NAME
SETLTNME DS    CL8              /SET LTERM DESTINATION NAME
SETPWD   DS    CL8              /SET PASSWORD
TERMLIST DS    CL16             DEFINE TERMINAL LIST
LENNMLST EQU   *
         EJECT
         REQUATE SAVE=YES
         ICLI  CLBBASE=0,CTBBASE=0,CTTBASE=0
         ISCD  SCDBASE=0
         END
```

Figure 4-10 (Part 3 of 3). IBM-Supplied 3741 Sign-On Exit Routine

CHAPTER 5. IMS/VS STORAGE ESTIMATES

This chapter provides guidelines for determining the general storage requirements for both the DB system and the DB/DC system. Worksheets and examples are provided within their respective section. When using the worksheets and examples to determine buffer pool requirements for high volume systems, the user should note that more storage may be required to attain desired performance than is indicated.

DATA BASE SYSTEM STORAGE REQUIREMENTS

Several major items comprise the main storage requirements for the operating system region in which the IMS/VS DB system operates.

1. IMS/VS Data Base system modules

2. IMS/VS Program Specification Block (PSB) and associated blocks

3. IMS/VS Data Base Description (DBD) and associated blocks

4. IMS/VS data base buffer pool

5. IMS/VS data base work pool

6. OS/VS modules

7. OS/VS control blocks, buffer pools, and work space

8. User's application program

Each of these items is discussed in detail so that the user can accurately estimate the OS/VS region main storage requirements for the DB system. A worksheet is provided on the following pages which can be used for accumulating the estimate. For a further discussion of all parameters, review the "Data Base Design Considerations" chapter in the _IMS/VS System/Application Design Guide_, and the "Data Base Description Generation" and "Program Specification Block Generation" chapters in the _IMS/VS Utilities Reference Manual_.

All main storage requirements defined in this chapter represent the virtual storage requirements. If you are running V=R, the real storage requirements equal the virtual storage requirements. If you are running V=V, the real storage requirements are a subset of the virtual storage requirements. The amount of real storage required is a function of the performance level you want. In general, an acceptable level of performance can be achieved when the real storage available is between 50% and 80% of the virtual storage required.

Reference Number	Description		Size
1	IMS Basic Modules.		_______
2	PSB Size.		_______
3	DMB Size.		_______
4	Data Base Buffer Pool Size.		_______
5	Data Base Work Pool.		_______
6	IMS/VS Data Base Organization Dependent Modules.		_______
7	OS/VS Data Base Organization Dependent Modules.		_______
8	OS/VS Control Blocks, Buffers, & Work Space.		_______
9	OS/VS Buffers	_______	
10	OS/VS Control Blocks & Work Space	_______	_______
11	Data Base System = Subtotal		_______
12	Application Program(s).	+	_______
13	Data Base System and Application Program(s). (Round to nearest multiple of 2K).		_______

IMS/VS MODULES -- BASIC

The initial set of IMS/VS modules is required, independent of the data base organizations used by the application program, and the manner in which the data bases are used. These modules represent the region controller and basic DL/I modules.

The storage requirement for these basic modules is about 25,000 bytes. For initialization, about 8,000 additional bytes of work space are required prior to loading your application program. These 8,000 bytes are subsequently available for other use.

IMS/VS PSB (PROGRAM SPECIFICATION BLOCK)

Associated with each application program is a PSB. One PSB is required for each data base system execution. The PSB, as it exists in IMSVS.PSBLIB, is converted to an internal format for use by DL/I. If the data base control blocks are obtained from IMSVS.ACBLIB, the necessary conversion has already been done. If the PSB is obtained from IMSVS.PSBLIB, the PSB is converted to an internal format prior to use. In any case, the size requirements are the same. The size of the PSB is calculated with the following formula:

PSB = PSB Prefix Size + Work Area Size +
      Sum of Data Base PCB Sizes + Index PCB Size.

where:

PSB Prefix Size = 60 bytes.

The following formula is used for calculating the size of the work area:

Work Area Size = (A + B + C + D + E) or (F) or (G).

Note: Round each computed value up to a multiple of 8.

where:

A = The largest of the following values:

1. 256 if any segment has PROCOPT = D.

2. 112 + (2* longest index segment) if any data bases referenced in this PSB in turn reference only prime index data bases.

3. 224 + (2* longest index segment) if any data bases referenced in this PSB in turn reference any secondary index data bases.

4. The largest logical child segment in any referenced data base which contains the physical key option.

5. The largest logical child/logical parent concatenated segment data length as it would appear in the the application I/O area for any referenced data base.

B = The largest of the following values:

1. The largest index segment referenced by any data base referenced in this PSB (data length + prefix size).

2. The longest HISAM VSAM root segment (data length + prefix size + 6).

3. 8 if none of the above.

C = The maximum length (prefix plus data) of the largest variable length or compressible segment in any data base referenced in this PSB.

D = The maximum length data base I/O area required to process a call. This value would be the largest of the following:

1. The largest segment which could be retrieved.

2. The largest concatenated segment which could be retrieved.

3. The largest path of segments that could be retrieved.

Note: This value is specifiable at PSBGEN time. A maximum value is calculated if no specification is made.

E = 0 if the region type is DLI or DBB and no data bases are being loaded.

= 96 if the region type is DLI or DBB and data bases are being loaded.

= 280 * (the maximum number of levels in any DBPCB) if the region type is MSG or BMP.

Note: This value is specifiable at PSBGEN time. If a value is not specified, a default value is calculated.

F = The long message queue buffer size if the PSB is used online; otherwise, 0.

G = The scratch pad area (SPA) size if the application program associated with this PSB is a conversational program; otherwise, 0.

Note: If the region type is DLI or DBB the value used is the sum of the values A + B + C + D + E. If the region type is MSG or BMP the value is the largest of the sum of values A + B + C + D + E or F or G.

The ACB utility generates an output message DFS593I describing the calculated work area sizes. The letters shown in the work area formula correspond to that message as follows:

A = NDX work area.

B = XIO work area.

C = SEG work area.

D = IOA work area.

E = SSA work area.

The following formula is used for calculating the size of a DB PCB:

Single Data Base PCB Size = 208 + (A*68) + (B*72) + (C*72) + (D*40) + (E*40) + (F*80) + (G*16) + H + (I*72).

where:

A = 1 if application program is PL/1.

= 0 if application program is another language.

B = Number of SENSEG statements in a data base PCB. This value must be 0 for GSAM data bases.

C = 1 plus the sum of the logical child segment and all of its superior segments in the second data base (that is, the logical parent plus all of its parents) for each logical child segment referenced by this PCB. For GSAM data bases, this value must be 0.

D = Number of hierarchical segment levels defined in this PCB. This value must be 0 for GSAM data bases.

E = Number of data set groups referenced either explicitly through a SENSEG statement or implicitly through a logical relationship. This value must be 0 for GSAM data bases.

F = 1 if an alternate processing sequence was specified for this PCB.

= 0 if an alternate processing sequence was not specified for this PCB, or if using GSAM data bases.

G = Total number of index data base references via INDICES operands on all SENSEGs for this PCB. This value must be 0 for GSAM data bases.

H = Length of key feedback as defined in the PCB macro. This value must be 8 for GSAM data bases.

I = 1 if HIDAM data base.

= 0 if other than HIDAM data base.

The following formula is used for calculating the size of the index PCB:

Index PCB Size = A(580 + B + C).

where:

A = 1 if there are any index data bases referenced explicitly or implicitly in this PSB.

= 0 if there are no index data bases referenced explicitly or implicitly in this PSB.

B = Length of longest index key.

C = Twice the longest index segment.

The total space requirement for this PSB includes:

1. Prefix size
2. Work area size
3. Sum of all data base PCB sizes
4. Index PCB size

Note: If the PSB is to be used online, the size requirements are satisfied in two requests. The sum of values 1, 3, and 4 above are obtained when the PSB in obtained from the IMS/VS ACBLIB data set. As long as the PSB remains in the PSB pool, this storage is required. Value 2 above is obtained whenever the application program is scheduled into a dependent region and the area is released upon application program termination. This area is satisfied from the PSBW pool.

IMS/VS DMB (DATA MANAGEMENT BLOCK)

One DMB is generated for each data base description (DBD) associated with the PSB being serviced. The space requirement is:

SPACE = the sum of all DMB sizes.

The following formula is used for calculating the size of a DMB:

DMB Size = 24 + (A*8) + (B*88) + (C*36) + (D*16) + (E*16) + (F*12) + (G*240) +(H*168) + (I*96) + (J*76) + K + L + M.

where:

A = Total number of DDNAMEs specified on DATASET statements in DBDGEN.

B = Total number of DATASET statements in DBDGEN.

C = Total number of SEGM statements in DBDGEN.

D = Total number of LCHILD statements plus total number of logical child segment definitions in DBDGEN.

E = Total number of operands specified on XDFLD statements for keywords SEGMENT=, SRCH=, SUBSEQ=, SOURCE=, and EXTRTN in DBDGEN. If the data base is HIDAM, add 2 to the value obtained.

F = Total number of FIELD and XDFLD statements in DBDGEN.

G = Total number of DDNAMEs specified on DATASET statements that reference ISAM data sets.

H = Total number of DDNAMEs specified on DATASET statements that reference OSAM data sets.

I = Total number of DDNAMEs specified on DATASET statements that reference SAM data sets.

J = Total number of DDNAMEs specified on DATASET statements that reference VSAM data sets.

K = Total size of all index CSECTs contained in the DBDGEN output. Default size is 24 bytes each.

L = Total size of all compression routine CSECTs contained in the DBDGEN output. Default size is 32 bytes each.

M = Size of RMVTAB CSECT generated by DBDGEN. Default size is 32 bytes if access is HDAM, 0 if access is other than HDAM.

Note: The ACB utility DFSUACB0 generates an output message DFS940I which indicates the storage requirements for the named DMB. The DMB does not exist for GSAM data bases.

IMS/VS DATA BASE BUFFER POOLS

There are three pools associated with the DL/I data base buffering facilities; the ISAM/OSAM buffer pool, the DL/I buffer handler pool, and the VSAM buffer pools. For batch execution (DLI or DBB region types), the ISAM/OSAM buffer pool size defaults to 7000 bytes, but is controlled by a parameter of the EXEC statement for the step.

The DL/I buffer handler pool default size is 4K bytes. Its minimum size is the larger of 1) the size of the largest VSAM buffer defined, or 2) the sum of 44 bytes plus 68 bytes per VSAM subpool defined plus, if VSAM subpools are defined, 268 bytes for each PST and each sequential node VSAM data base PCB, plus 32 bytes per DL/I trace table entry. For an explanation of how IMS/VS builds VSAM buffer pools, see the section "Defining the IMS/VS VSAM Buffer Pool" in the IMS/VS Installation Guide.

IMS/VS uses the shared resources option of VSAM for all VSAM data bases. The main storage required for VSAM control blocks and buffers can be obtained from OS/VS Virtual Storage Access Method (VSAM) System Information.

IMS/VS DATA BASE WORK POOL

The DL/I action modules dynamically obtain working storage to allow processing of some DL/I calls. The size of the storage obtained varies with the type of call being processed, for example, REPLACE, INSERT; and the size of the largest data base control interval or blocksize. Typical storage sizes are between 2K and 4K.

IMS/VS and OS/VS MODULES -- DATA BASE ORGANIZATION DEPENDENT

The following IMS/VS storage requirements depend on the data base access methods and processing options used by the application program. Figure 5-1 is provided to determine the IMS/VS and OS/VS access method storage requirements. The sum of all randomizing routine, index exit routine, and compression routine sizes must be added to the size calculated from Figure 5-1.

The processing option values, abbreviated in the following figures, are:

G	=	Retrieval	I	=	Insert
GS	=	Retrieval Sequential	R	=	Replace
LS	=	Load Sequential	D	=	Delete
L	=	Load	A	=	All = G, I, R, D

For each item below (if duplication from item to item, use both figures), a set of conditions is listed. If all conditions are met, the value should be included in the estimate. The sum of all values thus selected provides the total loaded module requirements. Do not select a given entry more than once.

1.	BASIC code (Required).	23,000
2.	Any data base PCB except HISAM PROCOPT = L.	30,000
3.	Any PROCOPT = I or L or A.	17,000
4.	Any PROCOPT = D or R or A.	23,000
5.	Any primary or secondary indexes.	5,000
6.	Any VSAM data bases. The VSAM module requirements must be added to this value. This information can be obtained from the OS/VS Virtual Storage Access Method (VSAM) System Information.	18,000
7.	Any PROCOPT = L and logical relationships in the data base being loaded.	4,000
8.	Any data bases using ISAM.	3,000
9.	Any HS type data base using VSAM.	10,000
10.	Any HD type data base.	7,000
11.	If any OSAM data set is present.	13,000
12.	If simple HISAM data base and PROCOPT = D or R.	1,000
13.	If any HSAM and PROCOPT = L. This value can be obtained from OS/VS Storage Estimates*. The access method is QSAM* with PUT LOCATE MODE processing.	
14.	If any HSAM and PROCOPT = GS. This value can be obtained from OS/VS Storage Estimates*. The access method is QSAM* with GET LOCATE MODE processing.	
15.	If any HSAM and PROCOPT = G. This value can be obtained from OS/VS Storage Estimates*. The access method is BSAM* with READ MODE processing.	
16.	If any ISAM data set with PROCOPT = L. This value can be obtained from OS/VS Storage Estimates*. The access method is QISAM* with LOAD MODE processing.	

Figure 5-1 (Part 1 of 2). IMS/VS and OS/VS Modules Supporting Data Base Functions

17. If any ISAM data set using BISAM**. This value can be obtained from OS/VS Storage Estimates*. The access method is BISAM* with READ K and WRITE KN processing.

18. If any ISAM data set using QISAM**. This value can be obtained from OS/VS Storage Estimates*. The access method is QISAM* with SCAN MODE processing.

* QSAM, BSAM, QISAM, and BISAM storage estimates can be obtained from either of these publications, depending upon the system (VS1 or VS2) under which you are running: OS/VS1 Storage Estimates, OS/VS2 Storage Estimates.

** See the "Data Base Design Considerations" chapter in the IMS/VS System/Application Design Guide for a description of when BISAM or QISAM is used to access data bases.

Figure 5-1 (Part 2 of 2). IMS/VS and OS/VS Modules Supporting Data Base Functions

OS/VS CONTROL BLOCKS, BUFFERS, AND WORK SPACE

This section describes the space requirements for OS/VS control blocks, buffers, and work space.

OS/VS Buffers

OS/VS buffers are required when QISAM load mode, QISAM scan mode, QSAM get mode, or QSAM put mode is used. The requirements are usually two physical block buffers for each OS/VS data set used in the IMS/VS Data Base system environment. The default of 2 is overridden by providing a DCB=BUFNO=X parameter in the appropriate data set DD statement.

OS/VS Control Blocks and Work Space

The OS/VS control blocks and work space requirements, within the OS/VS region used for the DB system execution, depend considerably on whether OS/VS1 or OS/VS2 is used.

OS/VS1 Requirements: All space requirements are fulfilled within the OS/VS1 partition. The following formula provides approximate needs:

(2,500 + TIOT + DEBs + IOBs) = bytes.

where:

TIOT = (28 + 16n + 4d) bytes.
n = number of DD statements.
d = number of I/O devices.

DEB = 160 bytes each -- one required for each SAM, ISAM, and OSAM data set.

IOB = 136 bytes each -- two required for each ISAM, OSAM, and SAM data set.

OS/VS2 Requirements: Space requirements are partially fulfilled within the OS/VS2 region, and partially fulfilled from system queue space (SQS).

Space in Region = (5,200 + IOBs) = bytes.

Space in SQS = (2,000 + TIOT + DEB) = bytes.

where:

IOB, DEB, and TIOT space requirements are the same as specified for OS/VS1.

Note: IMS/VS requirements in OS/VS2-1 (SVS) are essentially the same as for IMS/VS in OS/VS1.

DATA BASE SYSTEM STORAGE REQUIREMENTS EXAMPLE

The following environment is assumed for the calculation of main storage in this example. A worksheet is provided as Figure 5-2, and follows the discussion of this example.

1. Application program is 20,000 bytes.

   Enter on line 12 of the worksheet.

2. Basic IMS/VS system modules require 25,000 bytes.

   Enter on line 1 of the worksheet.

3. The PSB control block contains three data base PCBs. One PCB is for HSAM, one is for HISAM, and the third is for HIDAM. The HSAM PCB has PROCOPT GS, the HISAM has GRD, and the HIDAM has A. The length of the index segment is 20 bytes. The largest segment accessed is 100 bytes.

   PSB = PSB Prefix Size + Work Area Size + Sum of Data Base PCB Sizes + Index PCB Size.

   PSB Prefix Size = 60 bytes.

   Work Area Size = A + B + C + D + E.

   where:

   A = 256 bytes.
   B = 32 bytes.
   C = 8.
   D = 104 bytes.
   E = 0.

   Work Area Size = 256 + 32 + 104 + 8 = 400 bytes.

   Index PCB Size = 580 + 40 + 20 = 640 bytes.

   PSB Size = 60 + 400 + 640 + sum of data base PCB sizes.

It is assumed that the language used is not PL/I. The number of SENSEG statements in the first, second, and third PCBs is 5, 7, and 15 respectively. The number of hierarchical segment levels in the three PCBs is 2, 4, and 6 respectively. No logical parents are referenced. The number of data set groups in each PCB is one. The length of the longest concatenated key in each PCB is 20, 45, and 70 bytes, respectively.

```
PCB =   208 + (5*72) + (2*40) + (1*40) + 20 = 708 bytes.

PCB =   208 + (7*72) + (4*40) + (1*40) + 45 = 957 bytes.

PCB =   208 + (15*72) + (6*40) + (1*40) + 70 + 72 = 1710 bytes.

PSB =   60 + 400 + 640 + 708 + 957 + 1710 = 4475 bytes.
```

Enter this figure on line 2 of the worksheet.

4. Three DMBs are required for the three data bases referenced. The number of SEGM and FIELD statements for each of the three DBDs is 5 SEGM and 10 FIELD, 7 SEGM and 12 FIELD, and 17 SEGM and 35 FIELD statements, respectively.

```
DMB =   24 + (2*8) + (1*88) + (5*36) + (0*16) + (0*16) + (10*12) +
        (0*240) + (0*168) + (1*96) + (0*76) + 0 + 0 + 0 = 524 bytes.

DMB =   24 + (2*8) + (1*88) + (7*36) + (0*16) + (0*16) + (12*12) +
        (1*240) + (1*168) + (0*96) + (0*76) + 0 + 0 + 0 = 932 bytes.

DMB =   24 + (1*8) + (1*88) +(17*36) + (0*16) + (0*16) + (35*12) +
        (0*240) + (1*168) + (0*96) + (0*76) + 0 + 0 + 0 = 1320 bytes.
```

Adding DMBs together (524 + 932 + 1320) results in 2776 bytes.

Enter this figure on line 3 of the worksheet.

5. Data base buffer pool of 10,000 bytes is chosen.

   Enter on line 4 of the worksheet.

6. Data base work pool of 4000 bytes is chosen. Enter on line 5 of the worksheet.

7. IMS/VS organization module requirement is chosen from Figure 5-1.

   Three data bases -- one HSAM with PROCOPT = GS, one HIDAM VSAM with PROCOPT = A, and one HISAM with PROCOPT = GRD.

   Using Figure 5-1, the values selected are:

```
 1.   23000
 2.   30000
 3.   17000
 4.   23000
 5.    5000
 6.   18000
 7.    3000
 8.   10000
 9.    7000
10.   13000
     ------
     149000 TOTAL (Enter this figure on line 6 of the worksheet)
```

<u>Note</u>: Items 5 and 9 are selected because the HIDAM data base requires a VSAM primary index. An INDEX data base is an HS type data base. It is assumed for the example all OS/VS data management modules reside in the Link Pack Area (LPA) and do not require storage from the user region.

8. OS/VS control block and buffer requirements assuming OS/VS2, QSAM buffering for HSAM, and QISAM buffer for HISAM are:

QISAM buffers = 1 data base x 2 buffers x 1500 bytes = 3000 bytes.

QSAM buffers = 2 buffers x 1500 bytes = 3000 bytes.

Control blocks= [5200 +(14*136)] bytes = 7104 bytes.

Enter the buffer total on line 9, and control block and work space total on line 10 of the worksheet. Total of 13104 appears to the right of line 10.

In summary, total lines 1 through 10 of the worksheet. The total size of the sample Data Base system is 208335 bytes (line 11). This assumes that the dynamic block loading option (PARM='DLI...) was selected. Add to this the size of the application program(s), 20000 bytes (line 12); giving a total Data Base option, including the application program, of 228335 bytes. This value must be rounded to nearest multiple of 2K bytes.

The total requirement is 208K bytes.

Reference Number	Description		Size
1	IMS Basic Modules.		25000
2	PSB Size.		4475
3	DMB Size.		2776
4	Data Base Buffer Pool Size.		10000
5	Data Base Work Pool.		4000
6	IMS/VS Data Base Organization Dependent Modules.		149000
7	OS/VS Data Base Organization Dependent Modules. (LPA)		
8	OS/VS Control Blocks, Buffers, & Work Space.		
9	OS/VS Buffers	6000	
10	OS/VS Control Blocks and Work Space	7104	13104
11	Data Base System = Subtotal		208335
12	Application Program(s).	+	20000
13	Data Base System and Application Program(s). (Round to Nearest Multiple of 2K.)		228335 228K

Figure 5-2. Example Worksheet for Data Base System

DATA BASE SYSTEM MINIMUM STORAGE REQUIREMENTS EXAMPLE

The following environment is assumed for a minimum storage requirement example:

- One data base -- HISAM organization, single data group
- The data base has five segment-types, two fields each, eight-character key field, and five hierarchical levels
- No logical relationships
- No index data base
- COBOL application program for retrieving and inserting data

A worksheet, Figure 5-3, follows the discussion of this example.

1. Basic IMS/VS system modules require 25000 bytes.

   Enter on line 1 of worksheet.

2. The PSB control block contains one data base PCB. This PCB is for HISAM. The processing option for HISAM data base PCB is GI. The largest segment is 56 bytes.

   PSB = PSB Prefix Size + Work Area Size + PCB Size.

   where:

   PSB Prefix Size = 60 bytes.

   Work Area Size =A+B+C+D+E = 56 bytes.

   A=0, B=0, C=0, D=56, E=0.

   PCB Size = 208+A+B+C+D+E+F+G+H = 848 bytes.

   A=0, B=360, C=0, D=200, E=40, F=0, G=0, H=40.

   PSB Size = 60+56+848 = 964 bytes.

   Enter this figure on line 2 of the worksheet.

3. One DMB is required for the data base referenced. Assuming:

   DMB = 24+(A*8)+(B*88)+(C*36)+(F*12)+(G*240)+(H*168)
   = 24+16+88+180+120+240+168 = 836 bytes.

   Enter this figure on line 3 of the worksheet.

4. Data base buffer pool of 2000 bytes is chosen.

   Enter this figure on line 4 of the worksheet.

5. Data base work pool of 2000 bytes is chosen.

   Enter this figure on line 5 of the worksheet.

6. IMS/VS organization module requirement is chosen from Figure 5-1.

   Enter 86000 on line 6 of the worksheet.

7. OS/VS modules required are in the OS/VS LPA.

8. Assume an OS/VS buffer of 1500 bytes.

   Enter this figure on line 9 of the worksheet.

9. OS/VS control blocks are:

   OS/VS1

   OS/VS1 blocks = (2500+(28+(16*4)+(4*8)+(160*2)+(136*4).
   OS/VS1 blocks = 3500 bytes.

   OS/VS2

   OS/VS2 blocks = (5200+(136*4).
   OS/VS2 blocks = 5800 bytes.

   Enter each of the above figures on line 10 of the worksheet.

In summary, add lines 1 through 10 of the worksheet for each (OS/VS1 and OS/VS2) with a total of the DB system storage requirements entered on line 11 to be:

121800 bytes for OS/VS1

124100 bytes for OS/VS2

Note that minimum size for the application program must be at least 9K.

Therefore, using a minimum operating system OS/VS1 or OS/VS2, a batch-only IMS/VS DB system execution can operate on a 256K machine for OS/VS1 or a 348K machine for OS/VS2.

Reference Number	Description		Size	
1	IMS Basic Modules		25000	
2	PSB Size		964	
3	DMB Size		836	
4	Data Base Buffer Pool Size		2000	
5	Data Base Work Pool		2000	
6	IMS/VS Data Base Organization Dependent Modules		86000	
7	OS/VS Data Base Organization Dependent Modules (LPA)			
8	OS/VS Control Blocks, Buffers, and Work Space			
9	OS/VS Buffers	1500		
10	OS/VS Control Blocks and Work Space (OS/VS1) 3500, (VS2) 5800		OS/VS1 5000	VS2 7300
11	Data Base System = Subtotal		121800	124100
12	Application Program(s)	+		
13	Data Base System and Application Program(s). (Round to nearest multiple of 2K.)			

Figure 5-3. Example Worksheet for Minimum Data Base System

DATA BASE/DATA COMMUNICATION SYSTEM STORAGE REQUIREMENTS

The main storage requirements for the DB/DC system depend on the specifications set forth and the options selected in Stage 1 of IMS/VS system definition. In addition, the main storage requirements are affected by the values that appear in the parameter field of the JCL EXEC statements for the control, message processing, and batch-message processing job steps. The operating system programming system options and the contents of the resident areas also influence the main storage requirements.

It is assumed that the reader knows the operating system environment in which IMS/VS will be executed. The knowledge of this operating system environment must be applied by the reader to adjust estimates for loaded module occupancy, control blocks, and other factors. Where calculations of storage requirements involve operating system modules, work areas, or control blocks, no specific size values are provided. To obtain specific values, refer to OS/VS1 Storage Estimates, OS/VS2 Storage Estimates, or the appropriate OS/VS control block documentation.

A sample storage estimate is provided at the end of this section for an IMS/VS configuration intended to operate under VS2. Associated with the example are assumptions that define the operating system environment and the IMS/VS specifications on which the sample calculations are based. A similar set of assumptions should be prepared before attempting to calculate storage requirements for any IMS/VS DB/DC configuration.

The instructions for estimating main storage requirements are presented in two parts. The first part concerns the IMS/VS control region (CTL). The second part covers the two dependent region types: MSG (message processing) and BMP (batch-message processing).

The "Organization of Control Program" appendix of this publication illustrates the organization of the various elements of the control program.

When the IMS/VS resident monitor is included in a DB/DC system, an additional 5K of storage must be included in the control region for the monitor modules. In addition, space for work areas and output buffers must be included. In a VS/2 environment 2K of space must be included in the control region and 3K of space must be included in CSA for the monitor modules.

The space requirements for work areas and output buffers may be calculated as follows:

Monitor work areas: Control region

1. 256*(A +10)

   A= maximum number of concurrent I/O

2. $\frac{\text{Size of System TIOT Table}}{2}$

Monitor log work area: (CSA for VS/2)

   488 + (216 + buffer size) * A

   A= number of output buffers (minimum of 2)

The inclusion of the Interactive Query Facility (IQF) into the IMS/VS system affects the main storage requirements of the IMS/VS control region. Refer to the "Interactive Query Facility (IQF) With IMS/VS" chapter of this publication for IQF storage estimates.

CONTROL REGION

An understanding of the physical layout of the control region, its use of the supervisor services, and the structure of the control program nucleus will assist you in preparing the estimate of main storage requirements. Figure 5-4 shows a representation of the physical organization of the control region. (See Figure A-1 of the "Organization of Control Program" appendix of this publication for additional definition.)

Control Region Organization in VS/1

Partition

System Queue Space	IMS/VS Control Program Nucleus	IMS/VS Modules	IMS/VS Pools	IMS/VS Blocks	IMS/VS Working Storage

Control Region Organization in VS/2

Region

Local System Queue Space	IMS/VS Control Program Nucleus	IMS/VS Modules	IMS/VS Pools

CSA

IMS/VS Blocks	IMS/VS Modules	IMS/VS Pools	IMS/VS Working Storage

Figure 5-4. Control Region Organization

The actual division of the area is not precisely disciplined. For example, OS/VS working storage can exist in several non-contiguous areas. This representation establishes a framework within which calculations are performed. The following page is a worksheet which can be used for accumulating the control program region storage estimate.

Note: IMS/VS requirements in OS/VS2-1 (SVS) are essentially the same as for IMS/VS in OS/VS1.

Worksheet for Control Region Estimates

1. Control Program Nucleus	
a. Resident Code	_____
b. Generated Control Blocks	_____
2. IMS/VS Locally Loaded Modules	_____
3. Global Areas (CSA for VS/2)	
a. Control Blocks	_____
b. Program Specification Blocks	_____
c. Data Base Description Blocks	_____
d. Data Base Buffers	_____
e. Data Base Work Pool	_____
f. General Buffers	_____
g. DBLLOG Buffers	_____
h. System Log Buffers	_____
i. IMS/VS Globally Loaded Modules	_____
j. PSB Work Pool	_____
4. Buffer Areas	
a. Queue Buffers	_____
b. Format Block Pool	_____
c. Line Control Buffers	_____
d. Communication Work Area Pool	_____
5. Dynamic Storage Requirements -- OS/VS	_____
6. Dynamic Storage Requirements -- IMS/VS	_____

Region/Partition Size

Total Items 1,2,4-6 (VS/2)

Total Items 1-6 (VS/1)

CONTROL PROGRAM NUCLEUS

The first area to be calculated is the control program nucleus. The nucleus contains the control program executable code and generated control blocks. Figure 5-5 represents the physical organization of the control program nucleus. (See Figure A-2 of the "Organization of Control Program" appendix of this publication for additional definition.)

CONTROL PROGRAM ROOT	IMS/VS AND OS/VS GENERATED CONTROL BLOCKS	CONTROL PROGRAM OVERLAY REGION 1	CONTROL PROGRAM OVERLAY REGION 2

Figure 5-5. Control Program Nucleus (V=R)

The control program nucleus is organized to minimize the working set if virtual execution is desired, or as a planned overlay structure if real execution is desired. Control blocks generated during Stage 2 of IMS/VS system definition and supplied load modules are united to form the nucleus. The selection of supplied load modules and the generation of control blocks performed by system definition are directly related to the input statements. Certain modules and control blocks are always made a part of the control program nucleus. These are called "required" or "basic". Others are either optionally selected or, if control blocks, may be generated in multiples that exceed the basic requirements. The number of control blocks generated is related to the number of times a particular macro statement appears in IMS/VS system definition Stage 1 input.

Control Program Code

Figure 5-6 below shows the size of the basic and optional control program code. Total the values which apply to your configuration, and enter the sum on the supplied worksheet.

REF	DESCRIPTION	SIZE
1.	Basic code.	66000
2.	Conversational option selected by use of SPAREA macro statement.	V=V = 10500 V=R = 5100
3.	Paging option selected by OPTIONS operand on COMM macro.	800
4.	Message format services support:	
	a. Basic MFS.	24600
	Note: Basic MFS is included if any 274X, 3270, 3767, or 3600 terminal is defined in the system.	
	b. 274X or 3600 MFS.	1000
	c. MFS test facility specified by OPTIONS operand on COMM macro.	1000
5.	Message format services master terminal support selected by OPTIONS operand on COMM macro.	2300
6.	Resident portion of terminal device support selected through use of LINEGRP, LINE, and TERMINAL macro statements or through use of TYPE and TERMINAL macro statements (see Figure 5-7).	
7.	Select option A if V=R or B if V=V execution is desired.	
	a. Area for overlay regions 1 and 2. (VS/1 only)	V=R = 9000
	b. Area reserved for same code as a. but without planned overlay.	V=V = 78000
TOTAL BASIC AND OPTIONAL CODE		

Figure 5-6. Control Program Nucleus -- Basic and Optional Code

BTAM SUPPORTED DEVICES

• Required basic code	V=V = 2700 V=R = 1300
• 1050 Non-switched	1400
• 1050 Switched	1200
• 2260/2265 Non-switched, remote	1600
• 2740 Model 1, Non-switched	600
• 2740 Model 1, Switched	700
• 2740 Model 1, Non-station control	500
• 2740 Model 2, Non-switched	1600
• 2780 Non-switched	2200
• 2741 Non-switched	600
• 2741 Switched	1800
• 33/35 Teletypewriter (ASR)	1100
• 2770 Common code*	6600
• 2770 With MDI (050) attachment*	2200
• 2770, SYS/3, SYS/7 BSC, 3270 Remote common routine	600
• 2980 Non-switched	5000
• SYS/3 - SYS/7 - SYS/7 BSC Common Code	V=V = 6300 V=R = 4800
• SYS/3 - SYS/7 BSC common code	1600
• SYS/3	700
• SYS/7**	V=V = 3500 V=R = 2500
• SYS/7 BSC***	1700
• 3270 Local	2300
• 3270 Remote	7800
• 3275 Switched	3800
• 3275 Switched - 3741 Switched common code	600
• 3741 Switched ****	4000
• Common switched terminal routine	V=V = 2200 V=R = 550

VTAM REQUIREMENTS

• Required basic code	V=V = 7300 V=R = 5900
• Common VTAM code	4400
• 3270	6250
• 3600	8220
• 3614*****	3475
• 3767	6100
• 3770	8200
• 3790	8220

Figure 5-7 (Part 1 of 2). Control Program Nucleus -- Required Resident Device Code -- Select one entry value for each terminal type used and add the selected values.

GAM REQUIREMENTS

• 2260 Local	1300

BSAM REQUIREMENTS

• Local Card Reader/SYSOUT	3600

IMS/VS REQUIREMENTS

• 7770 Switched ******	1600

* Add to common code requirements the specific environment main storage requirements. Examples:

2270 with MDI	=	2770 common plus 2770 with MDI.
SYS/3	=	SYS/3, SYS/7, SYS/7 BSC common plus SYS/3 - SYS/7 BSC common plus SYS/3.

** Plus size of user-supplied CAAUZERO and CAAUTIPL.

*** Plus size of user-supplied SUBIPL.

**** Plus size of user-supplied DFSS3741.

***** Plus size of VTAM module BQKCIPH and user-written DFS36140.

****** Plus size of user-supplied DFSS7770, DFSI7770, and DFSO7770.

Figure 5-7 (Part 2 of 2). Control Program Nucleus -- Required Resident Device Code -- Select one entry value for each terminal type used and add the selected values.

<u>Control Program Nucleus -- Generated Control Blocks</u>

The specifications defined in Stage 1 of IMS/VS system definition directly influence the generation of control blocks. Figure 5-8 contains the storage requirement estimates based upon those specifications. The values obtained from this figure should be within 2 percent of the actual storage requirement. For an exact description of the control blocks represented by each item in Figure 5-8, refer to Figure 5-21. (See Figure A-4 of the "Organization of Control Program" appendix of this publication for additional information.)

REF	DESCRIPTION	SIZE
1.	Basic fixed control blocks	500
	SYSTEM OPTIONS DESCRIPTION	
2.	Each potential concurrent input/output request as specified in the MAXIO keyword of the IMSCTRL macro statement. (Save sets)	1008
3.	Each potential concurrent conversation-sum of main storage and direct access as specified in the SPAREA macro statement. (CCB)	48
4.	Each transaction class. (TCT)	80
	DATA COMMUNICATIONS DESCRIPTION	
5.	Each line group as specified by a LINEGRP macro statement (DCB):	40
	• For 7770 LINEGRP, add	36
	• For local reader line group, add	52
	• For each direct SYSOUT line group, add	52
	• For each spool SYSOUT line group, where: n = number of data sets assigned	56+92*(n-1)
	• For VTAM node	0
6.	Each communication line or pool (excluding the system console) as specified by a LINE or a POOL macro statement. (CLB)	124
7.	Each terminal, or each 1050 terminal complex, or each dial line subpool as specified by a TERMINAL or SUBPOOL macro statement. (CTB)	96
8.	Each terminal type, or each model, or each line, or within any one type, model, or line where there are different (CTT): • Translation requirements • Input/output buffer sizes • Screen sizes • Segment lengths • User output edit routines	36
9.	Each terminal type for which the terminal transmission code is unique within the system, or the translation requirements are unique. For example, 1050 and 2740 each have a unique transmission code; or 2740 translated to uppercase and lowercase are unique translation requirements. (Translation tables)	512
10.	Each logical terminal name as specified by a NAME macro statement. (CNT)	52

Figure 5-8 (Part 1 of 2). Control Program Nucleus -- Control Blocks

REF	DESCRIPTION	SIZE
11.	Each 2770 terminal. (CXB)	20
12.	Each physical terminal supported by the Message Format Service. (CIB)	68
13.	Each SYS/3 or SYS/7 station and each 3601, 3614, 3767, 3770, or 3790 operator station (CRB).	32

Figure 5-8 (Part 2 of 2). Control Program Nucleus -- Control Blocks

IMS/VS AND OS/VS LOADED MODULES -- CONTROL REGION

Depending on the terminal device support requirements and the data base organizations chosen, different OS/VS access method modules are selected for loading into the control region. All IMS/VS and OS/VS loaded modules that contain executable code can be placed in the system link pack area. This may reduce the main storage requirements of an IMS/VS DB/DC system. The detailed tables in the "Storage Estimates Source Data" section of this chapter contain the IMS/VS names of the modules represented by the selection tables. In VS/2, modules in global storage are loaded in CSA.

REF	DESCRIPTION	SIZE	
		GLOBAL	LOCAL
1.	Modules always loaded by the CTL region	155200	7000
2.	Terminal support		*
3.	Add if DL/I VSAM Support	18000	
4.	Add if CONVERSATION option (unpack rtn)	256	
5.	Add if DC MONITOR option	3000	2000
	TOTAL -- Enter in table of working papers	______	______

* See the appropriate OS/VS storage estimates publication, OS/VS1 Storage Estimates, or OS/VS2 Storage Estimates for calculating item 2.

Figure 5-9. Control Region -- Loaded Modules

REF	DESCRIPTION	SIZE	
		GLOBAL	LOCAL
1.	DL/I data base change logging	2000	
2.	Modules used in common by message queue manager and DL/I	6200	
3.	DL/I basic modules	129000	
4.	Miscellaneous modules	18000	7000
	TOTAL	155200	7000

Figure 5-10. Modules Always Loaded by the CTL Region

Refer to Figure 5-22 to determine the module names that comprise the list of always-loaded functions shown in Figure 5-10.

GLOBAL AREAS

Specific control blocks, pools, and IMS/VS modules require space in global storage. In a VS/1 environment the space is in the IMS/VS partition. In a VS/2 environment the space is in CSA (Common Service Area). See OS/VS2 Storage Estimates for information on specifying CSA storage.

GLOBAL CONTROL BLOCKS

For a description of the control blocks represented by the items in Figure 5-11, refer to Figure 5-23.

REF	DESCRIPTION	SIZE
1.	Basic fixed control blocks.	18600
2.	Each potentially active message or batch-message processing region as specified in the MAXREGN keyword of the IMSCTRL macro statement. (PSTs)	4096
	APPLICATION PROGRAM DESCRIPTION	
3.	Each application program as specified by an APPLCTN macro statement. (PDIR)	44
4.	Each transaction code as specified by a TRANSACT macro statement. (SMB)	68
5.	Each data base as specified by a DATABASE macro statement. (DDIR)	40

Figure 5-11. Global Control Blocks

GLOBAL BUFFER AREAS

System Log Buffers

The following formula is used to calculate storage requirements for the system log work area:

488 + A *(216 + B)

where:

A = number of output buffers (minimum of 2).

B = buffer size. Default is larger of: 1024 checkpoint log work area or size of long message queue LRECL (min 576) plus 24 bytes overhead.

IMS/VS BUFFERS

During the execution of the control program, buffer space is required for communication terminal input/output operations, data base management control blocks, conversation work areas, program description blocks, message queue management, system recovery (checkpoint/ restart), and data base input/output operations, and for miscellaneous use in command processing, message generation, and application scheduling. The sizes of these areas are specified in the EXEC statement for the control program nucleus.

At the time execution begins, the main storage requirements are summarized and a single area of dynamic storage is acquired unconditionally. The area thus acquired is partitioned into storage pools from which almost all IMS/VS dynamic requests are satisfied by an IMS/VS storage management routine. Figure 5-12 relates the specification of buffer sizes in the EXEC statement to their use by the control program nucleus. The letters which appear in the left-hand column correspond to those that appear in the supplied procedure named "IMS". (Refer to "The IMS/VS Procedure Library" chapter in this manual for details about the IMS/VS procedures.) Refer to Figure A-8 in the "Organization of Control Program" appendix of this publication for the layout of IMS/VS buffers.

PARM POSITION IN PROCEDURE	NAME	DESCRIPTION OF USE
QBUF	Queue Buffer	Buffers used by message queue management.
FBP	Format Buffer	Buffers used for Message Format Service control blocks.
PSB	PSB Pool*	Program description blocks stored here.
PSBW	PSB Work Pool*	Buffers used for Inter-Region Communications.
DMB	DMB Pool*	Data base description and data base management control blocks.
DBB	Data Base Buffer*	Data base input/output operation buffer.
DBWP	Data Base Work Pool*	Temporary storage required to process DL/I calls.
TPDP	Line Buffer	Communications line input/output operations buffer.
DYBN	DBLLOG Buffers*	DB LOG buffers for dynamic backout.
WKAP	General Buffer*	Miscellaneous requirements for command processing, application scheduling, working storage, conversation, system recovery.
MFS	MFSTEST	Maximum space available from the line buffer pool for use by the MFSTEST facility.
CWAP	Communication Work Pool	Temporary Storage Area for disk SPAs. Storage for incore SPAs while a conversation is active. Miscellaneous conversation work areas (pack, unpack commands: /EXIT, /REL, /STA, /HOLD).

* In VS/2 these buffers are in CSA storage (global).

Figure 5-12. Buffer Specifications in IMS Procedure

Sizes of the buffer pool areas are directly related to performance. There is a minimum size for each area. Below this minimum, full function is no longer available. The discussion that follows describes the calculation of the minimum pool size for function. It considers the performance enhancement effects of increasing pool sizes beyond the minimum values.

Maximum Dynamic Storage to be Used by IMS/VS ENQ/DEQ

The IMS/VS enqueue/dequeue routines are used to synchronize the operation of the data base buffer pool. The IMS/VS enqueue/dequeue routines are also used to control potential update requests ("HOLD" in data base retrieval calls). Another use of these routines is to isolate changed data base segments from possible retrieval by other programs during the period in which the program making the change could be backed out due to deadlock or application program failure.

Data base buffer pool management requires a maximum of three enqueues (held only during a single request) per message or batch-message processing region.

The DL/I action modules use the IMS/VS enqueue/dequeue routine to control data base changes. All segments retrieved in HOLD status are enqueued until the segments have been updated, or until another data base request releases them. In addition, any segment that has been updated is enqueued until the program that requested the update terminates, or, if the program is processing a transaction that has single processing mode, requests the next transaction.

The IMS/VS enqueue/dequeue routines obtain and release storage dynamically, as enqueue and dequeue requests are processed. If the amount of storage actually obtained reaches the specified maximum, no further enqueue requests are honored until sufficient storage is released by dequeue requests. Since the storage is obtained via GETMAIN requests, sufficient space within the control region must be reserved for this function. The following formula is used for calculating the amount of storage required for IMS/VS enqueue/dequeue routines:

Size of Storage for ENQ/DEQ Routines = I * N

where:

I = The value of the third subparameter (increment) of the CORE parameter of the IMSCTF system definition macro.

N = (32A * (B + C + D + E + 3F + G + 3H)) / I

where:

A = The number of concurrently scheduled regions.

B = The number of data base root segments that can be accessed to satisfy a given retrieval call. (Note: Count only the roots that could be accessed if the call were satisfied without having to search multiple data base records).

C = The maximum number of data base segments that can be retrieved in HOLD status in a single call.

D = The maximum number of segments that an application program can request to be reserved by the enqueue command code before it issues a corresponding dequeue DL/I call, or reaches a synchronization point.

E = The maximum number of data base segments that an application program can alter before it reaches a synchronization point.

F = The maximum number of data base segments that an application program can insert before it reaches a synchronization point.

G = The maximum number of data base segments that can be marked deleted by only their logical path, or only their physical path, due to an application program's delete call prior to the application program reaching a synchronization point.

H = The number of delete requests that can be made by an application program prior to the application program reaching a synchronization point.

The values for B, C, D, E, F, G, and H, above, can be estimated by use of a matrix that shows intent by data bases, similar to that shown in Figure 3-1 in the IMS/VS System/Application Design Guide, in conjunction with the data base descriptions that define the specified data bases.

Also, any data base segment types that are processed with an intent of Exclusive can be deducted from the above values.

Program and Data Base Description Buffers

Before an application program is scheduled into a MSG or a BMP region, the ACBs (application control blocks) required for this program must be loaded. The ACBs are further broken down into two groups: the PSB (program specification block) and the DMB (data management block). The PSB is subdivided into sections called PCBs (program communication blocks). There are two kinds of PCBs, a TP PCB (teleprocessing PCB) and a DB PCB (data base PCB). The TP PCB contains the identity for output message destinations. The DB PCB describes the application program's view of the data bases described by the DMB.

The PSBs and DMBs are loaded and managed in separate pools called PSB buffer pools and DMB buffer pools. The PSB buffer pool calculation is discussed first, followed by the DMB buffer pool space calculation.

PSB Buffer Pool: One PSB is required for each concurrently active application processing program. The functional minimum size of the PSB buffer pool is the size of largest PSB which must occupy that pool.

Calculating the minimum size of the PSB pool is tedious, but not complex. Determining an optimum size for the PSB buffer pool involves consideration not only of the sizes of all PSBs used by the system, but also the conflicts of intent toward particular segments in the data bases referenced by those PSBs. For example, although PSB^1 may be the largest PSB and PSB^2 the second largest, it may be unnecessary to reserve PSB pool space equal to the sum of PSB^1 and PSB^2 for concurrent execution because conflict of intent prohibits concurrent execution. If both were quite large, say 8K each, and PSB^3 (the next largest) were only 2K, then perhaps 10K is a reasonable value. However, if in addition PSB^1 and PSB^2 were low usage, and only the function were required, then 8K might be adequate. Since PSB^3 is third largest, at least a total of four PSBs could be resident for performance most of the time. If only PSB^3 were 2K and all others 1K or less, then at least seven PSBs could be resident most of the time.

The basic requirement is function. Having met the minimum functional main storage requirement, performance tradeoffs can be made at will. In general, the larger the PSB buffer pool, the better performance will be. Of course, a buffer pool size larger than the storage required for all PSBs to be concurrently resident provides no additional performance advantage.

PSB POOL CONSIDERATIONS IN AN OS/VS SYSTEM

When executing in an OS/VS system, the PSBs should be looked on as two separate control blocks. The first block is the PSB prefix and PCBs. The second block is a work area, made up of the current index maintenance area and segment work area, plus the additions which are the control region copy of the application program's call list, SSAs, and I/O area. The two areas are obtained separately at application program schedule time. The first part, the prefix and PCBs, is retained in the PSB pool. The work area is obtained from the PSBW pool when the application program is scheduled and is released when the application program terminates.

To determine the size of the PSB area in a VS system, use the following formula:

PSB Area = A+B+C+D

where:

A = PSB prefix.

B = Size of the TP PCBs.

C = Size of the DB PCBs.

D = Index PCB Size.

Items A, C, and D are calculated using the formula supplied in the preceding section under IMS/VS PSB (Program Specification Block).

The following formula is used to estimate the size of item B (Teleprocessing PCBs):

C = N(48J + 64)

where:

N = The number of TP PCBs in the PCB.

J = 0 if the application is not PL/I.
  = 1 if the application is PL/I.

To determine the size of the PSB work area, use the formula supplied in the preceding section under IMS/VS PSB (Program Specification Block).

<u>DMB Buffer Pool</u>: Each DB PCB in the PSB names a DBD. When resident in the control program region, the DBD is called a DMB (Data Management Block). When an application is active, all DMBs referenced explicitly by PCB statements in the PSB must be resident. In addition, all DMBs referenced implicitly must also be resident. This includes logically related DMBs and INDEX DMBs.

The functional minimum size for the DMB buffer pool is that required to store the largest complex of DMBs explicitly or implicitly used by a single application program. The size of any given DMB can be estimated using the formula supplied in the preceding section under IMS/VS DBD (Data Base Description).

As the demand for buffer space in the DMB pool exceeds available unallocated space, the data sets which comprise the least-used DMBs are closed, and the space occupied by the DMB is freed. DMBs are freed one at a time, until there is sufficient space available to satisfy the demand for a new DMB.

Each time a DMB is added to the buffer pool, the operating system data sets must be opened. Only those data sets that represent a data set group to which the application has data sensitivity are opened. Before releasing the space occupied by the DMB, those data sets are closed. The time involved to perform OPEN and CLOSE is substantial. Frequent exchange of DMBs causes a dramatic decrease in response time and overall performance. Message traffic must be carefully analyzed by DMB usage to determine optimum buffer size. It is recommended that the application design personnel at your installation consider the potential performance impact and storage requirements generated by the proliferation of data bases and the logical relationships among data bases. The system degradation caused by continual rotation of the DMB pool is significantly greater than that caused by rotating the PSB pool.

Data Base Buffer Pool

The input/output areas required for use of all data bases in the DB/DC system are acquired from the data base buffer pool. No part of the buffer pool is owned exclusively by a data base or an application program. As buffers are used for data base input/output operations, they are retained as long as possible. When the demand for new buffer space exceeds available unallocated space, the oldest active areas are freed to meet that demand. When sufficient space is freed, it may be necessary to consolidate it into a contiguous area. If this happens, only those buffers surrounded by the fragmented free space are relocated to permit consolidation. Use of a buffer, whether for real or logical input/output, causes it to become the most recently active. A single data base, used by several applications at the same time, can have several active buffers. Conversely, a single application can have several active buffers from several data bases. Note that "active", as used in this discussion, does not mean allocated or reserved; it means only that the data in the buffer area is current. All buffers could become inactive if the demand for a new buffer were sufficiently large. The demand for allocation of buffer space is directly related to how recently the data occupying the buffers was used. It is constrained by the total size of the buffer pool to be managed, as well as by the distribution of buffer sizes demanded.

The minimum functional size of the data base buffer pool is represented by the following formula:

Minimum Size of Data Base Buffer Pool = A+B+C

where:

A = 2 times the largest block size, excluding HSAM, plus 300.

B = Sum of each HSAM data set block size, plus 18 for each HSAM data set. The sum represents the maximum number of HSAM data sets that will be concurrently open.

C = 0 if HISAM with no logical relationships and no alternate index.

= T for all organizations with logical relationships or alternate indexing, the largest sum of the values calculated for every possible deletion path in all data bases. Each time a delete path enters a data base (including the first time, and every recurrence, into the data base in which delete processing began) develop a value using the following formula.

Delete Data Base Transit Formula:

T = 54+D+E

where:

D = 16 times the number of hierarchic levels in the data base entered.

E = Length of the maximum concatenated key of the data base entered.

For storage requirements for the DL/I VSAM buffer pool, see "IMS/VS Data Base Buffer Pools" in the "Data Base System Storage Requirements" section of this chapter.

Statistics on the operation of the data base buffer pool are available. They may be obtained through use of the /DISPLAY POOL command. A description of the /DISPLAY command appears in the IMS/VS Operator's Reference Manual. The information you receive from /DISPLAY provides a way to optimize the use of the data base buffer pool.

IMS/VS Data Base Work Pool

The DL/I action modules dynamically obtain working storage from the pool to allow processing of some DL/I calls. The size of the storage obtained varies with the type of call being processed, for example, REPLACE, INSERT; and the size of the largest data base control interval or blocksize. Typical storage sizes are between 2K and 4K. The total pool space should provide a minimum of 2K per potentially active message processing region or batch message processing region.

General Buffer Pool

The general buffer pool is used by checkpoint/restart and application scheduling. The minimum functional requirements for this general buffer pool are represented by the following formula:

Size of General Buffer Pool = A+B+C+D*(MAX(E,F,80)+28)

where:

The size must be greater than or equal to 5120 bytes.

A = 1024 bytes or the size of a long message buffer, whichever is larger, used by checkpoint/restart.

B = 124 bytes used by application scheduling.

C = 2048 bytes for miscellaneous system use.

D = 1 if system contains 2770 terminal with any of the following components: 2265, paper tape reader, or an 050 MDI; otherwise, D=0.

E = Largest value specified in the PTSEG= operand of a 2770 terminal statement.

F = Largest value specified in the MDISEG= operand of a 2770 terminal statement.

The size of this pool is particularly critical when a varying number of main storage conversations can be in process. Because transient requirements for application scheduling are met from the general pool, a marginal amount of storage could reduce throughput, or interlock the system for varying periods of time.

DBLLOG Buffers

The Data Base log buffers are used in writing and reading the disk data base log data set, IMSVS.DBLLOG. The minimum buffer size is 1K, and is increased in increments of 1K to a maximum of 32K. This data set is used for dynamic backout. The space allocated to the buffers affects system performance. Since it is a sequential data set, the more buffer space, the better the performance.

Queue Buffer Pool

Queue buffers are owned by the message queue manager. They are used for writing and reading all messages by communications terminal management, and by data base management when retrieving or inserting messages in behalf of an application program. In addition, they are used as an expansion to the QCB ENQ and QCB DEQ pointers in logical terminal blocks (CNTs) to provide additional queues for message output.

The storage requirement, then, is a function of the number of buffers plus a fixed amount of overhead. The default size for a queue buffer is 576 bytes. The following formula is used for calculating the size of the queue buffer pool:

Queue Buffer Pool Size = A*(B+40)+160.

where:

A = number of queue buffers.

B = size of queue buffers.

The 576-byte default queue buffer size value allows ten records per track on a 2314. Fixed overhead per buffer consists of 40 bytes for buffer management.

The number of queue buffers assigned by system definition is 4 plus 1 for every ten transaction codes or logical terminals. Both the size and the number of queue buffers can be varied at system definition. The minimum number of buffers that must be assigned is three. The minimum size that must be specified is limited by terminal line length plus overhead (192 bytes). A 576-byte buffer can hold twelve queue block records, three short message records, or one long message record.

A queue block record number is permanently assigned to a logical terminal when the first message is received that indicates that destination. From then on, all references to the destination may refer to that queue block record. Depending on the size of a message segment, or the average size of complete messages to a given destination, whichever is larger, either a short or a long message record is assigned to a given write request.

All buffers in the queue buffer pool are managed with a single "latest referenced" chain. Since the buffers are all the same size, no buffer need be moved. However, if a given buffer is at the bottom of the chain, and a block is requested that is not currently in the pool, the low block is written to disk, if necessary, and the requested block is read into its buffer space.

The only problem involved in having the minimum size queue buffer pool is one of disk contention. In small systems with low traffic volumes, the minimum size queue pool can be used; however, the average user should allocate at least eight buffers. If the number of buffers available exceeds the amount of message traffic, no access to the queue data sets is required. Thus, if there are more available queue buffers, there is potential for greater throughput.

For additional information on message queue management, see "Operation of Queues," in the "Design and Control of the Data Base/Data Communications System" chapter of the IMS/VS System/Application Design Guide.

Message Format Buffer Pool

The following factors should be considered when defining the format buffer pool size and the number of fetch request elements:

- Average size of format blocks.
- Total number of unique format blocks.
- Direct access device type.
- Number of 3270 terminals which will be using MFS concurrently.
- Response time required at terminals.
- Largest format block combination which must be in main storage at one time.
- One fetch request element is required for each active request and for each block that is in main storage. If all format blocks are 1000 bytes long and you have specified 10 fetch request elements, the maximum pool space used for the format blocks is 10,000 bytes. All requests for block space can require up to 8 additional bytes of space over the size of the block itself.

Format Block Pool Storage Estimates: The storage estimate for the format block pool is the sum of the fixed area, variable area, and format block space, calculated as follows:

FIXED AREA

OS/VS DCBs	352 bytes
Statistic Counters	80 bytes
Pool Control Blocks	128 bytes
Directory I/O Area	512 bytes
Total Fixed Area =	1072 bytes

VARIABLE AREA

Fetch Request Elements (FRE)
= 40 bytes per FRE

Directory Index

$$= \frac{\text{Total number of blocks}}{11} \times 12.$$

Resident Directory (optional) (see "Message/Format Service Utility" in the *IMS/VS Message Format Service User's Guide*)

= number of selected block names x 14

FORMAT BLOCK SPACE

Select largest of:

a. 14336.

b. (number of 3270 lines) x largest of: (2030 or average block size obtained by using the Format Block Pool formula calculations shown later in this chapter).

Therefore, the total Format Block Pool Storage Estimate =

Fixed Area + FRE + Directory Index + Resident Directory ± Format Block Space.

Format Block Pool

The following four formulas make reference to a FMT Set. The FMT Set is defined as the FMT descriptor and all MSG descriptors whose source (SOR=) format is the FMT descriptor.

These formulas do not consider literals that can be mapped to another literal, thus potentially reducing the actual size of the block as stored in the online format library. For example, the three literals: "ABC", "AB", and "BC" can be mapped (compacted) to a single string of "ABC", making the block literal section have three bytes rather than the seven bytes predicted in the following.

The following formula is used to calculate the 3270 DIF block size. This computation is to be performed by DEV statement level within the FMT descriptor:

$$Size = 20+A+B+C+2C^{1}+DE+6F+6G+6H+M+N+6P+2Q+T+V$$

where:

A= 10 if DEV statement specified PFK, PEN or CARD; otherwise, A=0

B= 8 if DEV statement has PEN=fieldname; otherwise, B=0

C= 12 if DEV statement has PFK= operand else C=0

C^{1}= number of PFK entries specified

D= length of longest PFK= literal

E= number of PFK= literals specified

F= number of DPAGE statements for DEV; minimum=1

G= total number of physical pages defined for device

H= total number of fieldnames specified in all DPAGE CURSOR= operands for the device

I= index to current physical page for following values

J_I = number of names DFLD statements for physical page

K= total number of unique named fields for FMT Set

L_I = 6 if PASSWORD DFLD present; otherwise, L_I =2

$$M= \sum_{i=1}^{i=G} (6J_I +2(K-J_I +1)+L_I)$$

N= combined length of all PEN= literal lengths +2 for DFLDS

P= number of DFLD statements with hi-intensity literal

Q= number of OPCTL= operands; otherwise, Q=0

R= total number of IF statements per TABLE; otherwise, R=0

S= number of IF statements with branch labels per TABLE; otherwise, S=0

U= index to current table

$$T= \sum_{u=0}^{u=Q} (7R+2S)$$

V= combined lengths of all literals for all TABLE(S); otherwise, V=0

The following formula is used to calculate the 274X, SC1, SC2, and 3600 DIF block size. This computation is to be performed by DIV statement level for 274X, SC1, SC2, and DEV statement level for 3600 within the FMT descriptor:

Size = 24+4A+8B+6C+E+F+6G+6H+2Q+T+V

where:

A= total number of DPAGE statements defined; otherwise, A=1

B= number of conditional DPAGE statements; otherwise, B=0

C= number of named DFLD statements for DIV if 274X, for DEV if 3600

D= total number of unique named fields for FMT set

E= 2 (D-C+1)

F= number of defined FTAB characters +1; otherwise, F=4 for 274X or F=1 for 3600

G= total number of skipped lines between field definition if DEV statement has MODE=RECORD defined and FORCE option is not defined; otherwise, G=0

H= number of undefined column areas (in RECORD mode) or position areas (in STREAM mode) of 1 byte or more between fields and FORCE option is not defined; otherwise, H=0

Q= number of OPCTL= operands per device; otherwise, Q=0

R= total number of IF statements per TABLE; otherwise, R=0

S= number of IF statements with branch labels per TABLE; otherwise, S=0

$$T = \sum_{u=0}^{u=Q} (7R+2S)$$

U= index to current TABLE

V= combined length of all literals for all TABLES; otherwise, V=0

The following formula is used to calculate the 3270 DOF block size. This computation is to be performed by DEV statement level within the FMT descriptor:

$$Size = 16+16A+2AB+9C+24C^{1}+15(D-1)+17E+F+5G+6G^{1}+6H$$

where:

A= number of DPAGE Statements or minimum value of 1

B= number of unique fieldnames defined in FMT SET

C= number of physical pages if SYSMSG= defined

C^{1}= number of DFLD fieldname statements for which no output message uses dynamic attribute modification for <u>first</u> physical page

D= number of DFLD statements with fieldnames for which an output message uses dynamic attribute modification $-C^{1}$

E= number of DFLD statements for literals

F= combined total of all literal lengths from E

G= number of separate undefined areas of 1 byte or more for all physical pages

G^{1}= quotient of division: $\frac{G}{51}$
Add 1 if remainder $\neq 0$

H= number of occurrences of unique fieldnames which are defined on more than one physical page within a DPAGE

The following formula is used to calculate the DOF block size for all other device types:

Size = 16+16A+2AB+12C+6D+8E+F+(8G+H)+14I+6J+K+8L+4M+N+8P+T

where:

A and B same as for 3270

C= number of fieldnames with ATTR=YES

D= number of fieldnames with no ATTR=YES

E and F same as for 3270

G= total number of separate unused areas of 2 lines or more. G=0 if FLOAT option specified. G=1 if vertical tab stop replaces NL characters for SC1. Unused area at end of physical page should only be added if SPACE option specified.

H= (3270P and 274X) number of lines of the largest unused area

H= (3600 devices) the value here is 4

I= (3270P only) number of internal pages required. To approximate: for an external page of 55 lines with 80 columns of data, 3 internal pages are required for Model 2 and 11 for Model 1.

I= (all other devices) number of DPAGE statements specified

J= number of DFLD fieldnames which span device physical lines

K= (36JP, 36PB, 36FP) if FORMS = literal specified, K=28 + length of literal

L= (36JP, 36PB, 36FB) number of EJECTs to perform; if no EJECT L=1

M= (36FB only) number of entries in SELECT=

N= (36DS only) number of entries in ORIGIN

P= (36DS only) number of cursor entries

T= (SCS1 only if HTAB OFFLINE or ONLINE)

50+6I+I(2QRS)

where: I= number of DPAGE statements
Q= number of horizontal tab stops
R= number of defined DFLD lines
S= number of physical pages per DPAGE

The following formula is used to calculate the MID block size:

Size = 18+2A+10B+6C+6D+E+2F+2G+16H+2I+2J

where:

- A= number of unique fieldnames in FMT Set
- B= number of SEG statements or minimum value of 1
- C= number of MFLD statements for fieldnames
- D= number of MFLD statements for literals
- E= combined total length of all literals from MFLD statements
- F= number of unique fieldname occurrences in more than 1 MFLD statement
- G= number of MFLD statements using the LTH=(pp,nn) option
- H= number of LPAGE statements defined; otherwise, H=1
- I= number of default literal MFLD; otherwise, I=0
- J= number of MFLD statements with EXIT= parameter defined; otherwise, J=0

The following formula is used to calculate the MOD block size:

Size = 16+28A+6B+2C+D+F+2H

where:

- A= number of LPAGE statements or minimum value of 1
- B= number of MFLD statements
- C= number of MFLD statements with literals
- D= combined total length of all literals from the MFLD statements with literals
- E= number of unique fieldnames for the FMT Set
- F= combined total length of all literals from the COND= operand of all LPAGE statements
- $G_{(i)}$= number of unique fieldnames in $LPAGE_{(i)}$
- H= $\sum_{i=1}^{i=A} (E-G_{(i)})$

Line Buffer Pool

Terminal input/output operations are performed from the storage assigned to the line buffer pool. The amount of storage required varies by terminal device type, terminal device model, and kind of output operation being performed. Minimum function for communications message handling can be supported by assignment of a value that is the largest of the three kinds of requirements. For performance, the line buffer pool should be large enough for one input or output buffer for each unbuffered line, and each buffered terminal, under all traffic conditions. A value that represents the peak concurrent demand for buffers may be excessive. A smaller value, although it results in less frequent line service, may be more appropriate. It is recommended that the value assigned is not less than the average or modal demand for buffers.

IMS/VS systems that include 3270 and message format service support have a more dynamic and application-dependent requirement for communications buffer pool space. The best method for determining a reasonable value for the pool size is by use of the /DISPLAY POOL command during actual execution. If the space currently in use is consistently only slightly less than the pool size, performance can normally be improved by increasing the pool size. The following factors influence communications buffer pool space requirements when 3270 and message format service are included in the system:

- For output

  Select the largest of:

  a. 4096

  b. Sum of:

     1) 3270 Local lines (largest of):

        a) maximum number of input characters x number of lines.

        b) 1250 x number of lines.

     2) 3270 Remote lines

        1250 x number of lines.

- For input

  a. The size of the largest field in the device input format.

  b. For option 3* input messages, input requires:

     18 x NS + 4 x NF + SF + 4 bytes.

     where:

     NS = number of segments in the message.

     NF = number of fields in the message.

     SF = the sum of the defined lengths of all fields in the message.

c. For options 1* and 2* input messages that do not have device input data mapped to message segments, such that segments are completed before data for succeeding segments is located, the maximum requirement is:

16 x NS + SF + 4 bytes.

d. For 3270 local line buffers, each started 3270 local line requires the amount of space specified in the BUFSIZE parameter during IMS/VS system definition. However, if this space is insufficient for an input message, this value is increased by 300 bytes. The value can be incremented more than once. When this happens, it is indicated by the printing of message DFS254 at the master terminal.

If the system includes the MFSTEST facility, line buffer pool space is used for format control blocks when terminals are in MFSTEST mode. A limit to the amount of space that will be used for this purpose is specified at system definition. It can be changed by the control region EXEC statement. The maximum value that can be used is the line buffer pool size -- 5000. Assuming that the MFSTEST mode is normally used for one or more terminals on a single line at one time, the value should be greater than the size of the largest MOD-DOF block combination that is to be used. Format control block sizes can be estimated by using the formulas shown later in this chapter. The Message Format Service utility lists the sizes of control blocks that have been created and placed in the format library. A recommended value for maximum space is 50% of the line buffer pool size. The higher the percentage specified, the greater the chance of performance degradation when terminals are operating in MFSTEST mode.

The MFS position of the parameter in the EXEC statement specifies the maximum space limit for MFSTEST usage in 1K increments.

* For a discussion of input message format options, see the _IMS/VS Message Format Service User's Guide_.

Communication Work Area Pool (CWAP)

The communication work area pool is used by command and conversation processing. Use the following formula to estimate its size:

CWAP Size = A+B+C+D

where:

A = A maximum of 2048 bytes used by command processing.

B = The largest of the following three values used for conversation processing. Zero, if conversational processing is not part of your system.

- Work area of 80 bytes
- Maximum direct access SPA + 56 bytes
- Maximum core SPA + 56 bytes

C = The number of in-core CCBs*the maximum core SPA size.

D = Temporary workspace =

2*(the number of in-core CCBs*the maximum core SPA size)+
3*(the number of disk CCBs*the maximum disk SPA size)

The number of concurrently processing conversations may be limited if this pool is too small.

Figure 5-13 shows the input buffer size requirements by device type and model, and the output buffer size requirements by type of operation. Short messages include both single segment output from application programs and responses to commands. For short messages, only the actual output buffer size needed is acquired. For long output messages, the values in the tables apply.

TERMINAL TYPE	BUFFER SIZE IN BYTES INPUT	OUTPUT
• 1050	148	204
• 2740 Model 1	148	204
• 2740 Model 2 Buffer 120 See TERMINAL	136	136
• 2740 Model 2 Buffer 248 macro statement	264	264
• 2740 Model 2 Buffer 440	456	456
• 2260/2265 Model 1 (2848 Model 3)	976	976
• 2260 Model 2 (2848 Model 1)	254	254
• 2260 Model 2 (2848 Model 2)	494	494
• 2770 line with basic 2772s	148	148
• 2770 line with buffer expansion 2772	276	276
• 2770 line with additional buffer expansion 2772	532	532
• 2780	416	416
• 2980 Non-switched Multipoint	100	100
• 2980 Non-switched Multipoint with RPQ4835503	200	200
• 3270 Local Display	See Note 1	7+data
• 3270 Local Printer	6	7+data
• 3270 Remote	392	92+data
• 3270 Switched	382	382+data
• 3270 VTAM	---	138+data
• 3600	---	158+Note 2
• 3614	---	156+Notes 2,3
• 3740	514	514
• 3767	---	392
• 3770	---	392
• 3790	---	158+Note 2
• VTAM Receive any buffers	144+Note 4	
• 7770 (User Specified) Max =(256,256)	50	50
• System Console	148	136
• 2741	148	204
• 33/35 Teletypewriter (ASR)	200	200
• SYS/3	2(10+data)+30	14+data
• SYS/7	38+data	11*+2(data)
• Local Card Reader	90	10
• Direct SYSOUT	10	See Note 1
• Spool SYSOUT	31	See Note 1

* If FEAT=PTTC/EBCD is specified in the STATION macro, then 11+data.

Notes:

1. User-defined at system definition. 306 minimum for 3270 local.

2. User-defined output buffer size. Refer to the OUTBUF parameter of the TERMINAL macro statement in the IMS/VS Installation Guide.

3. User-defined retention area size. Refer to the RETSIZE parameter of the TERMINAL macro statement in the IMS/VS Installation Guide.

4. User-defined at system definition. Refer to the RECANY parameter of the COMM macro statement in the IMS/VS Installation Guide.

Figure 5-13. Communications Input/Output Line Buffers

DYNAMIC STORAGE REQUIREMENTS -- CONTROL REGION

The dynamic storage requirements within the control region include work areas and control blocks. The majority of requirements are generated indirectly by the use of the OS/VS supervisor services. Some direct requirements for work areas and control blocks are generated by IMS/VS. Figure 5-14 summarizes the OS/VS requirements.

DESCRIPTION	SIZE
1. Work areas, save areas.	5000
2. OPEN/CLOSE work area.	*
3. BISAM IOB, one per concurrent operation using ISAM data sets.	*
4. BISAM channel programs, one per open ISAM data set.	*
5. BSAM IOB and channel programs, one per data set.	*
6. BTAM IOB and channel programs, one per open communication line.	*
7. If OS/VS1, add for control blocks.	2568
8. If OS/VS1, add for data extent blocks (DEBs) for each open data set. OSAM 92 bytes 7770 80+4*Lines bytes ISAM BSAM BTAM GAM VTAM (only 1 per IMS)	*
9. If OS/VS1, add for TIOT space.	*
10. 7770 IOB and channel program, one per open communication line.	104 bytes

* See the appropriate OS/VS storage estimates publication, <u>OS/VS1 Storage Estimates</u>, or <u>OS/VS2 Storage Estimates</u>, for calculating items 2 through 6, 8, and 9.

Figure 5-14. OS/VS Storage Requirements in Control Region

IMS/VS DYNAMIC STORAGE REQUIREMENTS

The IMS/VS requirements for work areas and control blocks from the dynamic area are summarized here:

DESCRIPTION	SIZE
1. Work areas.	288
2. If security specifications other than default, see the formula below to calculate storage for security tables.	
3. Use the formula for calculating storage for ENQ/DEQ routines to calculate dynamic storage requirements.	

TOTAL IMS/VS DYNAMIC REQUIREMENTS

The following formula is used for calculating the size of the security table area:

Security Table Area Size = AB + C(D/8) + A(E/8).

where:

A = Number of passwords.

B = Maximum length of password.

C = Number of unique sets of terminal security specifications. For example, assume logical terminals X and Y can enter /START command and transaction code PAY. Even though three logical terminals are involved in the security specification, there is only one unique set of requirements common to all.

D = Number of logical terminals in system.

E = Number of unique sets of password security specifications. For example; if passwords AA, BB, and CC are valid for use with transaction CALC, command /SET, and command /LOCK, there is only one unique password security specification.

MESSAGE AND BATCH-MESSAGE PROCESSING REGIONS

For the purposes of storage estimates, the message and batch-message processing regions are identical. Figure 5-15 represents, conceptually, the physical organization of these dependent regions during execution.

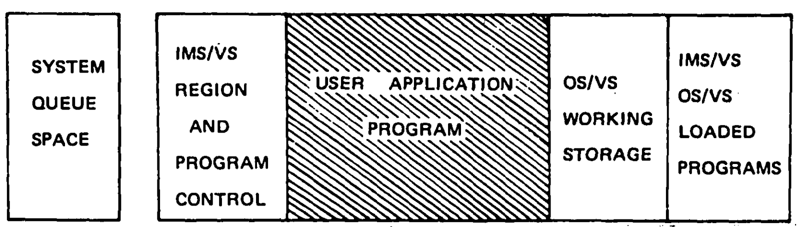


Figure 5-15. Message or Batch-Message Region Organization

The actual division of the areas shown in Figure 5-15 is not precisely disciplined. The shaded area represents the dynamic portion of the region that is available for use interchangeably by IMS/VS and user programs. It is one contiguous area in the center of the addressable space.

Figure 5-16 contains all the values necessary for calculation of the region size. Figure 5-16 then, is your worksheet for the storage estimates for message and batch-message processing required.

DESCRIPTION	OS/VS1 PART	OS/VS2 REGION
1. IMS/VS region and program control.	20,000	20,000
2. User application program area in Figure 5-15. Fill in program size.		
TOTAL		

Figure 5-16. Message or Batch-Message Region Size and Worksheet

See Figure 5-24 for an explanation of the values in Figure 5-16.

DATA BASE/DATA COMMUNICATION STORAGE REQUIREMENTS EXAMPLE

ENVIRONMENT

Storage requirements are based upon the environment outlined below:

OS/VS

- OS/VS2 V=V configuration.
- Step termination is resident in Link Pack Area (LPA) (IEFSD061).
- DL/I basic modules are in LPA because of anticipated frequency of concurrent batch and online processing (see Figure 5-10).
- HS and HD indexed, update function, and no write check are resident in LPA, because of frequency of use by concurrent batch and online processing.

IMS/VS

- Applications

  - 18 programs.

  - 23 transaction codes, one is conversational.

  - 6 data bases, HS and HD excluding HSAM, all stored on 2314 in 7000-byte blocks.

  - 1 transaction class.

- Terminals

  - 2740 Line Group, Non-switched

    Line 1: 1-2740 Model 1
    Line 2: 2-2740 Model 2

  - 2740 Line Group, Switched

    Line 1: 1-2740 Model 1

  - 1050 Line Group, Non-switched

    Line 1: 1-1050

  - 1050 Line Group, Switched

    Line 1: 1-1050
    Pool: 2-subpools

  - 2780 Line Group, Non-switched

    Line 1: 1-2780

    There is one logical name for each terminal or subpool, plus one for the master terminal.

- System Options

  - 6 concurrent IMS/VS subtasks can operate.

  - 3 concurrent message or batch-message regions can operate.

  - 11 concurrent exclusive control requests can be outstanding.

  - 3 main storage scratch pad areas of 100 bytes are to be available.

  - 6 direct access scratch pad areas of 150 bytes are to be available.

A control region worksheet is provided in Figure 5-19. In the "Control Region Calculation" discussion that follows, the term "worksheet" refers to Figure 5-19.

CONTROL REGION CALCULATION

The size of the CTL (control region) will be calculated first. Referring to Figure 5-6, the resident portion of terminal support is necessary to determine the size of the control program nucleus resident code. The total resident code for terminal device support from Figure 5-7 is 12600 bytes (Item 6 of Figure 5-6). Since the conversational option was elected, the total basic and optional code from Figure 5-6 is 167100 bytes. This value is entered on the worksheet at line 1a.

Referring to Figure 5-8 and the assumed environment, calculate the generated control blocks, line 1b of the worksheet:

Reference	Description		Size
1	Basic fixed control blocks		500
2	Six concurrent subtasks	6*1008	6048
3	Concurrent conversations	9*48	432
4	Transaction classes	1*80	80
5	Line groups	5*40	200
6	Lines and pools	5*124	620
7	Terminals	9*96	864
8	Different sets terminal attributes	6*36	216
9	Translation	6*512	3072
10	Logical terminal names	10*52	520
Size of generated control blocks			12552

Enter on the worksheet this total size of generated blocks on line 1b, and place the total of 1a plus 1b in the box to the right of 1b. The total size of the control program nucleus is 179652.

Line 2 of the worksheet is for locally loaded modules in the control region. Refer first to Figure 5-9. Terminal support of 7152 bytes is derived from OS/VS storage estimates. The total locally loaded module support to be entered in the box at line 2 of the worksheet is:

Always loaded	12000
Terminal support	7152
	19152 (19200 rounded)

Line 3h of the worksheet is for globally loaded modules. Note that the environment described contains certain modules in the resident LPA. The size of these modules can be found in Figure 5-10. The values shown in Figure 5-10 are included in the modules shown at line 1 in Figure 5-9. Therefore, a deduction must be made because some of them have been selected to go in the resident LPA. Line 1 of Figure 5-9 has been adjusted for DL/I Basic modules in the LPA.

The total globally loaded module support to be entered at line 3i of the worksheet is:

Always loaded (adjusted) 20700

The total DL/I code resident in the LPA is:

Always loaded 134000

Referring to Figure 5-11 and the assumed environment, calculate line 3a of the worksheet, global control blocks:

Reference	Description		Size
1	Basic fixed control blocks		18600
2	Three processing regions	3*4096	12288
3	Application programs	18*44	792
4	Transaction codes	23*68	1564
5	Data bases	6*40	240
	Size of global control blocks		33484

Enter on line 3a of the the worksheet.

Calculate the storage requirement for the system log work area using the formula shown in this chapter for system log buffers:

```
488 + A *(216 + B)

A= 2    The number of log buffers
B=1048  1024 + 24, checkpoint log workarea
        + overhead

488 + 2 *(216 + 1048)

= 3016
```

Enter the result on line 3h of the worksheet.

The buffer areas, lines 3b through 3g and lines 4a and 4c on the worksheet, are calculated next. The environment description contains 23 transaction codes and 10 logical terminal names. Using the default queue buffer size and calculation for the number of buffers, the buffer length is 576 and the number is 4 + [(23 + 10)/10] = 4 + 3.3, or 8 buffers. The size, calculated with the queue buffer pool size formula (shown under "Queue Buffer Pool" in this chapter), is 8(576 + 40) + 160, or 5,188 bytes. Enter 5188 on line 4a of the worksheet.

It is decided that both the program and data base description block buffer areas must be large enough to contain the two largest sets of those control blocks. For purposes of the example, assume PSBs are identical and refer to identically organized data bases. There are two data bases that are logically related. They are viewed by the application program as two structures in Figure 5-17.

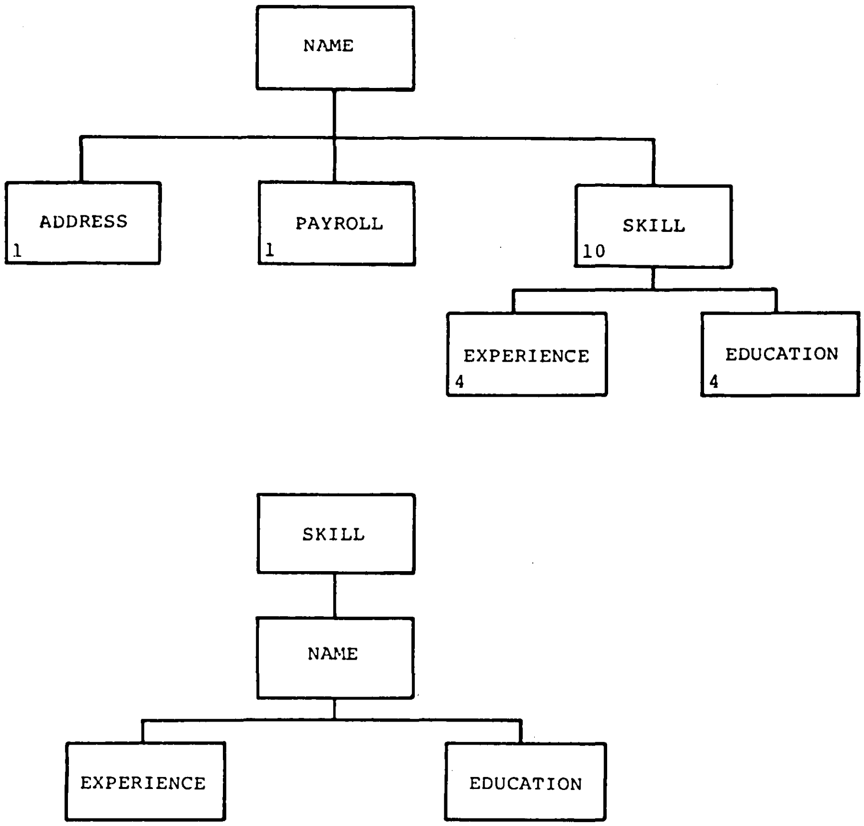


Figure 5-17. Hierarchic Structure for Two PSBs

The physical data bases through which these structures are viewed are shown in Figure 5-18.

Both are HD organization and are accessed using HDAM and HIDAM. Each consists of only one data set group. The length of each segment type is shown in Figure 5-18 in the lower right of each box. The length of the segment key is in the lower left of each box. The application programs have a processing option of ALL for all segment types. They are written in PL/I. Each application uses six alternate logical terminals.

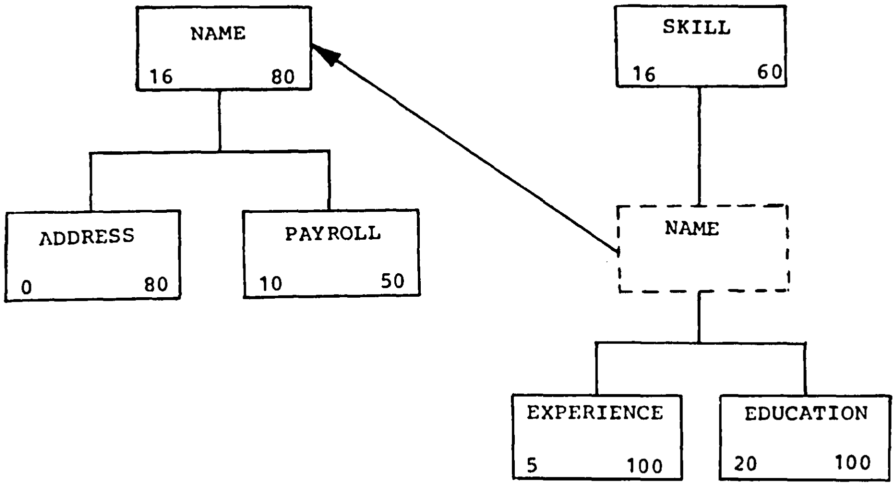


Figure 5-18. Two Data Bases Logically Related

Estimate the amount of storage required for the program isolation enqueue/dequeue routines using the formula shown in this chapter for storage estimates for IMS/VS ENQ/DEQ routines (size = I*N). The values for the calculation are as follows:

I = 1024 (The default increment).

N = (32A * (B + C + D + E + 3F + G + 3H)) / I

A = 3 (The number of scheduled regions)

B = 2 (The number of root segments that can be accessed in a retrieval call. Refer to the two data bases interrelated by logical relationships, Figure 5-19).

C = 2 (No path calls used. The concatenated segments making the logical relationship require two entires.)

D = 0 (None of the programs use the enqueue command code.)

E = 10 (Assumed from application programmer's estimate of 2.)

F = 6 (Assumed from application programmer's estimate of 3.)

G = 0 (NAME L/C segment has a Virtual Delete Rule.)

H = 6 (Assumed from application programmer's estimate of 3.)

N = (32*3 * (2 + 2 + 0 + 10 + 3*6 + 0 + 3*6)) / (1024);

= (96 * 50) / 1024;

= 4.7 Rounded up to the next whole number = 5

S = 1024 * 5 = 5120

This value is used in determining IMS/VS dynamic storage requirements.

Calculate the PSB size as described in the formula for determining PSB pool requirements shown in a preceding section of this chapter (PSB Area = A+B+C+D). The values for the calculation are as follows:

A =	58
B =	660
C =	2106
D =	428
PSB size A + B + C + D =	3252

The program description block buffer pool is then 8000 bytes (twice 3252, to the next 1000 bytes). Enter on line 3b of worksheet. The PSB work pool is 2000 bytes (1216 to the next 1000 bytes). Enter on line 3j of the worksheet.

Calculate the DMB size using the formula supplied in a preceding section of this chapter. Assume input to data base description by the Payroll data base with logical relationships using HIDAM, and the Skills Inventory data base with logical relationships using HDAM as defined in the IMS/VS Utilities Reference Manual.

DMB Size:

For physical Payroll data base -- HIDAM =	456
For index of Payroll data base =	572
For physical Skills Inventory data base -- HDAM =	464
DMB Size =	1492

The data base description block buffer pool is then 2,000 bytes. Enter value on line 3c of the worksheet.

For the data base buffer pool, calculate the size using the formula, shown in a preceding section of this chapter, for determining minimum size of a data base buffer pool (size=A+B+C). The values to be used in this calculation are as follows (assume minimum guideline is no contention between the two largest programs):

A = (2*7000) + 300 = 14300

B = 0 0

C = for the first application program data structure.

```
T = 54+(16*2)+26 = 112  (Payroll)
T = 54+(16*3)+52 = 154  (Skills Inv.)
                   266
```

Assume no name will appear in more than 10 skills data base records. = 2660

Data base buffer pool = 16960

The size of the data base buffer area is then 34000 bytes (twice 16960, to the next 1000 bytes). Enter value on line 3d of the worksheet.

Calculate the data base work pool size as described earlier in this chapter.

2000 bytes (minimum recommended size) x 3 message regions=6000

Enter value in line 3e of the worksheet.

Line control buffers are calculated using Figure 5-12. Terminal activity is forecast not to exceed .75 times the total buffer requirement for no contention on concurrent output of all lines.

2740 line with Model 2 Buffer 440		456
2740 line w/o Model 2	(2*204)	408
1050 line and pool	(3*204)	612
2780		416
System console		136
		2028

Line buffer pool size is 2000 bytes. Enter the value on line 4c of the worksheet.

Based upon the formula for calculating size of the general buffer pool, shown in a preceding section of this chapter, it is decided that 6000 bytes is adequate for the general buffer area. Enter on line 3f of the worksheet.

Based on the data base log buffer discussion in this chapter, 7000 bytes is allocated to this buffer. Enter the value on line 3g of the worksheet.

Dynamic storage requirements are calculated, using Figure 5-14 and the 288 bytes shown under "IMS/VS Dynamic Storage Requirements," as the sum of the two values. Starting in Figure 5-14:

Reference	Description	Size
1	Work Area	5000
2	Open/Close Work Area	2784
3	BISAM IOB	112
4	BISAM Channel Program	500
5	None	0
6	BTAM IOBs -- 5	760
7,8,9	Not OS/VS1	----
	TOTAL	9156

Enter the total on line 5 of the worksheet.

From the IMS/VS dynamic storage requirements using defaults with no security:

Reference	Description	Size
1	Work Area	288
2	No Security	0
3	ENQ/DEQ Routines	5120
	TOTAL	5408

Enter the total on line 6 of the worksheet.

1.	Control Program Nucleus		
	a. Resident Code	167100	
	b. Generated Control Blocks	12552	179652
2.	IMS/VS Locally Loaded Modules		19200
3.	Global Areas		
	a. Control Blocks	33484	
	b. Program Specification Blocks	8000	
	c. Data Base Description Blocks	2000	
	d. Data Base Buffers	34000	
	e. Data Base Work Pool	6000	
	f. General Buffers	6000	
	g. DBLLOG Buffers	7000	
	h. System Log Buffers	3016	
	i. IMS/VS Globally Loaded Modules	20700	
	j. PSB Work Pool	2000	122200
4.	Buffer Areas (local CTL-region storage)		
	a. Queue Buffers		5188
	b. Line Control Buffers		2000
	c. CWAP		1000
5.	Dynamic Storage Requirements -- OS/VS		9156
6.	Dynamic Storage Requirements -- IMS		5408
	Region size Total Items 1, 2, 4-6		220604
	CSA storage Total Item 3		123200

Figure 5-19. Worksheet for DB/DC Example

The total storage required is:

Control region (rounded)	221000
Add Link Pack Area	134000
TOTAL STORAGE	355000 bytes OS/VS2 478200 bytes OS/VS1

Control region including LPA

MESSAGE PROCESSING REGION CALCULATION

The size of the message processing region is determined using Figure 5-16.

DATA BASE/DATA COMMUNICATION SYSTEM MINIMUM STORAGE REQUIREMENTS EXAMPLE

The following environment is assumed for a minimum Data Base/Data Communications storage requirements example:

- One data base -- HISAM organization, single data set group.
- The data base has five segment-types, two fields each, eight-character key field, and five hierarchical levels.
- No logical relationships.
- PROCOPT = A.
- Three COBOL application programs with a total of six transaction codes.
- One non-switched communication line and one 2740 Model 1 communication terminal attached.
- No conversational application programs.
- Two concurrent subtasks.
- One message region.
- Three queue buffers.
- Segment length is 256.
- One logical terminal PCB.
- Four logical terminals.
- One transaction class.

A control region worksheet is provided as Figure 5-20. In the discussion that follows, the term "worksheet" refers to Figure 5-20.

The size of the CTL is calculated first. Referring to Figure 5-6, Item 2, the conversational option is not selected for this minimum environment. Item 6 is the resident portion of the terminal device support whose value is selected from Figure 5-7, 600 bytes for the 2740 Model 1 non-switched terminal plus 2700 bytes required basic code. The total basic and optional code value from Figure 5-6 is 78300 bytes for VS/1, and 147300 for VS/2 V=V, and is entered on line 1a of the worksheet.

Referring to Figure 5-8 and the assumed environment, calculate line 1b of the worksheet-generated control blocks:

Reference	Description		Size
1	Basic fixed control blocks		500
2	Two concurrent subtasks	2*1008	2016
3	Conversational	---	---
4	Transaction class	1*80	80
5	Line groups	1*40	40
6	Lines and pools	1*124	124
7	Terminals (subpools)	1*96	96
8	Different sets terminal attributes	1*36	36
9	Translation	1*512	512
10	Logical terminal names	4*52	208
	Size of generated control blocks		3612

Enter this total size of generated control blocks on line 1b of the worksheet, sum 1a plus 1b, and place the result in the box to the right of 1b. The total size of the control program nucleus, rounded to the nearest 1K bytes, is 160000 for VS/1 and 229000 for VS/2.

Line 2 of the worksheet is for loaded modules in the control area. Refer to Figure 5-9. Total locally loaded modules, from line 2 of worksheet, is 12000 bytes. Total globally loaded modules from Figure 5-9 is 155200. Enter on line 3i of worksheet. These modules can be placed in the virtual link pack.

Referring to Figure 5-11 and the assumed environment, calculate line 3a of the worksheet, global control blocks:

Reference	Description		Size
1	Basic fixed control blocks		18600
2	One processing region	1*4096	4096
3	Application programs	3*44	132
4	Transaction codes	6*68	408
5	Data bases	1*40	40
	Size of global control blocks.		23276

Enter on line 3a of the worksheet.

Calculate the storage requirement for the system log work area using the formula shown in this chapter for system log buffers:

488 + A *(216 + B)

A=2 the minimum number of log buffers
B=1048 1024 + 24, checkpoint log workarea + overhead

488 + 2 *(216 + 1048)

=3016

Enter the result on line 3h of the worksheet.

The buffer areas, lines 3b through 3g and lines 4a and 4c on the worksheet, are calculated next. Item 3b is for queue buffer pools. The minimum is used: 3(384 + 40) + 160 = 1432 bytes, and is entered on line 3b. Item 3b is for program description blocks and calculates as 890 bytes. Rounding up to the nearest 1000 bytes, enter 1000 for Item 3b on the worksheet. Calculating Item 3c, the result is 808 bytes for the DMB buffer pool. Rounding to the nearest 1000 bytes, place 1000 bytes in Item 3c of the worksheet. Data base buffer pool is Item 3d, where the default value of 7000 was used. Data base work pool is item 3e, with one message region the minimum recommended size of 2000 bytes was entered. Item 4c reflects the calculation for line buffer pool size.

2740 Model 1 output	204
System console input	148
System console output	136
	488 bytes

For the estimate of the general buffer pool, enter the minimum size of 6400 on line 3f of the worksheet.

Estimate the amount of storage required for the program isolation enqueue/dequeue routines using the formula shown in this chapter for storage estimates for IMS/VS ENQ/DEQ routines (SIZE=I*N). The values for the calculation are as follows:

I= 1024 (The default increment)

N= (32A*(B+C+D+E+3F+G+3H))/I

A=1 The number of scheduled regions.
B=1 The number of root segments that can be accessed in a retrieval call.
C=1 The number of data base segments that can be retrieved in HOLD status.
D=0 None of the programs use the enqueue command code.
E=1
F=1
G=0
H=1

N= (32*(1+1+0+1+3+0+3+/1024
 = (32*9)/1024
 = .28 rounded up to a whole number=1

S= 1024*1=1024

Dynamic storage requirements are calculated, using Figure 5-14 and the 288 bytes shown under "IMS/VS Dynamic Storage Requirements," as a sum of the two values. Starting in Figure 5-14:

Reference	Description	Size
1	Work areas	5000
2	OPEN/CLOSE line groups	1672
3	BISAM IOB	56
4	BISAM channel program	600
5	None	---
6	BTAM IOBs	152
7	Control blocks (OS/VS1)	2568
8	DEBs (OS/VS1) 1 ISAM 3 OSAM 1 BTAM	388
9	TIOT space (OS/VS1) 1 device in each DD statement 9 DD statements	336

Total OS/VS dynamic storage requirements: 10772 bytes OS/VS1
7480 bytes OS/VS2

Enter the result on line 5 of the worksheet.

From the IMS/VS dynamic storage requirements with no security:

Reference	Description	Size
1	Work Area	288
3	ENQ/DEQ Routines	1024
	TOTAL	1312

Enter the result on line 6 of the worksheet.

REF	DESCRIPTION	SIZE VS/1	SIZE VS/2
1.	Control Program Nucleus		
	a. Resident Code	78300	147300
	b. Generated Control Blocks	3612	3612
		81912	150912
2.	IMS/VS Locally Loaded Modules	12000	12000
3.	Global Areas		
	a. Control Blocks	23276	
	b. Program Specification Blocks	1000	
	c. Data Base Description Blocks	1000	
	d. Data Base Buffers	7000	
	e. Data Base Work Pool	2000	
	f. General Buffers	6400	
	g. DBLLOG Buffers	1024	
	h. System Log Buffers	3016	
	i. IMS/VS Globally Loaded Modules	155200	
	j. PSB Work Pool	1000	
		200916	
4.	Buffer Areas		
	a. Queue Buffers	1432	1432
	b. Line Control Buffers	488	488
5.	Dynamic Storage Requirements -- OS/VS	10772	7480
6.	Dynamic Storage Requirements -- IMS	1312	1312
	Region size OS/VS1	308308	
	Region size OS/VS2 (Total of items 1, 2, 4-6)		173624
	CSA storage OS/VS2		200916

Figure 5-20. Worksheet for Minimum DB/DC Example

In summary, the total control region storage requirement is:

309000 bytes for OS/VS1.

174000 bytes for OS/VS2.

A minimum message or batch-message region must be one of the largest from the following:

1. The OS/VS partition/region defined by the user's OS/VS system plus 20K.

2. The largest message processing program in the user's IMS/VS system plus 20K.

Using a minimum OS/VS1 or OS/VS2 system, in connection with the minimum IMS/VS DB/DC system, this teleprocessing execution can operate on a 348K machine for OS/VS1, or a 1024K machine for OS/VS2.

DATA BASE UTILITIES STORAGE REQUIREMENTS

This section provides the necessary data with which to estimate storage requirements for the IMS/VS Data Base utility programs that are involved with Data Base Recovery and Data Base Reorganization/Load processing functions of IMS/VS. The first four utilities' storage requirements refer to "Data Base Recovery" in the *IMS/VS Utilities Reference Manual*.

- Data Base Image Copy -- DFSUDMP0
- Data Base Change Accumulation -- DFSUCUM0
- Data Base Recovery -- DFSURDB0
- Data Base Backout -- DFSBB000

The next set of eight utilities' storage requirements refer to "Data Base Reorganization/Load Processing" in the *IMS/VS Utilities Reference Manual*.

- Data Base Physical Reorganization
  - HISAM Reorganization Unload -- DFSURUL0
  - HISAM Reorganization Reload -- DFSURRL0
  - HD Reorganization Unload -- DFSURGU0
  - HD Reorganization Reload -- DFSURGL0
- Data Base Logical Relationship Resolution
  - Data Base Pre-reorganization -- DFSURPR0
  - Data Base Scan -- DFSURGS0
  - Data Base Prefix Resolution -- DFSURG10
  - Data Base Prefix Update -- DFSURGP0

Note: Unless otherwise indicated, the following storage requirements pertain to OS/VS1 and OS/VS2 system options.

DATA BASE IMAGE COPY UTILITY -- DFSUDMPO

The formula supplied below must be used once for each data base image copy statement to be processed. The largest value thus obtained, rounded up to the nearest 2K multiple, can be used to estimate the region or partition size for a given execution of the Data Base Image Dump utility program.

Required Main Storage/Control Statement = 30,500+(A*(B+84))+(C*(D+84))+E
+F+G+H+I+J+K.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size. Default is 80.

C = Number of SYSPRINT buffers. Default is 2.

D = SYSPRINT data set block size. Default is 121.

E = 7498 if data base data set is ISAM. 0 if OSAM.

F = Buffer Space Required = (H*(I+136))+(J*(K+84)).

G = OS/VS control blocks and work space. See the formulas referred to under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

H = Number of data base data set buffers. Default is 2.

I = Data base data set block size.

J = Number of output data set buffers. Default is 2.

K = Output device data capacity, but limited to a maximum of 8191 bytes.

DATA BASE CHANGE ACCUMULATION UTILITY -- DFSUCUMO

The following formula can be used to estimate the region or partition size required for a given execution of the Data Base Change Accumulation program:

Required Main Storage = 21000+(A*(B+84))+(C*(D+84))+(E*(F+84))+(G*(H+84))
+(I*(J+84))+(K*(L+84))+N+120+(32*P)+O+Q+R.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size. Default is 80.

C = Number of SYSPRINT buffers specified. Default is 2.

D = SYSPRINT data set block size. Default is 121.

E = Number of DFSUCUMN buffers specified. Default is 2.

F = DFSUCUMN data set block size (normally device capacity, but limited to a maximum of 8191 bytes).

G = Number of DFSUCUMO buffers specified. Default is 2.

H = DFSUCUM0 data set block size.

I = Number of DFSUDD1 buffers specified. Default is 2.

J = DFSUDD1 data set block size (normally device capacity, but limited to a maximum of 8191 bytes).

K = Number of DFSULOG buffers specified. Default is 2.

L = DFSULOG data set block size.

N = 28800 if a DFSUCUMN DD statement was supplied; otherwise 0.

O = Number of db names specified on an ID control statement. Default is 16.

P = Number of DD names specified on an ID control statement. Default is 80.

Q = Amount of main storage for OS/VS sort, as specified in the EXEC statement parameters. Default is 100,000.

R = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

DATA BASE RECOVERY UTILITY -- DFSURDB0

The following formula can be used to estimate the region or partition size required for a given execution of the Data Base Recovery program:

Required Main Storage = 42500+(A*(B+84))+(C*(D+84))+(E*(F+84))+(G*(H+84))
+(I*(J+84))+K+L+M+(N*(O+136))+P+Q+S+T+U.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size. Default is 80.

C = Number of SYSPRINT buffers specified. Default is 2.

D = SYSPRINT data set block size. Default is 121.

E = Number of DFSUDUMP buffers specified. Default is 2.

F = DFSUDUMP data set block size.

G = Number of DFSUCUM buffers specified. Default is 2.

H = DFSUCUM data set block size.

I = Number of DFSULOG buffers specified if no DFSUDUMP or DFSUCUM supplied; otherwise 0.

J = DFSULOG data set block size if one is supplied; otherwise 0.

K = Data base buffer pool size specified. Default is 7000.

L = 7200 if DFSUCUM data set is supplied; otherwise 0.

M = 2000 if DFSULOG supplied and no DFSUDUMP supplied; otherwise 0.

N = Number of data base data set buffers specified. Default is 2.

O = Data base data set block size.

P = 7498 if data set to be recovered is ISAM; otherwise 0.

Q = PSB size calculation as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." The definition is as if a single PCB where PROCOPT = G had been defined. The PSB is sensitive to all segments in the data base.

S = DMB size as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements."

T = Size of the randomizing module if HDAM; otherwise 0.

U = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

DATA BASE BATCH BACKOUT UTILITY -- DFSBB000

The following formula can be used to estimate the region or partition size required for a given execution of the Data Base Batch Backout program:

Required Main Storage = 4280+A+B.

where:

4280 = Size of program DFSBB000.

A = Block size of input log tape.

B = Total of references 1 through 9 of the Data Base System worksheet in this chapter for the user's IMS/VS Data Base system.

HISAM REORGANIZATION UNLOAD UTILITY -- DFSURUL0

The following formula is to be used once for each control statement to be processed. The largest value thus obtained, rounded up to the nearest 2K multiple, can be used to estimate the region or partition size for a given execution of the HISAM Reorganization Unload program.

Required Main Storage = 61500+(A*(B+84))+(C*(D+84))+E+F+G.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size. Default is 80.

C = Number of SYSPRINT buffers. Default is 2.

D = SYSPRINT data set block size. Default is 133.

E = Block size of the OSAM data set.

F = Buffer Space Required = (H*(I+136))+(J*(K+84))+L for ISAM/OSAM.
= (H*(I+136))+(J*(K+84))+L+M for VSAM.

G = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

H = The number of ISAM data set buffers specified. Default is 2.

I = ISAM data set block size.

J = Number of output data set buffers. Default is 2.

K = Output device data capacity, but limited to a maximum of 16384 bytes.

L = Size of buffers required for DL/I as specified on EXEC statement. Default is 7K.

M = Buffers required by VSAM as specified by the DEFINE statement.

HISAM REORGANIZATION RELOAD UTILITY -- DFSURRLO

The following formula is to be used once for every ISAM/OSAM data set group to be reloaded. The largest value, rounded up to the nearest 2K multiple, can be used to estimate the region or partition required for a given execution of the HISAM Reorganization Reload utility program:

Required Main Storage = 11500+(A*(B+84))+(C*(D+84))+(E*(F+84))
+(G*(H+136))+I.

where:

A = Number of SYSPRINT buffers specified. Default is 2.

B = SYSPRINT data set block size. Default is 133.

C = Number of buffers specified on the associated DFSUINxx DD statement. Default is 2.

D = Associated DFSUINxx data set block size. Normally input device capacity, but limited to a maximum of 16384 bytes.

E = Number of buffers specified for the OSAM data set. Default is 2.

F = OSAM data set block size.

G = Number of buffers specified for the ISAM data set. Default is 2.

H = ISAM data set block size.

I = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

HD REORGANIZATION UNLOAD UTILITY -- DFSURGU0

The following formula can be used to estimate the region or partition size required for a given execution of the HD Reorganization Unload utility program:

Required Main Storage = 66500+(A*(B+84))+(2*C)+D+(40*E)+F+H+I+J+K+L+M.

where:

A = Number of SYSPRINT buffers specified. Default is 2.

B = SYSPRINT data set block size. Default is 121.

C = The smaller of (a) the output block size of the DFSURGU1 data set, or (b) the output block size of the DFSURGU2 data set. This is normally the smaller output device capacity, but limited to a maximum of 8191 bytes.

D = Specified buffer pool size. Default is 7000.

E = Number of SEGM statements in the DBD for this data base.

F = PSB size calculation as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." The definition is as if a single PCB with PROCOPT = G had been defined. The PSB is sensitive to all segments in the data base.

H = DMB size as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements."

I = 7498 if HISAM or HIDAM data base is being unloaded; otherwise 0.

J = Total buffer requirements for all ISAM data sets in the data base being unloaded.

K = Size of randomizing module if HDAM data base; otherwise 0.

L = 1000 if checkpoints are being taken or a restart is being done; otherwise 0.

M = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

HD REORGANIZATION RELOAD UTILITY -- DFSURGL0

The following formula can be used to estimate the region or partition size required for a given execution of the HD Reorganization Reload utility program:

Required Main Storage = 60000+(A*(B+84))+(C*(D+84))+(E*(F+84))+(G*(H+84)) +I+J+L+M+N+O.

where:

A = Number of SYSPRINT buffers specified. Default is 2.

B = SYSPRINT data set block size. Default is 121.

C = Number of DFSUINPT buffers specified. Default is 2.

D = DFSUINPT data set block size. Normally device capacity, but limited to a maximum of 8191 bytes.

E = Number of buffers specified for the DFSURWF1 data set. Default is 2.

F = DFSURWF1 data set block size.

G = Number of buffers specified for the DFSURCDS data set. Default is 2.

H = DFSURCDS data set block size.

I = Data base buffer pool size. Default is 7000.

J = PSB size calculation as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." The definition is as if a single PCB with PROCOPT = G and sensitive to all segments in the data base has been defined.

L = DMB size as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements."

M = 8632 if data base is HISAM or HIDAM; 1688 if the data base is HSAM; or, if HDAM, the size of the randomizing module.

N = Total buffer requirements for all ISAM data sets in the data base being reloaded. The default number of buffers for each data set is 2.

O = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

DATA BASE PRE-REORGANIZATION UTILITY -- DFSURPRO

The following formula is to be used to estimate the region or partition size required for a given execution of the Data Base Pre-reorganization utility program:

Required Main Storage = 30000+(A*(B+84))+(C*(D+84))+(E*(F+84))+G+I+J+K.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size.

C = Number of SYSPRINT buffers specified. Default is 2.

D = SYSPRINT data set block size.

E = Number of DFSURCDS buffers specified. Default is 2.

F = DFSURCDS data set block size.

G = PSB size as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." The definition is as if a single PCB with PROCOPT = G and sensitive to all segments in the data base has been defined. This calculation must be made once for every DBD name that appears in a control statement. The largest value obtained is the value to be used.

I = DMB size as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements." This calculation must be made once for every DBD name that appears in a control statement. The largest value obtained is the value to be used.

J = Data base buffer pool size specified. Default is 7000.

K = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

DATA BASE SCAN UTILITY -- DFSURGS0

The following formula is to be used to estimate the region or partition size required for a given execution of the Data Base Scan program:

```
Required Main Storage = 68500+(A*(B+84))+(C*(D+84))+(E*(F+84))
                        +(G*(H+84))+ (I*(J+84))+K+L+M+N+P+Q+R+S.
```

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size.

C = Number of SYSPRINT buffers specified. Default is 2.

D = SYSPRINT data set block size.

E = Number of buffers specified for DRFURWF1. Default is 2.

F = DFSURWF1 data set block size.

G = Number of DFSURCDS buffers specified. Default is 2.

H = DFSURCDS data set block size.

I = Number of DFSURSRT buffers specified; otherwise 0.

J = DFSURWF1 data set block size.

K = Data base buffer pool size specified. Default is 7000.

L = 7498 if HISAM or HIDAM data bases are to be scanned; otherwise 0.

M = Size of the largest randomizing module to be used.

N = PSB size calculation as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." This calculation must be made once for each data base that is to be scanned. The definition is as if a single PCB with PROCOPT = G and sensitive to all segments in the data base has been defined. The largest value obtained must be the value used.

P = DMB size as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements." This calculation must be made once for each data base that is to be scanned. The largest value obtained must be the value used.

Q = Total buffer requirements for all ISAM data sets that can be open simultaneously. If multiple calculations are necessary, the largest value obtained must be the value used.

R = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

S = Size of the largest segment to be scanned.

DATA BASE PREFIX RESOLUTION UTILITY -- DFSURG10

The following formula is to be used to estimate the region or partition size required for a given execution of the Data Base Prefix Resolution utility program:

Required Main Storage = 20000+(A*(B+84))+(C*(D+84))+(E*(F+84))
+(G*(H+84))+I+J.

where:

A = Number of SYSPRINT buffers specified. Default is 2.

B = SYSPRINT data set block size.

C = Number of buffers for the DFSURCDS data set specified. Default is 2.

D = DFSURCDS data set block size.

E = Number of DFSURWF2 buffers specified. Default is 2.

F = DFSURWF2 data set block size.

G = Number of DFSURWF3 buffers specified. Default is 2.

H = DFSURWF3 data set block size.

I = Amount of main storage for OS/VS SORT, if specified in the EXEC statement; otherwise 61440 bytes.

J = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

DATA BASE PREFIX UPDATE UTILITY -- DFSURGP0

The following formula is to be used to estimate the region or partition size required for a given execution of the Data Base Prefix Update utility program:

Required Main Storage = 72000+(A*(B+*84))+C*(D+*84))+(E*+*84))+G+H+I
+J+K+M+N.

where:

A = Number of SYSIN buffers specified. Default is 2.

B = SYSIN data set block size.

C = Number of SYSPRINT buffers specified. Default is 2.

D = SYSPRINT data set block size.

E = Number of DFSURWF3 buffers specified. Default is 2.

F = DFSURWF3 data set block size.

G = Data base buffer pool size specified. Default is 7000.

H = 7498 if any ISAM/OSAM data set groups are defined on DD statements.

I = Size of the largest randomizing module that will be used.

J = Total buffer requirements for all ISAM data sets that can be open simultaneously. If multiple calculations are necessary, the largest value obtained must be the value used. The default number of buffers for all data sets is 2.

K = PSB size calculation as described under "IMS/VS Program Specification Block" in the section "Data Base System Storage Requirements." The definition is as if a single PCB with PROCOPT = G and sensitive to all segments in the data base has been defined. This calculation must be made once for each data base that is to be updated. The largest value obtained must be the value used.

M = DMB size calculation as described under "IMS/VS Data Base Description" in the section "Data Base System Storage Requirements." calculation must be made once for each data base that is to be updated. The largest value obtained must be the value used.

N = OS/VS control blocks and work space. See the appropriate formula under "OS/VS Control Blocks, Buffers, and Work Space" in the section "Data Base System Storage Requirements."

SPOOL SYSOUT PRINT UTILITY -- DFSUPRT0

The following formula is to be used to estimate the region or partition for a given execution of the Spool SYSOUT Print utility program:

Required Main Storage = 3500+204*A+(8+(4xB)+(B*C))+(8+(4*D)+(D*E)) +(112+88*B)+(112+128*D).

where:

A = Number of spool data sets processed by the utility.

B = Number of buffers for SYSPRINT data set specified.

C = SYSPRINT data set block size.

D = Number of buffers for spool data set specified.

E = Spool data set block size.

Assumes basic QSAM modules resident.

STORAGE ESTIMATES SOURCE DATA

REF	DESCRIPTIONS
1.	Basic Fixed Control Blocks a. PSB Most Used QCB b. DMB Most Used QCB c. System Console CLB d. System Console CNT e. System Console Translate Table f. CVBs
2.	Maximum Concurrent Input/Output a. Save Area Set
3.	Concurrent Conversations a. CCB
4.	Transaction Class a. TCT
5.	Line Groups a. Access method DCB (BTAM, BSAM, 7770, GAM) b. Open list entry for each DCB
6.	Lines a. CLB b. SAP Wait Stack c. LERB
7.	Terminals a. CTB b. Average of 7 bytes per polling list entry c. CRB
8.	Unique Terminal Attribute Set a. CTT
9.	Unique Terminal Translation a. Pair of translate tables
10.	Each Logical Terminal a. CNT
11.	2770 Terminal a. CXB
12.	Message Format Service a. CIB
13.	3270 Switched Terminal a. CONFIG Table b. IDLIST List

Figure 5-21. IMS/VS Control Blocks in the Control Program Nucleus

Figure 5-22 lists the modules used to calculate the values in Figures 5-9 and 5-10.

1.	DL/I Logging	DFSRDBL0**		
2.	DL/I and MSG Q'ing	DFSAOS10** DFSAOS20**	DFSAOS30** DFSAOS50**	DFSAOCE0**
3.	DL/I	DFSDISM0** DFSDLOC0** DFSDXMT0** DFSDBCK0** DFSDVBH0**	DFSDHDS0** DFSDBH00** DFSDDLE0** DFSDSEH0** DFSFXC10**	DFSDLA00** DFSDLD00** DFSDLR00** DFSDLDV0**
4.	Misc	DFSPRPX0* DFSPRRG0* DFSIDSP0** DFSASK00** DFSREP00** DFSCPY00**	DFSFTIM0** DFSFUNL0** DFSFLRC0** DFSFMOD0* DFSFCTT0* DFSRBCP0*	DFSFLLG0** DFSFPLG0 DFSRDLG0** DFSFLST0* DFSFCST0*
5.	DL/I VSAM	DFSDVSM0**	DFSDVBH0	
6.	Conversational Option	DFSCONV0**		
7.	DC Monitor Option	DFSIMNT0	DFSMNTR0**	

* These modules are not re-entrant and may not be placed in the system link pack area.

** In VS/2 MVS these modules will be loaded into CSA.

Figure 5-22. Loaded Modules in CTL Region

1. Basic Fixed Control Blocks
   a. SCD
   b. OSAM IOB QCB in DFSIDS40
   c. OSAM DCBs for Queue Data Sets and Dynamic Log
   d. Background Write PST
2. Maximum Active Regions
   a. PST
3. Application Programs
   a. PSB Directory
4. Transaction Codes
   a. SMB
5. Data Bases
   a. DMB Directory

Figure 5-23. IMS/VS Global Areas (CSA in MVS)

DESCRIPTION	VS1 PARTITION	VS2 REGION
1. IMS/VS Region and Program Control		
a. RRC10	x	x
b. PR020	x	x
c. PCC20 *2	x	x
d. CPY00	x	x
e. REP00	x	x
f. ASK00	x	x
g. DIRCA	x	x
h. ATTACH	x	x
i. SSCD	x	x
j. PXPARMS	x	x

Figure 5-24. Message/Batch -- Message Region Contents

CHAPTER 6. COMMUNICATIONS WITH INTELLIGENT REMOTE STATIONS

INTRODUCTION

The IMS/VS System/Application Design Guide contains introductory and design information about IMS/VS Intelligent Remote Station Support (IRSS). This chapter describes the details of the communications interface between IMS/VS and IRSS terminals. IRSS support is available for the IBM System/3 Model 10 and the IBM System/7.

IMS/VS supports a System/3 attached on a binary synchronous (BSC) nonswitched polled line.

IMS/VS supports a System/7 attached on a start/stop (S/S), nonswitched, contention or polled line. Polling can be done using either programmed polling or autopoll.

IMS/VS also supports a System/7 attached on a BSC, nonswitched, contention or polled line.

IRSS stations may or may not have a restart facility for messages to IMS/VS. They are not expected to have a restart facility for messages from IMS/VS.

TERMINAL IDENTIFIERS

All terminal identifiers used in communication between IMS/VS and IRSS stations must be defined to IMS/VS in the TERMINAL macro during system definition. After the IRSS station has completely processed a message received from IMS/VS for a given terminal identifier, it must so inform IMS/VS. This action allows IMS/VS to transmit the next message, if any. An output message, partly or completely sent but not dequeued, is returned to the queue and retransmitted if an input message for the same identifier is received.

MESSAGE FORMATS

A message is divided into segments. Some messages are defined as single-segment messages and can never contain more than one segment; others are defined as multiple-segment messages and can contain one or more segments.

IMS/VS works with three types of messages:

- Transactions

  A transaction is a message to be processed by an application program. A transaction is defined at IMS/VS system definition as either a single- or a multiple-segment message.

- Message Switches

  A message switch is a message routed to a logical terminal for output by IMS/VS. It cannot be processed by an application program. A message switch is always defined as a multiple-segment message.

* Commands

  Commands control functions within IMS/VS. A command has a slash as the first significant character of its first segment. No other segment can have a slash as its first significant character. All commands normally allowed from a System/3 or System/7, except the /BROADCAST command, are defined as single-segment messages. /BROADCAST is defined as a multiple-segment message, but is different from other multiple-segment messages in two respects:

  1. /BROADCAST should contain at least two segments.

  2. The total length of all segments making up the /BROADCAST message must not exceed the size of the large message buffer as defined in the IMS/VS system definition.

The various commands available are described in the IMS/VS Operator's Reference Manual.

The remainder of this chapter is divided into three major sections. The first describes the interface between IMS/VS and a System/3 or System/7 attached on a BSC line. The second describes the interface between IMS/VS and a System/7 attached on a start/stop line. The third section contains examples of transmission sequences between IMS/VS and an IRSS station; no distinction between station type is made.

INTERFACE BETWEEN IMS/VS AND THE SYSTEM/3 OR SYSTEM/7 BSC

The interface between IMS/VS and a System/3 or System/7 consists of blocks of information transmitted across the communication line. Data blocks are used to transfer data. Synchronization blocks are used between IMS/VS and the System/3 or System/7 stations to inform each other about the status of terminals, completion of output, restart, and shutdown.

If IMS/VS detects interface errors, it transmits an EOT to stop the System/3 or System/7, and sends a message to the master terminal. If the System/3 or System/7 is restarted before IMS/VS is shutdown, it is restarted in emergency restart status (refer to "Shutdown/Restart Blocks" under "Synchronization Blocks").

The System/3 or System/7 is logically deactivated if any of the following categories of errors occur:

* Transmission errors
* Invalid data or synchronization block formats
* Invalid station or terminal identifier
* Invalid data block flag settings

The System/7 will also be logically deactivated if a load sequence error occurs.

DATA BLOCKS

A data block contains one or more segments belonging to one or more messages. A segment is fully transmitted by IMS/VS in one transmission, unless its size exceeds the user-specified transmission buffer size, in which case it is changed into multiple segments of the following format.

Block Format

D	A	Block identifier	One or more data segments
0	1	2	6

The D and A identify the block as a data block. The field contains the two characters D and A in uppercase EBCDIC.

Block identifier specifies the block for restart purposes. When an input message is enqueued, IMS/VS logs the block identifier with the message. IMS/VS transmits the last logged block identifier back to the System/3 or System/7 after a restart of IMS/VS. The System/3 or System/7 can also request this information to be transmitted, thus allowing resynchronization after a previous restart.

It is recommended that the block identifier be changed between blocks. If the first block received after a restart has the same block identifier as was used to restart, the block is considered retransmitted. This is described in more detail under "Data Segment Format."

Note: In a future release, the block identifier may be required to change between blocks.

Data Segment Format

Terminal Identifier	Msg Ident.	Flags	Length	Data
0	2	3	4	6

- Terminal Identifier
  - Received by IMS/VS: This value must correspond to the address given in a TERMINAL macro specified in IMS/VS system definition for the transmitting System/3 or System/7; otherwise, the System/3 or System/7 is logically deactivated.
  - Transmitted from IMS/VS: The address given in the TERMINAL macro for the outputting terminal is used as terminal identifier.
- Message Identifier

  This identifies a message within a block for error message and restart purposes. Error messages are described under "Synchronization Blocks."

The message identifier is logged with the block identifier by IMS/VS. In case of a restart of IMS/VS or an emergency restart of System/3 or System/7, the message identifier (together with the block identifier describing the last message enqueued) is transmitted to the System/3 or System/7. The System/3 or System/7 can then retransmit the identified block. Retransmission is not required if the identified message was not followed by any segments, or if these segments can be built into the next block.

The first input data block after a restart is considered retransmitted if its block identifier is the same as the one used to restart. The received block is scanned to find the first segment following the identified message, if any, thereby bypassing all segments already enqueued, in case of a retransmission.

- Flags

Bit	Meaning
0-4	Reserved.
5	Segment spanning flag: 0=Segment ends in this buffer. 1=Segment does not end in this buffer.
6	0=First part of a message. 1=Not the first part of a message.
7	0=Last part of a message. 1=Not the last part of a message.

All combinations of flag bits 5, 6 and 7 are valid except X'04' and X'06'.

"Part" in the above flag meanings, emphasizes that a segment can be changed to multiple segments as previously defined, and as indicated by the spanning flag.

The setting of the flag bits must correspond to the definition of the transaction in IMS/VS system definition. A transaction defined as a single-segment transaction to IMS/VS must have all flag bits off. A transaction defined as a multiple-segment transaction, as well as all message switches, can, but are not required to, consist of multiple segments. A command must follow the rules for that command defined by the IMS/VS system.

The setting of flag bits must also be consistent during the flow of data; that is, one message must be terminated before the next can start, or the station is logically deactivated. The segment spanning flag is set by IMS/VS whenever a segment spanned an IMS/VS queue buffer, or could not be contained in one transmission buffer. The segment spanning flag is ignored when received by IMS/VS.

- Length

Specifies the combined length of the length and data fields. All the data defined by this length must be within the block. This field is 2-byte binary.

- Data

The format of the data must correspond to the standard IMS/VS data formats.

EXAMPLES OF DATA BLOCK FORMATS

System/3 or System/7 Transmission to IMS/VS

Four data blocks are shown in this example. Data came from three terminals:

- Terminal T1: One message consisting of segments 1, 2, and 3.
- Terminal T2: Two messages, one consisting of segments 6 and 7, the other of segment 8.
- Terminal T3: One message consisting of segments 4 and 5.

```
|D A|BLK 1|T1| 1| 1|Length|Segment 1|T1| 1| 3|Length|Segment 2 |

|D A|BLK 2|T1| 1| 2|Length|Segment 3|T3| 2| 1|Length|Segment 4 |

|D A|BLK 3|T3| 1| 2|Length|Segment 5|T2| 2| 1|Length|Segment 6 |

|D A|BLK 4|T2| 1| 2|Length|Segment 7|T2| 2| 0|Length|Segment 8 |
```

IMS/VS Transmission to System/3 or System/7

Eight blocks are shown in this example. Data is destined for four terminals:

- Terminal T1: One message consisting of segments 1, 2 and 3.
- Terminal T2: One message consisting of segment 4.
- Terminal T3: One message consisting of segments 5, 6 and 7, each of which requires spanning.
- Terminal T4: One message consisting of segment 8.

```
|D A|BLK 1|T1| 1| 1|Length|Segment 1|T1| 1| 3|Length|Segment 2 |

|D A|BLK 2|T1| 1| 2|Length|Segment 3|T2| 2| 0|Length|Segment 4 |

|D A|BLK 3|T3| 1| 5|Length|Segment 5 (spanned)                |

|D A|BLK 4|T3| 1| 3|Length|Segment 5                          |
```

```
┌──────────────────────────────────────────────────────────────┐
|D A|BLK 5|T3| 1| 7|Length|Segment 6 (spanned)                 |
└──────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────┐
|D A|BLK 6|T3| 1| 3|Length|Segment 6                           |
└──────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────┐
|D A|BLK 7|T3| 1| 7|Length|Segment 7 (spanned)                 |
└──────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────┐
|D A|BLK 8|T3| 1| 2|Length|Segment 7|T4| 2| 0|Length|Segment 8 |
└──────────────────────────────────────────────────────────────┘
```

SYNCHRONIZATION BLOCKS

Synchronization blocks are used to transmit non-data control information between IMS/VS and System/3 or System/7. Only the formats described are transmitted by IMS/VS. Any input format different from those described below is ignored if received by IMS/VS.

General Block Formats

- Format A Unblocked

```
┌────────────────────────────────┐
| S | Y |Type |Flags|   Data     |
└────────────────────────────────┘
 0       2     3     4
```

- Format B Blocked

```
┌──────────────────────────────────────────────────────┐
| S | Y |Type |Flags|   Data    |Type |Flags| Data     |
└──────────────────────────────────────────────────────┘
 0       2     3     4
```

S and Y identify the block as a synchronization block. The field contains the characters S and Y in uppercase EBCDIC.

Type identifies the type of information contained in the block.

Value (hex)	Block Format	Description
80	A	Shutdown/restart block.
40	B	Status change block.
20	B	I/O synchronization block.
10	A	Error message block.
01	A	Load request (System/7 only).

All other type values are reserved.

Flags and data are described in the detailed description of the above blocks.

Shutdown/Restart Blocks

Format 1

```
┌──────────────────┐
|S | Y |80 |Flags  |
└──────────────────┘
 0  1   2   3
```

Format 2

```
┌─────────────────────────────────────────────┐
|S |Y |80 |Flags|Block identifier|Msg ident  |
└─────────────────────────────────────────────┘
 0  1  2   3     4                8
```

- Flags

Value	Meaning
X'80'	Cold start (format 1).
X'40'	Emergency restart (format discussed below).
X'20'	Emergency restart response (format 2).
X'10'	Normal restart (format 2).
X'08'	Shutdown request (format 1).
X'02'	System shutdown (format 1).
X'01'	Immediate shutdown request (format 1).

All other flag values are reserved.

Block identifier identifies the last received block causing a message to be queued.

Message identifier identifies the last message within the block to be queued.

Restart Messages: The restart message is sent by IMS/VS to a System/3 or System/7 when:

- Communication is started due to IMS/VS receiving a /START command with the line or pterm keywords, where the station pterm (the System/3 or System/7) was not explicitly stopped by a previous command. A station stopped condition is reset by including the station pterm in the /START command.

- Requested by the System/3 or System/7.

The restart message indicates either how IMS/VS was started or how previous communication was terminated.

- IMS/VS Cold Started

  When IMS/VS transmits the cold start message to the System/3 or System/7, the message indicates that IMS/VS was started with empty queues. The System/3 or System/7 must start its transmission with the first segment of a message; otherwise, the System/3 or System/7 is logically deactivated and the master terminal operator notified. If the System/3 or System/7 is reactivated before IMS/VS has been terminated, it is activated in an emergency restart status.

- IMS/VS Receives a Cold Start Message

  When IMS/VS receives a cold start message, any input message in progress is canceled. All output messages in progress are restarted from the first segment. The rules for System/3 or System/7 for starting data transmission apply as above.

- IMS/VS Emergency Restarted

  IMS/VS transmits an emergency restart message in format 2. The System/3 or System/7 has two options:

  1. It may retransmit the block identified in the restart message. IMS/VS starts processing with the first segment following the last segment of the identified message.

  2. If the System/3 or System/7 does not wish to retransmit the identified block, it can build the remaining segments in the block, if any, into some other block, and use a block identifier other than the one used to restart, in the first block transmitted.

- IMS/VS-Received Emergency Restart Message in Format 1

  An input message, if one is in progress, is canceled. All output messages in progress are retransmitted beginning with the first segment. IMS/VS responds by transmitting an emergency restart response message. The same emergency restart rules as above apply for starting communication.

- Normal Restart Message

  IMS/VS transmits the normal restart message to start communication if no other restart message is required. IMS/VS ignores a received normal restart message.

Shutdown Messages: Shutdown messages inform the receiving station that the transmitting station has started a procedure designed to terminate communication between the two stations in an orderly fashion. This is sent under the following conditions:

- Communication was terminated because IMS/VS received a /STOP, /PSTOP, or /PURGE command with the line or pterm keywords.

- Communication was terminated because IMS/VS received a /CHECKPOINT command for the system.

- Communication was terminated because of an error condition.

- Communication was terminated by request of the System/3 or System/7.

The types of shutdown messages are:

- Immediate Shutdown Request (from IMS/VS only)

  The IMS/VS master terminal operator has requested IMS/VS to terminate communication either by stopping the System/3 or System/7 or by requesting an IMS/VS shutdown procedure. This block requests System/3 or System/7 to stop transmitting data to IMS/VS when all messages in progress are completed.

  The System/3 or System/7 must inform IMS/VS of completion of messages received from IMS/VS, even though a shutdown is in progress. The master terminal operator may have requested IMS/VS to purge its queues before shutting down; hence, IMS/VS can continue transmitting data even though a shutdown is in progress. IMS/VS sends a system-shutdown message to inform the remote station when the shutdown procedure has been completed.

- Shutdown Request (to IMS/VS only)

  IMS/VS does not initiate transmission of a new output message after receipt of a shutdown request. IMS/VS transmits the system-shutdown message when all outstanding messages have been acknowledged by the System/3 or System/7 as being completed after all appropriate output has been sent.

- System Shutdown (from IMS/VS only)

  IMS/VS transmits this message to inform the System/3 or System/7 that communication is terminated normally.

Status Change Blocks

Status change blocks are used to specify a change in transmission mode between IMS/VS and a System/3 or System/7. Status change blocks may be sent as a result of using the line or pterm keywords with the following commands: /START, /STOP, /RSTART, /PSTOP, /PURGE, and /MONITOR.

S	Y	40	Flags	Terminal Identifier	40	Flags	Terminal Identifier	40	Flags	Terminal Identifier
0	1	2	3	4	6	7	8	10	11	12

- Flags

Value	Meaning
X'80'	Unable to operate with terminal (to IMS/VS only).
X'40'	Stop input from and output to terminal.
X'20'	Stop input from and start output to terminal.
X'10'	Start input from and output to terminal.
X'08'	Start input from and stop output to terminal.

  All other flag values are reserved.

  Terminal identifier specifies the status changing terminal.

The flag descriptions are as follows:

Value	Action
X'80'	The identified terminal is marked inoperable by IMS/VS and the master terminal operator is notified. Any input in progress on the specified terminal is cancelled. Any output in progress is postponed and will be retransmitted from the first segment when the terminal is restarted.
X'40'	Input and output are logically stopped, except system messages, which continue to be transmitted. A message in progress, in or out, is allowed to complete. Any input message received later is rejected, and an error message returned to the remote station. No output is initiated except system messages.
X'20'	Input is logically stopped while output is allowed to continue normally, or is started if required. An input message in progress is allowed to complete, but any later message is rejected, and an error message returned to the remote station.

X'10'	Input and output are logically restarted. Normal input and output are resumed.
X'08'	Input is allowed to continue normally or, if required, is started. Output is logically stopped. An output message in progress is allowed to complete.

I/O Synchronization Blocks

I/O synchronization blocks are used to allow the System/3 or System/7 and IMS/VS to synchronize I/O operations and maintain system integrity. I/O synchronization blocks also allow the System/3 and System/7 to optimize their resources by controlling when and what output is sent by IMS/VS.

S	Y	20	Flags	Terminal Identifier	20	Flags	Terminal Identifier	20	Flags	Terminal Identifier
0	1	2	3	4	6	7	8	10	11	12

- Flags

Value	Meaning
X'80'	Output completed (sent by System/3 or System/7).
X'40'	Input in progress (sent by System/3 or System/7).
X'20'	Input terminated (sent by System/3 or System/7).
X'10'	Send output (sent by System/3 or System/7; ASK message).
X'08'	No output available (sent by IMS/VS; NO-OUT message).
X'04'	Postpone output (sent by System/3 or System/7).
X'02'	Resume output (sent by System/3 or System/7).

All other flag values are reserved.

Terminal Identifier specifies the affected terminal, or is binary zeros (see flag values X'04' and X'02' below); the terminal identifier field must always be present, but is not verified for flag values X'10' and X'08'.

IMS/VS does not transmit I/O synchronization segments except for the NO-OUT message; it ignores a received NO-OUT message.

The flag descriptions are as follows:

Value	Action
X'80'	IMS/VS verifies that the identified terminal has an output message in progress. If so, the message is removed from the IMS/VS queue; otherwise, the segment is ignored.

X'40' This flag informs IMS/VS that the System/3 or System/7 is reading from the specified terminals but the first segment has not yet been sent to IMS/VS. IMS/VS stops sending output to the specified terminal until a full input message has been received from the System/3 or System/7 for the specified terminal. If an output message to the terminal was in progress when this block was received, it will be retransmitted later, beginning with the first segment. The segment is ignored if an input message from the terminal is in progress when the block is received.

X'20' This flag can be used to allow output to resume if it was stopped using the input-in-progress flag (X'40' above), and the System/3 or System/7 does not wish to send any data to IMS/VS.

X'10' This message is referred to as the "ASK" message. It is used by a System/3 or System/7 to reset the transmission limit counter if transmission limit was defined in IMS/VS system definition for the station. It is also used to ask for output to a station defined as "ASK" type in IMS/VS system definition. (See X'08' below.)

X'08' This message is sent by IMS/VS to respond to a request for output (value X'10') when no more output is available, if the System/3 or System/7 is defined in IMS/VS system definition as "ASK" type, unless transmission was terminated by a reached transmission limit.

X'04' Terminal identifier equals binary zeros: Postpone initiation of data messages to the System/3 or System/7 transmitting the request. Messages in progress are completed.

Terminal identifier does not equal binary zeros: Postpone initiation of data messages to the identified terminal. Any message in progress is completed.

Output initiation is resumed when IMS/VS receives an I/O synchronization message with the flag value X'02'.

X'02' Resume output initiation postponed by use of the above flag value (X'04').

A terminal identifier of binary zeros causes IMS/VS to resume output initiation to all terminals attached to the System/3 or System/7 transmitting the request.

A terminal identifier other than binary zeros causes IMS/VS to resume output initiation only to the identified terminal.

Error Blocks

Error blocks allow IMS/VS and the System/3 or System/7 to inform each other of errors pertaining to received data.

The error block format is as follows:

S	Y	10	Flags	Terminal Identifier	Msg Ident	Error Code
0	1	2	3	4	6	7

- Flags

Value	Meaning
X'00'	Error occurred on last block transmitted.
X'01'	Error occurred on previous block transmitted.
X'80'	Error message on last block is from user message table.
X'81'	Error message on previous block is from user message table.

All other bit settings are reserved.

The terminal identifier and message identifier are from the segment in error.

The error code is any four-character number in numeric-character notation when sent to or received from IMS/VS.

Error Message Sent by IMS/VS: An error block is sent whenever an error results while IMS/VS is processing an input segment. The message identifier from the segment causing the error message to be generated is added to the error message before transmitting it to the remote station. IMS/VS also reverts all involved resources to a first-segment status, causing all remaining segments of the message in error to be flushed.

IMS/VS causes a reverse interrupt (RVI) sequence to be transmitted if an error message was generated. IMS/VS then accepts one additional input block after transmitting RVI. An attempt to transmit more than one block results in a transmission error and the station is logically deactivated. The flags allow the remote station to determine in which block a given error was found.

Error Message Received by IMS/VS: An error message is accepted by IMS/VS if IMS/VS has transmitted a message to the System/3 or System/7 that has not yet been dequeued by a corresponding I/O synchronization block (output complete) received from the System/3 or System/7, or postponed because of an error or received input.

- Error message acceptable

  The logical terminal (CNT) from which the message causing the error was read is stopped. A message destined for the IMS/VS master terminal is generated. This message includes the name of the stopped CNT and the error code received from the remote station.

- Error message not acceptable

  The transmitting remote station is logically deactivated. The master terminal operator is notified. If the System/3 or System/7 is reactivated before IMS/VS has been shutdown, it is activated in an emergency restart status.

System/7 Load Request Block

A System/7 on a polled line can send IMS/VS a load request block to request that a load or IPL sequence be performed.

S	Y	01	Flags	Load Module Name

0 1 2 3 4

- Flags

Bit	Meaning
0	0=IMS/VS transmits only the load module. 1=IMS/VS transmits $UBIPL, followed by the load module, followed by an emergency restart block.

All other flag values are reserved.

Load module name is the name of a member in a PDS specified by the S7BSCLIB DD statement in the IMS procedure. The member must have been placed in the PDS using Format/7 (specifying 'CARD' output format), or other equivalent product producing the same format. IMS/VS reads the load module from the PDS and transmits the load module to the System/7.

INTERFACE BETWEEN IMS/VS AND A SYSTEM/7 START/STOP

The interface between IMS/VS and a System/7 consists of blocks of information transmitted across the communication line. Data blocks are used to transfer data. Synchronization blocks are used between IMS/VS and the System/7 stations to inform each other about the status of terminals, completion of output, restart, and shutdown. These blocks must be translated from transmission code to EBCDIC when received, and from EBCDIC to transmission code before being transmitted.

If IMS/VS detects interface errors, it transmits an EOT to stop the System/7, and sends a message to the master terminal. If the System/7 is restarted before IMS/VS is shut down, it is restarted in emergency restart status (refer to "Shutdown/Restart Blocks" under "Synchronization Blocks").

The System/7 is logically deactivated if any of the following categories of errors occur:

- Transmission errors
- Invalid data or synchronization block formats
- Transmission code/EBCDIC translation errors
- Invalid station or terminal identifier
- Invalid data block flag settings
- Load sequence errors

The System/7 may be logically deactivated due to its relatively short timeout cycle of 16.5 seconds. A timeout may first occur at the remote station and then IMS/VS if the system is so loaded that IMS/VS cannot process an input line buffer and respond to the station in a timely manner.

DATA BLOCKS

A data block contains one or more segments belonging to one or more messages. A segment is fully transmitted by IMS/VS in one transmission, unless its size exceeds the user-specified transmission buffer size, in which case it is changed into multiple segments of the following format.

Block Format

D	A	Block identifier	One or more data segments
0	1	2	6

The D and A identify the block as a data block. The field contains the two characters D and A in uppercase EBCDIC.

Block identifier specifies the block for restart purposes. When an input message is enqueued, IMS/VS logs the block identifier with the message. IMS/VS transmits the last logged block identifier back to the System/7 after a restart of IMS/VS. The System/7 can also request this information to be transmitted, thus allowing resynchronization after a previous restart.

Data blocks may be transmitted in PTTC/EBCD code or pseudobinary PTTC/EBCD code. Care must be taken to ensure that a transmitted character does not conflict with a line control character. All identifiers used must give the same result in EBCDIC regardless of transmission code.

It is recommended that the block identifier be changed between blocks. If the first block received after a restart has the same block identifier as was used to restart, the block is considered retransmitted. This is described in more detail under "Data Segment Format."

Note: In a future release, the block identifier may be required to change between blocks.

Data Segment Format

Terminal Identifier	Msg Ident.	Flags	Length	Data
0	2	3	4	8

- Terminal Identifier
  - Received by IMS/VS: This value must correspond to the address given in a TERMINAL macro specified in IMS/VS system definition for the transmitting System/7; otherwise, the System/7 is logically deactivated.
  - Transmitted from IMS/VS: The address given in the TERMINAL macro for the outputting terminal is used as terminal identifier.

- Message Identifier

  This identifies a message within a block for error message and restart purposes. Error messages are described under "Synchronization Blocks."

  The message identifier is logged with the block identifier by IMS/VS. In case of a restart of IMS/VS or an emergency restart of the System/7, the message identifier (together with the block identifier describing the last message enqueued) is transmitted to the System/7. The System/7 can then retransmit the identified block. Retransmission is not required if the identified message was not followed by any segments, or if these segments can be built into the next block.

  The first input data block after a restart is considered retransmitted if its block identifier is the same as the one used to restart. The received block is scanned to find the first segment following the identified message, if any, thereby bypassing all segments already enqueued, in case of a retransmission.

- Flags

Bit	Meaning
0-3	Must be all ones.
4	Reserved (should be zero).
5	Segment spanning flag: 0=Segment ends in this buffer. 1=Segment does not end in this buffer.
6	0=First part of a message. 1=Nonfirst part of a message.
7	0=Last part of a message. 1=Nonlast part of a message.

  All combinations of flag bits 5, 6, and 7 are valid except X'04' and X'06'.

  "Part" in the above flag meanings, emphasizes that a segment can be changed to multiple segments as previously defined, and as indicated by the spanning flag.

  The setting of the flag bits must correspond to the definition of the transaction in IMS/VS system definition. A transaction defined as a single-segment transaction to IMS/VS must have flag bits 4-7 off. A transaction defined as a multiple-segment transaction, as well as all message switches, can, but are not required to, consist of multiple segments. A command must follow the rules for that command defined by the IMS/VS system.

  The setting of flag bits must also be consistent during the flow of data; that is, one message must be terminated before the next can start, or the station is logically deactivated. The segment spanning flag is set by IMS/VS whenever a segment spanned an IMS/VS queue buffer, or could not be contained in one transmission buffer. The segment spanning flag is ignored when received by IMS/VS.

- Length

  Specifies the combined length of the length and data fields. All the data defined by this length must be within the block. This field is 4-byte EBCDIC hexadecimal notation. This format is chosen to avoid conflicts with line control characters. For example, if a segment is 108 bytes this length would, in EBCDIC hexadecimal, be '006C'.

- Data

  The format of the data must correspond to the standard IMS/VS data formats.

EXAMPLES OF DATA BLOCK FORMATS

System/7 Transmission to IMS/VS

Four data blocks are shown in this example. Data came from three terminals:

- Terminal T1: One message consisting of segments 1, 2, and 3.
- Terminal T2: Two messages, one consisting of segments 6 and 7, the other of segment 8.
- Terminal T3: One message consisting of segments 4 and 5.

```
|D A|BLK 1|T1| 1| 1|Length|Segment 1|T1| 1| 3|Length|Segment 2 |
```

```
|D A|BLK 2|T1| 1| 2|Length|Segment 3|T3| 2| 1|Length|Segment 4 |
```

```
|D A|BLK 3|T3| 1| 2|Length|Segment 5|T2| 2| 1|Length|Segment 6 |
```

```
|D A|BLK 4|T2| 1| 2|Length|Segment 7|T2| 2| 0|Length|Segment 8 |
```

IMS/VS Transmission to System/7

Eight blocks are shown in this example. Data is destined for four terminals:

- Terminal T1: One message consisting of segments 1, 2 and 3.
- Terminal T2: One message consisting of segment 4.
- Terminal T3: One message consisting of segments 5, 6 and 7, each of which requires spanning.
- Terminal T4: One message consisting of segment 8.

```
┌---------------------------------------------------------------┐
|D A|BLK 1|T1| 1| 1|Length|Segment 1|T1| 1| 3|Length|Segment 2 |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 2|T1| 1| 2|Length|Segment 3|T2| 2| 0|Length|Segment 4 |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 3|T3| 1| 5|Length|Segment 5 (spanned)                  |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 4|T3| 1| 3|Length|Segment 5                            |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 5|T3| 1| 7|Length|Segment 6 (spanned)                  |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 6|T3| 1| 3|Length|Segment 6                            |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 7|T3| 1| 7|Length|Segment 7 (spanned)                  |
└---------------------------------------------------------------┘

┌---------------------------------------------------------------┐
|D A|BLK 8|T3| 1| 2|Length|Segment 7|T4| 2| 0|Length|Segment 8 |
└---------------------------------------------------------------┘
```

SYNCHRONIZATION BLOCKS

Synchronization blocks are used to transmit non-data control information between IMS/VS and System/7. Only the formats described are transmitted by IMS/VS. Any input format different from those described below is ignored if received by IMS/VS. System/7 synchronization blocks must be transmitted in pseudobinary PTTC/EBCD code.

General Block Formats

- Format A Unblocked

```
┌────────────────────────────────┐
| S | Y |Type |Flags|    Data     |
└────────────────────────────────┘
 0        2      3     4
```

- Format B Blocked

```
┌──────────────────────────────────────────────────────┐
| S | Y |Type |Flags|    Data    |Type |Flags|  Data   |
└──────────────────────────────────────────────────────┘
 0        2      3     4
```

S and Y identify the block as a synchronization block. The field contains the characters S and Y in uppercase EBCDIC.

Type identifies the type of information contained in the block.

Value (hex)	Block Format	Description
80	A	Shutdown/restart block.
40	B	Status change block.
20	B	I/O synchronization block.
10	A	Error message block.
01	A	Load request.

All other type values are reserved.

Flags and data are described in the detailed description of the above blocks.

Shutdown/Restart Blocks

Format 1

```
┌───────────────────┐
|S | Y |80 |Flags   |
└───────────────────┘
 0   1   2   3
```

Format 2

```
┌────────────────────────────────────────────────────┐
|S |Y |80 |Flags|Block identifier|Msg ident |
└────────────────────────────────────────────────────┘
 0   1  2    3      4                  8
```

- Flags

Value	Meaning
X'80'	Cold start (format 1).
X'40'	Emergency restart (format discussed below).
X'20'	Emergency restart response (format 2).
X'10'	Normal restart (format 2).
X'08'	Shutdown request (format 1).
X'02'	System shutdown (format 1).
X'01'	Immediate shutdown request (format 1).

All other flag values are reserved.

Block identifier identifies the last received block causing a message to be queued.

Message identifier identifies the last message within the block to be queued.

Restart Messages: The restart message is sent by IMS/VS to a System/7 when:

- Communication is started due to IMS/VS receiving a /START command with the line or pterm keywords, where the station pterm (the System/7) was not explicitly stopped by a previous command. A station stopped condition is reset by including the station pterm in the /START command.

- Requested by the System/7.

The restart message indicates either how IMS/VS was started or how previous communication was terminated.

- IMS/VS Cold Started

  When IMS/VS transmits the cold start message to the System/7, the message indicates that IMS/VS was started with empty queues. The System/7 must start its transmission with the first segment of a message; otherwise, the System/7 is logically deactivated and the master terminal operator notified. If the System/7 is reactivated before IMS/VS has been terminated, it is activated in an emergency restart status.

- IMS/VS Receives a Cold Start Message

  When IMS/VS receives a cold start message, any input message in progress is canceled. All output messages in progress are restarted from the first segment. The rules for System/7 for starting data transmission apply as above.

- IMS/VS Emergency Restarted

  IMS/VS transmits an emergency restart message in format 2. The System/7 has two options:

  1. It may retransmit the block identified in the restart message. IMS/VS starts processing with the first segment following the last segment of the identified message.

  2. If the System/7 does not wish to retransmit the identified block, it can build the remaining segments in the block, if any, into some other block, and use a block identifier other than the one used to restart, in the first block transmitted.

- IMS/VS-Received Emergency Restart Message in Format 1

  An input message, if one is in progress, is canceled. All output messages in progress are retransmitted beginning with the first segment. IMS/VS responds by transmitting an emergency restart response message. The same emergency restart rules as above apply for starting communication.

- Normal Restart Message

  IMS/VS transmits the normal restart message to start communication if no other restart message is required. IMS/VS ignores a received normal restart message.

Shutdown Messages: Shutdown messages inform the receiving station that the transmitting station has started a procedure designed to terminate communication between the two stations in an orderly fashion. This is sent under the following conditions:

- Communication was terminated due to IMS/VS receiving a /STOP, /PSTOP or /PURGE command with the line or pterm keywords.
- Communication was terminated due to IMS/VS receiving a /CHECKPOINT command for the system.
- Communication was terminated due to an error condition.
- Communication was terminated by request of the System/7.

The types of shutdown messages are:

- Immediate Shutdown Request (from IMS/VS only)

  The IMS/VS master terminal operator has requested IMS/VS to terminate communication either by stopping the System/7 or by requesting an IMS/VS shutdown procedure. This block requests System/7 to stop transmitting data to IMS/VS when all messages in progress are completed.

  The System/7 must inform IMS/VS of completion of messages received from IMS/VS, even though a shutdown is in progress. The master terminal operator may have requested IMS/VS to purge its queues before shutting down; hence, IMS/VS can continue transmitting data even though a shutdown is in progress. IMS/VS sends a system-shutdown message to inform the remote station when the shutdown procedure has been completed.

- Shutdown Request (to IMS/VS only)

  IMS/VS does not initiate transmission of a new output message after receipt of a shutdown request. IMS/VS transmits the system-shutdown message when all outstanding messages have been acknowledged by the System/7 as being completed after all appropriate output has been sent.

- System Shutdown (from IMS/VS only)

  IMS/VS transmits this message to inform the System/7 that communication is terminated normally.

Status Change Blocks

Status change blocks are used to specify a change in transmission mode between IMS/VS and a System/7. Status change blocks may be sent as a result of using the line or pterm keywords with the following commands: /START, /STOP, /RSTART, /PSTOP, /PURGE, and /MONITOR.

S	Y	40	Flags	Terminal Identifier	40	Flags	Terminal Identifier	40	Flags	Terminal Identifier
0	1	2	3	4	6	7	8	10	11	12

- Flags

Value	Meaning
X'80'	Unable to operate with terminal (to IMS/VS only).
X'40'	Stop input from and output to terminal.
X'20'	Stop input from and start output to terminal.
X'10'	Start input from and output to terminal.
X'08'	Start input from and stop output to terminal.

  All other flag values are reserved.

  Terminal identifier specifies the status changing terminal.

The flag descriptions are as follows:

Value	Action
X'80'	The identified terminal is marked inoperable by IMS/VS and the master terminal operator is notified. Any input in progress on the specified terminal is canceled. Any output in progress is postponed and will be retransmitted from the first segment when the terminal is restarted.
X'40'	Input and output are logically stopped, except system messages, which continue to be transmitted. A message in progress, in or out, is allowed to complete. Any input message received later is rejected, and an error message returned to the remote station. No output is initiated except system messages.
X'20'	Input is logically stopped while output is allowed to continue normally, or is started if required. An input message in progress is allowed to complete, but any later message is rejected, and an error message returned to the remote station.
X'10'	Input and output are logically restarted. Normal input and output are resumed.
X'08'	Input is allowed to continue normally or, if required, is started. Output is logically stopped. An output message in progress is allowed to complete.

I/O Synchronization Blocks

I/O synchronization blocks are used to allow the System/7 and IMS/VS to synchronize I/O operations and maintain system integrity. I/O synchronization blocks also allow the System/7 to optimize their resources by controlling when and what output is sent by IMS/VS.

S	Y	20	Flags	Terminal Identifier	20	Flags	Terminal Identifier	20	Flags	Terminal Identifier
0	1	2	3	4	6	7	8	10	11	12

- Flags

Value	Meaning
X'80'	Output completed (sent by System/7).
X'40'	Input in progress (sent by System/7).
X'20'	Input terminated (sent by System/7).
X'10'	Send output (sent by System/7; ASK message).
X'08'	No output available (sent by IMS/VS; NO-OUT message).
X'04'	Postpone output (sent by System/7).
X'02'	Resume output (sent by System/7).

  All other flag values are reserved.

  Terminal identifier specifies the affected terminal, or is binary zeros (see flag values X'04' and X'02' below); the terminal identifier field must always be present, but is not verified for flag values X'10' and X'08'.

IMS/VS does not transmit I/O synchronization segments except for the NO-OUT message; it ignores a received NO-OUT message.

The flag descriptions are as follows:

Value	Action
X'80'	IMS/VS verifies that the identified terminal has an output message in progress. If so, the message is removed from the IMS/VS queue; otherwise, the segment is ignored.
X'40'	This flag informs IMS/VS that the System/7 is reading from the specified terminals but the first segment has not yet been sent to IMS/VS. IMS/VS stops sending output to the specified terminal until a full input message has been received from the System/7 for the specified terminal. If an output message to the terminal was in progress when this block was received, it will be retransmitted later, beginning with the first segment. The segment is ignored if an input message from the terminal is in progress when the block is received.
X'20'	This flag can be used to allow output to resume if it was stopped using the input-in-progress flag (X'40' above), and the System/7 does not wish to send any data to IMS/VS.

X'10'	This message is referred to as the "ASK" message. It is used by a System/7 to reset the transmission limit counter if transmission limit was defined in IMS/VS system definition for the station. It is also used to ask for output to a station defined as "ASK" type in IMS/VS system definition. (See X'08' below.)

X'08'	This message is sent by IMS/VS to respond to a request for output (value X'10') when no more output is available, if the System/7 is defined in IMS/VS system definition as "ASK" type, unless transmission was terminated by a reached transmission limit.

X'04'	Terminal identifier equals binary zeros: Postpone initiation of data messages to the System/7 transmitting the request. Messages in progress are completed.

Terminal identifier does not equal binary zeros: Postpone initiation of data messages to the identified terminal. Any message in progress is completed.

Output initiation is resumed when IMS/VS receives an I/O synchronization message with the flag value X'02'.

X'02'	Resume output initiation postponed by use of the above flag value (X'04').

A terminal identifier of binary zeros causes IMS/VS to resume output initiation to all terminals attached to the System/7 transmitting the request.

A terminal identifier other than binary zeros causes IMS/VS to resume output initiation only to the identified terminal.

Error Blocks

Error blocks allow IMS/VS and the System/7 to inform each other of errors pertaining to received data.

The error block format is as follows:

S	Y	10	Flags	Terminal Identifier	Msg Ident	Error Code
0	1	2	3	4	6	7

- Flags

Value	Meaning
X'00'	IMS/VS error message.
X'80'	Error message from user message table.

All other bit settings are reserved.

The terminal identifier and message identifier are from the segment in error.

The error code is any four-character number in numeric-character notation when sent to or received from IMS/VS.

Error Message Sent by IMS/VS: An error block is sent whenever an error results while IMS/VS is processing an input segment. The message identifier from the segment causing the error message to be generated is added to the error message before transmitting it to the remote station. IMS/VS also reverts all involved resources to a first-segment status, causing all remaining segments of the message in error to be flushed.

Error Message Received by IMS/VS: An error message is accepted by IMS/VS if IMS/VS has transmitted a message to the System/7 that has not yet been dequeued by a corresponding I/O synchronization block (output complete) received from the System/7, or postponed because of an error or received input.

- Error message acceptable

  The logical terminal (CNT) from which the message causing the error was read is stopped. A message destined for the IMS/VS master terminal is generated. This message includes the name of the stopped CNT and the error code received from the remote station.

- Error message not acceptable

  The transmitting remote station is logically deactivated. The master terminal operator is notified. If the System/7 is reactivated before IMS/VS has been shut down, it is activated in an emergency restart status.

Load Request Block

A System/7 on a polled line can send IMS/VS a load request block to request that a load or IPL sequence be performed.

S	Y	01	Flags	Load Module Name
0	1	2	3	4

- Flags

Bit	Meaning
0	0=IMS/VS transmits only the load module. 1=IMS/VS transmits UZERO and UTIPL, followed by the load module, followed by an emergency restart block.

  All other flag values are reserved.

  Load module name is the name of a member in a PDS specified by the S7LODLIB DD statement in the IMS procedure. The member must have been placed in the PDS using Format/7, or other equivalent product producing the same format. IMS/VS reads the lcad module from the PDS, translates the load module to line code, and transmits the load module to the System/7.

IMS/VS RESPONSES TO RECEIVED BLOCKS

IMS/VS normally responds to a received block with a circle Y, inviting the System/7 to transmit another block.

IMS/VS responds with a circle D under the following conditions:

- A logical error is detected in a received data block.
- A command completed message must be sent.
- A test message must be returned.
- IMS/VS has to transmit a shutdown-request message.

IMS/VS responds with a circle C when an unrecoverable error is detected. Some unrecoverable errors are permanent transmission error, undefined terminal identifier in a segment, and invalid flag sequence in data blocks. The IMS/VS master terminal operator is informed about the problem cause. The IMS/VS master terminal operator must enter a /START command to inform IMS/VS to resume communication with the affected System/7.

SAMPLE IRSS TRANSMISSION SEQUENCES

Figure 6-1 on the following pages contains sample transmission sequences between IMS/VS and an intelligent remote station (System/3 or System/7). The figure assumes the remote station was defined to IMS/VS as ASK-TYPE with a transmission limit either not specified or equal to 2 (both cases shown).

Specific differences between System/3, System/7 BSC, and System/7 S/S are not shown but are defined in the appropriate preceding sections; for example, an RVI precedes an error block sequence for System/3 and System/7 BSC versus a circle D for System/7 S/S.

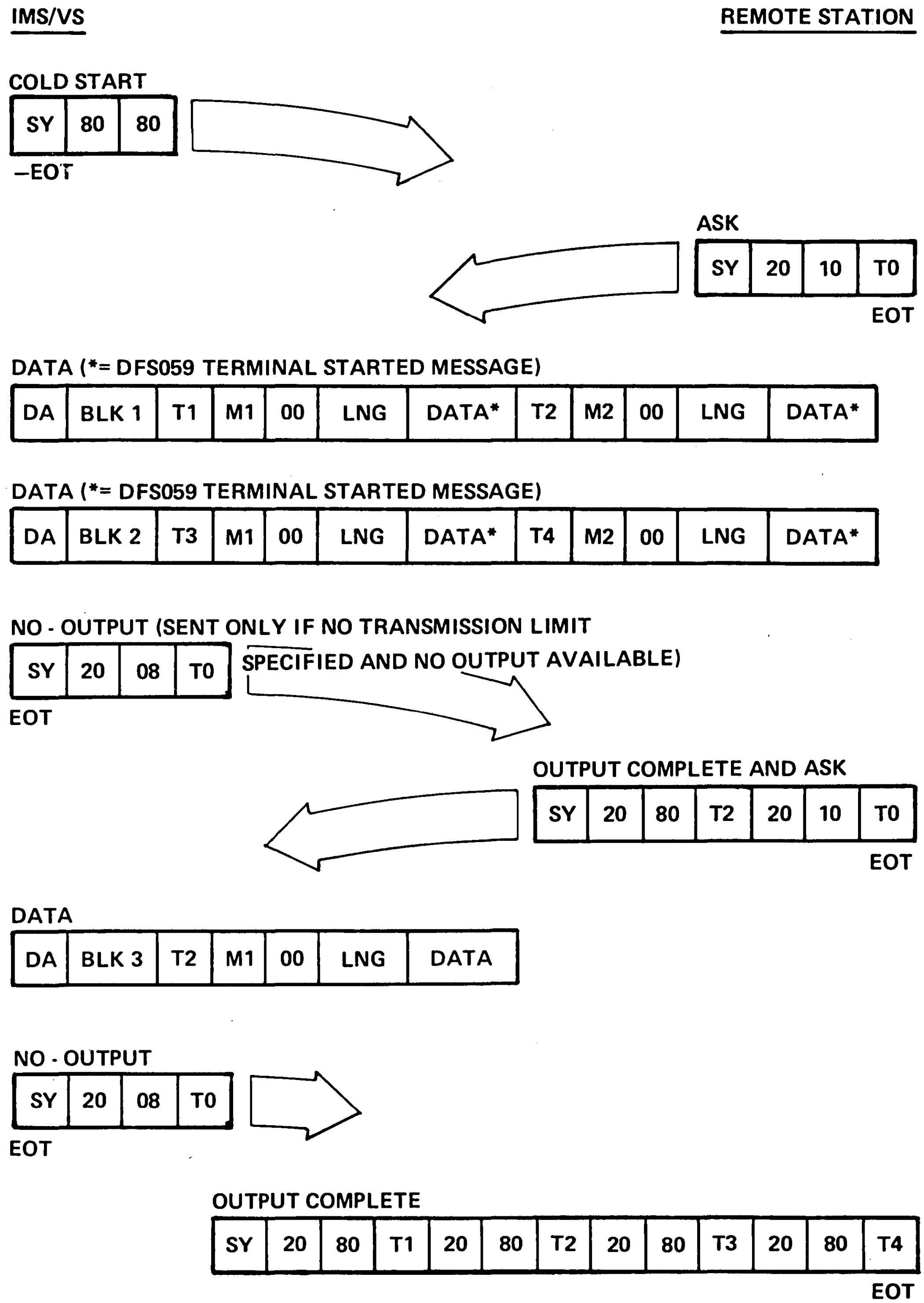


Figure 6-1 (Part 1 of 4). Sample IRSS Transmission Sequences

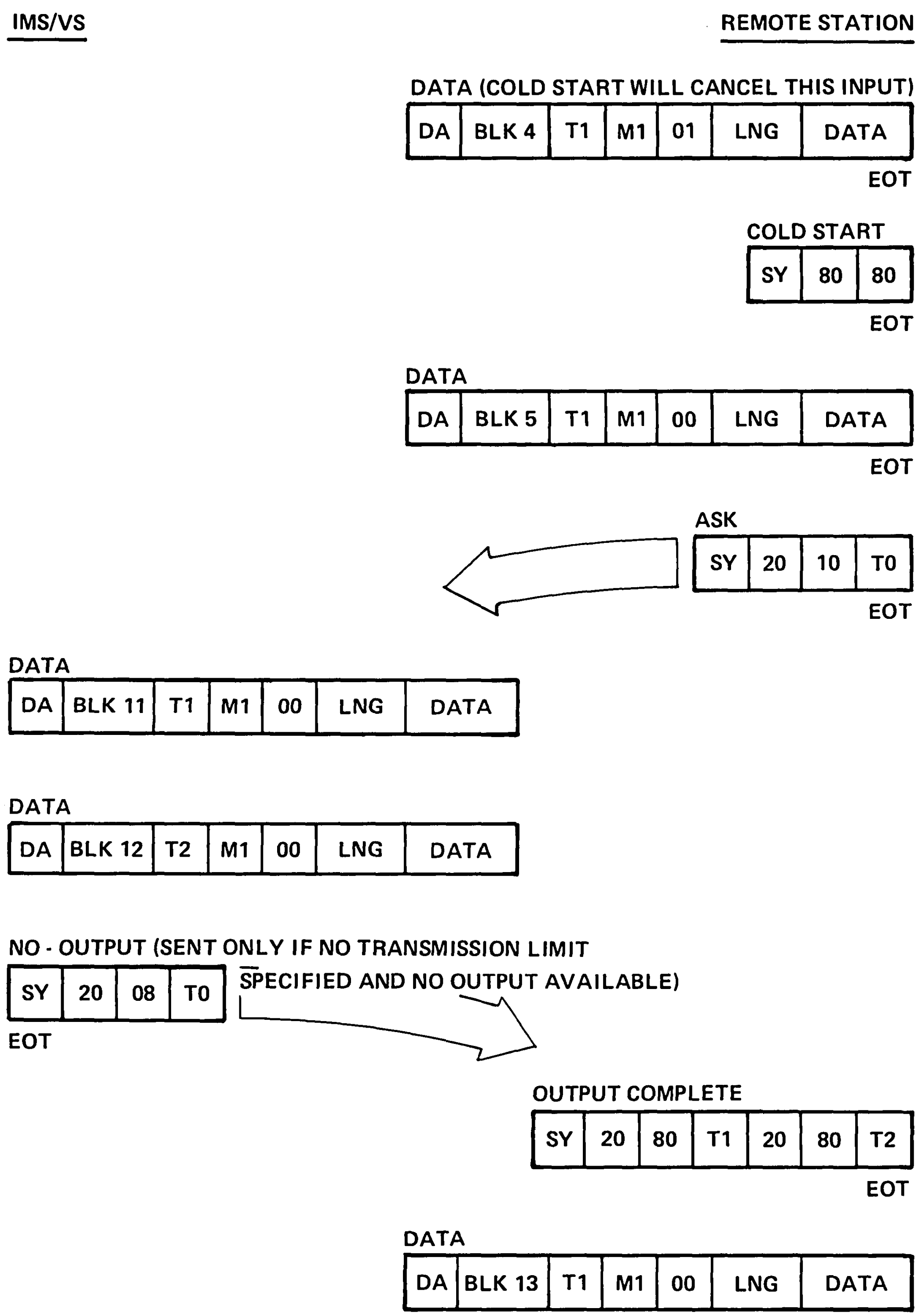


Figure 6-1 (Part 2 of 4). Sample IRSS Transmission Sequences

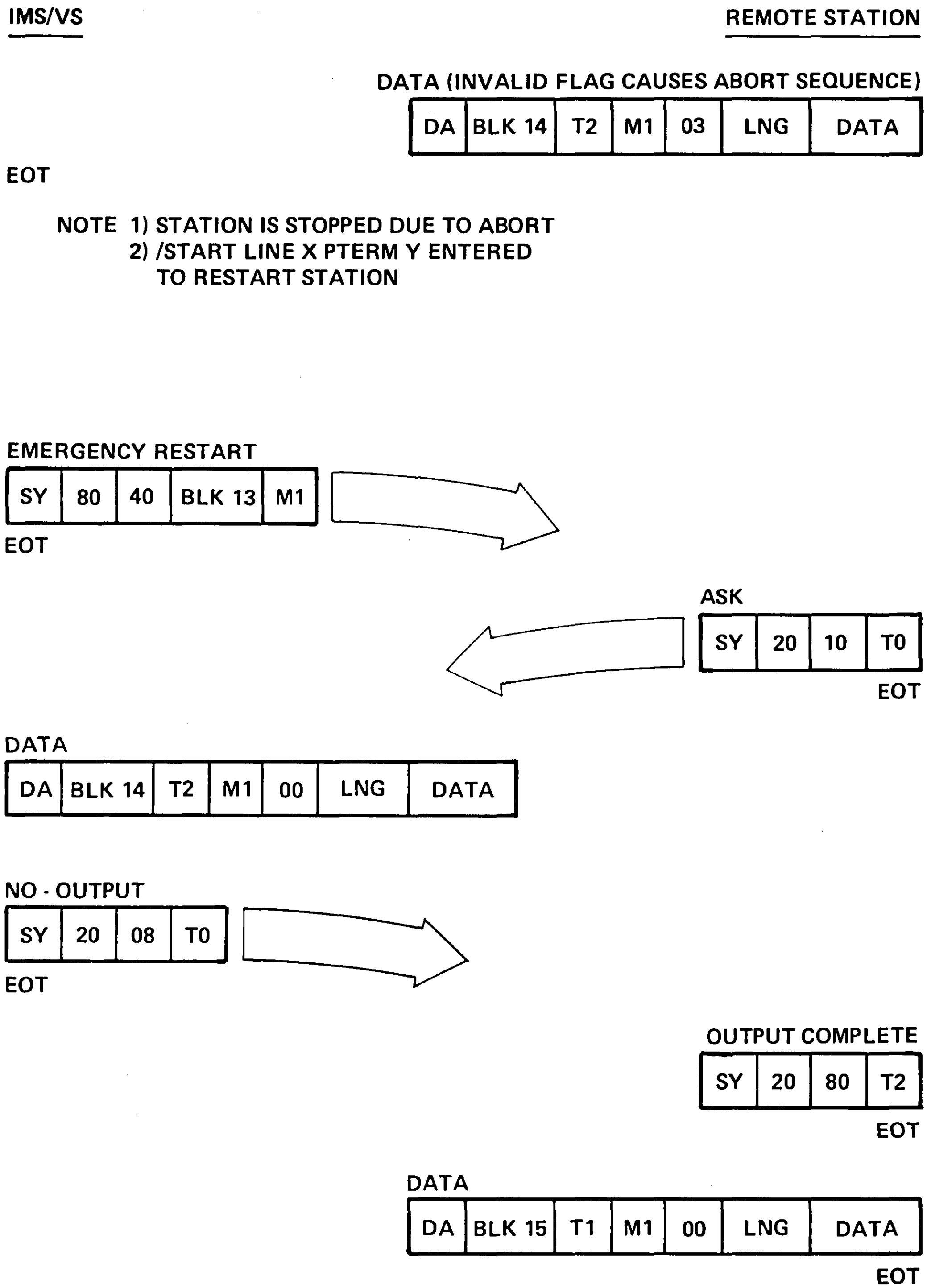


Figure 6-1 (Part 3 of 4). Sample IRSS Transmission Sequences

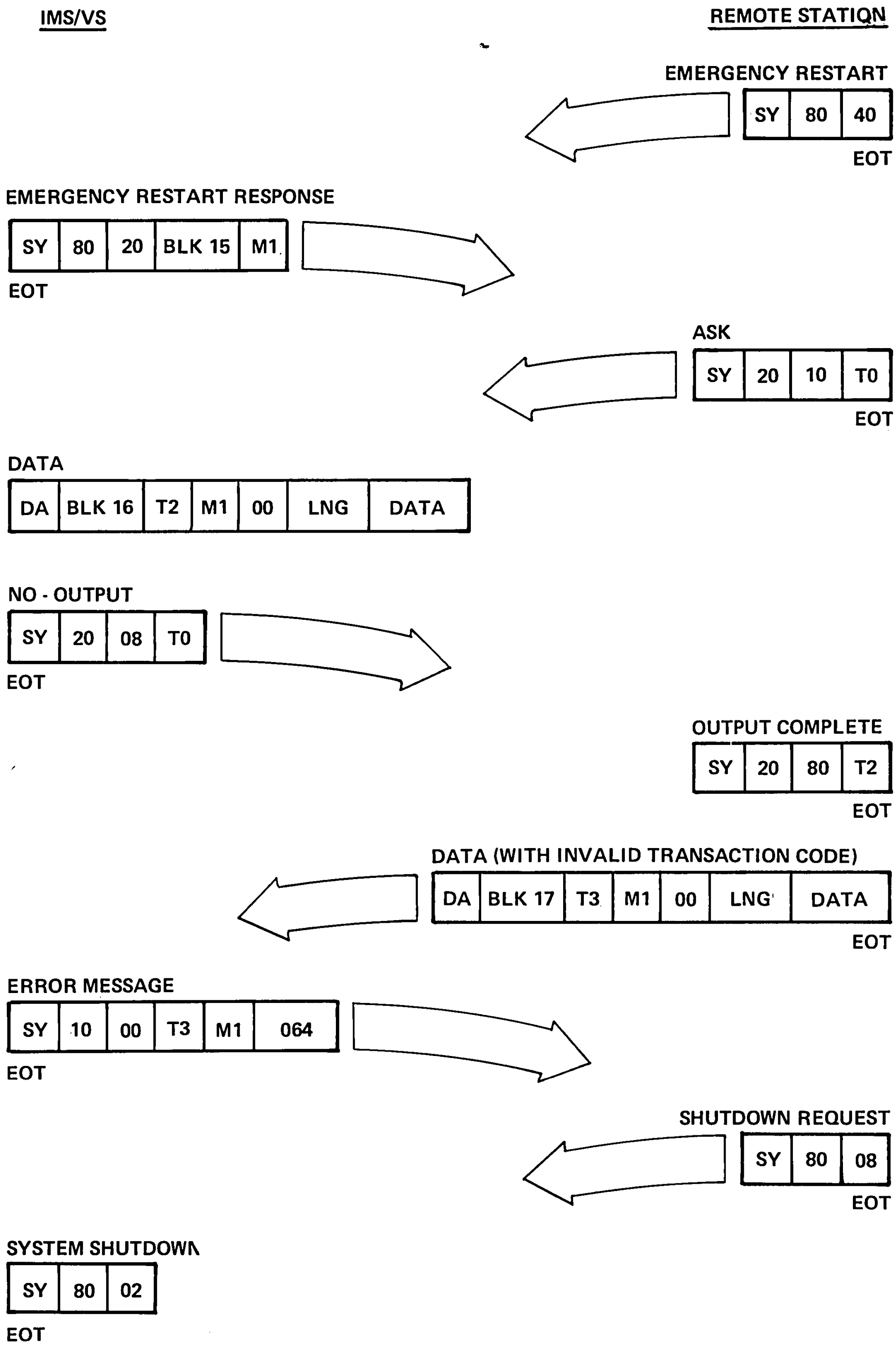


Figure 6-1 (Part 4 of 4). Sample IRSS Transmission Sequences

CHAPTER 7. INTERACTIVE QUERY FACILITY (IQF) WITH IMS/VS

INTRODUCTION

The Interactive Query Facility (IQF) is provided as an additional feature for users of IMS/VS with the full Data Base/Data Communication System. IQF offers the capability for spontaneous online query and retrieval and display of data maintained within IMS/VS data bases. IQF operates in a mode similar to a standard IMS/VS application program and uses IMS/VS resources for describing data, accessing data, and communicating with the user's terminal.

The IQF feature includes its own utility which creates the data bases used by IQF for resolving names, synonyms, and phrases appearing in the user's query.

Another function performed by the IQF utility is invoking IMS/VS PSB generation to generate a separate PSB for IQF use for each user-supplied PSB generation deck. The generated PSB will include PCBs for the IQF processor data bases. An IQF control card (described later in this chapter) is provided to allow the user to rename an existing PSB for use with IQF.

The IQF utility also creates and maintains IQF indexes.

CREATION OF IQF PROCESSOR DATA BASES

After performing IMS/VS system definition (described in the *IMS/VS Installation Guide*), including the IQF-required macro statements, the user must execute the IQF utility to create the following processor data bases: IQF System Data Base (required), IQF Phrase Data Base (required), and QINDEX Data Base(s) (optional). These data bases are described below.

- The System Data Base (sometimes referred to as the Field File) is a HISAM data base that contains system information from user-supplied IQF control cards and IMS/VS PSB generation and DBD generation decks. The purpose of this data base is rapid resolution of data base field names specified in the user's queries. The System Data Base is also used to provide column heading data and edit specifications for query output.

- The Phrase Data Base is a HIDAM data base that contains all the predefined phrases and null words provided by the user to tailor the IQF language to his requirements.

- The QINDEX Data Base(s) (optional) are HISAM data bases that provide an index to user-specified fields in the user's IMS/VS data bases. To conserve storage and time, two QINDEX Data Bases can be generated -- one with a large key field, and one with a small key field. The small key can be used to index all fields of its size or smaller; the large key can be used for other fields. The sizes of the two keys are under installation control.

Creation of these data bases requires that the user prepare a control card input deck for the IQF utility. The cards comprised in the deck are:

- IQF utility control statements
- IMS DBD statements
- IQF DBD extension statements
- IMS PSB statements
- IQF PSB extension statements

THE IQF UTILITY

The following programs comprise the IQF utility:

- Stage 1 System Creation (DMGSI1 and DMGSI2)
- System Data Base (Field File) Creation
- Index Creation/Update

The Stage 1 program processes the user's input control card deck and checks for validity and consistency. Depending upon the statements contained in the control card decks, Stage 1 produces job steps in a Stage 2 OS/VS job stream to perform some or all of the following functions:

- Allocate, catalog and create the IQF System Data Base describing the data bases to be queried

- Allocate, catalog and initialize the IQF Phrase Data Base to contain predefined phrases and null words

- Allocate, catalog and create the optional QINDEX Data Base(s) for IQF use

- Create or update index(es) stored in the QINDEX Data Base(s)

The Stage 1 program produces a listing of the input control card decks. A statement number appears in the listing to the left of each control statement. Any errors or warning conditions detected by Stage 1 appear in the listing following the printout of the control statements. The error or warning messages reference the statement number of the erroneous input statement. The user is cautioned to examine the output listing produced by Stage 1 before executing the Stage 2 OS/VS job stream.

The cataloged procedure for executing the IQF utility is described in an earlier chapter of this manual.

IQF UTILITY CONTROL STATEMENTS

The IQF utility control statements are described in the following sections.

THE QSYSFILE STATEMENT

The QSYSFILE statement specifies the data base name, volume(s) and space to be allocated for the processor data bases. The format of the user-coded QSYSFILE statement is:

```
| QSYSFILE |   {QFLDFILE }
|          |   {QPHFILE  }
|          |   {QINDEXS1 }
|          |   {QINDEXL1 }
|          |
|          | ,VOL=device=list¹[,INDEX=list¹]
|          | [,VOL2=list¹]
|          | ,SPACE=(CYL,(q1,(q2[,inc]),(q3[,inc])))
|          |
|          | [,MAXRTKEY= {value1}]
|          |             {32    }
|          |
|          | [,IXKEYLEN=(value2,value3)]
```

where:

QFLDFILE
: is the data base name of the IQF System Data Base.

QPHFILE
: is the data base name of the IQF Phrase Data Base. The data base name generated by the IQF utility for the index to the Phrase Data Base is QPHINDEX.

QINDEXS1
: is the data base name of the first QINDEX Data Base.

QINDEXL1
: is the data base name of the second QINDEX Data Base.

Note: If the IQF indexing feature is to be used, the QSYSFILE statement(s) for the QINDEX Data Base(s) must be included at creation time for the System Data Base. This causes the IQF utility to allocate space for the data base(s) and to initialize for subsequent index creation.

VOL=device=
: specifies the physical storage device type on which all data sets for this data set group are to be stored. A list of valid entries for this suboperand follows.

Device Name	"Devices"
Disk Facility	2314, 2319, 3330, or 3340
Fixed Head File	2305

list[1] specifies the volume serial number(s) of the volume(s) for a data set group as follows:

- Single-volume HISAM group
- Single-volume HIDAM group
- Multiple-volume HISAM group where the first volume in the list is also used for OSAM when the VOL2= is omitted
- Volumes of the ISAM portion of a HISAM group (volumes for OSAM portion are specified through VOL2= operand)
- Volumes of the OSAM portion of a HIDAM group (volumes for primary INDEX portion are specified through INDEX= and the VOL2= operands

INDEX=

list[1] specifies the volume serial number of the volume(s) used for the primary INDEX portion of a HIDAM data base. If VOL2= operand is also used, the suboperand specifies only the ISAM portion of the INDEX. Otherwise, the last volume in the VOL= suboperand list of the QPHFILE statement is also used for the OSAM portion of the INDEX.

VOL2=

list[1] specifies the volume serial number of the volume(s) used for the OSAM portion of a HISAM data set group or the OSAM portion of the primary INDEX of a HIDAM data set group.

<u>Note</u>: If the list suboperand consists of more than one volume serial number, the list is enclosed in parentheses and a comma is used to separate the serial numbers.

SPACE=CYL
specifies space allocation in cylinders as follows:

q1 allocation for the ISAM portion of a HISAM data set group or the ISAM portion of the primary INDEX of a HIDAM data set group.

q2 allocation for the OSAM portion of a HISAM data set group or the OSAM portion of the primary INDEX of a HIDAM data set group.

q3 allocation for the OSAM portion of a HIDAM data set group. This parameter is used only in the QPHFILE statement.

inc specifies secondary allocation for the OSAM or VSAM ESDS data set.

<u>Note</u>: The space allocation algorithms for the IQF processor data bases are discussed later in this chapter.

MAXRTKEY=

specifies the maximum size of the root key pointer field in a QINDEX Data Base. If the QINDEX Data Base capability is selected, this operand is optionally specified only in the QSYSFILE statement for the IQF System Data Base (QFLDFILE). If the operand is omitted, the default length is 32 bytes. A full file search will be required for any data base whose root key is greater than the value specified for this operand.

IXKEYLEN=

value2 specifies the maximum length index field for the QINDEXS1 Data Base.

value3 specifies the maximum length index field for the QINDEXL1 Data Base.

This operand is specified in the QSYSFILE statement for the IQF System Data Base (QFLDFILE) if either or both QINDEX Data Bases are used. If one QINDEX Data Base is used, then only the value2 operand is specified.

When creating the processor data bases, a QSYSFILE statement must be included in the control card input deck for the IQF System Data Base and the Phrase Data Base. If the IQF indexing feature is to be used, the QSYSFILE statement(s) for the QINDEX Data Base(s) must also be included. The applicable operands to be used in the QSYSFILE statement for each of the data bases are as follows:

Data Base Name	Operands
QFLDFILE	VOL=device=list[1][,VOL2=list[1]] ,SPACE=(CYL,(q1,(q2))) [,MAXRTKEY=value1] [,IXKEYLEN=(value2,value3)]
QPHFILE	VOL=device=list[1],INDEX=list[1] [,VOL2=list[1]] ,SPACE=(CYL,(q1,(q2[,inc]),(q3[,inc])))
QINDEXS1	VOL=device=list[1][,VOL2=list[1]] ,SPACE=(CYL,(q1,(q2)))
QINDEXL1	VOL=device=list[1][,VOL2=list[1]] ,SPACE=(CYL,(q1,(q2)))

THE OPTION STATEMENT

The OPTION statement specifies certain system defaults as described in the following discussion of the operands.

The format of the OPTION statement is:

```
| OPTION | [LINLIMIT=line limit]
|        |           200
|        |
|        | [,RECLIMIT=record limit]
|        |           0
|        |
|        | [,LIST= {YES} ]
|        |         {NO }
```

where:

LINLIMIT=
: is the maximum number of output lines produced by a query.

RECLIMIT=
: is the maximum number of logical records (data base path instances) retrieved from a data base by a query. If omitted, or if zero is coded, no limit is imposed.

LIST=
: specifies whether or not words in a query which are not recognized by the processor cause the query to terminate with an error message. The default option is to terminate.

Note: Both the LINLIMIT and RECLIMIT system defaults set through the OPTION statement can be overridden for a given query through the LIMIT command.

THE ** JOB STATEMENT

The ** JOB statement specifies the job execute statement to be included in the Stage 2 OS/VS job stream generated by the IQF utility (Stage 1).

The format of the ** JOB statement is:

** [jobname]	JOB	operands comments

where:

** must be punched in columns 1 and 2.

JOB must be preceded and followed by at least one blank.

Note: The operand field is the same as described in the OS/VS Job Control Language Reference Manual, GC28-0618.

If the ** JOB statement is not included, the following default card is generated in the Stage 2 job stream:

```
//IQFUTY    JOB 1,IQF,CLASS=A,
            MSGCLASS=A,
            MSGLEVEL=1
```

THE QINDXGEN STATEMENT

The QINDXGEN statement specifies index creation or update. Its presence in the control deck input to the IQF utility causes the IQF Index Creation/Update Utility program to be invoked.

The format of the QINDXGEN statement is:

```
| QINDXGEN |   CREATE
|          |   UPDATE
|          |
|          |                    [   ( A )   ]
|          | ,PCBN=pcb name     [ ( { D } ) ]
|          |                    [   ( M )   ]
|          |
|          | ,SEGN=segment name
|          |
|          |                    [   ( A )   ]
|          | ,FLDN=field name   [ ( { D } ) ]
|          |                    [   ( M )   ]
```

where:

CREATE
: specifies create (load) mode processing. This is the default.

UPDATE
: specifies update mode processing.

PCBN=
: specifies the IQF PCB name of the PCB describing the logical data base containing the indexed field. The same name should be used as that specified in the *QPCB statement.

SEGN=
: specifies the segment within the logical data base (PCB) containing the indexed field.

FLDN=
: specifies the indexed field. (A field that is indexed by IQF can be no greater than 250 bytes.)

FLDN always relates to the immediately preceding SEGN and SEGN to the most recent PCBN. The sequence is as follows:

```
PCBN=xxx,SEGN=xxx,
FLDN=xxx,FLDN=xxx,
FLDN=xxx,PCBN=xxx,
SEGN=xxx,FLDN=xxx,
SEGN=xxx,FLDN=xxx,
FLDN=xxx,FLDN=xxx
```

A processing action code, A, D, and M, can be specified at the data base (PCB) or field levels. The A, D, and M following the data base or field name indicate add, delete or modify processing. A code specified at the data base level supersedes any codes specified at the field level. If a code is not specified at either level, and the mode is CREATE, the default is "add". If the mode is UPDATE and the processing code is omitted, the default is "modify". If a code of D or M is specified in a statement with the CREATE mode code, the IQF Index Creation/Update Utility program assumes "add."

One or more QINDXGEN statements can be included in the IQF utility input deck. A separate statement can be used for each field to be indexed (or deleted) where it is desirable to process several fields in a single invocation of the IQF Index Creation/Update Utility program. It is also possible to repeat PCBN, SEGN and FLDN within a statement invocation. Also, a statement of both CREATE and UPDATE mode can be included in the input deck. For this case, however, the system will utilize a PROCOPT=A for processing the Index Data Base(s). (It should be noted that the creation of an index for a field not previously indexed results in a less efficient data structure.) place a non-blank character in column 72 and begin continuation with a keyword starting in column 16 of the following statement. (See the example in the discussion of "IQF Index Creation and Maintenance" provided later in this chapter.)

THE ENDUP STATEMENT

This statement must be entered. It indicates the end of input control card statements to the IQF utility.

	ENDUP	

IMS DBD STATEMENTS

The IMS DBD generation statements are described in the *IMS/VS Utilities Reference Manual*. The DBD control statements used for input to the IQF utility can be the same as those previously used to generate the DBDs described for an installation's data bases. Certain IQF statements are used, however, to expand the data base description to include additional field definitions, synonyms, column headings, etc. The DBD decks are used with the PSB decks by the IQF utility to create the System Data Base. The IQF DBD extension statements are described in the following section.

INTERACTIVE QUERY FACILITY (IQF) DBD EXTENSION STATEMENTS

IQF provides extensions to the DBD to define to IQF additional fields which are not defined to IMS/VS and to define synonyms, column headings and output masks for fields. The IQF DBD extension statements are *FIELD and *QFIELD. (An asterisk must always appear in column 1.) These statements are applicable only to the physical DBD deck.

Where FIELD statements are not present in the DBD deck, the *FIELD statement can follow a SEGM statement or a LCHILD statement.

The *FIELD Statement

This statement defines a field to IQF for use in a query. The field is not defined to IMS/VS. This capability can be used to subset an existing field or segment. The *FIELD statement must not be used to subset or bridge fields where packed decimal data (TYPE=P) is involved.

The format of the *FIELD statement is:

```
┌──────────────────────────────────────────────────────────────┐
|           |        |                                         |
|*          | FIELD  |                                         |
|           |        |                                         |
└──────────────────────────────────────────────────────────────┘
```

Note: The * in column 1 will cause IMS/VS DBDGEN to ignore this statement. There is thus no impact on user application programs sharing the data base(s).

The operands for the *FIELD statement are identical to those for the IMS/VS FIELD statement; the same rules and options apply. (See the *IMS/VS Utilities Reference Manual*.) It should be noted, however, that in IQF the following restrictions on data base field lengths apply to the TYPE= operands:

For TYPE = X	(hexadecimal data):	2 or 4 bytes
For TYPE = P	(packed decimal data):	1 to 31 digits
For TYPE = C	(alphameric data):	1 to 31 characters

All fields of a virtual logical child that are to be used in an IQF query must be defined by FIELD or *FIELD macro statements that refer to the data of the virtual logical child. (IQF does not automatically refer to field definitions provided for a real logical child and duplicate them under the virtual logical child at the appropriate offsets as does IMS.)

When a virtual logical child is defined, and when the user provides the virtual logical child in the input data stream provided to the IQF utility before the corresponding definition of the real logical child, the user must provide a FIELD or *FIELD macro statement for the virtual logical child such that the last byte of the virtual logical child data is included within the range of data defined by the FIELD or *FIELD macro statement.

The *QFIELD Statement

This statement specifies an output edit mask, column header or synonym for a field. The *QFIELD statement must immediately follow the FIELD or the *FIELD statement in the DBD deck.

The format of the *QFIELD statement is:

```
*    | QFIELD | [MASK=hh][,HEADER='header']
     |        | [,SYNONYM=(synonym,pcbname[,ALL])]
```

Note: The * in column 1 causes IMS DBDGEN to ignore this statement. There is thus no impact on user application programs sharing the data base(s).

where:

MASK=hh specifies a 1-byte output edit code.

The output mask byte is defined as follows:

00	Print as is.
01-3F	Reserved for future use.
40	Floating dollar sign, with no decimal places, left zero suppress, and commas every 3 non-zero places.
41	Invalid. Not to be used.
42	Same as 40, but with 2 decimal places.
43-7F	Invalid. Reserved for IBM World Trade Corporation use.
80	As is, with left-zero suppression.
81-BF	1-63 decimal places, left-zero suppression.
C0-FF	Invalid. Reserved for future use.

Note: Those masks pertaining to numeric editing such as decimal places, floating dollar signs, etc. are applicable only to packed decimal and hexadecimal fields.

HEADER=
specifies an output column header up to 20 bytes.

SYNONYM=

specifies a 1-word field synonym (maximum of 20 bytes) for the associated field name. The synonym is applicable to the associated field within the PCB named; pcbname is the IQF PCB name given in the *QPCB macro statement in the PSBGEN deck. Since the same field name can be used in FIELD macro statements in more than one segment in a DBDGEN, the "ALL" option can be used to indicate that the field synonym stands for the field in all segments within the PCB named. Multiple synonyms per field can be specified. (More than one synonym operand sublist can be specified per *QFIELD statement.)

Note: A synonym must: (1) be fewer than or equal to 20 alphameric characters, (2) start with an alpha character -- that is, A-Z, $, @, #, _(underscore), (3) be one word, without hyphenation, and (4) not be an IQF keyword.

IMS PSB STATEMENTS

The statements in the IMS PSB deck are described in the "PSB Generation" chapter of the IMS/VS Utilities Reference Manual. The PSB control statements used for input to the IQF utility can be the same as those previously used to generate PSBs for application programs. An optional IQF control statement (that is, *QPSBGEN) can be used to rename the PSB for use by IQF.

The user is cautioned that the IQF utility automatically includes PCBs for the IQF processor data bases -- that is, the System Data Base, Phrase Data Base, and (if defined) one or two Index Data Bases -- within the user-provided PSB deck. If the existing user PSB already contains the maximum number of PCBs that can be defined in a PSBGEN, the PSB should be restructured to accommodate the addition of the IQF PCBs. This may involve breaking the existing PSB into two or more PSBs. The manner in which the PSB is restructured is contingent upon what an installation wants to query through a given transaction code.

All IMS/VS PCB macro statements (PCB, SENSEG, PSBGEN) to be used by IQF must be contained within one card (columns 2-70). The user should examine his PSB generation deck(s) to ensure that the PCB statements meet this requirement. The user should also examine all SENSEG statements where the PROCOPT keyword has been coded. If PROCOPT is coded, 'GP' must be part of that PROCOPT to insure that returns from DL/I to IQF will be normal.

The PSB decks are combined with the DBD decks for creation of the System Data Base. The IQF PSB extension statements are described below.

INTERACTIVE QUERY FACILITY (IQF) PSB EXTENSION STATEMENTS

The IQF PSB extension statements are *QPCB and *QPSBGEN. (An asterisk must always appear in column 1.)

The *QPCB Statement

To associate the query with the appropriate logical data base (PCB), it is necessary to provide a PCB name for use by IQF. (PCB names referred to by IQF must be unique within a user's installation.) The *QPCB statement provides this function.

The format of the *QPCB statement is:

*	QPCB	PCBN=pcb name

Note: The * in column 1 of the QPCB statement causes IMS PSBGEN to ignore these statements. There is thus no impact on user application programs sharing the data.

where:

PCBN=
: specifies a 1- to 8-byte unique alphameric name to be associated with the PCB. This is the data base name to be used in QUERY commands for this data base.

The user is cautioned that he must insert the *QPCB statement immediately following each PCB statement in the PSB generation decks that pertain to a data base to be queried. In addition to providing a name for the PCB for use in IQF QUERY commands, this statement identifies PCBs sensitive to IQF processing. If the *QPCB statement is omitted, the IQF utility ignores the PCB.

The *QPSBGEN Statement

The optional *QPSBGEN statement follows the PSBGEN card. It provides the capability to rename a PSB input to the IQF utility without actually changing the name in the PSBGEN statement.

The format of the *QPSBGEN statement is:

```
┌──────────────────────────────────────────────────────────────┐
│          │         │                                         │
│*         │ QPSBGEN │ [PSBNAME=psb name] [,FFS=code]          │
│          │         │                                         │
└──────────────────────────────────────────────────────────────┘
```

where:

PSBNAME=
: specifies the PSB name to be used for IQF processing.

  Note: If this operand is used, the PSB name specified must also be coded in PSB=operand of the APPLCTN system definition macro-instruction.

FFS=
: specifies the name of the transaction code to be used for Full File Search (if any) associated with this PSB. If the code is an *, the Full File Search is performed by the same transaction code. This may cause checkpoint problems. If the code is not present, a Full File Search is not invocable by the transaction.

  The Full File Search transaction code must be specified to IMS/VS through the TRANSACT macro-instruction at IMS/VS system definition. If this transaction code is not an asterisk (*), it must be a non-conversational transaction code which uses the PSB named in the previous operand. In other words, the transaction code must have been specified at IMS/VS system definition time through a TRANSACT macro comprising the application description set which references the PSB named in the *QPSBGEN statement. The Full File Search is performed using the same PSB used during initial processing of the query in conversational mode.

  The capability to designate an alternate transaction code for the Full File Search (FFS) allows the installation to control when queries involving such an operation are to be executed. The master terminal operator can issue a /PSTOP for the FFS transaction code and any future Full File Search processing is queued for execution at a later time (when a /START command is issued).

  IQF informs the user that the query requires a Full File Search and requests him to reply "YES" or "NO", indicating whether or not he desires IQF to proceed. If a /PSTOP has been previously issued, the user's reply to the FFS response is accepted and queued for subsequent processing when a /START for the transaction code is issued. Depending upon the installation procedure, the terminal user may know when the FFS alternate transaction code has been /PSTOPped, or it may be necessary for him to communicate with the master terminal operator for this information.

A /STOP of the IQF conversational transaction code causes IMS/VS to reject the user's query. This /STOPs queuing of input only if the message to be queued originates at a terminal.

Under no condition should a terminal user attempt to enter a non-conversational transaction code for IQF.

FULL FILE SEARCH EXAMPLES

Case 1

```
*QPSBGEN    PSBNAME=PSB03,FFS=TRANCDX4
```

After a YES reply from the user terminal, IQF performs a program-to-program message switch using TRANCDX4. The user's system has defined TRANCDX4 as a non-conversational transaction code using the PSB name PSB03. IQF returns the message "QUERY HELD FOR LATER PROCESSING" and frees the input terminal by returning the SPA to IMS.

For this example, the system definition relating PSB and transaction codes might be as follows:

```
APPLCTN     PSB=PSB03,IQF=YES
TRANSACT    CODE=IQFTCDE,SPA=(1000,CORE),MODE=SNGL
TRANSACT    CODE=TRANCDX4
```

where IQFTCDE is used for conversational terminal input and TRANCDX4 is used internally by IQF for message switching to a non-conversational transaction code.

Case 2

```
*QPSBGEN    PSBNAME=PSB03,FFS=*
```

After a YES reply from the user terminal, IQF immediately starts full file searching. The user terminal remains in conversation for the duration of query processing.

Case 3

When a query is entered with an illegal (non-conversational) transaction code, the following IMS message is returned:

```
DFS080     MESSAGE CANCELED BY INPUT EDIT ROUTINE
```

Whenever this happens, the user should reenter the query with a valid IQF transaction code.

SUMMARY OF CONTROL STATEMENTS REQUIRED FOR PROCESSOR DATA BASES

IQF UTILITY CONTROL STATEMENTS

- QSYSFILE — Required 1 each for the System Data Base and the Phrase data base. If the IQF indexing feature is used, one QSYSFILE statement is required for each QINDEX data base.
- OPTION — Optional 1
- ** JOB — Optional 1
- QINDXGEN — Optional n
- ENDUP — Required 1

IMS DBD STATEMENTS

See the "DBD Generation" chapter of the _IMS/VS Utilities Reference Manual_.

IQF DBD EXTENSION STATEMENTS

- *FIELD — Optional n
- *QFIELD — Optional n

IMS PSB STATEMENTS

See the "PSB Generation" chapter of the _IMS/VS Utilities Reference Manual_.

IQF PSB EXTENSION STATEMENTS

- *QPCB — Required 1 for each PCB that the user wants to access IQF.
- *QPSBGEN — Optional n (1 for each PSB)

Note: Except for the ** JOB statement, IQF cards with an asterisk in column 1 can be kept in the input deck when it is used for IMS/VS system definition. The QSYSFILE, OPTION, and QINDXGEN cards, however, are not to be retained in the deck.

EXAMPLE OF CONTROL STATEMENTS FOR PROCESSOR DATA BASE CREATION

The following example shows the control statements required to create the System Data Base and to allocate and initialize the Phrase and QINDEX data bases.

```
//...     JOB
//        EXEC IQFUT
//SYSIN DD              *
          QSYSFILE      QFLDFILE,VOL=2314=999999,                          *
                        SPACE=(CYL,(20,(5))),                              *
                        MAXRTKEY=25,IXKEYLEN=(10,30)
          QSYSFILE      QPHFILE,VOL=2314=888888,                           *
                        INDEX=777777,                                      *
                        SPACE=(CYL,(10,(5,1),(20,2)))
          QSYSFILE      QINDEXS1,VOL=3330=666666,                          *
                        VOL2=555555,                                       *
                        SPACE=(CYL,(30,(10,1)))
          QSYSFILE      QINDEXL1,VOL=3330=555555,                          *
                        SPACE=(CYL,(10,(10,2)))
          OPTION        LINLIMIT=200,RECLIMIT=50
**        JOB           (6696),IQF,CLASS=A,MSGCLASS=A,MSGLEVEL=1
          DBD           NAME=VENDOR,ACCESS=HIDAM
            .
          FIELD         NAME=VENDNAM,...
*         QFIELD        MASK=00,HEADER='VENDOR NAME',                      *
                        SYNONYM=SUPPLIER
          FIELD         NAME=ADDRESS,...
*         FIELD         NAME=CITY,...
            .
            .
          DBDGEN
          FINISH
          END
          DBD           NAME=PAYROLDB,...
            .
          DBDGEN
          FINISH
          END
          PCB           TYPE=DB,DBDNAME=VENDOR ...
*         QPCB          PCBN=ORDERS
            .
            .
            .
          PCB           TYPE=DB,...
*         QPCB          PCBN=...
            .
            .
            .
          PSBGEN        LANG=ASSEM,PSBNAME=VENDFILE
*         QPSBGEN       PSBNAME=ORDRFILE,FFS=SUPLFFS
          END
          PCB           TYPE=DB,DBDNAME=PAYROLDB
*         QPCB          PCBN=PAYROLL
            .
            .
            .
            .
          PSBGEN        LANG=COBOL,PSBNAME=PAYONE
*         QPSBGEN       PSBNAME=QIQFPSB,FFS=*
          END
          ENDUP
```

IQF SYSTEM DATA BASE MAINTENANCE

If the user intends to add or delete data bases for IQF processing, or to define new fields in existing data bases, he must execute the IQF utility to recreate the IQF data bases (after scratching the old data set groups composing these data bases).

IQF INDEX CREATION AND MAINTENANCE

A facility is provided to update and create indexes using the QINDEX data bases. The QINDEX data base(s) must be allocated creation time to generate the system as illustrated in the preceding example. The indexes can be created or updated as required through the IQF utility.

The example below illustrates the control statements required to create an index and to update indexes.

```
//          JOB ...
//          EXEC IQFUT
//SYSIN     DD          *
            QINDXGEN    CREATE,PCBN=PAYROLL(A),                       *
                        SEGN=NAMEMAST,                                *
                        FLDN=EMPLOYEE
            QINDXGEN    UPDATE,PCBN=INVOICE,                          *
                        SEGN=DUEIN,                                   *
                        FLDN=INVONO(M)
**          JOB         (6696),IQF,CLASS=A,MSGCLASS=A,MSGLEVEL=1
            ENDUP
```

Note: Index creation can be combined in the same job step.

EXAMPLE OF STAGE 2 OS/VS JOB STREAM FOR CREATION OF IQF PROCESSOR DATA BASES (Output of Stage 1)

```
//IQFUTY JOB 1,IQF,CLASS=A,MSGCLASS=A,MSGLEVEL=1
// EXEC DBDGEN,MBR=QFLDFILE
//C.SYSIN DD *
           IQFDBD   FF=Y
           END
/*
// EXEC DBDGEN,MBR=QPHFILE
//C.SYSIN DD *
           IQFDBD   PH=Y
           END
/*
// EXEC DBDGEN,MBR=QPHINDEX
//C.SYSIN DD *
           IQFDBD   PI=Y
           END
/*
// EXEC DBDGEN,MBR=QINDEXS1
//C.SYSIN DD *
           IQFDBD   XS=(10,L),MRKL=25
           END
/*
// EXEC DBDGEN,MBR=QINDEXL1
//C.SYSIN DD *
           IQFDBD   XL=(30,L),MRKL=25
           END
/*
// EXEC PSBGEN,MBR=DMGFC1
//C.SYSIN DD *
           IQFPCB FF=Y,PH=Y,PSBN=DMGFC1,XS=(10,L),XL=(30,L)
           END
/*
// EXEC PSBGEN,MBR=DMGSIB
//C.SYSIN DD *
           IQFPCB FF=Y,PSBN=DMGSIB
           END
/*
// EXEC PSBGEN,MBR=QIQFPSB,
//C.SYSIN DD *
           IQFPCB FF=Y,PH=Y,XS=(10,A),XL=(30,A)
 PCB    TYPE=DB,DBDNAME=PAYROLDB,PROCOPT=GP,KEYLEN=22
*        QPCB   PCBN=PAYROLL
         SENSEG NAME=NAMEMAST,PARENT=0,PROCOPT=G
         PSBGEN LANG=ASSEM,PSBNAME=QIQFPSB,FFS=*
*   QPSBGEN     PSBNAME=QIQFPSB,FFS=*
         END
/*
```

```
//          EXEC IQFFC
//QFF       DD    DSN=IQFIFFDB,UNIT=2314,
//                VOL=SER=999999,
//                SPACE=(CYL,20),
//                DISP=(NEW,CATLG),DCB=(DSORG=IS)
//QFFOVF    DD    DSN=IQFOFFDB,UNIT=2314,
//                VOL=SER=999888,
//                SPACE=(CYL,(30,1)),
//                DISP=(NEW,CATLG),DCB=(DSORG=PS)
//QPHX      DD    DSN=IQFIXPDB,UNIT=2314,
//                VOL=SER=777777,
//                SPACE=(CYL,10),
//                DISP=(NEW,CATLG),DCB=(DSORG=IS)
//QPHOVF    DD    DSN=IQFOXPDB,UNIT=2314,
//                VOL=SER=888888,
//                SPACE=(CYL,(5,1)),
//                DISP=(NEW,CATLG),DCB=(DSORG=PS)
//QPH       DD    DSN=IQFPHFDB,UNIT=2314,
//                VOL=SER=888888,
//                SPACE=(CYL,(20,2)),
//                DISP=(NEW,CATLG),DCB=(DSORG=PS)
//QXS1      DD    DSN=IQFXS1DB,UNIT=3330,
//                VOL=SER=666666,
//                SPACE=(CYL,30),
//                DISP=(NEW,CATLG),DCB=(DSORG=IS)
//QXS1OV    DD    DSN=IQFXOVS1,UNIT=3330,
//                VOL=SER=555555,
//                SPACE=(CYL,(10,1)),
//                DISP=(NEW,CATLG),DCB=(DSORG=PS)
//QXL1      DD    DSN=IQFXL1DB,UNIT=3330,
//                VOL=SER=555555,
//                SPACE=(CYL,10),
//                DISP=(NEW,CATLG),DCB=(DSORG=IS)
//QXL1OV    DD    DSN=IQFXOVL1,UNIT=3330,
//                VOL=SER=555555,
//                SPACE=(CYL,(10,2)),
//                DISP=(NEW,CATLG),DCB=(DSORG=PS)
//FC1.SYSIN DD *
```

```
   QSYSFILE      QFLDFILE,MAXRTKEY=25,IXKEYLEN=(10,30)
    OPTION       LINELIMIT=200
    DBD          NAME=PAYROLDB,ACCESS=HISAM
    DATASET      DD1=PAYROLL,OVFLW=PAYROLOV,DEVICE=2314
    SEGM         NAME=NAMEMAST,BYTES=150,FREQ=1000,PARENT=0
    LCHILD       NAME=(SKILNAME,SKILLINV),PAIR=NAMESKIL,PTR=NONE
    FIELD        NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C
    DBDGEN
    FINISH
    END
    DBD          NAME=LOGICDB,ACCESS=LOGICAL
    DATASET      LOGICAL
    SEGM         NAME=SKILL,SOURCE=((SKILL,,SKILLINV))
    DBDGEN
    FINISH
    END
 PCB    TYPE=DB,DBDNAME=PAYROLDB,PROCOPT=GP,KEYLEN=22
*         QPCB   PCBN=PAYROLL
          SENSEG  NAME=NAMEMAST,PARENT=0,PROCOPT=G
          PSBGEN  LANG=ASSEM,PSBNAME=QIQFPSB
*   QPSBGEN      PSBNAME=QIQFPSB,FFS=*
          END
/*
//QUS2X1 EXEC   PSBGEN,MBR=DMGIU1
//C.SYSIN   DD   *
 IQFPCB FF=Y,XS=(10,L),XL=(30,L)
 PCB    TYPE=DB,DBDNAME=PAYROLDB,PROCOPT=GP,KEYLEN=22
*         QPCB   PCBN=PAYROLL
          SENSEG  NAME=NAMEMAST,PARENT=0,PROCOPT=G
 PSBGEN LANG=ASSEM,PSBNAME=DMGIU1
 END
/*
//L.SYSLMOD   DD   DSN=&&PSBTEMP(DMGIU1),UNIT=SYSDA,DISP=(NEW,PASS),     *
//               SPACE=(1024,(10,4,1))
//QUS2X2 EXEC   IQFIU
//IU1.SYSIN DD *
          PSBD
    QINDXGEN     CREATE,PCBN=PAYROLL(A),SEGN=NAMEMAST,FLDN=EMPLOYEE
/*
//
```

Note: Stage 1 Part 2 output is punched card only. This includes job steps associated with indexing.

STORAGE REQUIREMENTS

The main storage requirements for the DB/DC system depend on the specifications set forth and the options selected in Stage 1 of IMS/VS system definition. In addition, the main storage requirements are affected by the values which appear in the parameter field of the job control language EXEC statements for the control and batch processing regions. The OS/VS options and the contents of the resident areas also influence the main storage requirements.

Refer to the "IMS/VS Storage Estimates" chapter of this manual for storage allocations required by IMS/VS. (The figures referenced in the discussion that follows are included in that chapter.)

IMS/VS CONTROL REGION

The inclusion of IQF into the IMS/VS system affects the main storage requirements of the IMS/VS control region. The areas to be considered in calculating the storage requirements for this region are discussed in the following sections.

Control Program Code

Refer to Figure 5-6 for the size of the basic and optional control program code. To calculate the size of the control program code with IQF included, add the following to the basic code:

- 180 bytes for the IQF Transaction Edit module
- The optional code for conversational processing
- The optional code for paging
- Resident terminal device support code

Control Blocks

The specifications presented in Stage 1 of the IMS/VS system definition directly influence the generation of control blocks. Figure 5-8 contains the storage estimates based on those specifications. In calculating the storage requirements for each data base defined to the system, the user must consider the internal (processor) data bases used by IQF. These are the System Data Base, Phrase Data Base, and one or two optional QINDEX Data Bases. Although DATABASE macro statements are required only for the QINDEX data bases, the System and Phrase data bases must be considered in determining storage requirements.

Given an existing IMS/VS system to which IQF is to be added, the additional nucleus control blocks space required is as follows:

	Minimum Space Needed (bytes)
For each APPLCTN macro statement added because of IQF installation (one or more required):	40
For each TRANSACT macro statement added because of IQF installation (one or more required):	56
For each DATABASE macro statement added because of IQF installation (two, three or four required for IQF internal data bases; to this must be added the number of additional user data bases not already in IMS/VS):	36
The square of the total number of data bases included in the IMS/VS system definition minus the square of the number of data bases that existed before IQF was added to the system.	n

For example, if three IQF internal data bases are added to a system with five existing data bases, and no additional user data bases are added, then the impact on the nucleus control blocks for one IQF transaction is:

$$40 + 56 + (3 \times 36) + 8^2 - 5^2 = 243$$

Loaded Modules

Depending on the terminal device support requirements and the data base organizations chosen, different modules are selected for loading into the control region. If new terminals are added to the user's system configuration concurrent with the installation of IQF, refer to Figure 5-7 to determine storage requirements for the terminal support modules.

In the area of data base organization, IQF uses the HISAM and HIDAM organizations for its internal (processor) data bases. If IQF is to be added to an existing IMS/VS system where either (or both) of these data organizations was not previously used, then the storage requirements for these load modules must be considered. Refer to Figure 5-1.

IMS/VS Buffers

Inclusion of IQF in the IMS/VS system may require additional buffer pool space within the control program region. Refer to the discussion of buffers in the "IMS/VS Storage Estimates" chapter of this manual.

If the addition of IQF impacts message traffic, concurrent processing, data base processing intent, or terminal configuration, these factors must be considered in determining buffer storage space. The formulas presented in the "IMS/VS Storage Estimates" chapter can be used for calculating buffer storage requirements.

Refer to the formulas for calculating the sizes of PSBs and Data Base PCBs in the "IMS/VS Storage Estimates" chapter.

The formula described for calculating the size of Data Base PCBs can be used for the IQF internal data bases. The following values should be used:

	System	Phrase	QINDEX
A =	0	0	0
B =	5	10	2
C =	0	0	0
D =	3	6	2
E =	1	1	1
F =	0	0	0
G =	0	0	0
H =	37	100	**

** (6 + key length + MAXRTKEY)

The formula described for calculating the size of DMBs can be used for the IQF internal data bases. The following values should be used:

	System	Phrase	QINDEX
A =	2	3	2
B =	1	2	1
C =	5	11	2
D =	0	2	0
E =	0	0	0
F =	17	6	10
H =	1	1	1
I =	0	0	0
J =	0	0	0
L =	0	0	0
M =	0	0	0

Dynamic Storage Requirements

OS/VS requirements, for use in calculating additonal storage space for device or data base organization support required by the inclusion of IQF in the IMS/VS system, can be obtained from the appropriate OS/VS documentation.

IMS/VS MESSAGE PROCESSING REGION

The minimum message region size for IQF is 54K. This should handle 98 percent of the queries. It is assumed that the typical query will be less than 200 bytes long and mention fewer than 15 fields with an average field length of 10 bytes, and that sorting will not be performed. If sorting is performed, 2K bytes of the 50K will be available to hold the records.

A larger region may be required for the following reasons:

1. Many data fields

2. Large data fields

3. Sorting of a large quantity of records (collections of fields) or a quantity of large records

4. A complex query, generating a large amount of code

IMS/VS BATCH PROCESSING REGION

To run the IQF utility, a batch IMS/VS region of at least 250K is required.

The minimum region size of 250K for the IQF utility is based on a SORT work area size of 44K. If a larger work area size was specified at SYSGEN time, an appropriate increase must be made to the minimum region size for the IQF utility. Also, the IQF Index Utility program may require a further increase in the minimum region size. This potential increase can be calculated as follows: If A is equal to the number of times a value occurs in a field name being indexed and B is equal to the MAXRTKEY value specified at IQF system generation, calculate A(B+1.5)-8000. If the result is positive, the region size should be increased by the result (round up to the next multiple of 2K).

SECONDARY STORAGE

A maximum of 20 tracks of 2314 space is required for the IQF load modules.

IQF MODULE STORAGE (BYTES)

The following shows the number of bytes used by the different IQF modules at various stages of processing.

Modules	INPUT	COMPILE	RETENT	GENER	EXEC
Common Module Table (CMT)	700	700	700	700	700
SPA & Message	700	550	550	550	550
Control Program	1950	1950	1950	1950	1950
Message Interface	3100	3100	3100	3100	3100
Variable Message Builder	5000	5000	5000	5000	5000
Message Interface Work Area (WA)	450	450	450	450	450
Language Analyzer I		3000			
Language Analyzer I Work Area (WA)		300			
Edit Input Table WA		0			
Phrase Parameter Table (formerly EITWA)		400		400	
Edit Input Table		400			
Internal (IQF Processor) Data Base Interface-2		5328			
Internal (IQF Processor) Data Base Interface-2 WA		328			
Internal (IQF Processor) Data Base Interface-2 DL/I Buffers		200			
Field Information Table (20x52 bytes each)		1040		1040	1040
Query Path Description Table (10x20 bytes each)		200		200	200
Query Path Validation Table (25x32 bytes each)		700			
Retention			2100		
Internal (IQF Processor) Data Base Interface-1			2200		
Internal (IQF Processor) Data Base Interface-1 WA and Retention WA			600		
Internal (IQF Processor) Data Base Interface-1 DL/I Buffers			200		

Modules	Time Periods				
	INPUT	COMPILE	RETENT	GENER	EXEC
Language Analyzer II				500	
Language Analyzer II WA				60	
Func. Modules (List Total = 2800; Selection Criteria = 7500)				7500	
Generated Code Area				4096	4096
User Data Base Interface (UDI)					4400
UDI WA & Tables					2322
UDI DL/I Buffers					1720
UDI Logical Record (est.)					100
Sort & WA (if required)					2400
Sort Buffers (2K blocks)					2048
Storage Allocation Fragments	3000	3000	3000	3000	3000
Subtotal	14900	26646	19850	27211	33076
IMS/VS Region/Program Control and OS/VS Work Area (VS2)*	7200	7200	7200	7200	7200
TOTAL	22150	33846	27050	34411	40276

APPENDIX A. ORGANIZATION OF CONTROL PROGRAM

Figure A-1 below shows the general organization of the control program region in OS/VS1.

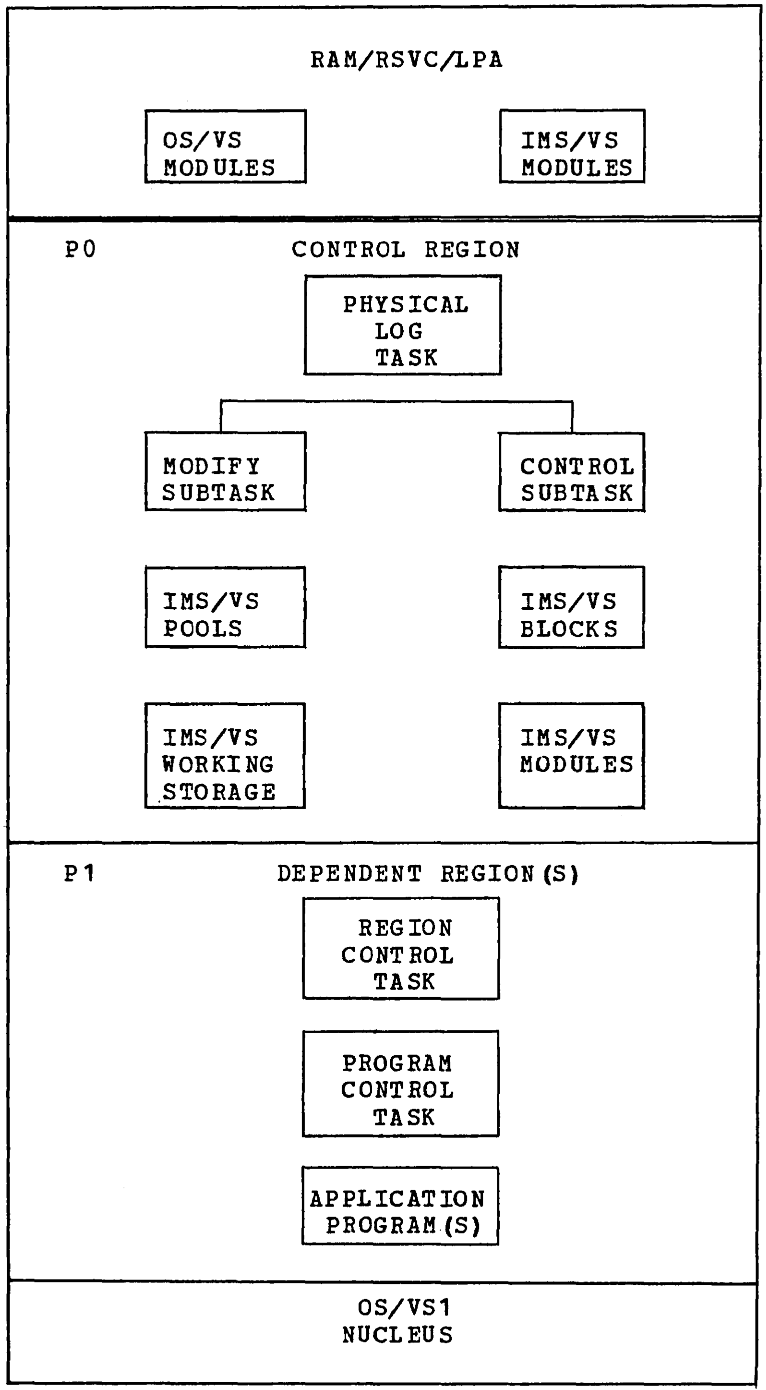


Figure A-1. IMS/VS System Structure in OS/VS1

Figure A-2 shows the general organization of the control program region in OS/VS2.

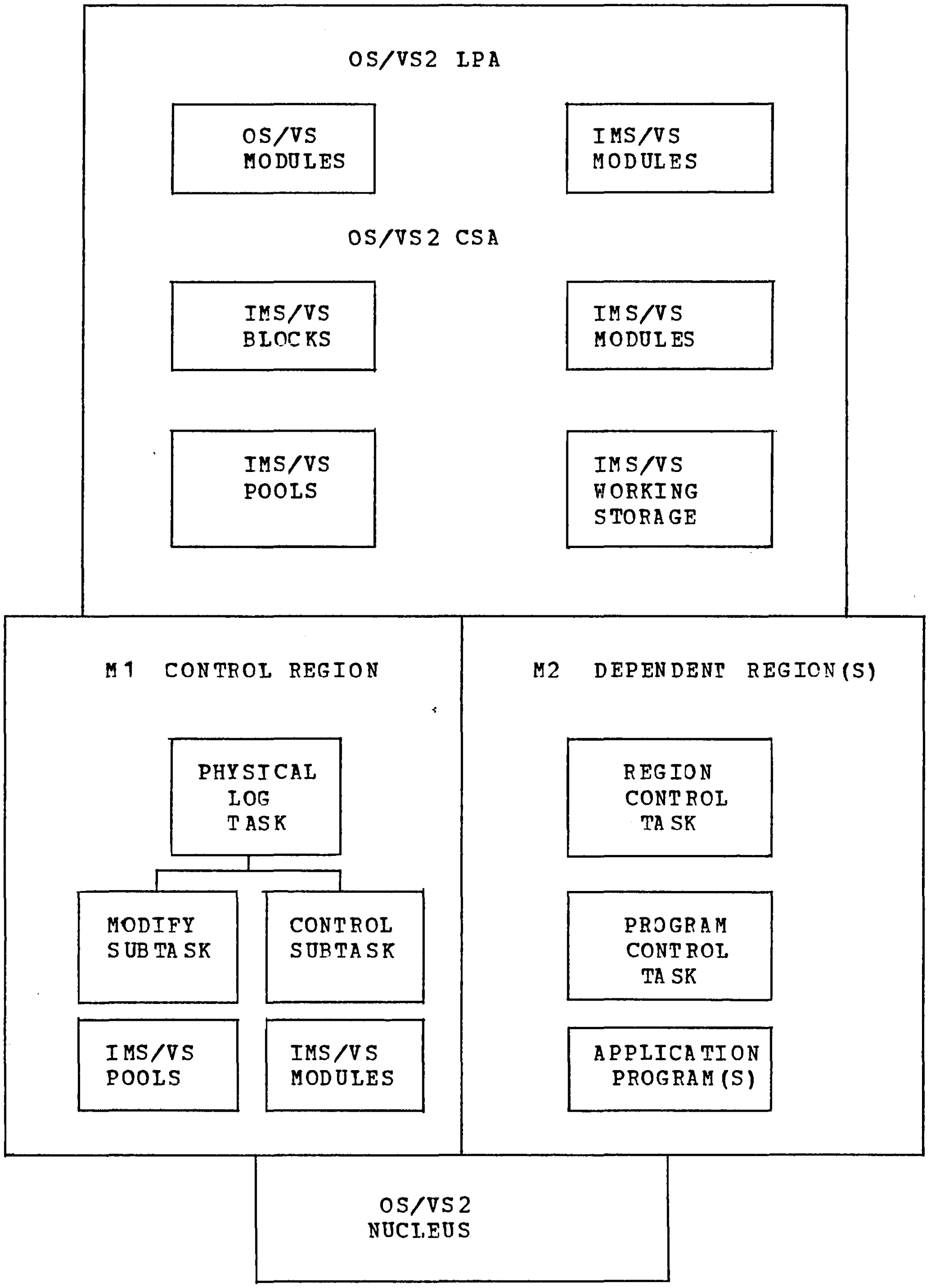


Figure A-2. IMS/VS System Structure in OS/VS2

CONTROL PROGRAM NUCLEUS

RESIDENT ROOT (See Figure A-4)
OVERLAY REGION (1 OR 2) (See Figures A-6 and A-7 for overlay region 1 and overlay region 2 contents)

Figure A-3. Control Program Nucleus Generation (VS1 V=R)

CONTROL PROGRAM ROOT

RESIDENT MAP
CONTROL BLOCKS (See Figure A-5)
CONTROL MODULES
DATA COMMUNICATION MODULES
DATA BASE MODULES

Figure A-4. Control Program Nucleus -- Root Generation (VS1 V=R)

CONTROL BLOCKS

COMMUNICATION LINE BLOCKS (CLB) (CLBDECB)
COMMUNICATION TERMINAL BLOCKS (CTB)
COMMUNICATION INTERFACE BLOCKS (CIB)
COMMUNICATION RESTART BLOCKS (CRB)
COMMUNICATION NAME TABLES (CNT)
COMMUNICATION TERMINAL TABLES (CTT)
COMMUNICATION VERB BLOCKS (CVB)
COMMUNICATION EXTENSION BLOCK (CXB)
MSG Q MGR CONTROL BLOCKS (Q DCBs; Q IOBs)
TRANSACTION CLASS TABLE (TCT)

Figure A-5. Control Program Nucleus -- Control Blocks Generation (VS1 V=R)

CONTROL PROGRAM OVERLAY REGION 1, SECTION 1

RSTO -- RESTART PROCESSING	RCPO -- CHECKPOINT PROCESSING

Figure A-6. Control Program Nucleus -- Contents of Overlay Region 1 Generation (VS1 V=R)

CONTROL PROGRAM OVERLAY REGION 2, SECTION 1

CLMO					
CMT1	MESSAGE	IDP0			
CMT2	GENERATION	IDP1			
CMT3		IDP2		IPCP	CHECKPOINT
CMT4		IDP3	DISPLAY	TERM	SHUTDOWN
ICA1		IDP4	COMMAND		
		IDP5	PROCESSING	CRSB1	
ICLE		IDP6		CRSB2	SYSTEM 3/
ICLG		IDP7		CRSH	SYSTEM 7
ICLH	TERMINAL	IDP8		CRSL1	PROCESSORS
ICLJ	COMMAND	IDP9		CRSN1	
		IDPA		CRSW	
		IDPB			
ICL1	PROCESSING	IRD1		CRSX	
ICL2	EXCEPT	CFEZ	TRACE EFFECTOR	CR2Z	
ICL3	DISPLAY	CFEZ1		CS7L	
ICL4		RNRE		CS7L2	
ICL5		RERE		CRS8	
ICL6					
ICL7		RBOI			
ICL8		RDBC			
ICL9		IECTLOPN			
		IECTCHGN			

CONTROL PROGRAM OVERLAY REGION 2, SECTION 2

ISMI -- SECURITY MAINTENANCE INITIALIZATION

Figure A-7. Control Program Nucleus -- Contents of Overlay Region 2 Generation (VS1 V=R)

IMS/VS BUFFERS

QUEUE
PROGRAM SPECIFICATION BLOCKS*
DATA MANAGEMENT BLOCKS*
DATA BASE BUFFERS*
TERMINAL BUFFERS
DATA BASE LOG BUFFERS*
FORMAT BLOCK BUFFERS
WORKING STORAGE*

* In OS/VS2 these buffers are in CSA.

Figure A-8. Control Program Region -- Buffer Areas

INDEX

SH20-9027-4

IMS/VS System Programming Reference Manual Printed in U.S.A. SH20-9027-4

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IMS/VS Version 1
System Programming
Reference Manual
SH20-9027-4

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

SH20-9027-4

Fold and Staple

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
P. O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IMS/VS System Programming Reference Manual Printed in U.S.A. SH20-9027-4