This publication explains when, why, and how to use IBM service aids to diagnose and fix failures in system or application programs. Each service aid is described in a separate chapter. The service aids are:

- **IFCDIP00** -- Initializes the SYS1.LOGREC data set.
- **IFCEREPO** -- Summarizes and prints records from the SYS1.LOGREC data set.
- **GTF (Generalized Trace Facility)** -- Traces selected system events such as SVC and I/O interruptions.
- **IMCJQDMP** -- Operates as a stand-alone program to format and print the system job queue.
- **IMBLIST** -- Formats and prints object modules, load modules, and CSECT identification records.
- **IMBMDMAP** -- Maps load modules.
- **IMCOSJQD** -- Operates as a problem program to format and print the system job queue.
- **IMDPRDMP** -- Formats and prints dumps, TSO swap data set, and GTF trace data.
- **IMAPTFLE** -- Generates JCL needed to apply to a PTF and/or applies the PTF.
- **IMDSADMP** -- Operates as a stand-alone program to produce a high-speed or low-speed dump of main storage.
- **IMASP2ZAP** -- Verifies and/or replaces instructions and/or data in a load module.

Information about how to write EDIT user programs is provided in a separate appendix.
Third Edition (March, 1972)

This is a major revision of, and obsoletes, GC28-6719-1 and Technical Newsletter GN28-2478. Text changes and illustration changes in chapters containing few changes are indicated by vertical lines to the left of the changes. Consult the Summary of Amendments following the Contents Directory for information about which chapters are new and which are changed.

This edition applies to release 21, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/360 and System/370 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department 958, PO Box 390, Poughkeepsie, N. Y. 12602. Comments become the property of IBM.

This publication is for system programmers and IBM programming systems representatives. It explains when, why, and how to use IBM service aids to diagnose and fix failures in system or application programs.

Each service aid is described in a separate chapter. The chapters are arranged so that the corresponding index tabs will appear in alphabetical order. The index tabs show the names of the programs minus the three-character component identifier (such as IMC). The form of the name shown on the index tab also appears in the index to help you locate the chapter you want.

Some information about service aids is not included in this publication, but is covered in the following publications:

IBM System/360 Operating System:

- **Service Aids Logic PLM, GY28-6721** -- describes the internal logic of the service aid programs (how they work).
- **Programmer's Guide to Debugging, GC28-6670** -- describes the dump-type output of the service aids.
- **Messages and Codes, GC28-6631** -- describes the numbered messages issued by the service aids.

You should also be familiar with the following publications:

IBM System/360 Operating System:

- **Utilities, GC28-6586** -- describes how to use utility programs to print certain types of service aid output.
- **Operator's Reference, GC28-6691** -- describes how to perform certain basic operations, such as loading a stand-alone program.
- **Job Control Language Reference, GC28-6704** -- describes how to use job control statements to override default parameters, use cataloged procedures, allocate space for data sets, etc.
4 Service Aids (Release 21)
Introduction
Explains the service aid concept; guides selection of a service aid; summarizes the ways to retrieve service aids.

Chapter 1: IFCDIP00
Initializes the SYS1.LOGREC data set.

Chapter 2: IFCEREPO
Summarizes and prints records from the SYS1.LOGREC data set.

Chapter 3: GTF (Generalized Trace Facility)
Traces selected system events such as SVC and I/O interruptions.

Chapter 4: IMCJQDMP
Operates as a stand-alone program to format and print the system job queue.

Chapter 5: IMBLIST
Formats and prints object modules, load modules, and CSECT identification records.

Chapter 6: IMBMDMAP
Maps load modules.

Chapter 7: IMCOSJQD
Operates as a problem program to format and print the system job queue.

Chapter 8: IMDPRDMP
Formats and prints dumps, TSO swap data set, and GTF trace data.

Chapter 9: IMAPTFLE
Generates JCL needed to apply a PTF and/or applies the PTF.

Chapter 10: IMDSADMP
Operates as a stand-alone program to produce a high-speed or low-speed dump of main storage.

Chapter 11: IMASPZAP
Verifies and/or replaces instructions and/or data in a load module.

Appendix: Writing EDIT User Programs
Tells how to write and use EDIT user programs.

Each chapter has its own Table of Contents.
GENERAL COMMENTS

- Program Design information has been moved to a new publication, the Service Aids Logic PLM, GY28-6721.
- Information relating to the Primary Control Program (PCP) has been deleted.
- Chapter order has been revised to accommodate several new chapters.

INTRODUCTION

References to services aids as a SYSGEN option have been deleted.

CHAPTER 1: IFCDIP00

IFCDIP00 has been moved from the Utilities SRL and rewritten. All information concerning the PARM parameter of IFCDIP00 has been deleted.

CHAPTER 2: IFCEREPO

IFCEREPO has been moved from the Utilities SRL and rewritten. All information concerning SDR records has been deleted. Information for the Reliability Data Extractor (RDE) and MES has been added.

CHAPTER 3: GTF (THE GENERALIZED TRACE FACILITY)

GTF is a new feature of the operating system that executes as a problem program and is invoked by the START command. It traces selected system events, such as IO interruptions, SIO operations, program interruptions, etc. A special feature of GTF, the GTRACE macro instruction, allows you to record user data in GTF's output buffers. GTF output can be printed and formatted using the EDIT function of IMDPRDMP, which is described in Chapter 8.

CHAPTER 4: IMCJQDMP

This chapter is essentially unchanged.

CHAPTER 5: IMBLIST

IMBLIST is a new service aid that formats and prints object modules and CSECT Identification Records (IDRs) and maps load modules. It assumes the function of IMAPTFLS, a service aid which is no longer documented in this publication.

CHAPTER 6: IMBADMAP

This chapter is essentially unchanged.

CHAPTER 7: IMCOSJQD

IMCOSJQD is a new service aid that dumps the system job queue data set (SYS1.SYSJOBQE), or formats and prints selected records from it. IMCOSJQD is identical in function to the IMCJQDMP service aid, but IMCOSJQD executes as a problem program whereas IMCJQDMP is stand-alone.
CHAPTER 8: IMDPRDMP

IMDPRDMP now includes the EDIT function, which formats and prints GTF output. The parameters of the EDIT control statement, which invokes the EDIT function, allow you to select records to be formatted; some of the parameters are: JOBNAME=, IO=, EXT.

The EDIT function also provides interfaces for user-written exit routines and format appendages. Exit routines examine every trace record to determine how it should be handled. Format appendages format and print specific types of user records. Information about how to write exit routines and format appendages is provided in the Appendix: Writing EDIT User Programs.

CHAPTER 9: IMAPTFLE

IMAPTFLE now allows you to include a Linkage Editor IDENTIFY control statement in the IMAPTFLE input stream; this is required for the application function and optional for the generate function. The purpose of the IDENTIFY statement is to flag specific CSECTs that are to be updated with PTFs.

CHAPTER 10: IMDGADMP

This chapter is essentially unchanged.

CHAPTER 11: IMASPZAP

IMASPZAP now provides a control statement, IDRDATA, that allows you to update the CSECT Identification Record of any module that is successfully updated with a REP operation.

APENDIX: Writing EDIT User Programs

This appendix provides all the information you need to write an exit routine or a format appendage for use with the EDIT function of IMDPRDMP and the GTRACE macro instruction. It describes the interfaces with EDIT, illustrates the use of the IMDMEDIT mapping macro instruction, shows samples of both exit routines and format appendages, discusses ways to avoid unrecoverable errors, and describes how to debug an exit routine.

Please note that change bars are not used in any chapter described as "new" in this summary of amendments.
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Areas Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAPIPFLE Improvements</td>
<td>Permits IMAPIPFLE to apply a PTF to OS/360 directly.</td>
<td>235-241, 243-245</td>
</tr>
<tr>
<td>2305 and 3330</td>
<td>Permits service aids to be used with these devices.</td>
<td>188, 207, 251</td>
</tr>
<tr>
<td>Multiprocessing Support</td>
<td>Permits IMDSADMP to dump the contents of both CPUs of the IBM System/360 Model 65 Multiprocessing System.</td>
<td>159, 166, 173-177, 179, 180, 184, 185, 187-189, 191, 194, 195</td>
</tr>
<tr>
<td>IMBDMDAP Improvement</td>
<td>Message improvement.</td>
<td>318</td>
</tr>
<tr>
<td>IMAPIPFLE Improvement</td>
<td>Message improvement -- blocksize error.</td>
<td>243</td>
</tr>
</tbody>
</table>

Summary of Amendments 9
## Summary of Amendments for GC28-6719-1
### OS Release 20

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Pages Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Generation</td>
<td>Permits six service aids to be added to the operating system during system generation.</td>
<td>2,16,17,183,280</td>
</tr>
<tr>
<td>SVC Dump</td>
<td>Permits IMDPRDMP to format and print system dumps.</td>
<td>25,26,29,32,33,37-39</td>
</tr>
<tr>
<td>System/370</td>
<td>Permits service aids to be used with the IBM System/370.</td>
<td>173-175,178-182,185,187-191,194,195,251</td>
</tr>
<tr>
<td>TSO</td>
<td>Permits IMDPRDMP to format and print TSO dumps and swap data sets.</td>
<td>1,11,13,26,28,31,33,34,39,40,42,46,47,77,83,134-155</td>
</tr>
<tr>
<td>IMDGADMP Improvement</td>
<td>The address of the input dump device can be specified from the operator console.</td>
<td>180,185</td>
</tr>
<tr>
<td>Restriction</td>
<td>Release 20 IMAPTFLE will not process a Stage I output tape from a release before release 19.</td>
<td>241,243</td>
</tr>
<tr>
<td>IMDPRDMP Program Design</td>
<td>The &quot;IMDPRDMP Program Design&quot; section of the &quot;IMDPRDMP&quot; chapter has been rewritten.</td>
<td>25,56-71</td>
</tr>
<tr>
<td>PRDMP</td>
<td>The PRDMP PROCLIB procedure for calling IMDPRDMP has been documented with examples.</td>
<td>43,44</td>
</tr>
<tr>
<td>IMDPRDMP examples</td>
<td>Examples on how to use the IMDPRDMP control statements and PRDMP PROCLIB procedure are included in the &quot;IMDPRDMP&quot; chapter.</td>
<td>44-47</td>
</tr>
<tr>
<td>MFT QCE Trace</td>
<td>IMDPRDMP formats and prints QCB traces for MFT users.</td>
<td>28</td>
</tr>
<tr>
<td>ONGO Clarification</td>
<td>When the ONGO verb of an IMDPRDMP control statement has no parameters specified, the original GO parameters are restored: QCSTRACE, LPAMAP, FORMAT, and PRINT ALL.</td>
<td>39</td>
</tr>
<tr>
<td>65MP Clarification</td>
<td>Occasionally only one prefix is shown on an IMDPRDMP listing. This occurs when the dump is initiated on one CPU, interrupted and then dispatched to the other CPU.</td>
<td>41</td>
</tr>
<tr>
<td>Messages</td>
<td>All messages have been altered, where necessary, to agree with the publication IBM System/360 Messages and Codes, GC28-6631.</td>
<td>48-55,192,193,223-227,242,243,266-268,285,286,315-318</td>
</tr>
</tbody>
</table>

(Part 1 of 2)
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Pages Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Name</td>
<td>For dumps that are formatted and printed by IMDPRDMP, the name of the module that invoked the dump is printed in the header of the dump listing.</td>
<td>77-79,82,83,102,120, 135-137,156-158,160</td>
</tr>
<tr>
<td>Output Comments</td>
<td>Within the IMDPRDMP formatted dump, a number of output comments may be printed to assist in reading and interpreting the dump. These comments are explained.</td>
<td>77,83,161-167</td>
</tr>
<tr>
<td>IMAPTFLE Region Size</td>
<td>IMAPTFLE requires a 46K region or partition.</td>
<td>236</td>
</tr>
<tr>
<td>MFT LPA Maps</td>
<td>MFT link pack area maps do not include resident SVC routines (IMBADMAP).</td>
<td>314</td>
</tr>
<tr>
<td>Control Blocks</td>
<td>Various changes have been made to system control blocks that are formatted and printed by IMDPRDMP.</td>
<td>84,85,87,88,106,107</td>
</tr>
<tr>
<td>Tables and Examples</td>
<td>Table and example numbers have been converted to figure numbers. All figures have been renumbered. See the figure list in each chapter.</td>
<td>All</td>
</tr>
</tbody>
</table>

(Part 2 of 2)
Service aids are programs designed to help system programmers and IBM programming system representatives diagnose and fix failures in system or application programs. Service aids have three general functions:

**Information Gathering**
- To dump main storage, use the stand-alone program IMDSADMP. Its output can be formatted and printed using IMDPRDMP.
- To trace system events such as SVC and I/O interruptions, use GTF (the Generalized Trace Facility). Its output can be formatted and printed using the EDIT function of IMDPRDMP.

**Formatting and Printing: Mapping**
- To summarize and print records in the SYS1.LOGREC data set, use IFCREP0.
- To format and print load module, use IMBMMDMAP or IMBLIST.
- To format and print object modules and CSECT identification records, use IMBLIST.
- To format and print the system job queue, use IMCJQDMP (stand-alone) or IMCOSJQD (problem program).
- For format and print IMDSADMP output, other system dumps, TSO swap data sets, and GTF trace output, use IMDPRDMP.

**Generating and Applying Fixes**
- To apply a PTF, use IMDPTFLE.
- To verify and/or replace instructions and/or data in a load module, use IMASPZAP.
- To initialize the SYS1/LOGREC data set, use IFCDIPO0.

For more detailed information about choosing a service aid, refer to the table in figure INTRO-1.
<table>
<thead>
<tr>
<th>SYMPTOM</th>
<th>INFORMATION GATHERING</th>
<th>MAPPING, FORMATTING, AND PRINTING</th>
<th>PATCHING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IMDSADMP</td>
<td>GTF</td>
<td>IMCJQD (Stand-Alone)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMDPRDMP</td>
<td>IMAPTFLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMBLIST</td>
<td>IMASPZAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFCEREPO</td>
<td>IFCDIPOD</td>
</tr>
<tr>
<td>WARM START FAILURE</td>
<td>1</td>
<td>1c,1d</td>
<td></td>
</tr>
<tr>
<td>Scheduler ABEND</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Writer ABEND</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Problem Program ABEND</td>
<td>3</td>
<td>4a</td>
<td></td>
</tr>
<tr>
<td>Recursive ABEND</td>
<td>1</td>
<td>1e,1e-4</td>
<td></td>
</tr>
<tr>
<td>Disabled Loop</td>
<td>1</td>
<td>1e,1e-4</td>
<td></td>
</tr>
<tr>
<td>Problem Program Loop</td>
<td>3</td>
<td>4a</td>
<td></td>
</tr>
<tr>
<td>Large Loop w/ I/O</td>
<td>1</td>
<td>1e,1e-4</td>
<td></td>
</tr>
<tr>
<td>DAR Loop</td>
<td>1</td>
<td>1e,1e-4</td>
<td></td>
</tr>
<tr>
<td>Hard Wait</td>
<td>1</td>
<td>1e,1e-4</td>
<td></td>
</tr>
<tr>
<td>Enabled Wait</td>
<td>1</td>
<td>1b,4</td>
<td></td>
</tr>
<tr>
<td>Reader/Interpreter Failure</td>
<td>1</td>
<td>1b,4</td>
<td></td>
</tr>
<tr>
<td>I/O Failure (e.g., console)</td>
<td>1</td>
<td>1a,1e-4</td>
<td></td>
</tr>
<tr>
<td>Allocation Failure</td>
<td>1</td>
<td>1b-4</td>
<td></td>
</tr>
<tr>
<td>Enqueued Job Lost</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Chain Scheduling Problem</td>
<td>1</td>
<td>1a,1e-4</td>
<td></td>
</tr>
<tr>
<td>Access Method Failure</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Data Mgt., Program Check</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Module Level Unknown</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>User Modification Unknown</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Applying PTF</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Applying Local Fix</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>APAR Documentation</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Print SYS1.DUMP</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>TSO Failure</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
<tr>
<td>Capturing System Before Re-IPL</td>
<td>1</td>
<td>1b,1e-4</td>
<td></td>
</tr>
</tbody>
</table>

**INFORMATION GATHERING**

**IMDSADMP**
1. Dumps the contents of main storage to a tape, which can be formatted and printed using PRDMP. (Note that IMDSADMP output may also be directed to a printer.)

**OTF (Generalized Trace Facility)**
1. Traces all system events.
2. Traces selected events, such as I/O interruptions, SIO operations, etc.
3. Traces user programs with GTRACE macro instruction.

**MAPPING, FORMATTING, AND PRINTING**

**IMDPRDMP**
1. Formats and prints the following from SADMP high-speed output:
   a. Link Pack Area,
   b. Queue Control Block trace,
   c. Major Control Blocks,
   d. Selected Areas of Main Storage,
   e. Operating System Nucleus.
2. Formats and prints TSO control blocks and main storage from a SYS1.DUMP data set.
3. Formats and prints TSO Swap data set(s).
4. Formats and prints selected records from the GTF Trace data set or from trace buffers in a SYS1.DUMP or SADMP output data set. Records are selected by keywords such as:
   a. JOBNAME,
   b. I/O,
   c. SVC,
   d. SIO.

**IMBMDMAP**
1. Maps the operating system nucleus,
2. Maps a failing module,
3. Maps the link pack area.

**IFCEREPO**
Selects, formats, and prints error records in the SYS1,LOGREC data set.
1. Selects records by record type, such as:
   a. Machine check and/or inhibit
   b. Outboard
2. Selects records by device type or device address.
3. IFCEREPO can also select records by model number or time of creation.

**PATCHING**

**IMAPTFLE**
1. Generates control statements and JCL needed to apply PTFs; the application function also invokes the linkage editor.

**IMASPZAP**
Verifies or replaces instructions or data in a load module stored on a direct access storage device.
1. Modifies code or data in a load module.
2. Sets traps by inserting invalid instructions or user-written SVCs.
3. Dumps load modules by CSECT to allow examination of the text.
4. Dumps selected data to verify the count, key, and contents of the data.

14 Service Aids (Release 21)
All service aids except IMDSADMP and IMCJQDMP execute as problem programs under the operating system. They are automatically transferred to SYS1.LINKLIB during system generation. IMDSADMP and IMCJQDMP are stand-alone programs that must be retrieved from the distribution library before they can be used.

Retrieving IMCJQDMP

IMCJQDMP resides as an object module in distribution library SYS1.DN554A. Before you can load it into the system as a stand-alone program, you must retrieve it from the distribution library. To do this you can either transfer the module onto punch cards using the IEBPTPCH utility, or copy it to magnetic tape using the IEBGENER utility.

Retrieving IMDSADMP

IMDSADMP resides as a macro definition in distribution library SYS1.MACLIB. The easiest way to retrieve IMDSADMP is to specify the MACLIB macro instruction at system generation; IMDSADMP will automatically be transferred to the SYS1.MACLIB data set in the operating system.

If you choose not to create a SYS1.MACLIB data set at system generation, you can retrieve IMDSADMP by three other methods:

- If you want to retrieve IMDSADMP and execute it all in the same step, you can treat the distribution library as a private macro library. Figure INTRO-2 shows the job control statements needed to do this.
- You can copy IMDSADMP from the distribution library into a private library.
- You can punch IMDSADMP from the distribution library onto cards using the IEBPTPCH utility.

//ASMSAD JOB MSGLEVEL=(1,1)
//STEP EXEC ASMFC
//ASM.SYSLIB DD DSN=SYS1.MACLIB,DISP=OLD
//ASM.SYSIN DD *
IMDSADMP END
/*

Figure INTRO-2. Sample JCL Statements Need to Assemble IMDSADMP Directly from the Distribution Library
Chapter 1: IFCDIP00
Initializes the SYS1.LOGREC data set.
INTRODUCTION

INPUT TO IFCDIP00
The SYS.LOGREC Data Set
Job Control Statements

RUNNING IFCDIP00
Reinitializing SYS1.LOGREC
Changing Space Allocation for SYS1.LOGREC

Figures

Figure DIP00-1. Reinitializing the SYS1.LOGREC Data Set
Figure DIP00-2. Changing the Space Allocation for SYS1.LOGREC
IFCDIPO00 is a service aid that runs under the IBM System/360 Operating System. IFCDIPO00 has three applications:

1. Initializing the SYS1.LOGREC data set during system generation. This application is discussed in the publication IBM System/360 Operating: System Generation.

2. Reinitializing the SYS1.LOGREC data set. During processing, some types of errors may destroy the SYS1.LOGREC header and make the data set unusable; IFCDIPO00 can then be used to reinitialize the SYS1.LOGREC data set.

3. Modifying the space allocation for the SYS1.LOGREC data set. In some situations, the SYS1.LOGREC data set may be too large or too small for the system using it; IFCDIPO00 can then be used to increase or decrease the space allocation for SYS1.LOGREC.
Input to IFCDIP00

The input to IFCDIP00 consists of the SYS1.LOGREC data set and job control statements.

The SYS1.LOGREC Data Set

The SYS1.LOGREC data set consists of a header record followed by environment records.

The header record is created by IFCDIP00; it keeps track of the number and location of the environment records.

The environment records are generated by the outboard recording routine (OBR), the miscellaneous data recorder (MDR), the recovery management routines MCH and CCH, and the reliability data extractor program RDE, and the environment recording routines SER0 and SER1. Each record reflects the condition that prevailed in the system when an error occurred.

Job Control Statements

IFCDIP00 is run and controlled by job control statements; no user or utility control statements are needed.
Running IFCDIPOO

You run IFCDIPOO by providing job control language procedures to reinitialize and reallocate the SYS1.LOGREC data set. The following sections contain detailed examples of reinitializing and reallocating SYS1.LOGREC.

Reinitializing SYS1.LOGREC

Figure DIP00-1 is an example of the job control statements needed to reinitialize the SYS1.LOGREC data set using IFCDIPOO.

```plaintext
//INSERLOG JOB
// EXEC PGM=IFCDIPOO
//SERERDS DD DSN=SYS1.LOGREC,UNIT=2311,DISP=(OLD,KEEP),
//VOL=SER=111111
```

Figure DIP00-1. Reinitializing the SYS1.LOGREC Data Set

Control Statements for Figure 1

The JOB statement initiates the job; the job name INSERLOG has no significance.

The EXEC statement specifies the program name (PGM=IFCDIPOO).

The SERERDS DD statements specifies the output (SYS1.LOGREC) data set; the DSN=NAME must be SERERDS.

Changing Space Allocation for SYS1.LOGREC

IFCDIPOO may be used in conjunction with the IEHPROGM utility to increase or decrease the space allocated for the SYS1.LOGREC data set. First the SYS1.LOGREC data set is scratched and uncataloged, using IEHPROGM; then, using IFCDIPOO, the data set is reallocated with increased or decreased space specifications; and, finally, the newly allocated data set is reinitialized.

If you use the preceding procedure and an error occurs after the SYS1.LOGREC data set has been scratched, but before it has been reallocated, the IFCDIPOO job will be terminated and the system will be marked ineligible for IPL procedures. To solve this problem, do one of the following:

- Use the IBCMPRS utility to restore the system and thereby restore the SYS1.LOGREC data set. After the SYS1.LOGREC data set has been restored, you can reinitialize the system and reallocate SYS1.LOGREC.
- Execute the reallocate operation on another IBM System/360 Operating System, if one is available.
Figure DIP00-2 is an example of reallocating the SYS1.LOGREC data set.

```
//RELGREC JOB
//SCR EXEC PGM=IEHPROGM
//DDI DD UNIT=2311,VOLUME=SER=111111,DISP=OLD
//SYSIN DD *
   SCRATCH DSNAME=SYS1.LOGREC,VOL=2311=111111
   UNCATLG DSNAME=SYS1.LOGREC
/*
//R EXEC PGM=IFCDIP00
//SERERDS DD DSNAME=SYS1.LOGREC,UNIT=2311,DISP=(NEW,CATLG),
// VOL=SER=111111,SPACE=(allocation,CONTIG)
```

Figure DIP00-2. Changing the Space Allocation for SYS1.LOGREC
Chapter 2: IFCEREPO
Summarizes and prints records from the SYS1.LOGREC data set.
INTRODUCTION
Editing and Writing Selected Records
Accumulating Selected Records
Summarizing Selected Records
Processing Records Produced on Different Machine Models

INPUT TO IFCEREPO
Environment Records

RUNNING AND CONTROLLING IFCEREPO
Job Control Statements
Keyword Parameters for IFCEREPO

IFCEREPO EXAMPLES
Example 1: Printing Machine Check Records
Example 2: Writing Machine Check Records onto a 7-Track Magnetic Tape
Example 3: Printing and Accumulating Machine Check and Channel Inboard Records
Example 4: Printing and Accumulating Machine Check Records Contained in a History Data Set
Example 5: Printing Recently Generated Machine Check Records and Accumulated Machine Check Records

IFCEREPO OUTPUT
Format of Edited Records

Figures

Figure EREPO-1. Job Control Statements
Figure EREPO-2. Output Record Printout Structure
Figure EREPO-3. Sample Printout -- Outboard Data Editing and Printing Section
Figure EREPO-4. First Sample Printout -- TCAM Data Editing and Printing Section
Figure EREPO-5. Second Sample Printout -- TCAM Data Editing and Printing Section
Figure EREPO-6. Machine-Check Summary
Figure EREPO-7. Channel Inboard Summary
Figure EREPO-8. I/O Outboard Summary
Figure EREPO-9. TCAM I/O Outboard Summary
Introduction

IFCEREPO is a service aid that runs under the IBM System/360 Operating System. You can use IFCEREPO to:

- Select and format environment records from the SYS1.LOGREC data set and write them to an output device. The environment records on the SYS1.LOGREC data set are generated by the error environment recording programs OBR, SERO, SER1, MDR, by the recovery management programs CCH, and MCH, and by the reliability data extractor program RDE.
- Select environment records from the SYS1.LOGREC data set and accumulate them on a history data set.
- Write the records accumulated on the history data set to an output device.
- Summarize the information contained in the records on the SYS1.LOGREC data set or the history data set.
- Process (edit, write, accumulate, and summarize) records produced on different machine models.

Editing and Writing Selected Records

You can use IFCEREPO to retrieve selected environment records from the SYS1.LOGREC data set or a history data set, edit them, and write them to an output device. After the record is written to the output device, it is cleared to hexadecimal zeros on the SYS1.LOGREC data set unless you specify otherwise. If the input data set is the history data set, the records remain unchanged. The selection of records that IFCEREPO will process is based on the following factors:

- Record type: you can specify any type of environment record, or any combination of types.
- Model number: you can specify the model number of any computing system that is writing records on the SYS1.LOGREC data set; this specification is useful when several computing systems are writing records on the same SYS.LOGREC data set.
- Time period: you can specify that IFCEREPO only process records that were generated on certain dates.
- Devices: you can specify that IFCEREPO process records that are related to a specific device or device type.

Accumulating Selected Records

You can use IFCEREPO to move selected environment records from the SYS1.LOGREC data set to a history data set; this enables you to accumulate specific types of environment records on different volumes or on the same volume. When you move an environment record from the SYS1.LOGREC data set to the history data set, the environment record or the SYS1.LOGREC data set, is cleared to hexadecimal zeros unless you specify otherwise.
Summarizing Selected Records

You can use IFCEREPO to extract pertinent data from selected records and print the data in the form of a summary. The contents of the summary depend on the type of error you monitor.

Processing Records Produced on Different Machine Models

You can use IFCEREPO to edit, write, summarize, and accumulate environment records for any IBM System/360 or IBM System/370 model that supports the IBM System/360 Operating System. In addition, any SYS1.LOGREC data set or history data set generated on one system can be printed on another system.
Input to IFCEREPO

The input to IFCEREPO consists of environment records located on the SYS1.LOGREC or history data sets.

Environment Records

You can use IFCEREPO to process six types of environment records:

1. Machine check records - which are produced and stored in SYS1.LOGREC by the system environment recorders SER0 and SER1, and by the machine check handler (MCH). They record machine check interruptions caused by malfunctions in the central processing unit.

2. Inboard records - which are produced and stored by SER0, SER1, and by the channel check handler (CCH). They record input/output interruptions caused by specific channel failures.

3. Outboard records - which are produced and stored by the outboard recorder. They record permanent errors on input/output devices, and terminal statistics and errors for TCAM.

4. Miscellaneous data records - which are produced and stored by the miscellaneous data recorder (MDR). They record errors that are not reflected in any other record type.

5. System initialization (IPL) records - which are produced and stored in the SYS1.LOGREC data set by the reliability data extractor (RDE) programs. They record information related to each system initialization.

6. System termination (EOD) records - which are produced and stored in the SYS1.LOGREC data set by the reliability data extractor (RDE) programs. They record information related to each system termination. For a complete explanation of RDE see the publication, IBM System/360 Operating System: RDE Guide, GC28-6741.
Running and Controlling IFCEREPO

You run and control IFCEREPO by job control statements and by specifying keyword parameters on the EXEC statement of your IFCEREPO procedure; no user or utility control statements are needed.

Job Control Statements

IFCEREPO Figure 1 shows the job control statements necessary for running IFCEREPO.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB Statement</td>
<td>This statement initiates the job.</td>
</tr>
<tr>
<td>EXEC Statement</td>
<td>This statement specifies the program name and keyword parameters necessary to control the function of the program.</td>
</tr>
<tr>
<td>SERLOG DD Statement</td>
<td>This statement defines the input data set as being the SYS1.LOGREC data set. Either a SERLOG DD statement or the ACCIN DD statement must be included for each application of the IFCEREPO program.</td>
</tr>
<tr>
<td>ACCIN DD</td>
<td>This statement defines the input data set as being a history data set. Either an ACCIN DD statement or the SERLOG DD statement must be included for each application of the IFCEREPO program.</td>
</tr>
<tr>
<td>EREPPT DD Statement</td>
<td>This statement defines the edited output data set. It must be included with each application of the program.</td>
</tr>
<tr>
<td>ACCDEV DD Statement</td>
<td>This statement defines an accumulated output data set. The accumulated data set can reside on a direct access device. Space must be allocated for a new output data set that is to reside on a direct access volume. Space cannot be allocated for an existing output data set.</td>
</tr>
</tbody>
</table>

Notes: The SERLOG, ACCIN, EREPPT, and ACCDEV DD statements define sequential data sets.

If records produced on different machine models are to be processed, a JOBLIB DD statement is required to define the original system's link library.

Figure EREP0-1. Job Control Statements
Keyword Parameters for IFCEREPO

You can specify the following keyword parameters to control the functions of the IFCEREPO program.

\[
\text{PARAM} = \begin{cases} 
\text{TYPE} = [M][C][O][T][I][E], \\
\text{MOD} = (\text{nnn}[,\text{nnn}...]), \\
\text{MES} = \{N,Y\}, \\
\text{VOLID} = (\text{VOLID1, VOLID2, VOLID3, VOLID4}) \\
\text{CUA} = (\text{CUU}[,\text{CUU}]), \\
\text{DEV} = \text{NNNN}, \\
\text{DATE} = ([\text{YYDDD}][,\text{YYDDD}]), \\
\text{ZERO} = \{N,Y\}, \\
\text{PRINT} = \{\text{PS, PT, SU, NO}\}, \\
\text{ACC} = \{Y,N\}, \\
\text{HIST} = \{N,Y\}, \\
\text{TERMN} = 1-8 \text{ chars}, \\
\text{M67} = \{1,2\}, \\
\text{RDESUM} = \{N,Y\}.
\end{cases}
\]

**TYPE**

specifies the type of records to be processed.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Machine-check records</td>
</tr>
<tr>
<td>C</td>
<td>Channel inboard records</td>
</tr>
<tr>
<td>O</td>
<td>I/O outboard records</td>
</tr>
<tr>
<td>T</td>
<td>T-type records</td>
</tr>
<tr>
<td>I</td>
<td>IPL Records</td>
</tr>
<tr>
<td>E</td>
<td>EOD Records</td>
</tr>
</tbody>
</table>

A combination of records can be specified. For example, PARM=(TYPE=MC,...). If no record type is specified, all record types are processed.
MOD

indicates that all records created on the model or models specified are to be processed. The operand is to be right justified and may be up to three digits in length.

MES

indicates that error statistics for specific volume/serials are to be summarized and printed. This parameter is valid only for the 3410 and 3420 tape subsystems, when "TYPE=O" is coded, or when no record type is specified.

VOLID

indicates specific volumes for error statistics (MES) processing. A maximum of four volumes can be specified. If this parameter is not coded and MES=Y is coded, all volumes will be processed.

If no model numbers are specified all models are accepted for processing.

CUA (maximum of two)

indicates that the selected record types that are related to the specific channel(s) and unit(s) are to be processed.

DEV (maximum of one)

indicates that selected record types that are related to a specific device type are to be processed.

if DEV is not specified, all selected records (as specified in the TYPE subparameter) are processed regardless of the device type.

if DEV =3410 or DEV =3420 is specified, both devices will be included in the report.

DATE (maximum of one set)

indicates that all of the selected record types generated within a specific period of calendar time are to be processed. The date is written yyddd yyddd where yyddd represents the year and the day (of the year) when the time period begins and yyddd represents the year and the day when the period ends.

If no date is specified, all selected records are processed regardless of when they were generated.

ZERO

indicates whether input records in the SYS1.LOGREC data set are to be cleared with hexadecimal zeros after they are processed. Records are not cleared to zeros in the history data set.

Note: It is possible to use the same operating system on several machines. Before moving the system packs to another machine, the operator must use the EREP program to copy the SYS1.LOGREC data set to tape so that the environmental data can later be related to the system that generated it.
PRINT

indicates how records are to be processed and written.

Code   Meaning
SU   Suppress full printing (print summary only).
PT   Suppress summary printing (print full record only).
NO   Suppress full printing and summary printing.
PS   Print full record and summary.

ACC

indicates whether selected records are to be accumulated in a history data set. If ACC=Y is coded; ZERO=Y must be coded if the input data set is SYS1.LOGREC.

HIST

indicates whether the input data set is a history data set. If HIST=Y is coded, the input data set must be defined with an ACCIN DD statement.

If HIST is not coded HIST=N is assumed and the input data set will be the SYS1.LOGREC data set.

TERMN

indicates the OBR and TCAM records are to be selected by terminal name. Up to eight characters may be specified.

If TERMN is not coded all terminal names are selected.

M67

indicates which Model 67 records are to be processed.

If M67 is not coded mod 1 Model 67 records are processed.

RDESUM

indicates that the IFCEREP0 summary function for RDE records is to be run. The summary function produces an IPL report and a hardware error report. This parameter can be coded only if RDE has been selected during system generation. For a complete explanation of RDE see the publication IBM System/360 Operating System: RDE Guide, GC28-6741.
IFCEREPO Examples

The following examples show some of the typical uses of the IFCEREPO program.

Example 1: Printing Machine Check Records

In this example:

- Machine check records are printed in a full record format.
- The records on SYS1.LOGREC are zeroed.

```
//JOBA
// JOB
// EXEC PGM=IFCEREPO,PARM='TYPE=M,ZERO=Y,PRINT=PT,ACC,N'
//SERLOG DD DSN=SYS1.LOGREC,DISP=(OLD,KEEP)
//EREPT DD SYSOUT=A
```

Control Statements for Example 1

The EXEC statement specifies (1) that machine check records are to be processed, (2) the type of printout (full record), (3) no accumulation is to take place.

The SERLOG DD statement defines the input (SYS1.LOGREC) data set.

The EREPPT DD defines the edited output data set (printer assumed).
Example 2: Writing Machine Check Records onto a 7-Track Magnetic Tape

In this example:

- Date-dependent machine check records are written in full record and summary formats onto a 7-track magnetic tape at a density of 200 bits per inch.
- The SYS1.LOGREC data set is zeroed.

```
//JOB
//SERLOG
//EREPT
//JOB
EXEC PGM=IFCEREPO,PARM='TYPE=M,DATE=(62110,62117),ZERO=Y,PRINT=PS,ACC=N'
/DD DNAME=SYS1.LOGREC,DISP=(OLD,KEEP)
/DD DNAME=ERRDATA,UNIT=2400-2,LABEL=(),NL,
     DCB=(DEN=0,TRTCH=C),DISP=(NEW,CATLG)
/*
```

Control Statements for Example 2

The EXEC statement specifies (1) that machine check records are to be processed, (2) the type of printout (full record and summary), (3) the applicable time period, and (4) that no accumulation is to take place.

The SERLOG DD statement defines the input (SYS1.LOGREC) data set.

The EREPPT DD statement defines the output data set.
Example 3: Printing and Accumulating Machine Check and Channel Inboard Records

In this example:

- Machine check and channel inboard records are printed in a full record and summary format.
- Machine check and channel inboard records are accumulated on a history data set.
- The records on SYS1.LOGREC are zeroed.

```bash
//JOB        JOBA
//          EXEC PGM=IFCEREPO,PARM='TYPE=MC,ACC,Y,PRINT=PS,ZERO=Y'
//SERLOG     DD DSNNAME=SYS1.LOGREC,DISP=(OLD,KEEP)
//EREPPT     DD SYSOUT=A
//ACCDEV     DD DSNNAME=ACUMSET,UNIT=2311,DISP=(NEW,CATLG),
            VOLUME=SER=111112,SPACE=(TRK,(40,10))
/*
```

Control Statements for Example 3

The EXEC statement specifies (1) that machine check and channel inboard records are to be processed, (2) the type of printout (full record and summary), and (3) accumulation on a history data set.

The SERLOG DD statement defines the input (SYS1.LOGREC) data set.

The EREPPT DD statement defines the output data set.

The ACCDEV DD statement defines the accumulated (history) output data set. The set is cataloged for ease of retrieval.
Example 4: Printing and Accumulating Machine Check Records Contained in a History Data Set

In this example:

- Machine check records in the history data set are printed in a full record format.
- Machine check records in the history data set are moved to a second (output) history data set.

```
//JOB        JOBA
// EXEC       PGM=IFCEREPO,PARM='TYPE=M,HIST=Y,PRINT=PT,ACC=Y'
//ACCIN DD    DSN=HISTRYIN,DISP=(OLD,CATLG)
//EREPT DD    SYSOUT=A
//ACCDEV DD   DSN=EXISTACC,DISP=(MOD,CATLG)
/*
```

Control Statements for Example 4

The EXEC statement specifies (1) that machine check records are to be processed, (2) a history data set is the input data set, (3) the type of printout full record, and (4) accumulation.

The ACCIN DD statement defines the input (history) data set.

The EREPPT DD statement defines the output data set.

The ACCDEV DD statement defines the accumulated (history) output data set.
Example 5: Printing Recently Generated Machine Check Records and Accumulated Machine Check Records

This example is a two-step job. Together the job steps produce a printout of machine check records from the SYS1.LOGREC data set and machine check records from a history data set.

In the first job step (STEP1):

- Machine check records on SYS1.LOGREC are edited and printed in a full record format.
- Machine check records on SYS1.LOGREC are accumulated on a history data set.
- The records on SYS1.LOGREC are zeroed.

In the second job step (STEP2):

- Machine check records in the history data set, updated in STEP1, are printed in a full record format.

//JOB JOB
//STEP1 EXEC PGM=IFCEREPO,PARM='TYPE=M,PRINT=PT,ACC=Y,ZERO=Y'
//SERLOG DD DSNAME=SYS1.LOGREC,DISP=(OLD,CATLG)
//EREPTT DD SYSOUT=A
//ACCDEV DD DSNAME=HISTORY,DISP=(MOD,CATLG)
/
//STEP2 EXEC PGM=IFCEREPO,PARM='TYPE=M,PRINT=PT,HIST=Y,ACC=N'
//ACCIN DD DSNAME=HISTORY,DISP=(OLD,CATLG)
//EREPTT DD SYSOUT=A
/

Machine Records (for comparison)

Control Statements for Example 5

STEP1

The EXEC statement specifies (1) that machine check records are to be processed, (2) the type of printout, and (3) accumulation.

The SERLOG DD statement defines the input (SYS1.LOGREC) data set.

The EREPPT DD statement defines the output data set.

The ACCDEV DD statement defines the accumulation (history) data set.

STEP2

The EXEC statement specifies (1) that machine check records are to be processed, (2) a history data set is the input data set, (3) the type of printout (full record), and (4) no accumulation.

The ACCIN DD statement defines the input (history) data set.

The EREPPT DD statement defines the output data set.
You can use IFCEREPO to write output to any output device supported by the basic sequential access method (BSAM). The output is written as 120-byte records with a control character as the first character of each record. After the records are written to the output device, they are normally cleared to hexadecimal zeros in the SYS1.LOGREC data set; the space occupied by the cleared records cannot be reused until the entire SYS1.LOGREC data set is cleared. You can, however, specify that the records remain uncleared in your procedure for running IFCEREPO.

Format of Edited Records

Figure EREPO-2 shows the printed format of an edited output record.

<table>
<thead>
<tr>
<th>Program heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td>Record type</td>
</tr>
<tr>
<td>Record data</td>
</tr>
<tr>
<td>Additional data</td>
</tr>
</tbody>
</table>

Figure EREPO-2. Output Record Printout Structure

Program heading identifies the IFCEREPO program on the first page of the listing:

- ENVIRONMENT RECORD EDITING AND PRINTING PROGRAM

Program section identifies the program section that is generating the printout. Valid program sections are:

- CPU (MC) DATA EDITING AND PRINTING SECTION
- INBOARD DATA EDITING AND PRINTING SECTION
- OUTBOARD DATA EDITING AND PRINTING SECTION
- MDR DATA EDITING AND PRINTING SECTION
- TCAM OUTBOARD DATA EDITING AND PRINTING SECTION

Model identifies the IBM System/360 Model or System/370 for which the printout is applicable. Valid entries are:

- Model 40, 50, 65, 75, 85, 91, 95, 135, 145, 155, 165, or 195 for channel inboard records. (Model 67 and 95 channel inboard records appear as Model 65 and 91 records, respectively.)
Note: SER can produce channel inboard records on any of the SER supported models. CCH can produce channel inboard records on IBM Systems/360 Models 65, 75, 85, 91, and 195, and IBM System/370 Models 135, 145, 155 and 165. The channel recording facilities of some MCH programs can produce channel inboard records on these models when CCH is not in the system or when the channel error cannot be recorded by CCH.

Source

identifies the error environment or recovery management program that generated the record placed in the SYS1.LOGREC data set. Valid sources are:

- RECORD ENTRY SOURCE - OBR
- RECORD ENTRY SOURCE - SER0
- RECORD ENTRY SOURCE - SER1
- RECORD ENTRY SOURCE - MCH
- RECORD ENTRY SOURCE - CCH
- RECORD ENTRY SOURCE - MDR

Record type

indicates the type of printout. Valid types are:

- TYPE - CPU
- TYPE - INBOARD
- TYPE - OUTBOARD
- TYPE - MDR

Record data

is a listing of the edited record from the input data set. This data, which constitutes the bulk of the printout, is the programming data and machine data collected at the time of the error.

Additional data

is a listing of records that were recorded in the SYS1.LOGREC data set while the program was being executed.

The heading:

- THE FOLLOWING RECORDS WERE GENERATED WHILE EXECUTING EREP

is followed by a printout of the records.

Figure EREP0-3 shows a sample outboard printout of an environment record that was processed by the outboard data editing and printing section of the utility program. The record was generated by the OBR program on an IBM System/360 Model 30, 40, 50, 65, 67, 75, 85, 91, 95, or 195 and on an IBM System/370 Model 135, 145, 155, or 165 (indicated by UNIVERSAL in the printout). The device failure occurred on a 2311 disk with a channel and unit address of 190.
Figures EREPO-4 and EREPO-5 show samples of a TCAM outboard printout of an environment record that was processed by the TCAM outboard data editing and printing section. The record was generated by the OBR program on an IBM System/360 Model 30, 40, 50, 65, 67, 85, 91, or 195, and on an IBM System/370 Model 135, 145, 155, or 165 (indicated by UNIVERSAL in the printout).

Note: The format for the MDR record is variable and requires special editing modules from the specific sub-types. Because of this variation, no sample printouts are shown for MDR record editing.
---RECORD ENTRY TYPE - UNIT CHECK
OS RELEASE xxx
SOURCE- OUTBOARD
DAY YEAR HH MM SS.TH
DATE- xxx xx TIME- xx xx xx xx
MODEL- xxx SERIAL NO. xxxxxx
JOB IDENTITY xxxxxxx
xxxxxxxxxxxxxxxx
OBR RECORD CONVERTED TO THE STANDARD FORMAT
MULTIPROCESSOR - CPU xx

DEVICE TYPE xxx
PRIMARY CHANNEL UNIT ADDRESS xxx
ALTERNATE CHANNEL UNIT ADDRESS xxx
PHYSICAL DRIVE x
PHYSICAL CONTROL UNIT x
VOLUME LABEL xxx

FAILING CCW xx x x x x
K CA US CS CT
CSW xx x x x x

LAST SEEK ADDRESS x x x x x x

UNIT STATUS

<table>
<thead>
<tr>
<th>UNIT ADDRESS</th>
<th>CHANNEL STATUS</th>
<th>STATISTICAL DATA</th>
<th>STATISTICAL DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
<td>x x x x x x x x</td>
</tr>
</tbody>
</table>

SENSE BYTE DATA

<table>
<thead>
<tr>
<th>SENSE BYTE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>x x x x x x x x</td>
</tr>
</tbody>
</table>

Figure EREP0-3. Sample Printout -- Outboard Data Editing and Printing Section
Figure EREPO-4. First Sample Printout -- TCAM Data Editing and Printing Section

Figure EREPO-5. Second Sample Printout -- TCAM Data Editing and Printing Section
Machine-Check Summary: A machine-check summary can be generated on IBM System/360 Models 40, 50, 65, 67, 85, 91, 95, and 195, and on IBM System/370 Model 135, 145, 155, 165. A summary consists of:

- Items that provide clues as to the type of machine malfunction.
- Parity information for registers in the diagnostic scan-out area (logout area), general purpose registers, and floating point registers.
- The status of binary triggers recorded in the logout area.

Notes: For the model 85, only the error triggers are summarized.

Figure EREPO-6 shows the format of a machine-check summary. Each summarized item is listed with its frequency of occurrence.

<table>
<thead>
<tr>
<th>MOD xx MACHINE-CHECK SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF RECORDS EXAMINED = 10</td>
</tr>
<tr>
<td>TITLE</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>ROBAR SUMMARY (UP TO FIRST 10)</td>
</tr>
<tr>
<td>AAAAA</td>
</tr>
<tr>
<td>BBBB</td>
</tr>
<tr>
<td>CCC</td>
</tr>
<tr>
<td>LOGOUT REG PARITY CHECK SUMMARY</td>
</tr>
<tr>
<td>REG A</td>
</tr>
<tr>
<td>REG B</td>
</tr>
<tr>
<td>REG C</td>
</tr>
<tr>
<td>CHECKS AND INDICATORS SUMMARY</td>
</tr>
<tr>
<td>ROAR CHECK</td>
</tr>
<tr>
<td>LSAR PTY CHECK</td>
</tr>
<tr>
<td>H DECODE CHECK</td>
</tr>
<tr>
<td>D/8 CHECK</td>
</tr>
</tbody>
</table>

Figure EREPO-6. Machine-Check Summary

Channel Inboard Summary: A channel inboard summary can be generated on IBM System/360 models 40, 50, 65, 75, 85, 91, and 195 and IBM System/370 Model 165. (Model 67 and Model 95 channel inboard summaries are identified as Model 65 and 91 summaries, respectively.) Channel inboard records are summarized according to channel address. Each channel summary contains:

- The addresses of devices connected to the channel (a maximum of 10 devices).
- The status of hardware elements (pertaining to the channel) in the logout area.
- A summary of failing CCW command codes (a maximum of 24 entries). (The 24th CCW command code entry is a logical OR of the remainder of the failing command codes, if any.)
Figure EREPO-7 shows the format of channel inboard summary. Each summarized item is listed with its frequency of occurrence.

Table:

<table>
<thead>
<tr>
<th>Title</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SUMMARY OF DEVICE ADDRESSES</strong>&lt;br&gt;(MAX 10 ENTRIES)</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>5</td>
</tr>
<tr>
<td>190</td>
<td>6</td>
</tr>
<tr>
<td>1FO</td>
<td>5</td>
</tr>
<tr>
<td>UNDET.</td>
<td>4</td>
</tr>
<tr>
<td><strong>SUMMARY OF CMND CODES</strong>&lt;br&gt;(MAX 24 ENTRIES)</td>
<td></td>
</tr>
<tr>
<td>CMND CODES</td>
<td>TOTAL</td>
</tr>
<tr>
<td>'01'</td>
<td>7</td>
</tr>
<tr>
<td>'02'</td>
<td>6</td>
</tr>
<tr>
<td>'03'</td>
<td>3</td>
</tr>
<tr>
<td>'14'</td>
<td>4</td>
</tr>
<tr>
<td><strong>SUMMARY OF HARDWARE LOGOUT</strong></td>
<td></td>
</tr>
<tr>
<td>IF PARITY</td>
<td>8</td>
</tr>
<tr>
<td>LWR WR</td>
<td>6</td>
</tr>
<tr>
<td>IF TAG CHK</td>
<td>2</td>
</tr>
<tr>
<td>WO PARITY CHK</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure EREPO-7. Channel Inboard Summary

I/O Outboard Summary: An I/O outboard summary can be generated on IBM System/360 Models 30, 40, 50, 67, 75, 85, 91, 95, and 195, and IBM System/370 Models 135, 145, 155 and 165. I/O outboard summaries are organized according to device address; however, the order of appearance of the summaries is determined by the order in which device addresses are encountered in the OBR records selected for summarization. Where TCAM is used the summary will appear in CUA (channel unit address) and line (terminal name) sequence. Each I/O outboard summary contains:

- Volume labels (a maximum of 10 entries).
- A summary of failing CCW command codes (a maximum of 24 entries). (The 24th CCW command code entry is a logical OR of the remainder of the failing command codes, if any.)
- The sense bits (a maximum of 6 bytes)

Note: Selected records can be edited and written, accumulated, and/or summarized in one execution of the program.

Figure EREPO-8 shows the format of an I/O outboard summary. Each summarized item is listed with its frequency of occurrence.

Figure EREPO-9 shows the format of the TCAM I/O outboard summary. All totals reference the CUA/line. All subtotals reference terminal names. Individual errors appear under their type of error for every terminal. Graphic errors always appear on the third line under their type of error.
SUMMARY OF I/O OUTBOARD ENVIRONMENT RECORDS FOR DEVICE 031

TOTAL NUMBER OF RECORDS 005

DEVICE TYPE 2311

VOLUME LABELS ENCOUNTERED (MAXIMUM OF 10 ENTRIES)

<table>
<thead>
<tr>
<th>VOL. LABEL</th>
<th>22222</th>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOL. LABEL</td>
<td>22222</td>
<td>002</td>
</tr>
<tr>
<td>VOL. LABEL</td>
<td>22224</td>
<td>002</td>
</tr>
</tbody>
</table>

CCW COMMAND CODES ENCOUNTERED (MAXIMUM OF 24 ENTRIES)

<table>
<thead>
<tr>
<th>CMND</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>005</td>
</tr>
</tbody>
</table>

SENSE BYTE SUMMARY

<table>
<thead>
<tr>
<th>BYTE 0</th>
<th>BYTE 1</th>
<th>BYTE 2</th>
<th>BYTE 3</th>
<th>BYTE 4</th>
<th>BYTE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMND REJ</td>
<td>0</td>
<td>DATA CHK</td>
<td>0</td>
<td>UNSAFE</td>
<td>1</td>
</tr>
<tr>
<td>INTV REQ</td>
<td>0</td>
<td>TRK OVERF</td>
<td>0</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>BUS OUT</td>
<td>0</td>
<td>CYL END</td>
<td>1</td>
<td>SERIAL CH</td>
<td>3</td>
</tr>
<tr>
<td>EQUIP CHK</td>
<td>0</td>
<td>INV SEQ</td>
<td>2</td>
<td>TAG LINE</td>
<td>4</td>
</tr>
<tr>
<td>DATA CHK</td>
<td>0</td>
<td>REC UNFND</td>
<td>0</td>
<td>ALU CHK</td>
<td>0</td>
</tr>
<tr>
<td>OVERRUN</td>
<td>0</td>
<td>FILE PROT</td>
<td>3</td>
<td>UNSEL STA</td>
<td>0</td>
</tr>
<tr>
<td>TRK COND</td>
<td>0</td>
<td>MISG A MK</td>
<td>4</td>
<td>BIT</td>
<td>6</td>
</tr>
<tr>
<td>SEEK CHK</td>
<td>1</td>
<td>OVFL INC</td>
<td>5</td>
<td>BIT</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure EREP0-8. I/O Outboard Summary
---RECORD ENTRY TYPE - UNIT CHECK  SOURCE - OUTBOARD  MODEL-  SERIAL NO. XXXXXX
OS RELEASE  xxx
DAY YEAR  HH MM SS TH  JOB IDENTITY  XXXXXXXX
DATE-  xxx xx  TIME-  xx xx xx xx

OBR RECORD CONVERTED TO THE STANDARD FORMAT  MULTIPROCESSOR - CPU  xx

DEVICE TYPE  XXXX
PRIMARY CHANNEL UNIT ADDRESS  XXXX
ALTERNATE CHANNEL UNIT ADDRESS  XXXX
PHYSICAL DRIVE  x
PHYSICAL CONTROL UNIT  x
VOLUME LABEL  xxxxxx

CC CA FL CT  K CA US CS CT
FAILING CCW  xx XXXXXX xx xx XXXX  CSW  xx XXXXXX xx xx XXXX
M B B C C H H R
LAST SEEK ADDRESS  x  XXXX XXXX XXXX XX

UNIT STATUS  CHANNEL STATUS  STATISTICAL DATA  STATISTICAL DATA
YYYYYYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYYYYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYYYYYY YYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYYYYYY YYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX
YYYY YYYY  x  YYYYYYYYYYYY YYYY YYYY YYYY YYYY XXXX  YYYY YYYY YYYY YYYY XXXX

SENSE BYTE DATA

<table>
<thead>
<tr>
<th>BYTE 0</th>
<th>BYTE 1</th>
<th>BYTE 2</th>
<th>BYTE 3</th>
<th>BYTE 4</th>
<th>BYTE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY YYYYYYYY  x  YYYY YYYY YYYY  x  XXXXXXXX  YYYY YYYY YYYY  x  YYYY YYYY YYYY  x  XXXXXXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY  x  YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x  YYYY YYYY YYYY YYYY YYYY  x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure EREPO-9. TCAM I/O Outboard Summary
Chapter 3: GTF (Generalized Trace Facility)
Traces selected system events such as SVC and I/O interruptions.
The Generalized Trace Facility (GTF) is a feature of OS/360 that allows you to trace selected system events. It also allows you to create your own user trace records and include them in the trace output. The trace output, when formatted and printed by the EDIT function of IMDPRDMP, is useful in determining and diagnosing problems that may arise while using the operating system.
Features

GTF operates as a system task under the operating system; it is compatible with all configurations of the operating system. If the TRACE option has been selected at system generation, the OS Trace facility will function normally except during GTF processing, when OS Trace processing will be suspended.

GTF can trace any or all of the following system events:

- Input/output interruptions (IO)
- START I/O operations (SIO)
- Supervisor Call interruptions (SVC)
- Program interruptions (PI) (including SSM)
- External interruptions (EXT)
- Dispatcher task-switch operations (DSP)

If you choose IO or SIO, you can supply specific device names in response to a prompting message; GTF will then filter out all IO or SIO events that are not associated with the devices you specified. Similarly, you can supply specific SVC numbers when you choose SVC tracing, and specific program interrupt codes when you choose PI tracing.

GTF will ordinarily ignore traceable events that are associated with its own task, but you can request that such events be included as part of the trace output (TRC). You can also request that a timestamp be included in each trace record (TIME=YES).

GTF trace output can be maintained in main storage (MODE=INT) or directed to a data set on an external storage device (MODE=EXT). The output device may be any magnetic tape or direct access device supported by the operating system.

If data is maintained internally or written to a direct access output device, it is "wrapped". That is, when the buffers or available tracks become full, GTF will overlay previously stored or written information beginning at the first buffer or block.

Any abnormally terminating user who has requested ABEND processing will be supplied with formatted trace data as part of the ABEND dump if GTF was active with MODE=INT when ABEND was given control. Similarly, trace data will be provided for SNAP dumps if the user has included the SDATA=TRT parameter in the SNAP macro.
Use the START command to initiate GTF processing. By specifying certain optional parameters, you can choose whether the trace records should be recorded internally or externally, whether or not they should be time-stamped, and whether or not GTF should terminate if it encounters errors while gathering trace information. You can also select trace options, either by entering them directly through the console or by retrieving them from SYS1.PARMLIB where you have stored them.

Using the START Command

Figure GTF-1 shows the general format of the START command as it is used to start GTF.

```
START procnamel.identifierl,[devaddr],[volser],[parmvalue]
[,keyword=option1,...,keyword=option1[,REG=size]]
```

Figure GTF-1. General Format of the Start Command for GTF

The following discussion describes the parameters of the START command as they are used for GTF.

```
procname

devaddr

volser

parmvalue
```

defines one of the two cataloged procedures (GTF and GTFSNP) described in the next section.

indicates the address of the device to which trace output is to be written, if you have specified MODE=EXT. If you have specified MODE=INT, omit this field.

defines the volume serial number of the direct access storage pack to which trace output is to be written, if you have specified MODE=EXT. If you specified MODE=INT, omit this field.

overrides the value specified in the PARM= parameter of the EXEC statement in the cataloged procedure GTF or GTFSNP. This field may contain any combination of the following parameters:

```
MODE=(INT
  )
```

defines where the trace data is to be maintained. If you omit this parameter, GTF will assume the default specified in the cataloged procedure (MODE=EXT) and write the trace data on the SYS1.TRACE data set. When MODE=EXT is in effect, you will be prompted to supply trace options unless you have specified a member of SYS1.PARMLIB where trace options are stored.
When MODE=INT is in effect, the trace data is maintained in main storage, and GTF will not prompt you to supply trace options. It will gather basic data (similar to that contained in the OS trace table) for the following events:

- Dispatcher entries
- External interrupts
- I/O interrupts, including program-controlled interrupts.
- Program interrupts
- SIO operations
- SVC interrupts.

When any task in the system terminates abnormally and the ABEND routine is invoked, GTF will suspend tracing until the ABDUMP program can format the trace data as part of the dump output. Trace events missed during ABEND processing will be counted in a special control record that will be included in the trace buffers. If ABEND is not invoked, tracing will continue unaffected. If you specified MODE=(INT,S), GTF will not pause for ABEND or SNAP processing, and the trace buffers will not be formatted.

TIME=YES requests that every logical trace record be timestamped (in addition to the block time stamp associated with every block of data). This record timestamp will be four bytes of timer units for systems without Time-of-Day Clock support; for systems with Time-of-Day Clock support, the record timestamp will be the clock value at the time the record was constructed. Note that if no timer option is present in the system, this parameter will be ignored and a warning message will be issued.

If you code TIME=NO, or if you omit this parameter, GTF will not timestamp individual records.

DEBUG=YES requests that every logical trace record be created. If you specify DEBUG=YES, most errors of this kind will cause GTF to issue an error message and then terminate, so that the contents of the GTF buffers immediately prior to the error will be unchanged. If you have named the GTFSNP procedure in the START command, a SNAP dump will be produced if GTF terminates abnormally.

If you specify DEBUG=NO, or if you omit this parameter, GTF will not terminate immediately, but instead will initiate error recovery procedures. For more information about error recovery procedures, refer to the section "GTF Error Recovery Handling" later in this chapter.
keyword=option

You may use this parameter to override specific parameters in the IEFRDER DD statement in the cataloged procedure. For example:

- To specify a different name for the trace data set, code DSNAME=newname.
- To prevent the system from sending mount messages to the operator's console when specifying MODE=INT, code DSN=NULLFILE.
- To request more than two output buffers, code DCB=(BUFNO=number).
- To modify the GTF buffer size code, DCB=(BLKSIZE=number). The block size cannot be less than 350 bytes.
- To specify an existing data set as the output data set, code DISP=OLD. (Note: If you specify DISP=MOD, GTF will change the data set disposition to OLD.)

Do not use this parameter to request DCB=OPTCD=C; GTF does not support chain-scheduling.

REG=size

supplies a region size for GTF. This will override the value specified in the REGION= parameter of the EXEC statement in the cataloged procedure.

Using the GTF Cataloged Procedures

The START command for GTF names one of two cataloged procedures supplied in SYS1.PROCLIB. The first, GTF, contains job control statements as shown in Figure GTF-2. The second, GTFSNP, is identical to cataloged procedure GTF except that the SNAPDUMP DD statement, shown as optional in Figure GTF-2, is supplied, and the default region size is 30K.

//GTF PROC REG=26
//IEFPROC EXEC PGM=IHLGTF,REGION=&REG.K,
// PARM='MODE=EXT,DEBUG=NO,TIME=NO'
//IEFRDER DD DSNAME=SYS1.TRACE,UNIT=SYSDA,
// SPACE=(3500,20),DISP=(NEW,KEEP)
//SYSPRINT DD SYSOUT=A,SPACE=(TRK,(1,1))
///SYSLIB DD DSN=SYS1.PARMLIB (membername),
/// DISP=SHR]
///SNAPDUMP DD SYSOUT=A

Figure GTF-2. The GTF Cataloged Procedure

PROC Statement

defines the default region size for the symbolic REGION= parameter in the EXEC statement. This default value is used if you do not specify a region size in the START command.

EXEC Statement

calls for the execution of GTF. The REGION parameter is specified as a symbolic parameter so that you can vary it according to need.
IEFDRER DD Statement

defines the trace output data set. If you do not override this statement in the START command, the trace output data set will have the name SYS1.TRACE; it will be directed to a direct access device with sufficient allocation to allow the data set to contain twenty 3500-byte physical blocks.

SYSPRINT DD Statement

defines the GTF message data set.

SNAPDUMP DD Statement (Optional in the cataloged procedure GTF, supplied in GTFSNP.)

causes GTF to issue the SNAP macro to dump the nucleus and the GTF region if an error condition causes GTF to terminate. This statement increases GTF's region size requirements by 4K.

SYSLIB DD Statement (Optional)

defines a member in the SYS1.PARMLIB data set that contains GTF options. If such a member exists, GTF will not prompt you to supply options, but will use the options in the member.

Specifying GTF Trace Options

When you start GTF with MODE=EXT, you will receive the following message:

IHL100A SPECIFY TRACE OPTIONS.

Use the following format to specify the events to be recorded during GTF execution:

TRACE=option1[,option2]...[,optionx]

You can specify any of the following trace option values:

{SYS {SYSP}
{SYSM}

SYS requests that comprehensive trace data be recorded for the following system events:

• I/O interrupts
• SVC interrupts
• Program interrupts
• External interrupts
• Start I/O operations

Note: Dispatcher task switching must be requested separately through the DSP keyword. Similarly, program-controlled interrupt must be requested separately through the PCI keyword.
SYSM requests that minimal trace data be recorded for all system events listed above. SYSP requests further prompting for IO, SIO, SVC, and PI; that is, if you specify SYSP, GTF will prompt you to supply specific device addresses, SVC numbers, or program interrupt codes. Comprehensive trace data will be recorded for events associated with the devices or interrupts that you specify; all other events will be filtered out and ignored. If SYS and SYSM, or SYS and SYSP, are both specified, SYS will be ignored. Similarly, if SYSP and SYSM are both specified, SYSP will be ignored.

\{SIO\} 
\{SIOP\}

SIO requests comprehensive recording for system SIO operations on all devices. SIOP requests further prompting for specific devices for which trace data should be recorded.

This keyword will be ignored if SYS, SYSM, or SYSP has also been specified.

\{IO\} 
\{IOP\}

IO requests comprehensive recording for all I/O interrupts except program-controlled interrupts, which must be requested separately through the PCI keyword. IOP requests further prompting for specific devices for which I/O interrupts should be recorded.

This keyword will be ignored if SYS, SYSM, or SYSP has also been specified.

\{SVC\} 
\{SVCP\}

SVC requests comprehensive recording for all SVC interrupts. SVCP requests further prompting for specific SVC numbers for which trace data should be recorded.

This keyword will be ignored if SYS, SYSM, or SYSP has also been specified.

\{PI\} 
\{PIP\}

PI requests comprehensive recording for all program interrupts. PIP requests further prompting for specific interrupt codes for which trace data should be recorded.

This keyword will be ignored if SYS, SYSM, or SYSP has also been specified.

EXT

requests comprehensive recording for all external interrupts. This keyword will be ignored if SYS, SYSM, or SYSP has also been specified.
DSP requests that a trace record be created whenever the dispatcher is entered for task switching. The trace data collected will be comprehensive unless you have requested SYSM.

USR requests that all data passed to GTF via the GTRACE macro be recorded with the system data in the trace data set.

PCI requests that all program-controlled I/O interrupts be recorded. This keyword will be ignored unless IO, IOP, SYS, SYSM, or SYSP is also specified. If you have specified IOP or SYSP, program-controlled I/O interrupts will be recorded only for those devices that you supplied in response to a prompting message.

PRC requests tracing of trace events associated with the trace task while operating under GTF's task control block. Such events will be traced according to the GTF trace options selected while starting GTF. If this keyword is not specified, GTF task events will be filtered out and not recorded.

SSM requests all program interrupts caused by SSM instructions to be recorded. This keyword is effective only in a multiprocessing environment, and only when PI, PIP, SYS, SYSM or SYSP is also specified.

Prompting
When you specify SYSP, IOP, SIOP, SVCP, or PIP as trace options, GTF will prompt you to supply specific values. These values are:

SIO=(devaddr1[,devaddr2][...devaddr50])

specifies up to 50 device addresses for which you want SIO operations traced. All other SIO operations will be filtered out. If you have specified SIOP or SYSP, and do not specify SIO= in response to the prompting message, no SIO filtering will take place.

IO=(devaddr1[,devaddr2][...devaddr50])

specifies up to 50 device addresses for which you want I/O interruptions traced. All other IO interruptions will be filtered out. If you have specified IOP or SYSP, and do not specify IO= in response to the prompting messages, no IO interruption filtering will take place.

SVC=(svcnum1[,svcnum2][...svcnum50])

specifies up to 50 SVC numbers that you want traced. All other SVC numbers will be filtered out. If you have specified SVCP or SYSP, and do not specify SVC= in response to the prompting message, no SVC filtering will take place.
PI=(code1[,code2][...,code15])

specifies up to 15 program interrupt codes that you want traced. All other program interruptions will be filtered out. If you have specified PIP or SYSP, and do not specify PI= in response to this prompting message, no program interruption filtering will take place.

IO=SIO=(devaddr1[,devaddr2][...,devaddr50])

specified after requesting SYSP or both IOP and SIOP, names up to 50 device addresses for which you want GTF to trace both IO and SIO events. All other IO and SIO events, except those requested specifically by IO= or SIO=, will be filtered out.

Note that in each case GTF imposes a limit on the number of specific values you can supply through prompting. If you exceed this limit, GTF will issue a message and you must respecify all values.

Figure GTF-3 shows an example of an exchange between GTF and the operator when GTF is being started.

```
START GTF,,,(MODE=EXT),REG=34

00 IHL100A SPECIFY TRACE OPTIONS
   r00,'TRACE=SYSP,USR'

01 IHL101A SPECIFY TRACE EVENT KEYWORDS--SVC=,IO=,SIO=,PI=
   r01,'SVC=(1,2,3,4,10),IO=(191,192),PI=

02 IHL102A CONTINUE TRACE DEFINITION OR REPLY END
   r02,'SIO=282,END'

IHL103I TRACE OPTIONS SELECTED--SYSP,USR
   IHL103I SVC=(1,2,3,4,10),IO=(191,192),SIO=(282)

03 IHL125A RESPECIFY TRACE OPTIONS OR REPLY U
   r03,'U'
```

Figure GTF-3. GTF messages and operator replies while starting GTF.

Storing Trace Options in SYS1.PARMLIB

You can save time in starting GTF by storing one or more set combinations of trace options as members in SYS1.PARMLIB. GTF will not prompt you to supply trace options, but will look in SYS1.PARMLIB if you include a SYSLIB DD statement in the GTF or GTFSNP cataloged procedures.
Figure GTF-4 shows the job control statements and utility control statements needed to add trace options to SYS1.PARMLIB using IEBUPDTE. For full descriptions of the statements, refer to the publications IBM System/360 Operating System: Utilities, GC28-6586, and Job Control Language Reference, GC28-6703.

```
//GTFPARM JOB MSGLEVEL=(1,1)
//  EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSPRINT=A
//SYSUT2 DD DSN=SYS1.PARMLIB,DISP=SHR
//SYSIN DD DATA
  TRACE=SYS,USR
  SVC=(1,2,3,4,10),IO=(191,192),SIO=282,PI=15
  TRACE=IO,SIO,TRC
  TRACE=SYS,PCI,SSM
```

Figure GTF-4. Adding Trace Options to SYS1.PARMLIB Using IEBUPDTE.

A sample //SYSLIB DD statement to be included in the GTF or GTFSNP cataloged procedure might look like this:

```
//SYSLIB DD DSN=SYS1.PARMLIB(GTFA),DISP=SHR
```
Calculating Storage Requirements

GTF's region requirements vary according to the GTF options that you specify.

If you have requested MODE=INT, you must specify a minimum region size of 16K bytes of main storage. This minimum will provide you with four 1024-byte buffers. If you need more buffers, you must specify 1K of additional storage for each buffer. If you use the GTFSNP cataloged procedure, or if you use an installation-defined procedure that contains a SNAPDUMP DD statement, you must add 4K to the minimum region size.

If you have requested MODE=EXT, you must specify a minimum region size of 26K. For larger regions, use the following formula to compute your region requirements. Note that all intermediate values must be rounded up to the nearest 2K multiple. The final region size that you calculate must also be rounded up to the nearest 2K multiple.

\[
\text{region} = 16K + n(b+8) + 88(n) + m + a
\]

Where:

- **16K**
  - minimum main storage required for minimal trace.

- **n**
  - number of trace buffers, ordinarily two unless you have specified more in the START command.

- **b**
  - the size of the trace buffers, ordinarily 3500 bytes unless you have specified a different value in the START command. (Note: When trace output is directed to a direct access device, the buffer size should equal the track size. This is necessary to prevent too much previously stored data from being lost when the trace data is "wrapped". The 8 additional bytes are needed for the GTF buffer prefix.

- **88**
  - the size of the input/output block (IOB); one IOB is required for each buffer.

- **m**
  - total main storage required to process GTF options requested. In some cases, several GTF options are contained within one module. Even if you request two or more GTF functions that are contained in the same module, you only need to provide enough space for one copy of the module. Refer to Figure GTF-5 for a summary of GTF options, the modules that contain them, and the amount of main storage required for each module.

Chapter 3: IHLGTF 65
To calculate \( m \), add together the storage requirements for each module that you will need, and add 1K to the total if you have requested filtering for any option. For example, if you specify EXT, SVCP, and USR:

\[
m = 2K + 8K + 1K + 0.5K \\
m = 11.5K
\]

The amount of main storage required for ABEND or SNAP processing. If you have requested either ABEND or SNAP, or both, when starting GTF, this value is 4K. If you have not requested ABEND or SNAP, this value is zero.

<table>
<thead>
<tr>
<th>GTF OPTIONS SELECTED</th>
<th>MODULES REQUIRED</th>
<th>MAIN STORAGE REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSM[,DSP][,PCI]</td>
<td>IHLSYSV or IHLSYS</td>
<td>1K</td>
</tr>
<tr>
<td>DSP</td>
<td>IHLTPED</td>
<td>2K</td>
</tr>
<tr>
<td>IO=</td>
<td>IHLTSIO</td>
<td>1K</td>
</tr>
<tr>
<td>SVC</td>
<td>IHLTSVC</td>
<td>8K</td>
</tr>
<tr>
<td>SYS[,DSP][,PCI][,SSM]</td>
<td>IHLTPED, IHLTSIO, and IHLTSVC</td>
<td>11K</td>
</tr>
<tr>
<td>USR</td>
<td>IHLTSVR</td>
<td>1K</td>
</tr>
<tr>
<td>IOP</td>
<td>IHLTFL</td>
<td>1K</td>
</tr>
</tbody>
</table>

Figure GTF-5. Main Storage Requirements for GTF Options, By Module. Note that TRC can be considered to require 0 (zero K) bytes of main storage.
Recording User Data

If you want your own trace data to be recorded in the GTF trace buffers, you can specify that data in the GTRACE macro instruction. In one invocation of GTRACE, an application program can record up to 256 bytes of data in a GTF trace buffer. Secure data should not be recorded using the GTRACE macro since security protection cannot be guaranteed. Note, however, that GTRACE can record only data that has the same protect key as the GTRACE user.

GTRACE will be effective only when GTF is active, when it is directing its output to an external data set, and when it is accepting user data -- that is, when GTF has been started with MODE=EXT and TRACE=USR specifications.

Printing User Data

Like other trace data, information recorded by the GTRACE macro can be printed by the EDIT function of IMDPRDMP. Usually user data will be printed in hexadecimal, since EDIT cannot format records not created by GTF. However, you can write format appendages to format specific types of user data records.

Every time you issue GTRACE to create a user record, you specify which format appendage should process it; you do this by including the optional FID (format identifier) parameter in the GTRACE invocation. The FID corresponds to the last two hexadecimal characters in the name of the format appendage, IMDUSRxx.

Coding the GTRACE Macro

Figure GTF-6 shows the general format of the GTRACE macro, standard form.

[symbol] GTRACE DATA=address, LNG=number, ID=number[, FID=value]

Figure GTF-6. The General Format of the GTRACE Macro, Standard Form

The parameters in the macro are described below.

DATA=address

gives the main storage address of the data to be recorded.

LNG=number

specifies the number of bytes (1 to 256) to be recorded from the address specified in the DATA= parameter. The number may be specified in decimal or in hexadecimal (as X'number').

Chapter 3: IHLGTF 67
ID=value

is the identifier to be associated with the record. ID values are assigned as follows:

0 to 1023 -- user events
1024 to 4095 -- reserved

The value may be specified in decimal or in hexadecimal (as X'value').

FID=value

indicates the format appendage that is to format this record when the trace output is processed by the EDIT function of IMDPRDMP. FID values are assigned as follows:

0 (or FID= parameter omitted) -- record to be dumped in hexadecimal
1 to 80 -- user format identifiers
81 to 255 -- reserved

The value may be specified in decimal or in hexadecimal (as X'value').

Figure GTF-7 shows how the GTRACE macro might be coded to record 200 bytes of data, beginning at the address of AREA, with an event identifier of 37 and to be formatted by the format appendage with the name IMDUSR64.

GTRACE DATA=AREA, LNG=200, ID=37, FID=100

Figure GTF-7. An Example of the GTRACE Macro.

For more details about the GTRACE macro instruction, consult the publication IBM System/360 Operating System: Supervisor Services and Macro Instructions, GC28-6646.
GTF Error Recovery Handling

GTF recognizes all errors that occur while building a trace record as potentially recoverable. Whether recovery takes place or not depends on what you code in the START command.

If you specify DEBUG=YES, GTF will not attempt error recovery. It will issue an error message and then terminate, so that the contents of the GTF buffers immediately prior to the error will be preserved.

If you specify DEBUG=NO, GTF will initiate the following error procedures:

For minor errors in the routine that builds the trace record (the build routine), GTF flags the field that led to the error and continues processing. It does not issue a message to the operator's console or disable the function that caused the error; instead, it proceeds as if no error had occurred. All errors that occur while building an SVC record fall into this category.

For severe errors in the build routine, GTF flags the entire record that was being built, issues a message to the console, and continues processing with the function that caused the error suppressed.

For errors in the routine that filters trace events, GTF suppresses filtering for future events of the same type, issues a message to the console, and continues processing.

Errors that occur outside the build and filter routines are not recoverable; they result in immediate abnormal termination of GTF.

Note that the termination of GTF will never cause termination of a user's task.
GTF Output

GTF creates two kinds of records: trace records and control records.

Trace Records

GTF creates trace records for each system event you select. The records have the general format shown in Figure GTF-8.

```
length 00 AID FID Timestamp EID DATA
```

- **length** indicates the total length of the record.
- **00** always zero.
- **AID** defines whether the data record is a trace record or a GTF control record.
  - X'FF' -- trace record
  - X'00' -- GTF control record
  - X'01' to X'FE' -- reserved
- **FID** is the format identifier, a one-byte hexadecimal number that identifies the program that will format the trace record during EDIT execution. (For information on specifying the FID in the GTRACE macro, refer to the section "Coding the GTRACE Macro" in this chapter.)

  If this field is zero, the trace record will not be formatted, but will be dumped in hexadecimal.

Figure GTF-8. Fields in a trace record.

The fields in the record are described as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>total length of the record</td>
</tr>
<tr>
<td>AID</td>
<td>always zero</td>
</tr>
<tr>
<td>FID</td>
<td>defines whether the data record is a trace record or a GTF control record</td>
</tr>
<tr>
<td>Timestamp</td>
<td>optional (8 bytes)</td>
</tr>
<tr>
<td>EID</td>
<td>format identifier (1 byte)</td>
</tr>
<tr>
<td>DATA</td>
<td>trace data (up to 256 bytes)</td>
</tr>
<tr>
<td>Application Identifier</td>
<td>(1 byte)</td>
</tr>
<tr>
<td>Always zero</td>
<td>(2 bytes)</td>
</tr>
<tr>
<td>Number of bytes in trace record</td>
<td>(2 bytes)</td>
</tr>
</tbody>
</table>
timestamp

If TIME=YES was specified in the START command and a timer option is in effect in the system, a time stamp will be included in this eight-byte field. If GTF is executing on a system without the Time-Of-Day Clock, the time stamp will be four bytes of timer units, right justified. On a system with Time-Of-Day clock support, the value in the record will be the clock value at the time the record was constructed.

EID

defines the event that caused the trace record to be created. It is not present in GTF control records. You can determine the EID of a trace record by issuing the IMDMEDIT mapping macro, which is described in the Appendix: Writing EDIT User Programs.

data

This field contains the trace data gathered for the requested event. The length of this field varies according to the event being traced.

Figures GTF-9 through GTF-13 are examples of trace output as processed by the EDIT function of IMDPRDMP. In all the examples, fields flagged with hhhhhhhhh are hexadecimal representations, and fields flagged with ccccccccc are alphameric characters. N/A signifies that the field label does not apply to this particular record. For explanation of the fields in the records, refer to the Programmer's Guide to Debugging, GC28-6670.
Figure GTP-9. Format of Comprehensive Trace Records for DSP, IO (including PCI), SIO, PI, EXT, and SVC (MFT and MVT)
Figure GTF-10. Format of Comprehensive Trace Records for DSP, IO (including PCI), SIO, PI (including SSM) EXT, and SVC (Model 65 Multiprocessing)
Figure GTF-11. Format of Minimal Trace Records for DSP, IO (including PCI), SIO, PI, EXT, and SVC (MFT and MVT)
Figure GTF-12. Format of Minimal Trace Records for DSP, IO (including PCI), SIO, PI (including SSM), EXT, and SVC (Model 65 Multiprocessing)

```
(a) DSP RES PSW hhhhhhh hhhhhhh R15/R0 hhhhhhh hhhhhhh R1 hhhhhhh NUA hhhhhhh NUB hhhhhhh
(b) (IO ) OLD PSW hhhhhhh hhhhhhh CSW hhhhhhh hhhhhhh RQE TCB {********} OLA hhhhhhh OLB hhhhhhh
               hhhhhhh N/A
(b) (PCI) SIO CC/DEV/CAW hhhhhhh hhhhhhh CSW hhhhhhh hhhhhhh RQE TCB {********} OLA hhhhhhh OLB hhhhhhh
               hhhhhhh N/A
(b) PGM OLD PSW hhhhhhh hhhhhhh R15/R0 hhhhhhh hhhhhhh R1 hhhhhhhh OLA hhhhhhhh OLB hhhhhhh
(b) SSM LK hh OPSW hhhhhhh hhhhhhh R15/R0 hhhhhhh hhhhhhh R1 hhhhhhh OLA hhhhhhh OLB hhhhhhh
(b) EXT OLD PSW hhhhhhh hhhhhhh R15/R0 hhhhhhh hhhhhhh R1 hhhhhhh OLA hhhhhhh OLB hhhhhhh
(b) SVC OLD PSW hhhhhhh hhhhhhh R15/R0 hhhhhhh hhhhhhh R1 hhhhhhh OLA hhhhhhh OLB hhhhhhh
```

Figure GTF-13. Hexadecimal Format Records

```
HEXFORMAT
USER
SYSTEM
SUBSYS

(a) AID hh FID hh EID hh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
(b) hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh hhhhhhh
   hhh...```
Control Records

GTF produces two types of control records: timestamp records and lost data records. The first record in every block of trace output is a timestamp record. A lost data record appears to signal trace events that were not recorded because the GTF buffers were full or because GTF has temporarily suspended operations during ABEND or SNAP processing. Figure GTF-14 shows the general format of a timestamp record.

<table>
<thead>
<tr>
<th>length</th>
<th>00</th>
<th>AID</th>
<th>FID</th>
<th>reserved</th>
<th>timestamp</th>
<th>date</th>
<th>options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>4 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>4 bytes</td>
<td>4 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure GTF-14. General Format of a Timestamp Control Record.

The fields in the record contain the following information:

length

total length of the record in bytes.

00

always zero.

AID

For control records this field is always zero.

FID

For timestamp control records, this field is always X'04'.

reserved

reserved for future use.

timestamp

timer units (in hexadecimal) representing the time when the control record was constructed. If GTF is running on an MFT system with no timer option, this field is zero.

date

year and julian day, in hexadecimal. The format is X'00 yy dd dc', where c is the packed decimal sign.

options

GTF options in effect. For detailed information about this field, see Figure APNDX-2 in the Appendix.
Figure GTF-15 shows the general format of a lost event record.

```
length  00  AID  FID  reserved  timestamp  events  bytes (optional)
                4 bytes
                4 bytes
                1 byte
                1 byte
                2 bytes
                2 bytes
```

Figure GTF-15. General Format of a Lost Event Record.

The fields in the record contain the following information.

**length**

- total record length in bytes.
- 00
- always 00

**AID**

- always 00 in control records

**FIC**

- format identifier. Valid values are:
  - X'05' -- events lost because buffers full.
  - X'06' -- events lost because GTF disabled temporarily.

**reserved**

- reserved for future use.

**timestamp**

- timer units (in hexadecimal) representing the time when the control record was constructed. If GTF is running on an MFT system with no timer option, this field is zero.

**events**

- number of traceable events lost (in hexadecimal).

**bytes (present only in records created under buffer-full condition)**

- number of bytes of data lost (in hexadecimal). This field is not formatted by the EDIT function of IMDPRDMP.
Chapter 4: IMCJQDMP
Operates as a stand-alone program to format and print the system job queue.
INTRODUCTION • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
IMCJQDMP is a service aid program that produces a formatted copy of the contents of the IBM System/360 operating system's job queue data set (SYS1.SYSJOBQE). The program operates in stand-alone mode; that is, it is independent of any operating system.

It may be said that system control is centered in the job queue. Its component tables and blocks store the dynamic environmental descriptions that regulate the processing of all jobs submitted to the operating system. Detailed descriptions and layouts of the record types which may be encountered in the job queue data set may be found in the following publications: IBM System/360 Operating System: MVT Job Management, GY28-6603, and Control Program with MFT, GY27-7128.

IMCJQDMP may be used to dump the entire job queue, or the user may optionally specify selected portions of it for printing.
Function of IMCJQDMP

In determining the cause of a job or system failure, it is often desirable to know precisely what was contained in the job queue, or in specific portions of it, at the time of such failure.

For example, the user may attempt to initiate a warm start, and fail. A warm start failure tends to be a critical problem, as it is dependent upon job queue structure for its proper functioning. A dump of the job queue would be an invaluable aid in tracing the cause of such a failure.

There are also the instances in which the Scheduler ABEND 080 occurs, indicating an I/O error on the job queue data set. This ABEND is often caused by an invalid TTR-address being used to access the job queue. A job queue dump provides precise information as to the address of each record, and, in addition, allows access to certain queue records which are chained together by a TTR-address contained in a primary record. Such information is vital in determining the cause of the I/O failure.

In many other situations, it may be necessary to interpret and examine the main storage chains reflected in the control blocks contained in the job queue.

Optimally, this information should be made available to the user:

- Without disturbing the prevailing status of the job queue;
- Whether or not the system is operational;
- Without prior knowledge of the exact location of the job queue data set on its assigned direct access volume;
- On a record-by-record basis, according to direct access volume address; and
- Conveniently formatted for ready access and interpretation.

The IMCJQDMP program is designed to supply specialized job queue dumps incorporating all these features.

The program functions in stand-alone fashion, a circumstance which is particularly beneficial in instances where the system is involved in the failure. Since it does not function under the operating system, it is not enqueued upon the job queue data set and, therefore, does not alter the existing status of the records that are to be dumped. The printed queue records reflect precisely what they contained at the time of malfunction. Nor is it required that the user know the explicit address of SYS1.SYSJOBQE. Only the address assigned to the direct access device on which the volume containing the job queue is mounted need be supplied to the dump program. The program determines the address of the job queue data set by reading the queue volume's VTOC (volume table of contents). The VTOC contains data set control blocks (DSCBs) corresponding to each data set and to contiguous blocks of unassigned tracks on the volume.
When the queue has been found by IMCJQDMP, records are read and, according to the user's exercise of the available options, are either serially or selectively identified by type and address, formatted, and written to the chosen output device. This may be either a 1403 printer or an unlabeled 9- or 7-track magnetic tape volume. Printing the tape output of IMCJQDMP is discussed under "Tape Output Processing."
Retrieving IMCJQDMP

The Job Queue Dump program is supplied in object module form, together with an absolute loader. The program resides on the OS/360 Distribution Library packs as a member (IMCJQDMP) of component library SYS1.DN554A. In preparing the program for use, the module IMCJQDMP must be punched from the component library or copied to a nonlabeled magnetic tape. The card deck or tape may then be used to load the program for execution.

The JCL statements for punching the program from the component library are shown in Figure JQDMP-1. This example assumes that the distribution libraries are cataloged; if they are not, add the UNIT and VOL=SER parameters to the SYSUT1 data definition statement.

```
//QDUMP JOB      MSGLEVEL=(1,1)
//STEP EXEC      PGM=IEBPTPCH
//SYSPRINT DD    SYSOUT=A
//SYSUT1 DD      DSN=SYS1.DN554A(IMCJQDMP),DISP=OLD,
//                DCB=(BLKSIZE=3600,LRECL=80,RECFM=FB)
//SYSUT2 DD      UNIT=2540-2
//SYS1N DD
//PUNCH
/* PUNCH
```

Figure JQDMP-1. Sample JCL Statements Needed to Punch IMCJQDMP from Component Library SYS1.DN554A
IMCJQDMP may be used with any S/360 or S/370 CPU, and requires about 18K bytes of main storage for execution. I/O device requisites are a card reader (or, optionally, a 2400 tape drive) for initial program loading (IPL); one of the following consoles: 1052, 3066, 3210, 3215, or 5450; and one of the following DASD devices -- 2311, 2312, 2313, 2314, 2318, 2319, 2301, 2303, 2305, or 3330 -- for input, and either a 1403 printer or a 2400 tape drive for output. Figure JQDMP-2 describes the flow of processing when IMCJQDMP is used.

Figure JQDMP-2. Flow of Processing for IMCJQDMP
Job Queue Format

Input to IMCJQDMP is the system's job queue data set (SYS1.SYSJOBQE), which is maintained on a permanently resident direct access volume. The job queue is composed of control records and work queues, created and updated by diverse system components. Figure JQDMP-3 shows the format of a job queue.

![Job Queue Format Diagram]

The job queue data set consists of 76 work queues:

- 15 input queues, one for each job class.
- 36 output queues, one for each output class.
- 1 free-track queue, from which work queue space is assigned as needed. Immediately after job queue initialization, the entire data set consists of free tracks.
- 1 automatic SYSIN batching (ASB) queue.
- 1 TSO Background Reader queue.
- 1 remote job entry (RJE) queue.
- 1 HOLD queue for temporarily dequeued jobs.
- 21 reserved queues.
These work queues consist of assigned logical tracks. A logical track may be defined as an area of contiguous space in the data set large enough to contain a 20-byte logical track header (LTH) record, followed by a predetermined number of 176-byte data records. Figure JQDMP-4 describes the format of a logical track header record.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex</th>
<th>Dec</th>
<th>Field Content</th>
<th>Queue Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>Reserved</td>
<td>HOLD Queue</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>Reserved</td>
<td>ASB Queue</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>11</td>
<td>First Logical Track of the Job</td>
<td>Output class queues</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>16</td>
<td>Last Logical Track of the Next Job</td>
<td>RJE Queue</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>21</td>
<td>Jobclass of the Job</td>
<td>Input class Queues</td>
</tr>
</tbody>
</table>

Figure JQDMP-4. Logical Track Header (LTH) Record Format

In Figure JQDMP-4 and subsequent figures, where applicable, byte size of a field is shown in the upper right corner; offset from the beginning of the record, in hexadecimal and decimal notation, is given along the left margin.

Content of the type field in an LTH record indicates the type of queue to which the logical track has been assigned:

<table>
<thead>
<tr>
<th>Field Content</th>
<th>Queue Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HOLD Queue</td>
</tr>
<tr>
<td>2</td>
<td>ASB Queue</td>
</tr>
<tr>
<td>3-38</td>
<td>Output class queues</td>
</tr>
<tr>
<td>39</td>
<td>RJE Queue</td>
</tr>
<tr>
<td>40-54</td>
<td>Input class Queues</td>
</tr>
</tbody>
</table>
To keep track of individual work queues, a control area in the job queue data set maintains a series of 36-byte minor queue control records (QCRs) -- one QCR for each work queue arrayed upon the job queue (see Figure JQDMP-5), plus a master QCR (see Figure JQDMP-6).

### Figure JQDMP-5. Example of Minor Job Queue Control Record

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address of Last LTH of Highest Priority Entry on Queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1B</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1C</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Address of ECB for First Task Requesting Work</td>
<td>20</td>
<td>32</td>
<td>3</td>
</tr>
</tbody>
</table>

### Figure JQDMP-6. Master Job Queue Control Record Format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-byte Disk Address of the Master QCR</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MBBCCHHR</td>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Number of Logical Tracks in the Job Queue Data Set</td>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Number of Tracks Reserved for Canceling of Job Steps When Queue Is Full</td>
<td>10</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Displacement of Last Available Logical Track</td>
<td>14</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Number of QCRs per Physical Track</td>
<td>18</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>Number of Records per Logical Track</td>
<td>1C</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Number of QCRs on the Mixed Track</td>
<td>20</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>Number of Logical Tracks In the Free-track Queue</td>
<td>Reserved</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of Tracks Reserved for Any Initiator</td>
<td>Reserved</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Displacement of First Track Containing Only Job Queue Records</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of Job Queue Records per Physical Track</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of Logical Tracks for Each Problem Program Partition</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Address of First Record on First Track Containing Only Job Queue Records</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

90 Service Aids (Release 21)
To use IMCJQDMP, initial program load (IPL) the program from the card reader or from the tape unit on which a tape-copy of the deck is mounted. This is done by setting the LOAD UNIT dials on the console control panel to the unit address of the card reader or the tape drive, and depressing the LOAD key on the control panel. When loading has been accomplished, the program enters a wait state, indicated by the lighting of the WAIT light on the console. Pressing the console request or enter key at this point results in the console message:

```
IMC000A ENTER O=XXXD,Q=YYY(S) OR PRESS INTERRUPT KEY FOR O=00E,Q=191
```

Message IMC000A is a request for parameters giving specifications for the desired dump. If the operator responds by depressing the external interrupt key without entering a device identification command through the console, the dump output will be written to the 1403 printer assigned device address 00E; input will be read from the direct access volume mounted on the disk drive assigned device address 191.

**Device Identification Command**

If the device identification command is entered, its format is:

```
O=xxxd,Q=yyy,,SELECT
```

where

- `O=xxxd` is the output address parameter;
- `Q=yyy` is the input address parameter;
- `SELECT` (or `S`) indicates that selective rather than full printing of the job queue is desired.

**Output Address Parameter**

The output address parameter may be omitted entirely. If it is, the output address will default to the 1403 printer at device address 00E. If the parameter is entered, it must precede the input address parameter. In making the entry,

```
xxx
```

is replaced with the address of the desired output device. Valid choices are the 1403 printer and the 2400 tape drive.
indicates the output device type. The character T is entered if a 2400 tape drive address has been specified in the xxx field. Example: O=182T.

The d field is omitted if output is to go to the 1403 printer. Example: O=00E.

Input Address Parameter

In the input address parameter,

YYY

is replaced with the address of the direct access device upon which the volume containing the SYS1.SYSJOBQE data set has been mounted.

Selective Dumping Parameters

If an entire job queue data set is to be dumped, the SELECT (or S) parameter is omitted from the device identification command.

If the SELECT (or S) parameter is included in the command, the program will issue the console message:

IMC001A SPECIFY SELECT PARAMETERS

and wait for a reply. The two valid parameters, QCR= and JOBNAME=, and their possible values are discussed separately.

QCR= Parameter

The QCR= parameter specifies that a particular work queue within the job queue data set is to be dumped. When this parameter is specified, the dump output listing will contain the data set's master queue control record and the queue records associated with the named work queue. The possible values for the QCR= parameter are:

\[
\text{QCR=} \begin{cases} 
\text{ASB} \\
\text{CLASS=}y \\
\text{FREE} \\
\text{HOLD} \\
\text{RJE} \\
\text{SYSOUT=}x \\
\text{SUBMT} 
\end{cases}
\]

where:

- y is replaced with one of the 15 input job class indicators, A through 0, and
- x is replaced with one of the 36 output class indicators, A through Z and 0 through 9.

If FREE is the value used, the master QCR and all logical tracks enqueued upon the free-track queue are dumped. The output listing for any of the other values will include the associated minor QCRs as well.
The values CLASS= and SYSOUT= must be completed with the system-assigned symbol of the particular input or output class desired. Examples:

QCR=SYSOUT=C

will result in a dump of the job queue's master QCR, the C-class output work queue's minor QCR, and the logical tracks assigned to the C-class output work queue.

QCR=RJE

will produce output consisting of the job queue's master QCR, the RJE work queue's minor QCR, and its assigned logical tracks.

JOBNAME= Parameter

The JOBNAME= parameter signifies to IMCJQDMP that the fifteen input work queues are to be searched for logical track areas assigned to the named job or jobs. Associated system message blocks and data set blocks will also be dumped. From one to four jobnames, enclosed in parentheses, may be specified in the value field of the parameter. Example:

JOBNAME=(TAX,NUMBER)

will produce a dump listing containing the assigned logical track areas, the system message blocks and the data set blocks, if any, associated with jobs named TAX and NUMBER, respectively.

Combining QCR= and JOBNAME= Parameters

The time required to search out the records associated with a particular job may be considerably reduced if the input class is known to the dump program. This passing of class information to IMCJQDMP may be effected by using the QCR= and JOBNAME= parameters in combination. For instance:

QCR=CLASS=G,JOBNAME=(LIST)

will cause only the class G input work queue to be searched for records concerned with the job named LIST.

Completion Message

After the selective dump parameters have been accepted, IMCJQDMP performs the requested task. When the operation has been completed, message IMC001A is reissued. Additional selective dump parameters may be entered if more information is desired. When all user requests have been fulfilled:

END

is entered through the console. The message:

IMC004I DUMP COMPLETED
is then issued. Note that when no selective dump parameters are entered, the program ends automatically after dumping the full job queue, issuing message IMC004I at completion of the operation.

Tape Output Processing

For magnetic tape output, IMCJQDMP creates 121-byte records, one record to a block. Each record contains a machine control character in its first byte.

Figure JQDMP-7 gives a sample of the job control statements needed to print IMCJQDMP 9-track tape output with the IEBPTPCH utility program.

```
//PRINT JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=2400,LABEL=(,NL),VOL=SER=QDUMPT,
//       DISP=(OLD,KEEP),DCB=(RECFM=F,BLKSIZE=121,LRECL=121)
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT PREFORM=M
/**
```

Figure JQDMP-7. Sample JCL Needed to Print 9-Track JQDMP Tape Output

For 7-track tape printing, there is an additional consideration. Initial program loading of IMCJQDMP generates a system reset which, on a 7-track control unit, has the following effect:

1. Mode is set to 800 bits per inch.
2. If the data conversion feature is present in the control unit, the data converter is turned on.
3. The translator is turned off.
4. Odd parity is established.

When the dump output has been written to 7-track tape, therefore, the following additional DCB parameters should be coded on the SYSUT1 DD statement for the IEBPTPCH utility if the data conversion feature exists:

```
DEN=2,TRTCH=C
```

If the data conversion feature is not included in the system, the TRTCH keyword must be omitted.

Standard Label Processing

For output to magnetic tape, IMCJQDMP automatically performs standard label processing; the user has no option to bypass this function. The extent of the label processing is confined to protecting security protected data sets and tapes with unexpired expiration dates; and, if multiple-volume output is produced, to maintaining standard labeled tapes, provided the first volume of IMCJQDMP output has standard labels.
When verifying that a mounted tape has standard labels, IMCJQDMP will read the labels (if present) in the density set for the tape drive; therefore, the user must be careful to ensure that the labels on the tape were recorded in the same density as the recording density set for the tape drive on which the tape is mounted. If the recording density for the drive is different from that of the tape, IMCJQDMP will assume that the tape has no labels, and will create non-labeled tape output.

**Standard Labeled Output**

If the user desires standard labeled output, he must mount a standard labeled tape. IMCJQDMP checks for an IBM standard volume label (VOL1) and the standard data set header label one (HDR1). Any user labels will be ignored and destroyed if the tape is eventually used for IMCJQDMP output. If the mounted tape contains a security protected data set, IMCJQDMP will request a new tape. If the expiration date in the HDR1 label has not occurred, IMCJQDMP will request permission to use the tape; if the operator's reply is negative (M), a new tape is requested. Otherwise, the tape will be used, and will contain standard labels, with the VOL1 label remaining the same as it was when the tape was mounted. The header and trailer labels will be created to be compatible with OS/360 standard labels, with a data set name of "JQDUMP."

**Non-Labeled Output**

If the user desires non-labeled tape output, the first volume mounted must be non-labeled. A non-labeled tape, to the IMCJQDMP program, is a tape that does not have a first record of 80 characters whose first four characters are equal to "VOL1." If the first record on the first volume is a standard volume label, processing as outlined in "Standard Labeled Output" will occur.

**Abnormal Termination of JQDMP**

It is conceivable that a condition can arise that will prevent IMCJQDMP from running to normal completion. Indeed, it may be the same error condition that caused the system to malfunction; that is, I/O error on the queue device, or invalid chaining of queue records. Under unrecoverable error conditions, the program comes to a halt in a wait state. The type of error encountered by the program may be determined by examination of the contents of the program status word (PSW) which was current at the time of the malfunction.

The PSW is a doubleword, having the following format:

**Program Status Word**

<table>
<thead>
<tr>
<th>System Mask</th>
<th>Key</th>
<th>AMWP</th>
<th>Interruption Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ILC</th>
<th>CC</th>
<th>Program Mask</th>
<th>Instruction Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>33</td>
<td>34 35 36</td>
<td>39 40 63</td>
</tr>
</tbody>
</table>

The publication IBM System/360 Principles of Operation, GA22-6821, gives a comprehensive description of each of the fields in the PSW. For the purpose of locating the cause of trouble in running IMCJQDMP, the user would be concerned mainly with the contents of the instruction address field, bits 40 through 63, in the event of a program check error, or with the interruption code, bits 16 through 31, if there has been an unrecoverable I/O error.
By displaying the contents of the instruction address register (IAR) on the system maintenance panel of the console, the address in main storage of the pertinent PSW can be obtained. The two low-order bytes of the IAR will be set according to the pattern:

\[
0Dnn
\]

where \( nn \) will contain the hexadecimal value of the location where the PSW was stored at the time the error condition was discovered.

For example, should a program check occur, the IAR will be set to 0D28, indicating that the double word at location hex 28 will contain the Program Interrupt old PSW. (A note for users of doubleword fetch machines, such as M65 or M75: The IAR is updated by 8 after an interrupt, and this must be subtracted from the IAR setting to obtain the true location to be checked. In this example, for instance, the reading would be 0D30, and subtracting hex 8 would give the true 0D28 location.)

If the IAR display indicates 0D20, inspecting the interruption code in the PSW stored at hexadecimal location 20 will indicate the nature of the I/O error:

<table>
<thead>
<tr>
<th>IC Content</th>
<th>Error Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>x'00'</td>
<td>Channel end, device end, and unit check bits are all off in a stored channel status word (CSW).</td>
</tr>
<tr>
<td>x'02'</td>
<td>Invalid track-per-cylinder count in the format 4 DSCB (data set control block) of the queue volume.</td>
</tr>
<tr>
<td>x'03'</td>
<td>I/O error during write operation to output device or system console. The number of retries for recoverable tape I/O errors is set at 20.</td>
</tr>
<tr>
<td>x'20'</td>
<td>I/O error during read operation from SYS1/SYSJOBQE data set. The number of retries for recoverable DASD I/O errors is set at 16.</td>
</tr>
<tr>
<td>x'26'</td>
<td>I/O error during read operation from system console.</td>
</tr>
</tbody>
</table>
IMCJQDMP Output

IMCJQDMP dumps the contents of job queue records in hexadecimal representation, with six 4-byte words appearing in a line of printed output. In addition, translatable EBCDIC characters are printed in a one-character-per-byte format at the end of the printline. EBCDIC characters which cannot be interpreted in print are represented by periods. Record identification is shown on the sample listing page depicted in Figure JQDMP-8.
<table>
<thead>
<tr>
<th>TTR</th>
<th>NN</th>
<th>TYPE</th>
<th>DISP</th>
<th>SYSJOBGE DL/MF</th>
<th>PAGE 0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001</td>
<td>QCR</td>
<td>QSTR</td>
<td>0000</td>
<td>00CD00CD 02000001 0C067C1 0191018D 00060006 05800000 05000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000002</td>
<td>QCR</td>
<td>MSTR</td>
<td>0018</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000003</td>
<td>QCR</td>
<td>MSTR</td>
<td>0018</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000004</td>
<td>QCR</td>
<td>MSTR</td>
<td>0018</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000005</td>
<td>QCR</td>
<td>MSTR</td>
<td>0018</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TTR</th>
<th>NN</th>
<th>TYPE</th>
<th>DISP</th>
<th>SYSJOBGE DL/MF</th>
<th>PAGE 0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>000202</td>
<td>QCR</td>
<td>QSTR</td>
<td>0009</td>
<td>00CD00CD 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000203</td>
<td>QCR</td>
<td>RESV</td>
<td>0018</td>
<td>00CD00CD 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000204</td>
<td>LTH</td>
<td>OCR</td>
<td>0009</td>
<td>00CD00CD 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
<tr>
<td>000205</td>
<td>LTH</td>
<td>OCR</td>
<td>0009</td>
<td>00CD00CD 00000000 00000000 00000000 00000000 00000000</td>
<td><em>.............</em></td>
</tr>
</tbody>
</table>

Figure JQDMP-8. Sample of IMCJQDMP Output Listing
Record Identification

Record identification on the listing includes:

TTR

The direct access address, relative to the beginning of SYS1.SYSJOBQE, is supplied for both QCR and logical track records.

NN

Supplied for logical track records only, this address is a binary number assigned relative to the beginning of the specific work queue in which the printed record resides. Starting with an assignment of 1 for the first logical track header allotted to the queue, the NN address increases by 1 for each additional record in the work queue.

TYPE

Figure JQDMP-9 lists the types of queue records dumped by IMCJQDMP, and the listing identification given to each recognizable type. QCRs and LTHs are identified through their position in the job queue's structure. Identification for records from the logical track area is obtained from the ID field, hexadecimal offset 03 (byte 4) of each record. Recognizable ID values are listed in the figure. Unidentifiable nonzero records -- the job file control block (JFCB), job file control block extension (JFCBX), and system output class directory (SCD) -- are printed without type labeling.

QCR ID

Each queue control record is further labeled with the name of the unique work queue with which the QCR is associated. Figure JQDMP-10 lists the identification given by IMCJQDMP to each work queue type.

DISP

Indicates the displacement, or position, within a queue record of the next hexadecimal word to be printed on the listing. The first word of the first printed line for a given record is at displacement 0000; the first word of the second printed line, if one exists, is displacement 0018 hex (24 decimal).
<table>
<thead>
<tr>
<th>Hex ID Value</th>
<th>Output Type ID</th>
<th>Job Queue Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>ACT</td>
<td>Account Control Table</td>
</tr>
<tr>
<td>15</td>
<td>DSB</td>
<td>Data Set Block</td>
</tr>
<tr>
<td>0F</td>
<td>DSENQ</td>
<td>Data Set Enqueue Table</td>
</tr>
<tr>
<td>07</td>
<td>DNSMT</td>
<td>Data Set Name Table</td>
</tr>
<tr>
<td>00</td>
<td>JCT</td>
<td>Job Control Table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Job File Control Block (JFCB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Job File Control Block Extension (JFCBX)</td>
</tr>
<tr>
<td>0A</td>
<td>LTH</td>
<td>Logical Track Header</td>
</tr>
<tr>
<td>02</td>
<td>PCT</td>
<td>Procedure Override Table</td>
</tr>
<tr>
<td>0C</td>
<td>QCR</td>
<td>Queue Control Record</td>
</tr>
<tr>
<td>03</td>
<td>SFT</td>
<td>System Output Class Directory (SCD)</td>
</tr>
<tr>
<td>05</td>
<td>CRT</td>
<td>Step Control Table</td>
</tr>
<tr>
<td>06</td>
<td>RESRV</td>
<td>Step Control Table Extension</td>
</tr>
<tr>
<td></td>
<td>RJE</td>
<td>Step Input Output Table</td>
</tr>
<tr>
<td></td>
<td>SMB</td>
<td>System Message Block</td>
</tr>
<tr>
<td></td>
<td>SLT</td>
<td>Volume Table</td>
</tr>
</tbody>
</table>

Figure JQDMP-9. Queue Record Type Identification

<table>
<thead>
<tr>
<th>Output QCR ID</th>
<th>Corresponding Work Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASB</td>
<td>Automatic SYSIN Batching Queue.</td>
</tr>
<tr>
<td>CLS=y</td>
<td>System Input Job Class Queues; y is replaced with the appropriate class, A-O.</td>
</tr>
<tr>
<td>HOLD</td>
<td>Hold queue.</td>
</tr>
<tr>
<td>MASTR</td>
<td>Master QCR.</td>
</tr>
<tr>
<td>OUT=x</td>
<td>System Output Job Class Queues; x is replaced with the appropriate class, A-Z or 0-9.</td>
</tr>
<tr>
<td>RESRV</td>
<td>Reserved QCRs.</td>
</tr>
<tr>
<td>RJE</td>
<td>Remote Job Entry Queue.</td>
</tr>
<tr>
<td>SUBMT</td>
<td>TSO Background Reader Queue.</td>
</tr>
</tbody>
</table>

Figure JQDMP-10. Queue Control Record Identification

Zero Records in the Dump

Records in each logical track are read and dumped sequentially. When a record in the logical track area contains only binary zeroes, its TTR and NN positions are given, but the record is not dumped. The notation:

ENTIRE RECORD CONTAINS BINARY ZEROES

is printed on the listing. A second contiguous zero-filled record would be similarly treated. But when three or more contiguous zero-filled records are encountered, only the first is treated as outlined above. Subsequent records are bypassed until a nonzero record or a logical track header, whichever occurs first, is encountered. Then the TTR and NN of the last zero-filled record and the listing message:

ZERO RECORDS SUPPRESSED

are printed. The number of suppressed records may be computed by subtracting the NN of the first such record from that of the last.
Contents of the Dump Listing

If an entire job queue is being dumped, the output listing is produced in two sections. The first contains all queue control records; the second, the logical track area records.

If selective dumping of a job queue data set is stipulated, the program prints the specified parameters on the top of an output page, then follows with the appropriate QCRs and logical track area records. When particular job names are given as the selective dump parameters (see "Using The Job Queue Dump Program"), the records associated with each job are collected and printed under the given name. Each data set block (DSB) is printed immediately following the related step input/output table (SIOT) and labeled as such. The system message block (SMB) chain is printed as the last records for a given job.
Operational Considerations

- The time required to produce a full job queue dump is dependent upon space allocated to the SYS1.SYSJOBQE data set. The time required for this stand-alone operation may be reduced by using the tape output option of the program. In this way, the operating system may be more quickly brought back into service and the queue dump tape printed with a system utility program such as IEBPTPCH. Figure JQDMP-11 shows the execution time difference between tape and printer output for various queue devices.

<table>
<thead>
<tr>
<th>Queue Device</th>
<th>Printer (1403)</th>
<th>Tape (2400)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2311</td>
<td>11.3 minutes</td>
<td>4.0 minutes</td>
</tr>
<tr>
<td>2314</td>
<td>19.5 minutes</td>
<td>6.9 minutes</td>
</tr>
<tr>
<td>2301</td>
<td>49.5 minutes</td>
<td>17.4 minutes</td>
</tr>
</tbody>
</table>

Figure JQDMP-11. IMCJQDMP Execution Time per 100 Tracks of Input, As a Function of the Output Device
Chapter 5: IMBLIST
Formats and prints object modules, load modules, and CSECT identification records.
Contents

INTRODUCTION .................................................. 107
FEATURES ..................................................... 108
EXECUTING LIST ............................................... 109
Listing a Load Module ......................................... 109
Listing an Object Module .................................... 110
Listing CSECT Identification Records ...................... 111
OUTPUT ....................................................... 112
EXAMPLES
Example 1: Listing Several Object Modules ............... 119
Example 2: Using the LISTLOAD Control Statement ....... 120
Example 3: Using the LISTIDR Control Statements ......... 121
Example 4: Verifying an Object Deck ....................... 122
Example 5: Combining LISTOBJ, LISTLOAD, and LISTIDR .... 123

Figures

Figure LIST-1. Sample Module Summary for LISTLOAD .... 112
Figure LIST-2. Sample LISTLOAD Output - Load Module Map
(Part 1 of 2) ................................................. 113
Figure LIST-2. Sample LISTLOAD Output - Load Module Map
(Part 2 of 2) ................................................. 114
Figure LIST-3. Sample LISTLOAD Output - Cross-Reference
Listing (Part 1 of 2) ........................................... 115
Figure LIST-3. Sample LISTLOAD Output - Cross-Reference
Listing (Part 2 of 2) ........................................... 116
Figure LIST-4. Sample LISTOBJ Output ........................ 117
Figure LIST-5. Sample LISTIDR Output ........................ 118
IMBLIST is a service aid that operates as a problem program under the IBM System/360 Operating System. It produces the following kinds of output that can help you debug complex programs:

- A formatted listing of an object module.
- A formatted listing of a load module.
- A load module cross reference listing.
- A formatted listing of all information in a load module's CSECT identification records (IDRs).
- A listing of all program modifications for a load module or library.
Features

IMBLIST can help you solve programming problems in several ways.

If you want to verify an object module, you can use IMBLIST to obtain a formatted listing of it. The listing contains SYM records produced by TESTRAN (if there are any), the external symbol dictionary (ESD), the relocation dictionary (RLD), the text of the program containing instructions and data, and the END record.

If you are interested in the relationships of control sections in a load module, you can use IMBLIST to get a listing of the load module along with its module map and cross-reference listing. You can then examine the control sections in the load module, the overlay structure, and the cross-references for each control section.

If you want to trace modifications to the executable code in a control section, you can use IMBLIST to produce a formatted listing of all information in the load module's CSECT identification records (IDRs). An IDR provides the following information:

• It identifies the version and modification level of the language translator and the date that each control section was translated. (Translation data is available only for control sections that were produced by a translator that supports IDR generation.)

• It identifies the version and modification level of the linkage editor that built the load module and gives the date the load module was created.

• It identifies by date modifications to the load module performed by INASPEAP.

An IDR also may contain optional user-supplied data associated with the executable code of the control sections.
Executing IMBLIST

You control IMBLIST processing by supplying control statements in the input stream. You must code the control statements according to the following rules:

- Leave column 1 blank, unless you want to supply an optional symbolic name. A symbolic name must be terminated by one or more blanks.

- If a complete control statement will not fit on a single card, end the first card with a comma and continue on the next card. Begin all continuation cards in columns 2 - 16. You must not split parameters between two cards; the only exception is the MEMBER parameters, which may be split at any internal comma.

Listing a Load Module

Use the LISTLOAD control statement to get a formatted listing of a load module. The format of this statement is:

```
LISTLOAD [OUTPUT=MODLIST] [,TITLE=('title',position)]
    [XREF] [BOTH]
 [,DDN=ddname] [,MEMBER=(list,...) ]
```

The parameters of the LISTLOAD control statement are as follows:

**OUTPUT=type**

specifies the type of load module listing to be produced. OUTPUT=MODLIST requests a formatted listing of the control and text records of a load module, including its External Symbol Dictionary and Relocation Dictionary Records. OUTPUT=XREF requests a module map and cross-reference listing for the load module. OUTPUT=BOTH requests both a formatted listing of the load module and its map and cross-references. If this parameter is omitted, OUTPUT=BOTH will be assumed.

**TITLE=('title',position)**

specifies a title, from one to forty characters long, to be printed below the heading line on each page of output. (The heading line identifies the page number and the type of listing being printed, and is not subject to user control.) The position subparameter specifies whether or not the title should be indented; if TITLE=('title',1) is specified, or if the position parameter is omitted, the title will be printed flush left, that is, starting in the first column. If you want the title indented from the margin, use the position parameter to specify the number of characters that should be left blank before the title. Note: Do not punctuate your title with commas, since IMBLIST recognizes these as delimiters. Anything that follows an embedded comma in a title will be ignored.
DDN=ddname

identifies the DD statement that defines the data set containing the input module. If the DDN= parameter is omitted, IMBLIST will assume SYSLIB as the default ddname.

MEMBER={member1,...membern}

identifies the input object module(s) by membername or alias name. To specify more than one object module, enclose the list of names in parentheses and separate the names with commas. If you omit the MEMBER= parameter, IMBLIST will print all modules in the data set.

Listing an Object Module

Use the LISTOBJ control statement to obtain a listing of an object module. The format of this control statement is:

LISTOBJ [TITLE=('title',position)] [,DDN=ddname] [,MEMBER={member1,...membern}]

TITLE=('title',position)

specifies a title, from one to forty characters long, to be printed below the heading line on each page of output. (The heading line identifies the page number and the type of listing being printed, and is not subject to user control.) The position parameter specifies whether or not the title should be indented; if TITLE=('title',1) is specified, or if the position parameter is omitted, the title will be printed flush left, that is, starting in the first column. If you want the title indented from the margin, use the position parameter to specify the number of characters that should be left blank before the title. Note: Do not punctuate your title with commas, since IMBLIST recognizes these as delimiters. Anything that follows an embedded comma in a title will be ignored.

DDN=ddname

identifies the DD statement that defines the data set containing the input module. If the DDN= parameter is omitted, IMBLIST will assume SYSLIB as the default ddname.

MEMBER={member1,...membern}

identifies the input object module(s) by membername or alias name. To specify more than one object module, enclose the list of names in parentheses and separate the names with commas. CAUTION: You must include the MEMBER= parameter if the input object modules exist as members in a partitioned data set. If you do not include the MEMBER= parameter, IMBLIST will assume that the input data set is organized sequentially, and that it contains a single, continuous object module.
Listing CSECT Identification Records

Use the LISTIDR control statement to get a formatted listing of a module's CSECT identification record (IDR). The format is:

LISTIDR [ OUTPUT={IDENT } ] [,TITLE=('title',position)] [,DDN=ddname] [,MEMBER={ (member1,...membern)} ]

**OUTPUT= type**

specifies whether IMBLIST should print all CSECT identification records or only those containing IMASPZAP data and user data. If you specify OUTPUT=ALL, all IDRs associated with the module will be printed. If you specify OUTPUT=IDENT, IMBLIST will print only those IDRs that contain IMASPZAP data or user-supplied data. If you omit this parameter, IMBLIST will assume a default of OUTPUT=ALL.

**TITLE=('title',position)**

specifies a title, from one to forty characters long, to be printed below the heading line on each page of output. (The heading line identifies the page number and the type of listing being printed, and is not subject to user control.) The position parameter specifies whether or not the title should be indented; if TITLE=('title',l) is specified, or if the position parameter is omitted, the title will be printed flush left, that is, starting in the first column. If you want the title indented from the margin, use the position parameter to specify the number of characters that should be left blank before the title. Note: Do not punctuate your title with commas, since IMBLIST recognizes these as delimiters. Anything that follows an embedded comma in a title will be ignored.

**DDN=ddname**

identifies the DD statement that defines the data set containing the input module. If you omit the DDN= parameter, IMBLIST will assume SYSLIB as the default ddname.

**MEMBER={ (member1,...membern)}**

identifies the input load module(s) by membername or alias name. To specify more than one load module, enclose the list of names in parentheses and separate the names with commas. If you omit the MEMBER= parameter, IMBLIST will print all modules in the data set.
IMBLIST produces a separate listing for each control statement that you specify. The first page of each listing always shows the control statement as you entered it. The second page of the listing is a module summary, unless you requested LISTOBJ; in that case, no module summary will be produced, and the second page of the listing will be the beginning of the formatted output.

The module summary gives the member name (with aliases), the entry point, the linkage editor attributes, and system status index information (SSI) for the module being formatted. Figure LIST-1 shows a typical module summary.

```
*** MODULE SUMMARY ***

MEMBER NAME: PL1LOAD
MAIN ENTRY POINT: 000720
** ALIASES **
SECONDARY ENTRY POINT ADDRESSES ASSOCIATED WITH ALIASES:

*** LINKAGE EDITOR ATTRIBUTES OF MODULE ***

** BIT STATUS **
0 NOT-REN 1 NOT-REDS 2 NOT-CYVT 3 NOT-TEST
4 NOT-OL 5 BLOCK 6 EXEC 7 MULTI-RCD
8 NOT-DC 9 ZERO-ORG 10 RP > ZERO 11 RLD
12 EDIT 13 NO-SYMS 14 F-LEVEL 15 NOT-REFR

MODULE SSI: NONE
```

Figure LIST-1. Sample Module Summary for LISTLOAD

The third page of the listing (or, for LISTOBJ, the second page) is the beginning of the formatted output itself.

For LISTLOAD, this consists of the load module and/or the module map and cross-reference listing. Figure LIST-2 shows an example of LISTLOAD module map output. Figure LIST-3 shows an example of the cross-reference listing for the same module.

For LISTOBJ, the body of the listing consists of the object module listing, the module's external symbol dictionary, and its relocation dictionary. Figure LIST-4 shows an example of LISTOBJ output.

For LISTIDR, the third page of the listing begins a complete list of all CSECT identification records for the module. Figure LIST-5 shows an example of LISTIDR output.

Complete descriptions of the fields in the formatted output listings can be found in the publication IBM System/360 Operating System: Linkage Editor (E) Program Logic Manual, GY28-6610, and Linkage Editor (F) Program Logic Manual, GY28-6667.
Figure LIST-2. Sample LISTLOAD Output - Load Module Map (Part 1 of 2)

Chapter 5: IMBLIST 113
Figure LIST-2. Sample LISTLOAD Output - Load Module Map (Part 2 of 2)
### NUMERICAL MAP AND CROSS-REFERENCE LIST OF LOAD MODULE PL1LOAD

#### CONTROL SECTION

<table>
<thead>
<tr>
<th>LMOD LOC</th>
<th>NAME</th>
<th>LENGTH</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>PL1TC02</td>
<td>4B6</td>
<td>SD</td>
</tr>
<tr>
<td>4B8</td>
<td>PL1TC02A</td>
<td>260</td>
<td>SD</td>
</tr>
<tr>
<td>718</td>
<td>IHEMAIN</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>728</td>
<td>IHENTRY</td>
<td>0C</td>
<td>SD</td>
</tr>
<tr>
<td>738</td>
<td>C</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>738</td>
<td>A</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>740</td>
<td>IHESPRT</td>
<td>38</td>
<td>SD</td>
</tr>
</tbody>
</table>

### ENTRY

<table>
<thead>
<tr>
<th>LMOD LOC</th>
<th>CSECT LOC</th>
<th>IN CSECT</th>
<th>REFERS TO SYMBOL AT LMOD LOC</th>
<th>CSECT LOC</th>
<th>IN CSECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>PL1TC02</td>
<td>PL1TC02A</td>
<td>4B8</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>20</td>
<td>PL1TC02A</td>
<td>IHEASAC</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>4B8</td>
<td>24</td>
<td>PL1TC02A</td>
<td>IHSAADB</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>4B8</td>
<td>20</td>
<td>PL1TC02A</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>4B8</td>
<td>30</td>
<td>PL1TC02A</td>
<td>IHESAFA</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>4B8</td>
<td>44</td>
<td>PL1TC02A</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>500</td>
<td>48</td>
<td>PL1TC02A</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>504</td>
<td>4C</td>
<td>PL1TC02A</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>508</td>
<td>50</td>
<td>PL1TC02A</td>
<td>IHELEOA</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>508</td>
<td>54</td>
<td>PL1TC02A</td>
<td>IHELEOB</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>58</td>
<td>PL1TC02A</td>
<td>IHEIOCPT</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>514</td>
<td>5C</td>
<td>PL1TC02A</td>
<td>IHEIOBC</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>518</td>
<td>60</td>
<td>PL1TC02A</td>
<td>IHESAFB</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>51C</td>
<td>64</td>
<td>PL1TC02A</td>
<td>AA</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
<tr>
<td>520</td>
<td>68</td>
<td>PL1TC02A</td>
<td>C</td>
<td>730</td>
<td>00</td>
</tr>
<tr>
<td>524</td>
<td>6C</td>
<td>PL1TC02A</td>
<td>B</td>
<td>738</td>
<td>00</td>
</tr>
<tr>
<td>528</td>
<td>70</td>
<td>PL1TC02A</td>
<td>A</td>
<td>740</td>
<td>00</td>
</tr>
<tr>
<td>52C</td>
<td>74</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>600</td>
<td>148</td>
<td>PL1TC02A</td>
<td>A</td>
<td>740</td>
<td>00</td>
</tr>
<tr>
<td>604</td>
<td>150</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>614</td>
<td>15C</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>618</td>
<td>160</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>620</td>
<td>168</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>634</td>
<td>16C</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>638</td>
<td>174</td>
<td>PL1TC02A</td>
<td>IHESPRT</td>
<td>748</td>
<td>00</td>
</tr>
<tr>
<td>640</td>
<td>178</td>
<td>PL1TC02A</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>718</td>
<td>00</td>
<td>IHEMAIN</td>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>720</td>
<td>08</td>
<td>IHENTRY</td>
<td>IHESAFC</td>
<td>UNRESOLVED</td>
<td></td>
</tr>
</tbody>
</table>

| LENGTH OF LOAD MODULE | 780 |

### NUMERICAL MAP AND CROSS-REFERENCE LIST OF LOAD MODULE PL1LOAD

#### PSEUDO REGISTER

<table>
<thead>
<tr>
<th>VECTOR LOC</th>
<th>NAME</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>IHEQINV</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>IHEQERR</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>IHEQDIR</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>IHELDIR</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>IHEQSLA</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>IHEQSLB</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>PL1TC02B</td>
<td>4</td>
</tr>
<tr>
<td>1C</td>
<td>PL1TC02C</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>IHEQDIR</td>
<td>4</td>
</tr>
</tbody>
</table>

| LENGTH OF PSEUDO REGISTERS | 24 |

---

Figure LIST-3. Sample LISTLOAD Output - Cross Reference Listing (Part 1 of 2)
### Alphabetical Map of Load Module PL1LOAD

<table>
<thead>
<tr>
<th>NAME</th>
<th>LMOD LOC</th>
<th>LENGTH</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>740</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>B</td>
<td>738</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>C</td>
<td>730</td>
<td>04</td>
<td>SD</td>
</tr>
<tr>
<td>IHENTRY</td>
<td>720</td>
<td>0C</td>
<td>SD</td>
</tr>
<tr>
<td>IHESPF</td>
<td>748</td>
<td>3B</td>
<td>SD</td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>436</td>
<td>SD</td>
</tr>
<tr>
<td>PL1TC02A</td>
<td>488</td>
<td>260</td>
<td>SD</td>
</tr>
</tbody>
</table>

### Pseudo Register

<table>
<thead>
<tr>
<th>NAME</th>
<th>VECTOR LOC</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>IHEQERR</td>
<td>04</td>
<td>4</td>
</tr>
<tr>
<td>IHEQINV</td>
<td>00</td>
<td>4</td>
</tr>
<tr>
<td>IHEQLWF</td>
<td>0C</td>
<td>4</td>
</tr>
<tr>
<td>IHEQLWO</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>IHEQSLA</td>
<td>08</td>
<td>4</td>
</tr>
<tr>
<td>IHEQSPR</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>PL1TC02B</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>PL1TC02C</td>
<td>1C</td>
<td>4</td>
</tr>
</tbody>
</table>

### Alphabetical Cross-Reference List of Load Module PL1LOAD

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>AT LMOD LOC</th>
<th>CSECT LOC</th>
<th>IN CSECT</th>
<th>IS REFERRED TO BY LMOD LOC</th>
<th>CSECT LOC</th>
<th>IN CSECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>740</td>
<td>00</td>
<td>A</td>
<td>528 70</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>740</td>
<td>00</td>
<td>A</td>
<td>600 148</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>738</td>
<td>00</td>
<td>B</td>
<td>528 54</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>730</td>
<td>00</td>
<td>C</td>
<td>528 68</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>IHESPF</td>
<td>748</td>
<td>00</td>
<td>IHESPF</td>
<td>528 74</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>IHESPF</td>
<td>748</td>
<td>00</td>
<td>IHESPF</td>
<td>668 150</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>IHESPF</td>
<td>748</td>
<td>00</td>
<td>IHESPF</td>
<td>614 15C</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>IHESPF</td>
<td>748</td>
<td>00</td>
<td>IHESPF</td>
<td>620 168</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>4E0 28</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>4F8 2C</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>4F8 40</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>500 48</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>504 4C</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>60C 154</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>618 160</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>624 15C</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>630 178</td>
<td>PL1TC02A</td>
<td></td>
</tr>
<tr>
<td>PL1TC02</td>
<td>00</td>
<td>00</td>
<td>PL1TC02</td>
<td>71B 10</td>
<td>PL1TC02A</td>
<td></td>
</tr>
</tbody>
</table>

******END OF MAP AND CROSS-REFERENCE LISTING******

Figure LIST-3. Sample LISTLOAD Output - Cross Reference Listing
(Part 2 of 2)
Figure LIST-4. Sample LISTOBJ Output
<table>
<thead>
<tr>
<th>CSECT</th>
<th>YR/DAY</th>
<th>IMASP2AP DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMP1</td>
<td>71/329</td>
<td>FIX12345</td>
</tr>
<tr>
<td>SAMP2</td>
<td>71/329</td>
<td>LEVEL003</td>
</tr>
<tr>
<td>SAMP3</td>
<td>71/329</td>
<td>PATCH001</td>
</tr>
<tr>
<td>SAMP4</td>
<td>71/329</td>
<td>PATCH002</td>
</tr>
<tr>
<td>SAMP4</td>
<td>71/329</td>
<td>PATCH003</td>
</tr>
</tbody>
</table>

This load module was produced by linkage editor 360SED521 at level 21.01 on day 329 of year 71.

<table>
<thead>
<tr>
<th>CSECT</th>
<th>TRANSLATOR</th>
<th>VR MD</th>
<th>YR/DY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMP1</td>
<td>360SAS037</td>
<td>21 00</td>
<td>71/329</td>
</tr>
<tr>
<td>SAMP2</td>
<td>360SAS037</td>
<td>21 00</td>
<td>71/329</td>
</tr>
<tr>
<td>SAMP3</td>
<td>360SAS037</td>
<td>21 00</td>
<td>71/329</td>
</tr>
<tr>
<td>SAMP4</td>
<td>360SAS037</td>
<td>21 00</td>
<td>71/329</td>
</tr>
<tr>
<td>SAMP5</td>
<td>360SAS037</td>
<td>21 00</td>
<td>71/329</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CSECT</th>
<th>YR/DAY</th>
<th>USER DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMP1</td>
<td>71/329</td>
<td>CHANGE LEVEL 01</td>
</tr>
<tr>
<td>SAMP2</td>
<td>71/329</td>
<td>VERSION 6</td>
</tr>
<tr>
<td>SAMP3</td>
<td>71/329</td>
<td>FIX LEVEL 2735</td>
</tr>
<tr>
<td>SAMP4</td>
<td>71/329</td>
<td>SORT SUBROUTINE</td>
</tr>
<tr>
<td>SAMP5</td>
<td>71/329</td>
<td>CARD SCANNING SUBROUTINE</td>
</tr>
</tbody>
</table>

Figure LIST-5. Sample LISTIDR Output
Example 1: Listing Several Object Modules

In this example, IMBLIST is used to list all object modules contained in the data set named OBJMODS, three specific object modules from another data set called OBJMOD, and finally all object modules in OBJMOD.

```plaintext
//OBJLIST JOB MSGLEVEL=(1,1)
//LISTSTEP EXEC PGM=IMBLIST
//SYSPRINT DD SYSOUT=A
//OBJLIB DD DSN=OBJMODS,DISP=OLD
//OBJSDS DD DSN=OBJMOD=DISP=OLD
//SYSIN DD *
  LISTOBJ DDN=OBJSDS, TITLE=('OBJECT MODULE LISTING OF OBJSDS',20)
  LISTOBJ DDN=OBJLIB, MEMBER=(OBJ1,OBJ2,OBJ3), TITLE=('OBJECT MODULE LISTING OF OBJ1 OBJ2 OBJ3'),20)
  LISTOBJ DDN=OBJLIB, TITLE=('OBJ MOD LISTING OF ALL MODS IN OBJLIB',20)
/*
SYSPRINT DD Statement

defines the message data set. This statement must be included; if it is omitted, IMBLIST will produce no output.

OBJLIB and OBJSDS DD Statements

define input data sets that contain object modules.

SYSIN DD Statement

defines the data set in the input stream containing IMBLIST control statements.

LISTOBJ Control Statement #1

instructs IMBLIST to format the data set defined by the OBJSDS DD statement, treating them as a single continuous object module. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTOBJ Control Statement #2

instructs IMBLIST to format three members of the partitioned data set defined by the OBJLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTOBJ Control Statement #3

instructs IMBLIST to format the entire data set defined by the OBJLIB DD statement, treating it as a sequential data set. It also specifies a title for each page of output, to be indented 20 characters from the left margin.
```
Example 2: Using the LISTLOAD Control Statement

In this example, IMBLIST is used to produce formatted listings of several load modules.

```
//LOADLIST JOB MSGLEVEL=(1,1)
//LISTSTEP EXEC PGM=IMBLIST
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//LOADLIB DD DSNAME=LOADMOD,DISP=OLD
//SYSIN DD *

LISTLOAD OUTPUT=MODLIST,DDN=LOADLIB,
      MEMBER=TESTMOD,
      TITLE=('LOAD MODULE LISTING OF TESTMOD',20)
LISTLOAD OUTPUT=XREF,DDN=LOADLIB,
      MEMBER=(MOD1,MOD2,MOD3),
      TITLE=('XREF LISTINGS OF MOD1 MOD2 AND MOD3',20)
/*

In this example:

SYSPRINT DD Statement

defines the message data set.

SYSLIB DD Statement

defines an input data set, SYS1.LINKLIB, that contains load modules to be formatted.

LOADLIB DD Statement

defines a second input data set.

SYSIN DD Statement

defines the data set (in the input stream) containing the IMBLIST control statements.

LISTLOAD Control Statement #1

instructs IMBLIST to format the control and text records, including the external symbol dictionary and relocation dictionary records, of the load module TESTMOD in the data set defined by the LOADLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTLOAD Control Statement #2

instructs IMBLIST to produce a module map and cross-reference listing of the load modules MOD1, MOD2, and MOD3 in the data set defined by the LOADLIB DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

/*
LISTLOAD Control Statement #3

instructs IMBLIST to produce a formatted listing of the load module and its map and cross-reference listing. Because no DDN= parameter is included, the input data set is assumed to be the one defined by the SYSLIB DD statement. Because no MEMBER= parameter is specified, all load modules in the data set will be processed. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.

Example 3: Using the LISTIDR Control Statement

In this example, IMBLIST is used to list the CSECT identification records in several load modules.

```
//IDRLIST JOB MSGLEVEL=(1,1)
//LISTSTEP EXEC PGM=IMBLIST
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSN=SYS1.LINKLIB,DISP=OLD
//LOADLIB DD DSN=LOADMODS,DISP=OLD
//SYSIN DD *
  LISTIDR TITLE=('IDR LISTINGS OF ALL MODS IN LINKLIB',20)
  LISTIDR OUTPUT=IDENT,DDN=LOADLIB,MEMBER=TESTMOD
  TITLE=('LISTING OF MODIFICATIONS TO TESTMOD',20)
  LISTIDR OUTPUT=ALL,DDN=LOADLIB,MEMBER=(MOD1,MOD2,MOD3),
  TITLE=('IDR LISTINGS OF MOD1 MOD2 MOD3',20)
/*
In this example:

SYSPRINT DD Statement
defines the message data set.

SYSLIB DD Statement
defines the input data set SYS1.LINKLIB, which contains load modules to be processed.

LOADLIB DD Statement
defines a second input data set.

SYSIN DD Statement
defines the data set (in the input stream) containing the IMBLIST control statements.

LISTIDR Control Statement #1

instructs IMBLIST to list all CSECT identification records for all modules in SYS1.LINKLIB (this is the default data set since no DDN= parameter was included). It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTIDR Control Statement #2

instructs IMBLIST to list CSECT identification records that contain IMA$P2AP or user-supplied data for load module TESTMOD. TESTMOD is a member of the data set defined by the LOADLIB DD statement. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.
LISTIDR Control Statement #3

instructs IMBLIST to list all CSECT identification records for load modules MOD1, MOD2, and MOD3. These are members in the data set defined by the LOADLIB DD statement. This control statement also specifies a title for each page of output, to be indented 20 characters from the left margin.

Example 4: Verifying an Object Deck

In this example, IMBLIST is used to format and list an object module included in the input stream.

//LSTOBJDK JOB
//EXEC PGM=IMBLIST
//OBJDECK DD SYSOUT=A
//SYSIN DD *
   LISTOBJ DDN=OBJDECK,
       TITLE=('OBJECT DECK LISTING FOR MYJOB', 25)
/*

SYSPRINT DD Statement

defines the message data set.

OBJDECK DD Statement

defines the input data set, which follows immediately. In this case the input data set is an object deck.

SYSIN DD Statement

defines the data set containing IMBLIST control statements, which follows immediately.

LISTOBJ Control Statement

instructs IMBLIST to format the data set defined by the OBJDECK DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.
Example 5: Combining LISTOBJ, LISTLOAD, and LISTIDR

An unsuccessful attempt has been made to link edit an object module with two load modules to produce one large load module. This example shows how to use IMBLIST to verify all three modules.

```
//LISTOBJ JOB MSGLEVEL=(1,1)
// EXEC PGM=IMBLIST
//SYSPRINT DD SYSOUT=A
//OBJMOD DD DSN=MYMOD,DISP=OLD
//LOADMOD1 DD DSN=YOURMOD,DISP=OLD
//LOADMOD2 DD DSN=HISMOD,DISP=OLD
//SYSIN *
LISTOBJ DDN=OBJMOD,
   TITLE=('OBJECT LISTING FOR MYMOD',20)
LISTLOAD DDN=LOADMOD1,OUTPUT=BOTH,
   TITLE=('LISTING FOR YOURMOD',25)
LISTIDR DDN=LOADMOD1,OUTPUT=ALL,
   TITLE=('IDRS FOR YOURMOD',25)
LISTLOAD DDN=LOADMOD2,OUTPUT=BOTH,
   TITLE=('LISTING FOR HISMOD',25)
LISTIDR DDN=LOADMOD2,OUTPUT=ALL,
   TITLE=('IDRS FOR HISMOD',25)
/*
SYSPRINT DD statement
defines the message data set.

OBJMOD DD statement
defines an input load module data set.

LOADMOD1 and LOADMOD2 DD Statements
define input load module data sets.

SYSIN DD statement
defines the data set containing IMBLIST control statements, which follows immediately.

LISTOBJ Control Statement
instructs IMBLIST to format the data set defined by the OBJMOD DD statement. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTLOAD Control Statement #1
instructs IMBLIST to format all records associated with the data set defined by the LOADMOD1 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

LISTIDR Control Statement #1
instructs IMBLIST to list all CSECT identification records associated with the data set defined by the LOADMOD1 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.
```
LISTLOAD Control Statement #2

instructs IMBLIST to format all records associated with the data set defined by the LOADMOD2 DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.
Maps load modules.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>129</td>
</tr>
<tr>
<td>CHARACTERISTICS OF THE LOAD MODULE MAP</td>
<td>130</td>
</tr>
<tr>
<td>Load Module and Nucleus Maps</td>
<td>130</td>
</tr>
<tr>
<td>Link Pack Area Maps</td>
<td>130</td>
</tr>
<tr>
<td>Specialized Maps</td>
<td>130</td>
</tr>
<tr>
<td>INPUT TO MDMAP</td>
<td>131</td>
</tr>
<tr>
<td>Load Modules</td>
<td>131</td>
</tr>
<tr>
<td>The Modular Concept</td>
<td>131</td>
</tr>
<tr>
<td>Control Sections</td>
<td>131</td>
</tr>
<tr>
<td>Object Modules</td>
<td>131</td>
</tr>
<tr>
<td>External Symbol Dictionaries</td>
<td>132</td>
</tr>
<tr>
<td>Relocatable Load Dictionaries</td>
<td>133</td>
</tr>
<tr>
<td>Text</td>
<td>133</td>
</tr>
<tr>
<td>Linkage Editor Output</td>
<td>133</td>
</tr>
<tr>
<td>Load Module Attributes</td>
<td>135</td>
</tr>
<tr>
<td>Programmer-Assigned Attributes</td>
<td>135</td>
</tr>
<tr>
<td>Linkage Editor-Assigned Attributes</td>
<td>136</td>
</tr>
<tr>
<td>Load Modules with Overlay Characteristics</td>
<td>136</td>
</tr>
<tr>
<td>MVT Link Pack Area</td>
<td>137</td>
</tr>
<tr>
<td>MFT Resident Reenterable Load Module Area</td>
<td>137</td>
</tr>
<tr>
<td>Nucleus</td>
<td>138</td>
</tr>
<tr>
<td>EXECUTING MDMAP</td>
<td>140</td>
</tr>
<tr>
<td>JCL Statements</td>
<td>140</td>
</tr>
<tr>
<td>EXEC Statement Parameters</td>
<td>140</td>
</tr>
<tr>
<td>OUTPUT FORMATS</td>
<td>143</td>
</tr>
<tr>
<td>Numerical ESD Listing</td>
<td>143</td>
</tr>
<tr>
<td>Numerical RLD Listing</td>
<td>144</td>
</tr>
<tr>
<td>Alphabetic ESD Listing</td>
<td>144</td>
</tr>
<tr>
<td>Alphabetic RLD Listing</td>
<td>145</td>
</tr>
<tr>
<td>MDMAP EXAMPLES</td>
<td>146</td>
</tr>
<tr>
<td>Example 1: Mapping an MVT Link Pack Area and Nucleus</td>
<td>146</td>
</tr>
<tr>
<td>Example 2: Mapping the ESDs of a Load Module, Using the Relocation Option</td>
<td>148</td>
</tr>
<tr>
<td>Example 3: Mapping a Load Module with the DEBUG and Relocation Options</td>
<td>150</td>
</tr>
<tr>
<td>OPERATIONAL CONSIDERATIONS</td>
<td>152</td>
</tr>
</tbody>
</table>
Figures

Figure MDMAP-1. Structure of an Object Module .......... 132
Figure MDMAP-2. Creation of a Load Module by the Linkage Editor .. 134
Figure MDMAP-3. Structure of a Load Module With Overlay ........ 136
Figure MDMAP-4. Upper Main Storage After IPL of an MVT System .... 137
Figure MDMAP-5. Lower Main Storage Organization in MFT ........ 138
Figure MDMAP-6. A Nucleus After IPL .................. 139
Figure MDMAP-7. IBMMDMAP Execution JCL ................ 140
Figure MDMAP-8. Snap Dumps Taken When the DEBUG Parameter Is Used .. 141
Figure MDMAP-9. Excerpts from the Map Resulting From Example 1 .. 147
Figure MDMAP-10. Excerpts from the Map Resulting From Example 2 .. 149
Figure MDMAP-11. Excerpts from the Map Resulting From Example 3 .. 151
IMBMDMAP, the Load Module Mapping service aid program, operates under the control of IBM Operating System/360, and provides the facility for mapping:

- A system's nucleus;
- The load modules included in an MVT link pack area or an MFT resident reenterable load module area; or
- Load modules previously link edited into a partitioned data set.

In determining the cause of problems in the execution of system component programs or complex user problem programs, the load module maps produced by IMBMDMAP, used in conjunction with main storage dumps, constitute powerful debugging aids. They enable the user to readily locate and identify individual control sections and their entry points, and to verify load module attributes and aliases.
Characteristics of the Load Module Map

A load module map contains edited information regarding the control sections, entry points, aliases, external references, attributes, type codes, overlay segments and hierarchy designations for each load module for which a map is requested.

Load Module and Nucleus Maps

A map of load modules from a partitioned data set (PDS) or a map of a nucleus consists of external symbol dictionary (ESD) and relocatable load dictionary (RLD) items, sorted first to numeric order by location, and then to alphabetic order by name. ESD and RLD items are discussed more fully under their respective headings in this chapter.

Link Pack Area Maps

A map of an MVT link pack area (LPA) contains contents directory entries (CDEs); that of an MFT resident reenterable load module area (analogous to the MVT link pack area) contains loaded program request block (LPRB) entries. The nature of these entries is discussed under the headings "MVT Link Pack Area" and "MFT Resident Reenterable Load Module Area" in this chapter.

In a map of either area type, the entries are sorted numerically, then alphabetically; and the length, entry points, and relative addresses of each module in the area are listed.

Specialized Maps

The user can request a map containing only ESD items in numeric sequence when executing IMBMDMAP. Or, an address relocation value may be specified to the program; that value will be assigned as the map's base address, and the result will be the printing of an absolute main storage location for each record, providing an added storage dump debugging aid. Or, the user may request a map that includes a series of "snapshot" dumps, taken at strategic points in time during IMBMDMAP's execution, and useful in determining the cause of certain load module structural problems -- including those which might arise during execution of IMBMDMAP.
Input to MDMAP

Input to IMBMDMAP may be a load module, a link pack area (MVT), a resident reenterable load module area (MFT), or any OS/360 nucleus. The following sections describe the contents and function of each type of input.

Load Modules

A load module is composed of all the edited modules (object, load, or an intermix of both types) that are input to the linkage editor for a given linkage. In addition to text items, a load module contains composite ESD and RLD entries. Any load module is both relocatable and executable.

The Modular Concept

Every program is designed to fulfill a particular purpose. In achieving that purpose, a program can be divided into logical functional units. Each of these units, defined as a section of coding that performs a specific task or several related functions, can be termed a module.

Control Sections

A module contains one or more control sections. A control section, or CSECT, is a unit of instructions and data that, within itself, is an entity. All elements of a control section (CSECT) are loaded and executed in a constant relationship to one another. A CSECT is, therefore, the smallest separately relocatable unit of a program.

Object Modules

Each module within a program can be separately assembled or compiled by a language translator. During this processing, references between the module's component control sections are unresolved. Object modules, the output of the language translator, consist of control dictionaries and text. Control dictionaries contain the information necessary to resolve cross-references between control sections and modules. A module's text area contains its instructions and data. Figure MDMAP-1 illustrates the structure of an object module. An object module is relocatable, but not executable.
External Symbol Dictionaries

An external symbol dictionary (ESD) entry identifies and defines the position of the external symbols contained, or referred to, in a module. Each entry is classified as either an external name or an external reference.

External Names

An external name is a defined value within the module, bearing a name that can be referred to by any control section or by any separately assembled or compiled module. There are four types of external names:

a. Control Section Name: The symbolic name of a control section. The ESD entry specifies the name, the assembled origin, and the length of a control section. The defined value of the symbol is the address of the first byte of the control section.

b. Entry Name: A name within a control section defining a point in the coding unit where processing may begin, or "enter." The ESD entry specifies the assembled address of the name and identifies the control section to which it belongs.

c. Blank or Named Common Area: A control section used to reserve a main storage area (containing no data or instructions) for CSECTs supplied by other modules, or as a center for communication between modules within a program. The ESD entry specifies the name and length of a named common area. The name field of a blank common area contains blanks.

d. Private Code: An unnamed control section. The ESD entry specifies the assembled address and assigned length of the area. The name field contains blanks. Since it has no name, a private code area cannot be referred to by any other control section.
External References

An external reference is a symbol referred to in a given module, but defined as an external name in another module. The ESD dictionary for the current module specifies the name only.

Relocatable Load Dictionaries

Relocatable load dictionaries (RLDs) contain information about address constants within the module. Each RLD entry identifies an address constant by:

- Indicating its location within the module, and
- Identifying the ESD symbol whose contents are used in determining the value of the address constant.

For a detailed discussion of ESD and RLD items, see the publication, IBM System/360 Operating System: Linkage Editor and Loader, GC28-6538.

Text

A text item includes the addresses of the instructions and data in a module, and indicates the ESD entry defining the CSECT in which the subject text is contained.

Linkage Editor Output

The linkage editor's output, a completed load module, is placed in a partitioned data set (SYSLMOD library) as a named member. In addition to its member name, the load module may carry as many as sixteen other names, or aliases. Under MFT it can contain up to 524,288 bytes; MVT allows larger modules. Figure MDMAP-2 illustrates the relationship of input to output of the linkage process.

In linking the input modules, the linkage editor resolves all references between control sections, just as if they had been assembled as a single module. The output load module contains the information necessary to load and relocate the module in main storage, and to compute the relocated value of location-dependent address constants. When it places the load module in the output module library, the linkage editor stores the module's member name, aliases, and attribute control information in the library's PDS directory.
Figure MDMAP-2. Creation of a Load Module by the Linkage Editor
Load Module Attributes

Each load module has specific characteristics, or attributes, which are used by the control program when the module is loaded for execution. Some of these attributes are programmer-specified; others are assigned by the linkage editor as a result of information obtained during its processing of the module.

Programmer-Assigned Attributes

Attributes that can be assigned to a load module by the programmer, and the characteristics assumed by the load module under each assignment, are:

<table>
<thead>
<tr>
<th>ASSIGNED ATTRIBUTE</th>
<th>LOAD MODULE CHARACTERISTICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reenterable</td>
<td>Executable by more than one task at a time; cannot be modified by any other load module during execution; cannot modify itself unless disabling techniques (such as the ENQ macro instruction, the test and set (TS) instruction, etc.) are used to prevent another routine from using this load module.</td>
</tr>
<tr>
<td>Serially Reusable</td>
<td>Executable by only one task at a time; will initialize itself and/or will restore any altered instructions or data before a new task takes control.</td>
</tr>
<tr>
<td>Refreshable</td>
<td>Cannot be modified by itself or by any other load module during execution, since it must be capable of being replaced by a new copy during execution without changing the results of processing.</td>
</tr>
<tr>
<td>Scatter Format</td>
<td>Suitable for either block loading (placement in main storage in one contiguous block of space); or scatter loading (possible placement in main storage, by control section, in non-contiguous areas), thus taking better advantage of available storage space.</td>
</tr>
<tr>
<td>Hierarchy Format</td>
<td>Suitable for either block or scatter loading into either hierarchy 0 or hierarchy 1, as specified to the linkage editor when hierarchy support is included in the system.</td>
</tr>
<tr>
<td>Not Editable</td>
<td>Cannot be reprocessed by the linkage editor; that is, cannot be link edited again into a larger load module.</td>
</tr>
<tr>
<td>Only Loadable</td>
<td>Can be brought into main storage only by use of the LOAD macro instruction.</td>
</tr>
<tr>
<td>Downward Compatible</td>
<td>Can be reprocessed by either level E or level F of the linkage editor.</td>
</tr>
<tr>
<td>Overlay</td>
<td>Structured as directed by linkage editor OVERLAY statements.</td>
</tr>
<tr>
<td>Test</td>
<td>Applies only to Assembler Language programs that are to be tested; causes inclusion of the test symbol dictionary.</td>
</tr>
</tbody>
</table>
Linkage Editor-Assigned Attributes

Linkage editor-assigned attributes, and the load module characteristics that result, are:

<table>
<thead>
<tr>
<th>ASSIGNED ATTRIBUTE</th>
<th>LOAD MODULE CHARACTERISTICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Format</td>
<td>Suitable for block loading only.</td>
</tr>
<tr>
<td>Not Executable</td>
<td>Assigned when errors that would prevent successful execution of the load module are detected during linkage editing.</td>
</tr>
</tbody>
</table>

Load Modules with Overlay Characteristics

When a load module contains overlay characteristics, the linkage editor structures the module somewhat differently, incorporating segment and entry tables (SEGTABs and ENTABs) into the text.

The single segment table created by the linkage editor for an overlay program structure is used to keep track of:

- the relationship of the segments in the program;
- which segments are in main storage, or in the process of being loaded; and
- other control information.

Entry tables are linkage editor-generated for control program use in determining the segment to be loaded in response to a branch instruction or one of the macro instructions used to transfer control between overlay segments. Figure MDMAP-3 illustrates the structure of a load module containing overlay characteristics.

![Figure MDMAP-3. Structure of a Load Module With Overlay](image)
MVT Link Pack Area

The link pack area in MVT is a required feature of the system. It resides in upper main storage and contains reenterable routines from the linkage and supervisor call libraries (SYS1.LINKLIB and SYS1.SVCLIB).

MVT link pack area routines are available to all tasks requiring them, and thus need not be separately loaded into the various regions of main storage. Figure MDMAP-4 shows the arrangement of the library routines in an MVT link pack area. The types 3 and 4 SVC routines operate in the Supervisor state; the others generally operate in the same state as the calling routine.

In MVT systems, the link pack area control queue (LPACQ) is composed of contents directory entries (CDEs), which are linked together. Each CDE on the LPACQ describes a routine resident in the LPA, giving the name, entry point and other attributes.

For a more detailed explanation of the link pack area control queue and associated control blocks, consult the publications, MVT Supervisor, SY28-6659, and System Control Blocks, GC28-6628.

Figure MDMAP-4. Upper Main Storage After IPL of an MVT System

MFT Resident Reenterable Load Module Area

Under MFT, the resident reenterable load module area is a SYSGEN option. If the option is selected, the access method routines from SYS1.SVCLIB and the routines from SYS1.LINKLIB which are to be made resident are loaded during system initialization. Figure MDMAP-5 shows the relative location of these routines in main storage.
In MFT systems with the resident routine option selected, a queue of request blocks (RBs), called the reenterable load module queue, is maintained. Each request block describes a resident routine and can be either a loaded program request block (LPRB) or a loaded request block (LRB). For a more detailed description of the resident reenterable load module area and the associated control blocks, consult the publications Control Program with MFT, GY27-7128, and System Control Blocks, SC28-6628.

For ease of reference, the term "link pack area" will hereafter be used to denote either the MVT link pack area or the MFT resident reenterable load module area.

**Nucleus**

The nucleus, a member of the partitioned data set SYS1.NUCLEUS, is the resident portion of a control program. It is loaded into the fixed area of main storage at IPL time. The nucleus contains:

- all task supervision routines, except the nonresident types 3 and 4 SVC routines;
- the data management I/O supervisor and BLDL routine;
- the resident recovery management routines; and
- small transient areas into which certain nonresident SVC routines and I/O error handling routines, all resident in SYS1.SVCLIB, are loaded as needed.
Figure MDMAP-6 shows the layout of a nucleus after IPL. All control programs assign the nucleus to lower main storage.

Figure MDMAP-6. A Nucleus after IPL
Executing MDMAP

IMBMDMAP runs in the problem program mode under any of the OS/360 control programs.

IMBMDMAP can be executed by use of the job control statements described in the next section. Main storage requirements for executing IMBMDMAP are variable, depending upon the number of ESD and RLD items present in the module being mapped. The average execution requires about 35Ki; an extremely complex module might require 70K.

JCL Statements

The statements required for executing IMBMDMAP are shown in Figure MDMAP-7.

**Table: JCL Statements**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB Statement</td>
<td>This statement initiates the JOB.</td>
</tr>
<tr>
<td>EXEC Statement</td>
<td>This statement specifies the program name:</td>
</tr>
<tr>
<td></td>
<td>PGM=IMBMDMAP</td>
</tr>
<tr>
<td></td>
<td>The statement may, in addition, contain from one to four parameters,</td>
</tr>
<tr>
<td></td>
<td>randomly coded in any combination, to designate particular</td>
</tr>
<tr>
<td></td>
<td>specialization of a map's format. These parameters are:</td>
</tr>
<tr>
<td></td>
<td>*PARM='LINKPACK,BASIC,DEBUG,hhhhh'</td>
</tr>
<tr>
<td>SYSPRINT DD Statement</td>
<td>This statement defines a sequential message data set, such as</td>
</tr>
<tr>
<td></td>
<td>SYSOUT. The device defined for SYSPRINT may be a system</td>
</tr>
<tr>
<td></td>
<td>output class, system output device, magnetic tape volume, or direct</td>
</tr>
<tr>
<td></td>
<td>access volume.</td>
</tr>
<tr>
<td>//ddname DD Statement</td>
<td>This statement defines the load module to be mapped. One such</td>
</tr>
<tr>
<td></td>
<td>statement must be supplied for each load module for which an</td>
</tr>
<tr>
<td></td>
<td>map is to be produced. The statement's format is:</td>
</tr>
<tr>
<td></td>
<td>//ddname DD DSN=YYY(ZZZ),DISP=SHR</td>
</tr>
<tr>
<td></td>
<td>in which //ddname is any unique ddname,</td>
</tr>
<tr>
<td></td>
<td>YYY is the dsname of the partitioned data set in which the load</td>
</tr>
<tr>
<td></td>
<td>module to be mapped resides; e.g., SYS1.NUCLEUS.</td>
</tr>
<tr>
<td></td>
<td>ZZZ is the name or alias of the member load module -- resident in</td>
</tr>
<tr>
<td></td>
<td>the YYY data set -- to be mapped.</td>
</tr>
<tr>
<td></td>
<td>For example,</td>
</tr>
<tr>
<td></td>
<td>DSN=SYS1.NUCLEUS (IEANUC01) would accomplish mapping of the nucleus,</td>
</tr>
<tr>
<td></td>
<td>IEANUC01 is the name of a load module contained in the date set</td>
</tr>
<tr>
<td></td>
<td>called SYS1.NUCLEUS.</td>
</tr>
<tr>
<td>//SNAPDUMP DD Statement</td>
<td>This statement defines a sequential output data set, to be used</td>
</tr>
<tr>
<td></td>
<td>as output for the SNAP dumps taken as a result of the DEBUG parameter</td>
</tr>
<tr>
<td></td>
<td>(discussed under &quot;EXEC Statement Parameters&quot; in this section). This</td>
</tr>
<tr>
<td></td>
<td>DD statement is required only if the DEBUG parameter is specified on</td>
</tr>
<tr>
<td></td>
<td>the EXEC statement. The device specified for SNAPDUMP may be a system</td>
</tr>
<tr>
<td></td>
<td>output class or any system output device.</td>
</tr>
<tr>
<td>//SYSABEND DD Statement or //SYSUDUMP DD Statement</td>
<td>These statements define a sequential output data set, to be used as output for the ABEND dump issued by IMBMDMAP as a result of the DEBUG parameter (discussed under &quot;EXEC Statement Parameters&quot; in this section). One of these statements is required only if the DEBUG parameter is specified on the EXEC statement. The device specified for either SYSABEND or SYSUDUMP may be a system output class or any system output device.</td>
</tr>
</tbody>
</table>

Figure MDMAP-7. IMBMDMAP Execution JCL

EXEC Statements Parameters

The parameters associated with IMBMDMAP's EXEC statement are:
PARM='LINKPACK'

indicates that a map of the link pack area of main storage under an MVT or MFT environment is to be produced. To obtain a complete map of all LPA modules (that is, to pick up the resident SVC routines), the nucleus currently resident in main storage must also be mapped. The user must, therefore, include in the jobstream a DD statement for the nucleus currently in core when requesting a link pack area map.

PARM='BASIC'

specifies that the resultant map is to contain only numerically-ordered external symbol dictionary items. Neither the alphabetic ESD nor either of the RLD listings is produced. The LINKPACK BASIC map will contain only numeric CDE or LPRB items.

PARM='hhhhhh'

where hhhhhh is a hexadecimal address of from one to six characters, and represents a relocation, or base, address. This parameter causes the program to add this value to the relative address of each mapped item, thus providing an absolute main storage address for the output listing. This does not apply to mapping a nucleus, which already has relocated addresses.

PARM='DEBUG'

provides for up to seven "snapshots" of main storage, taken at strategic intervals during execution of IMBMDMAP. These dumps are useful in the debugging of module construction problems, including any which may arise during the running of IMBMDMAP. The content of each dump, and the interval at which it is taken, are described in Figure MDMAP-8.

<table>
<thead>
<tr>
<th>DUMP NUMBER</th>
<th>CONTENT</th>
<th>EXECUTION INTERVAL WHEN TAKEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ESD entries</td>
<td>After read</td>
</tr>
<tr>
<td></td>
<td>RLD entries</td>
<td>After read and first sorting pass</td>
</tr>
<tr>
<td>2</td>
<td>TRANSLATE table</td>
<td>After read (nucleus only)</td>
</tr>
<tr>
<td></td>
<td>SCATTER table</td>
<td>After read (nucleus only)</td>
</tr>
<tr>
<td>3</td>
<td>IPLTABLE</td>
<td>Simulation of IPL conditions to ensure accuracy of map (nucleus only)</td>
</tr>
<tr>
<td>4</td>
<td>TRANSLATE Table</td>
<td>After read</td>
</tr>
<tr>
<td></td>
<td>SCATTER Table</td>
<td>After read or after IPL relocation (nucleus only)</td>
</tr>
<tr>
<td>5</td>
<td>ESD entries</td>
<td>After relocation (scatter loading</td>
</tr>
<tr>
<td></td>
<td>RLD entries</td>
<td>After second sorting pass</td>
</tr>
<tr>
<td>6</td>
<td>ESD entries</td>
<td>After EXEC parameter relocation if specified</td>
</tr>
<tr>
<td></td>
<td>RLD entries</td>
<td>After EXEC parameter relocation if specified</td>
</tr>
<tr>
<td>7</td>
<td>CDE Table or LPRB Table</td>
<td>If MVT link pack area is being mapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If MFT link pack area is being mapped</td>
</tr>
</tbody>
</table>

Figure MDMAP-8. Snap Dumps Taken When the DEBUG Parameter Is Used
Of course, the production of certain of these dumps depends upon the nature of the area being mapped. For example, maps 2, 3, and 4 would not be provided if the modules in the nucleus were not being mapped.

The jobstream must include a SNAPDUMP DD statement, defining a sequential message data set, for output of these dumps.

Additionally, the DEBUG parameter produces:

- a hexadecimal dump of each mapped module, the text portion of which may be truncated;
- a dump of each involved PDS directory; and:
- an ABEND dump, if a SYSABEND or SYSUDUMP DD statement has been provided, since IMBMDMAP terminates with a user 100 ABEND code. The SYSABEND statement provides the user with a more complete main storage dump than the SYSUDUMP statement.
The printed output of IMBMDMAP is a formatted listing, giving the user detailed information about the CSECTs contained in each mapped load module. The "regulation" map -- that is, one not limited or expanded through the use of parameters -- provides this information in four sections.

Numerical ESD Listing

A map, ordered by main storage location, of:

- all attributes assigned to a given load module,
- all aliases assigned to the load module,
- the module's primary entry point,
- all CSECTs and entry points within the load module, and
- all external references within the load module.

CSECTs are listed to the left of a page; entry points, in two sets of columns, in the center; and external references to the right. Each CSECT is identified, in addition to name, address, and length, by one of six type codes:

- CM (Common) indicates that the name defines a common area, named or unnamed. A constant, $BLK COM, is assigned to the name field if the area is unnamed. For example,

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>LENGTH</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BLK COM</td>
<td>080090</td>
<td>000008</td>
<td>CM</td>
</tr>
</tbody>
</table>

- LR (Label Reference) indicates that the name defines a label, or symbol, within a control section.

- PC (Private Code) indicates that the name defines the beginning of an unnamed control section. A constant, $PRIVATE, is assigned to the name field of such CSECTs on the listing.

- PD (Private Code Marked Delete) indicates that this is an ENTAB or a SEGTAB. The code is used with modules having the overlay attribute.

- PR (Pseudo Register) defines an area external to the load module, but referred to within it, for which storage is allocated at the time the load module is executed.

- SD (Section Definition) indicates that the CSECT name defines the beginning of a named control section.

Two additional mutually exclusive CSECT definitions exist. When applicable, they appear immediately to the right of the type column on the listing. They are:
• SEG (Segment), a column heading under which appears the overlay segment in which a CSECT is contained. This is used with modules that have the overlay attribute. An example of the use of this identification is:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>LENGTH</th>
<th>TYPE</th>
<th>SEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SEGTPB</td>
<td>010A20</td>
<td>00004C</td>
<td>PD</td>
<td>01</td>
</tr>
<tr>
<td>IEKAA01</td>
<td>010A70</td>
<td>000114</td>
<td>SD</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEKXRS</td>
<td>017B78</td>
<td>0000E8</td>
<td>SD</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEKVFP</td>
<td>0225C8</td>
<td>000A60</td>
<td>SD</td>
<td>0D</td>
</tr>
<tr>
<td>IERP25</td>
<td>023028</td>
<td>000244</td>
<td>CM</td>
<td>0D</td>
</tr>
</tbody>
</table>

• HIERARCHY1, a constant printed beside the TYPE when a load module has been link edited with hierarchy designation, and the CSECT has been marked for loading into Hierarchy 1. Hierarchy 0 loading is indicated by the absence of such a notation. An example of the use of this identification is:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>LENGTH</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBTST05</td>
<td>000000</td>
<td>00000C</td>
<td>SD</td>
</tr>
<tr>
<td>IMBTST06</td>
<td>000010</td>
<td>00000C</td>
<td>SD</td>
</tr>
<tr>
<td>IMBTST07</td>
<td>000020</td>
<td>00000C</td>
<td>SD</td>
</tr>
<tr>
<td>IMBTST08</td>
<td>000030</td>
<td>00000C</td>
<td>SD</td>
</tr>
</tbody>
</table>

Numerical RLD Listing

A map, in order by main storage location, of the RLD items within a load module. Column headings on the listing are:

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>REL ADDR</th>
<th>IN CSECT</th>
<th>REFERS TO</th>
<th>IN CSECT</th>
</tr>
</thead>
</table>

This line of column headings may be interpreted as:

"In this location (column 1) . . . at this relative address (column 2) . . . in this control section (column 3) . . . there is a reference to the area identified by this name (column 4) . . . which resides in this control section (column 5)."

Alphabetic ESD Listing

The same as the numerical ESD listing but sorted to order by ESD name rather than by location.
Alphabetic RLD Listing

The same as the numerical RLD listing but sorted to order by reference name (column 4) rather than by location.

A map of a link pack area is formatted to give the following information, appearing in two sets of columns on a listing page:

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>LENGTH</th>
<th>NAME</th>
<th>EP ADR</th>
<th>EP REL ADR</th>
</tr>
</thead>
</table>

where EP ADR is entry point address, and EP REL ADR is entry point relative address.
MDMAP Examples

The following examples and figures illustrate the job control language statements needed to produce sample configurations of IMBMDMAP maps; excerpts from the resulting maps show the headings and data for the edited portions of the information.

Example 1: Mapping an MVT Link Pack Area and Nucleus

This example shows the statements for a map of an MVT nucleus and link pack area. Figure MDMAP-9 shows the resulting map.

```plaintext
//JOB JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IMBMDMAP,PARM='LINKPACK'
//DD1 DD DSN=SYS1.NUCLEUS(IEANUC01),DISP=SHR
//SYSPRINT DD SYSOUT=A
/*
```

146 Service Aids (Release 21)
Figure MDMAP-9. Excerpts From the Map Resulting From Example 1
Example 2: Mapping the ESDs of Load Module, Using the Relocation Option

The statements in this example will produce a basic map of a load module in SYS1.LINKLIB with relocation of the relative addresses to base address 10A20. Only the numerically arranged ESD entries are produced. See Figure MDMAP-10 also.

```
//JOB3 JOB MSGLEVEL=(1,1)
//MAPMOD EXEC PGM=IMBMDMAP,PARM='BASIC,10A20'
//DD3 DD DSN=SYS1.LINKLIB(IEFAS061),DISP=SHR
//SYSPRINT DD SYSOUT=A
```
**Figure MDMAP-10. Excerpts From the Map Resulting From Example 2**
Example 3: Mapping a Load Module with the DEBUG and Relocation Options

This example includes the DEBUG and relocation options. Because of the nature of the module being mapped, IMBMDMAP produced (in addition to the regulation map items) hexadecimal dumps of the PDS directory and of the load module, and snapdumps 1, 5, and 6 of the series described under the DEBUG parameter discussion. Figure MDMAP-11 shows excerpts from the listing.

```
//JOB4
//DEBUGDMP JOB MSGLEVEL=(1,1)
//DD4 EXEC PGM=IMBMDMAP,PARM='DEBUG,7FFFF'
//SYSPRINT DD DSN=SYS1.LINKLIB(IEFW21SD),DISP=SHR
//SNAPDUMP DD SYSOUT=A
/*
```

150 Service Aids (Release 21)
**LOAD MODULE MAP**  
**VERSION 0 LEVEL 0.19C**

**IMBMMDMP - SYS1.LINKLIEFSD061**  
**EP=010210 DATE=70.091 NUMERICALLY BY ESD ITEM**

**ATTRIBUTES** - REUS, RENT

**NAME** - IEFS0061  
**ALIASES** - IEFS0065 IEFS0164 IEFS0207 IEFS0065

**CSECT**  
**ENTRY**  
**EXT REF**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>LENGTH</th>
<th>TYPE</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>REL ADR</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>REL ADR</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>REL ADR</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEFS0061</td>
<td>010A20</td>
<td>000784</td>
<td>SD</td>
<td>QMTMSG</td>
<td>01110A</td>
<td>000035</td>
<td>IEFS0065</td>
<td>011120</td>
<td>00004E</td>
<td>IEFS0066</td>
<td>01116F8</td>
<td>000123</td>
</tr>
<tr>
<td>IEFS0064</td>
<td>011110</td>
<td>000025</td>
<td>SD</td>
<td>IEFS0164</td>
<td>011180</td>
<td>000026</td>
<td>IEFS0065</td>
<td>011188</td>
<td>000029</td>
<td>IEFS0068</td>
<td>01118C8</td>
<td>000100</td>
</tr>
<tr>
<td>IEFSTRL</td>
<td>011210</td>
<td>00004E</td>
<td>SD</td>
<td>IEFSTRL</td>
<td>011210</td>
<td>000026</td>
<td>IEFSLST</td>
<td>011120</td>
<td>000023</td>
<td>IEFD0066</td>
<td>01118C8</td>
<td>000100</td>
</tr>
<tr>
<td>IEFD0065</td>
<td>011180</td>
<td>000026</td>
<td>SD</td>
<td>IEFD0066</td>
<td>011188</td>
<td>000029</td>
<td>IEFD0067</td>
<td>01118C8</td>
<td>000100</td>
<td>IEFD0068</td>
<td>01118C8</td>
<td>000100</td>
</tr>
<tr>
<td>IEFJU1</td>
<td>011210</td>
<td>000026</td>
<td>SD</td>
<td>IEFJU1</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU2</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU3</td>
<td>01126F8</td>
<td>000014</td>
</tr>
<tr>
<td>IEFJU1</td>
<td>011210</td>
<td>000026</td>
<td>SD</td>
<td>IEFJU1</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU2</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU3</td>
<td>01126F8</td>
<td>000014</td>
</tr>
<tr>
<td>IEFJU1</td>
<td>011210</td>
<td>000026</td>
<td>SD</td>
<td>IEFJU1</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU2</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU3</td>
<td>01126F8</td>
<td>000014</td>
</tr>
<tr>
<td>IEFJU1</td>
<td>011210</td>
<td>000026</td>
<td>SD</td>
<td>IEFJU1</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU2</td>
<td>01126F8</td>
<td>000014</td>
<td>IEFJU3</td>
<td>01126F8</td>
<td>000014</td>
</tr>
</tbody>
</table>

**Figure MDMAP-11. Excerpts From the Map Resulting From Example 3**
Operational Considerations

You should pay careful attention to the following points when using IMBKMDMAP:

- A maximum of sixteen aliases will be printed, since the linkage editor assigns no more than sixteen.
- Link pack area maps for MFT do not include resident SVC routines.
- If the DEBUG parameter is used, a SNAPDUMP DD statement must be included in the jobstream.
- A DD statement is required for each load module to be mapped.
- To map a nucleus load module, the nucleus must be a member of a partitioned data set named SYSn.NUCLEUS, or incorrect will result.
- To obtain a main storage dump in the event of abnormal termination when using the DEBUG parameter, a SYSABEND or SYSUDUMP DD statement must be included in the jobstream.
Chapter 7: IMCOSJQD

Operates as a problem program to format and print the system job queue.
INTRODUCTION .................................................. 157

STARTING OSJQD .......................... 158
Restarting the System .................. 158
Invoking OSJQD by JCL ................. 159
Invoking OSJQD from the System Console .... 159

CONTROLLING OSJQD .................. 162
Defining the Input Data Set ............ 162
Using the Control Statements .......... 162

OSJQD OUTPUT .......................... 164
Record Identification Headings ........ 166
Output Comments ...................... 167
Error Recovery Procedures ............ 167

JCL AND CONTROL STATEMENT EXAMPLES ........ 169
Example 1: Dumping the Input Job Queues .......... 169
Example 2: Searching the Input Job Queues for a Specific Job 169
Example 3: Dumping the Entire Job Queue ........ 169

Figures

Figure OSJQD-1. Sample Job Control Statements Used to Invoke OSJQD .............. 159
Figure OSJQD-2. An Example of a User-Written Cataloged Procedure to Call OSJQD from the System Console .... 160
Figure OSJQD-3. A Sample Exchange Between the Operator and OSJQD 161
Figure OSJQD-4. OSJQD Execution Time as a Function of Output and Input Devices .......... 164
Figure OSJQD-5. Sample Job Control Statements and Control Statements Used to Print a 9-Track Tape Containing OSJQD Output ................. 164
Figure OSJQD-6. Sample OSJQD Output, Showing Output Comments .... 165

Chapter 7: IMCOSJQD 155
IMCOSJQD is a service aid that formats and prints the contents of the system job queue data set (SYS1.SYSJOBQE). IMCOSJQD is similar in function to the standalone service aid IMCJQDMP; however, IMCOSJQD operates as a problem program under the operating system, using standard access methods. IMCOSJQD can therefore be used without disrupting normal operating system processing; this is a great advantage in a large installation where stopping and restarting the operating system can take a long time.

To save even more time, you can specify that IMCOSJQD output should be stored temporarily on tape rather than printed immediately. The tape can be printed later, at your convenience.

You can use IMCOSJQD to dump the entire job queue, or you can select specific queues within the job queue and their associated logical tracks.
Starting IMCOSJQD

IMCOSJQD resides in the linkage library (SYS1.LINKLIB data set). You can invoke it either through job control statements in the input stream or through the system console.

In almost every case you will run IMCOSJQD to produce a listing that will help you diagnose a problem connected with the job queue. If the problem is relatively minor, and the system can continue processing, you can schedule IMCOSJQD immediately. For more severe problems, when the operating system cannot continue processing, you must restart the system before running IMCOSJQD.

Restarting the System

If the system goes down, first try a system restart (warm start); that is, IPL without reformattting the job queue. If the restart fails, take action as suggested below:

If your installation has a volume containing an alternate SYS1.SYSJOBQE data set, restart the system, requesting that that volume be formatted as the new job queue data set. Then run IMCOSJQD, specifying the original job queue data set as input.

If your installation has more than one operating system, and time is not critical, mount the volume containing the job queue on another system. Then run IMCOSJQD on that system, specifying the transferred data set as input.

If you cannot use an alternate volume, or if the volume containing the job queue data set cannot be moved, dump the job queue data set to another direct access volume with a different volume serial number, as follows:

1. Execute the IBCDMPRS utility to dump the SYS1.SYSJOBQE data set to a direct access device. Use IBCDMPRS control statements like those shown in the following example:

```
DUMP JOB DUMP 2314 ONTO 2314
  DUMP FROMDEV=2314,FROMADDR=230,
   TODEV=2314,TOADDR=232,
   VOLID=ALTQUE
END
```

For more information about the IBCDMPRS utility program, refer to the publication IBM System/360 Operating System, Utilities, GC28-6586.

2. Restart the operating system, specifying that the job queue should be reformatted. This will establish a fresh job queue.

3. Run IMCOSJQD, specifying the new direct access data set as input.
Invoking OSJQD by JCL

Figure OSJQD-1 shows an example of job control statements used to invoke IMCOSJQD. The statements are described below.

```plaintext
//DUMP JOB MSGLEVEL=(1,1) // EXEC PGM=IMCOSJQD // OSJQDIN DD DSNAME=SYS1.SYSJOBQE, UNIT=2314,VOL=SER=111111,DISP=SHR // OSJQDOUT DD UNIT=2400,DISP=(NEW,KEEP), DD DSNAME=QUEUEOUT,LABEL=(,NL) // SYSPRINT DD SYSOUT=A //SYSIN DD *1 /* Figure OSJQD-1. An Example of Job Control Statements Used to Invoke IMCOSJQD

EXEC Statement
calls for the execution of IMCOSJQD.

OSJQDIN DD Statement
defines the job queue to be processed. Note that the DD statement that defines the input data set must be named OSJQDIN.

OSJQDOUT DD Statement
defines the output data set. In this case the output data set, named QUEUEOUT, resides on a tape device. Note that the DD statement that defines the output data set must be named OSJQDOUT.

SYSPRINT DD Statement
defines the IMCOSJQD message data set.

SYSIN DD Statement (optional)
defines the data set that contains IMCOSJQD options. In this case, the options follow the job control statements in the input stream. If this statement is omitted, the operator will be prompted to supply options.

Invoking OSJQD from the System Console

If you wish, you can include the job control statements shown in Figure OSJQD-1 as a cataloged procedure in the procedure library (SYS1.PROCLIB data set); this allows the operator to initiate IMCOSJQD processing from the console.

Use the IEBUPDTE Utility to include your IMCOSJQD cataloged procedure in SYS1.PROCLIB. The name you specify in the ADD control statement for IEBUPDTE is the name of the procedure that you must specify in the START command. For information on using IEBUPDTE, refer to the publication IBM System/360 Operating System: Utilities, GC28-6586.
Figure OSJQD-2 shows an example of a cataloged procedure that calls IMCOSJQD.

```plaintext
//OSJBQDMP PROC REG=20,D='SYS1.SYSJOBQE',U=2314,VS=111111,
// DSP=SHR,UN=2400,DISP=(NEW,KEEP),DSN=QUEUEOUT
// EXEC PGM=IMCOSJQD,REGION=&REG.K
//OSJQDIN DD DSNAME=&D,UNIT=U,VOL=SER=VS,DISP=DS
//OSJQDOUT DD UNIT=UN,DISP=DS,DSNAME=DSN
//SYSPRINT DD SYSOUT=A
```

Figure OSJQD-2. An Example of a User-Written Cataloged Procedure to Call IMCOSJQD from the System Console

**PROC Statement**

defines the name of the cataloged procedure and default values for any symbolic parameters included in the remaining statements in the procedure. In this case, the defaults are as follows: the input data set is SYS1.SYSJOBQE, the output data set is QUEUEOUT, and the region size is 20K. Note that you can specify any name for the procedure on the PROC statement.

**EXEC Statement**

calls for the execution of IMCOSJQD, and specifies the region size by a symbolic parameter. (The default region size specified in the PROC statement is 20K; this is the minimum region size required for IMCOSJQD processing.)

**OSJQDIN DD Statement**

defines the input data set. In this case, symbolic parameters permit the operator to specify an input data set or accept the defaults specified in the PROC statement.

**OSJQDOUT DD Statement**

defines the output data set. In this case, symbolic parameters permit the operator to specify an output data set or accept the defaults specified in the PROC statement.

**SYSPRINT DD Statement**

defines the message data set.

Note that the SYSIN DD statement has been omitted from this cataloged procedure; as a result the operator will be prompted to supply options when he starts IMCOSJQD.
Figure OSJQD-3 shows an example of an exchange between the operator and IMCOSJQD while starting IMCOSJQD. Note that in this example the operator made an error the first time he selected dump parameters, and IMCOSJQD prompted him to correct his error.

```
start osjbqmp,,,reg=24
  
  00 IMC001A SPECIFY SELECT PARAMETERS OR END
  r00,'qcr=cls=c'
  01 IMC002A COMMAND ERROR - ENTER QDUMP PARAMETERS
  r01,'qcr=class=c'
  00 IMC001A SPECIFY SELECT PARAMETERS OR END
  r00,'qcr=class=g'
  
  IMC005I SPECIFIED QUEUE IS EMPTY
  02 IMC001A SPECIFY SELECT PARAMETERS OR END
  r02,'qcr=class=a,jobname=(myjob,youjob,hisjob)'
  
  IMC006I THESE JOBS NOT FOUND
  HISJOB
  03 IMC001A SPECIFY SELECT PARAMETERS OR END
  r03,'qcr=class=a,jobname=(myjob,herjob)'
  
  04 IMC001A SPECIFY SELECT PARAMETERS OR END
  r04,'end'
  IMC004I QDUMP COMPLETE
```

Figure OSJQD-3. A sample exchange between operator and IMCOSJQD.
Controlling OSJQD

You control IMCOSJQD processing by defining the input data set and by supplying control statements.

Defining the Input Data Set

In most cases, the input to IMCOSJQD will be the system job queue, SYS1.SYSJOBQ. However, IMCOSJQD will accept as input any data set on a direct access device that has the format of the system job queue. This feature is useful when you have transferred the contents of the SYS1.SYSJOBQ data set to another volume, as described earlier in "Preparing to Use IMCOSJQD".

Using the Control Statements

Several control statements allow you to specify how much of the job queue you want IMCOSJQD to format and print. You can enter these control statements in two ways:

- If you invoke IMCOSJQD with JCL and include a SYSIN DD *, you can include control statements as cards in the input stream. If you want more than one dump operation, you must supply a separate card for each dump. IMCOSJQD will process the cards sequentially and produce a separate output listing for each one. (Blank cards will be ignored.) IMCOSJQD will terminate when it reaches end-of-file.

- If you start IMCOSJQD from the console, or if you omit the SYSIN DD * statement from the JCL, IMCOSJQD will prompt you to supply dump options. In reply you should define one dump operation fully. IMCOSJQD will prompt you again when it has finished processing the first dump, and you can then define a new dump operation. If you want to terminate IMCOSJQD processing, you must wait for a prompting message and reply END. (See Figure OSJQD-3.)

There are four IMCOSJQD control statements: QCR=, JOBNAME=, ALL, and END.

QCR= \{ ASB, CLASS=y, FREE, HOLD, RJE, SYSOUT=x, SUBMIT \}

specifies that the job queue data set's master queue control record and the queue records associated with the named work queue should be formatted and printed. The parameters are mutually exclusive; if you want more than one specific work queue, you must request separate dump operations for each.

For each QCR= option, IMCOSJQD dumps the master queue control record, the requested minor queue control record, and the logical tracks associated with that minor queue. The QCR= options and the minor queue control records they request are as follows:
ASB - Automatic SYSIN Batching Queue
CLASS=y - An input job queue (A through O)
FREE - Free Track Queue
HOLD - Hold Queue
RJE - Remote Job Entry Work Queue
SYSOUT=x - An output job queue (A through Z and 0 through 9)
SUBMIT - TSO Background Reader Queue

JOBNAME=(jobname1,...,jobname4))

requests IMCOSJQD to search all fifteen input work queues for logical track areas assigned to the specified jobname(s). These will be dumped along with associated system message blocks and data set blocks.

Note that searching all the input work queues for a job is a time-consuming operation. To reduce this time, use the QCR=CLASS=x control statement in combination with the JOBNAME= control statement to specify the input class of the requested job(s). For this purpose both control statements may be coded on a single card or entered as a single reply to a prompting message. An example of such an entry is:

QCR=CLASS=B,JOBNAME= (NEWJOB)

ALL

requests a dump of the entire job queue. This is the default option; it will take effect if the operator replies to the message prompting him for dump options by entering r xx,'U'.
IMCOSJQD Output

IMCOSJQD output can be directed either to a printer device or to a scratch tape, from which it can be printed later. Immediate printing can take a long time, so in most cases you should direct IMCOSJQD's output to a tape. Figure IMCOSJQD-4 shows the differences in execution time per 100 tracks between tape and printer output for various devices on which the job queue can reside.

<table>
<thead>
<tr>
<th>Output Device</th>
<th>Queue Device</th>
<th>Printer (1403)</th>
<th>Tape (2400)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2311</td>
<td>11.3 minutes</td>
<td>4.0 minutes</td>
</tr>
<tr>
<td></td>
<td>2314</td>
<td>19.5 minutes</td>
<td>6.9 minutes</td>
</tr>
<tr>
<td></td>
<td>2301</td>
<td>49.5 minutes</td>
<td>17.4 minutes</td>
</tr>
</tbody>
</table>

Figure OSJQD-4. IMCOSJQD Execution Time per 100 Tracks of Input as a Function of Output and Input Devices

Once IMCOSJQD's output is on a scratch tape, you can print it at any time using IEBPTPCH. Figure OSJQD-5 shows an example of the job control statements needed for this operation. For more information, refer to the publication IBM System/360 Operating System, Utilities, GC28-6586.

```
//PRINT JOB MSGLEVEL=(1,1)
//EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=2400, LABEL=(,NL), VOL=SER=QDUMPT,
// DISP=(OLD,KEEP), DCB=(RECFM=F, BLKSIZE=121, LRECL=121)
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT PREFORM=M
/*
Figure OSJQD-5. Sample JCL and Control Statements Used to Print a 9-Track Tape Containing IMCOSJQD Output

Figure OSJQD-6 shows a sample listing of a job queue as produced by IMCOSJQD.
<table>
<thead>
<tr>
<th>TTR</th>
<th>NN</th>
<th>TYPE</th>
<th>DISP</th>
<th>SYSJOBQUE</th>
<th>CUMF</th>
<th>PAGE 0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001</td>
<td>QCR</td>
<td>0000</td>
<td>C000000C C000000C C0000000 C0000000 C0000000 C0000000 C0000000 C0000000</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000002</td>
<td>MSTR</td>
<td>0018</td>
<td>C0000000 C0000000 C0000000 C0000000 C0000000 C0000000</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000003</td>
<td>QCR</td>
<td>0000C</td>
<td>C000000C C000000C C000000C C000000C C000000C</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000004</td>
<td>ASB</td>
<td>0018</td>
<td>C000000C C000000C C000000C C000000C</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000005</td>
<td>OUT=A</td>
<td>0018</td>
<td>C000000C C000000C C000000C C000000C</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TTR</th>
<th>NN</th>
<th>TYPE</th>
<th>DISP</th>
<th>SYSJOBQUE</th>
<th>CUMF</th>
<th>PAGE 0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>000202</td>
<td>QCR</td>
<td>0000</td>
<td>C000000C C000000C C0000000 C0000000 C0000000 C0000000</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000203</td>
<td>RESRV</td>
<td>0018</td>
<td>C0000000 C0000000 C0000000</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000204</td>
<td>LTH</td>
<td>0000</td>
<td>E2E2E2E2 4BE2E2E2 E5E2E2E2 E7E2E2E2 E4E2E2E2</td>
<td>*SYS1.SYSVLOGX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000205</td>
<td>OUT=B</td>
<td>0018</td>
<td>C000000C C000000C C000000C C000000C C000000C C000000C C000000C C000000C</td>
<td>* ..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000206</td>
<td>ENTIRE RECORD CONTAINS BINARY ZEROS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000207</td>
<td>ZER RECORDS SUPPRESSED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000208</td>
<td>ZER RECORDS SUPPRESSED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sample OSJQD-6. Sample OSJQD Output, Showing Output Comments
Record Identification Headings

For each record, IMCOSJQD supplies information under the following headings:

**TTR**

Direct access address, relative to the beginning of the data set, for QCR and logical track records.

**NN** (not supplied for queue control records)

Gives the sequence number of the logical track record within the specific work queue. This is a hexadecimal number assigned to each new record as it is added to the queue. The first logical track header record in the queue is always 1; for each new record added to the queue, the value of NN is increased by 1.

**TYPE**

Identifies the record type. IMCOSJQD recognizes the record type in two ways: queue control records and logical track header records are identified through their position in the structure of the job queue. Records from the logical track area are identified by the value in the ID field of each record (byte 4, at offset X'03').

The following table shows the type labels and their significance. Where applicable, the ID field value is also shown.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ID</th>
<th>RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCR</td>
<td>-</td>
<td>Queue Control Record.</td>
</tr>
<tr>
<td>LTH</td>
<td>-</td>
<td>Logical Track Header Record</td>
</tr>
<tr>
<td>ACT</td>
<td>01</td>
<td>Account Control Table</td>
</tr>
<tr>
<td>DSB</td>
<td>15</td>
<td>Data Set Block</td>
</tr>
<tr>
<td>DSENQ</td>
<td>0F</td>
<td>Data Set Enqueue Table</td>
</tr>
<tr>
<td>DSNT</td>
<td>07</td>
<td>Data Set Name Table</td>
</tr>
<tr>
<td>JCT</td>
<td>00</td>
<td>Job Control Table</td>
</tr>
<tr>
<td>POT</td>
<td>0A</td>
<td>Procedure Override Table</td>
</tr>
<tr>
<td>SCT</td>
<td>02</td>
<td>Step Control Table</td>
</tr>
<tr>
<td>SCTX</td>
<td>0C</td>
<td>Step Control Table Extension</td>
</tr>
<tr>
<td>SIOT</td>
<td>03</td>
<td>Step Input Output Table</td>
</tr>
<tr>
<td>SMB</td>
<td>05</td>
<td>System Message Block</td>
</tr>
<tr>
<td>VOLT</td>
<td>06</td>
<td>Volume Table</td>
</tr>
</tbody>
</table>

If no TYPE identifier is shown in the listing, the record is either a job file control block (JFCB), job file control block extension (JFCBX), or system output class directory (SCD), etc.
The column headed TYPE in the listing also identifies the name of the specific work queue associated with a queue control record. The following table shows the work queue identifiers and their significance.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASB</td>
<td>Automatic SYSIN Batching Queue</td>
</tr>
<tr>
<td>CLS=y</td>
<td>System Input Job Class Queues; y is the class identifier (A through 0).</td>
</tr>
<tr>
<td>HOLD</td>
<td>Hold queue</td>
</tr>
<tr>
<td>MASTR</td>
<td>Master queue control record.</td>
</tr>
<tr>
<td>OUT=x</td>
<td>SYSTEM Output Class Queues; x is the class identifier (A through Z and 0 through 9).</td>
</tr>
<tr>
<td>RESRV</td>
<td>Reserved queue control records.</td>
</tr>
<tr>
<td>RJE</td>
<td>Remote Job Entry Queue.</td>
</tr>
<tr>
<td>SUBMT</td>
<td>Background Reader Queue</td>
</tr>
</tbody>
</table>

DISP gives the displacement within a record of the next hexadecimal word to be printed on the listing. The first word of the first printed line for a given record has a displacement of X'0000'; the first word of the second printed line, if one exists, has a displacement of X'0018'.

Output Comments
IMCOSJQD does not dump records that consist entirely of binary zeroes. Instead, when it comes to an all-zero record, it prints

ENTIRE RECORD CONTAINS BINARY ZEROES

and supplies TTR and NN information as described in the previous section. If IMCOSJQD comes to subsequent all-zero records, it will stop printing records until it comes to the next non-zero record or the next logical track header record. To indicate that all-zero records are not being printed, IMCOSJQD prints

ZERO RECORDS SUPPRESSED

See Figure OSJQD-6 for an example of an output listing showing these comments.

Error Recovery Procedures
IMCOSJQD error recovery depends on what kind of dump is being produced, what record was being read when the error occurred, and how many times the error has already occurred.
If you have requested a full dump (by specifying ALL when starting IMCOSJQD), IMCOSJQD will attempt to recover from all errors except those that occur while reading the master queue control record. To recover, IMCOSJQD prints an output error indicator, attempts to print the record associated with the error, and proceeds by reading the next record. If IMCOSJQD could not read the record associated with the error, it prints an appropriate output error indicator on the output listing, and then continues processing with the next queue record.

IMCOSJQD will permit up to 20 consecutive errors to occur before abandoning its attempts to recover. After the twentieth consecutive error, however, it will issue message IMC016I (PERMANENT I/O ERROR ON OSJQDIN), print the contents of the SYNAD buffer, and obtain the next dump option.

If you have requested a selective dump, or if an error occurs while reading the master queue control record, IMCOSJQD does not attempt to recover from any errors. It prints the record associated with the error or an output error indicator, issues message IMC016I, prints the contents of the SYNAD buffer, and obtains the next dump option. It does this by searching the SYSIN data set, if control statements were entered from the input stream, or by prompting the operator to supply dump options, if control statements were entered from the console. It will not terminate processing unless it encounters an END control statement or an end-of-file condition.

The error messages and their meanings are as follows:

badttr - INVALID TTR
IMCOSJQD will print this line in place of the record it could not find, followed by the contents of the SYNAD buffer.

UNABLE TO READ RECORD
An input/output error occurred while IMCOSJQD was trying to read a queue record. IMCOSJQD prints the TTR and NN values associated with the record, and substitutes this message for the contents of the record itself. The message is followed by the contents of the SYNAD buffer.

I/O ERROR READING FOLLOWING RECORD
An input/output error occurred while IMCOSJQD was trying to read a queue record; the error did not prevent IMCOSJQD from reading the record. IMCOSJQD prints this message to indicate that the record contains an error, and follows it with the record itself and the contents of the SYNAD buffer. It also prints the TTR and NN values associated with the record.

INVALID LENGTH RECORD
IMCOSJQD has encountered a record which is not a standard length (for a normal queue record, standard length is 176 bytes; for logical track header records, 20 bytes; for queue control records, 36 bytes). IMCOSJQD prints this message, followed by the record and its associated TTR and NN values. No SYNAD information is included.
JCL and Control Statement Examples

The following examples illustrate some of the functions that IMCOSJQD can perform.

Example 1: Dumping the Input Job Queues

This example shows how to format and print three input job queues, the automatic SYSIN batching queue, and two output job queues. Note that the only JCL statement shown is the SYSIN DD statement; for an example of the other JCL statements required to invoke IMCOSJQD, see Figure OSJQD-1.

```
//SYSIN DD *
QCR=CLASS=A
QCR=CLASS=B
  QCR=CLASS=C
  QCR=ASB
QCR=SYSOUT=A
QCR=SYSOUT=B
/*
```

Note that each control statement requests a separate queue, and that the control statements are entered in free form.

Example 2: Searching the Input Queues for a Specific Job

This example shows how to combine the QCR= and JOBNAME= control statements to search a limited number of queues for specific jobs. Note that the only JCL statement shown is the SYSIN DD statement; for an example of the other JCL statements required to invoke IMCOSJQD, see Figure OSJQD-1.

```
//SYSIN DD *
  QCR=CLASS=A,JOBNAME=(MYJOB,YOURJOB,HISJOB,HERJOB)
/*
```

Note that the maximum of four jobnames are specified in the JOBNAME= control statement.

Example 3: Dumping the Entire Job Queue

This example shows how to dump the entire job queue. Note that the only JCL statement shown is the SYSIN DD statement; for an example of the other JCL statements required to invoke IMCOSJQD, see Figure OSJQD-1.

```
//SYSIN DD *
  ALL
/*
```

Coding the ALL control statement has the same effect as replying r xx,'U' to message IMC001A.
Chapter 8: IMDPRDMP

Formats and prints dumps, TSO swap data set, and GTF trace data.
Figures

Figure PRDMP-1. IMDPRMDP Input and Output .................................. 176
Figure PRDMP-2. IMDPRMDP Function and Format Control Statements,
                  Standard and Abbreviated Forms .................................. 183
Figure PRDMP-3. Format of the EDIT Control Statement, Showing
                  All Valid Keywords .................................................... 188
Figure PRDMP-4. Priorities and Effects of EDIT Keywords Used
                  to Select Records by Trace Event Type ................................ 192
Figure PRDMP-5. Number of Lines of EDIT Output Per Buffer
                  As a Function of Maximum Buffer Size and
                  Trace Type ........................................................................ 195
Figure PRDMP-6. The Cataloged Procedure PRDMP ................................ 196
Figure PRDMP-7. Sample Queue Control Block Trace .......................... 197
Figure PRDMP-8. Sample MVT Link Pack Area Map ............................ 198
Figure PRDMP-9. Sample MVT Major Control Block Format ................. 199
Figure PRDMP-10. Sample MFT Major Control Block Format ............... 200
Figure PRDMP-11. Sample TSO Control Block Format (Part 1 of 3) ...... 201
Figure PRDMP-11. Sample TSO Control Block Format (Part 2 of 3) ...... 202
Figure PRDMP-11. Sample TSO Control Block Format (Part 3 of 3) ...... 203
Figure PRDMP-12. Sample TCB Summary for MVT or MFT With Subtasking 204
Figure PRDMP-13. Sample TCB Summary for MFT Without Subtasking .... 205
Figure PRDMP-14. Sample Dump - General Format ............................ 206
Figure PRDMP-15. Sample EDIT for Trace Data Set ........................... 207
IMDPRDMP is a service aid that prints system dump and trace information. Its principal function is to save you time; it does this by producing formatted output that you can scan quickly and easily. Within certain limits, it even allows you to suppress formatting and printing of information that does not interest you.

IMDPRDMP can process the following kinds of input:

- Dump data sets. These include:
  - IMDSADMP high-speed dump data set.
  - SYS1.DUMP data set.
  - TSO dump data set.
- TSO swap data sets.
- GTF trace data. This may exist as:
  - GTF external trace data set (usually called SYS1.TRACE).
  - GTF trace data in buffers within a main storage dump.
Figure PRDMP-1 shows the general characteristics of these types of input and how they relate to IMDPRDMP processing.

**INPUT**

- **Control Statements**
  - SYSIN or Console
  
  See Figure PRDMP-2.

- **Data Sets**
  - DUMPS
  - IMDSADMP
  - HI-speed output.

  or

  - SYS1.DUMP or TSO dump output

- SWAP
  - TSO Swap data sets.

- TRACE
  - GTF External trace data set.

**IMDPRDMP**

- Formats and prints input data sets.

**OUTPUT**

- Formatted output

- Messages

**NOTES:**

**Input DD Statements:**

- //SYSIN – Control statements.
- //TAPE or //anyname – Dump data sets and GTF trace data sets.

**Output DD Statements:**

- //PRINTER – Formatted output.
- //SYSPRINT – IMDPRDMP messages.

Figure PRDMP-1. IMDPRDMP Input and Output
Functions

You vary the formatting and printing of a dump by supplying IMDPRDMP control statements. You can enter these either as replies to prompting messages issued to the console, or as cards in the input stream.

The control statements provide the following functions:

Formatting Control Blocks

You can specify one control statement (FORMAT) that will cause IMDPRDMP to format all major system control blocks for each task in the system. When printed, the formatted output will look like a SYSABEND dump. Note: IMDSADMP low-speed dump tapes can be printed using IMDPRDMP, but they will not be formatted.

Editing GTF Trace Data

IMDPRDMP can format GTF trace data either as records in the trace data set or as buffers contained in a dump data set. You can edit trace data by specifying special keywords in the EDIT control statements. You can also write exit programs to inspect the data before IMDPRDMP formats it. Suggestions on how to write a user exit program will be provided in the Appendix: Writing EDIT User Programs.

Dumping the TSO Swap Data Set

If a failure occurs in the TSO subsystem or in the operating system, it is important to capture the TSO SWAP data set quickly so that TSO can be restarted without undue delay. You can do this by executing IMDPRDMP against a SWAP data set and a dump data set, and directing its output to tape. The tape may be printed later, at your convenience.

Clearing SYS1.DUMP

You can use IMDPRDMP to transfer the contents of the SYS1.DUMP data set to another data set for later formatting and printing at a more convenient time. This allows you to clear the SYS1.DUMP data set and resume processing without pausing to print the contents.

Selective Printing

In a single control statement called PRINT, you can specify precisely what areas of main storage you want IMDPRDMP to print. IMDPRDMP will format and print control blocks that are associated with specified areas of main storage, unless you specify only PRINT NUCLEUS or PRINT STORAGE.
PRINT allows you to specify printing of main storage areas that are associated with:

- A certain jobname.
- The current task.
- The task terminated by the damage assessment routine (DAR), where applicable.

You can also choose printing of the nucleus, system queue area, and/or all of allocated main storage.

Other control statements provide the following functions:

Resident System Module Mapping

IMDPRDMP can generate a link pack area map (MVT) or a resident reenterable load module area map (MFT). These maps describe resident system modules that were loaded into main storage by the nucleus initialization program (NIP). If you request a map, it will be printed on a separate page or pages of the IMDPRDMP formatted dump listing. These maps are useful in diagnosing system failures that occurred in program modules residing outside the user's region or partition.

Queue Control Block Trace

IMDPRDMP can provide a separate listing of the formatted queue control blocks for all task control blocks in the system. This listing, known as a QCB trace, may be used to resolve problems arising from task contention or system interlock.
Job Control Language Statements

Job control statements are important in determining what functions IMDPRDMP is to perform. This section describes the JCL statements that have special significance in executing IMDPRDMP. For more complete information about using JCL statements, refer to the publication IBM System/360 Operating System: Job Control Language Reference, GC28-6704.

**JOB Statement**

initiates the job, and provides the opportunity to override the default region size. IMDPRDMP requires a minimum region size of 64K. In most cases it executes more efficiently if its region size is larger than the minimum.

**EXEC Statement**

calls for the execution of IMDPRDMP and specifies certain actions that IMDPRDMP should take. The operands are:

**PGM=IMDPRDMP**

identifies IMDPRDMP to the system. This is the only required operand.


n should be used only when the input is a dump data set. It specifies what IMDPRDMP should do if it detects a permanent I/O error or format error while processing a dump.

- 0 -- print the nucleus (and the system queue area in MVT)
- 1 (or n not specified) -- print the entire input data set.
- 2 -- read the next control card from the SYSIN data set, or request control statements from the operator.

T specifies that the operator should be prompted to supply a title for the listing. If T is not specified, no prompting will occur.

**FREEnnn** specifies the size of the work space within IMDPRDMP's region or partition, excluding the size of the root module, control module, service modules, and input buffer area. nnn is the number of K-bytes in the work space. The default is 8K. This value is usually adequate; however, if the input data set is very large or complex, use the FREEnnn parameter to specify a larger work space. Also, if you need additional storage for a work area in an EDIT user program, use the FREEnnn parameter to reserve it.

**LINECNT=nn** specifies the number of lines per page to be printed on the output listing. The value specified for nn may be any decimal integer greater than 10. If this parameter is omitted, LINECNT=58 is assumed.
S instructs IMDPRDMP to issue a message which the operator may reply to at any time during processing. In his reply, the operator may stop IMDPRDMP from processing the current input data set and start a new phase of IMDPRDMP execution.

ER=x specifies what action the EDIT portion of IMDPRDMP should take if it detects an error in an exit or format routine while editing trace data from a dump or trace data set. The valid values of x and their meanings are:

0 -- EDIT will display in hexadecimal the record associated with the error and ignore the faulty routine in subsequent processing. If the error was in a format routine, all subsequent records that require processing by the same format routine will be ignored. If the error was in an exit routine, record formatting will continue.

1 -- EDIT will display in hexadecimal the record associated with the error and ignore the faulty routine in subsequent processing. If the error was in a format routine, all subsequent records that require processing by the same format routine will be dumped in hexadecimal. If the error was in an exit routine, record formatting will continue.

2 -- EDIT will display in hexadecimal the record associated with the error; EDIT will then terminate, and the next IMDPRDMP verb will be executed.

3 -- EDIT will allow ABEND to get control if a program check occurs in an exit or format routine. (If ER=3 is not specified, EDIT will issue the SPIE macro before entering the exit routine or format appendage and thus bypass ABEND processing.) If the recognized error is not a program check, the associated record will be dumped in hexadecimal; then EDIT will terminate and the next IMDPRDMP verb will be executed.

If this value is not included in the PARM= parameter list, a value of ER=2 will be assumed. Note that ER=1 and ER=2 are the same for exit programs.

Input DD Statements

{TAPE } DD Statement
{anyname}

defines an input dump or trace data set, which may reside on direct access storage or on tape. If the input data set is a dump, you can specify any ddname. Remember, however, that for ddnames other than TAPE, you must use a NEWDUMP control statement to identify the input data set. You can define any number of input data sets, as long as each is identified by a different ddname, and each ddname except TAPE is specified in a separate NEWDUMP control statement.

If the input is a GTF trace data set, the ddname must be the same as the one specified in the DDNAME parameter of the EDIT control statement. You can define any number of trace data sets, provided that you identify each data set with a unique ddname and a separate EDIT control statement.
Here are some of the parameters that you may use to describe each input data set; note that you may also need other parameters to describe certain types of input data set. For more information about DD statement parameters, refer to the publication Job Control Language Reference, GC28-6704.

* DSNAME=name (for direct access only)
  VOL=SER=volser
  UNIT=ddd
  * LABEL=|,NL| (for tape only)
|,SL|
  DISP=OLD
  DCB=(BUFNO=number,BLKSIZE=size) (for trace data sets only)

* If the input is a trace data set on a standard label tape, you must include the DSNAME= parameter and code the LABEL= parameter as LABEL=|,SL|.

Use the DCB parameter to specify a greater blocksize or more input buffers, or both, if you think the default values will be inadequate. The default blocksize is 3500 bytes; the default number of buffers is 2.

Do not specify a file sequence number in the LABEL= parameter if you intend to use the NEWTAPE or NEWDump FILESEQ=x control statement.

If you omit the TAPE DD statement, IMDPRDMP assumes that the input data is in the SYSUT1 data set, and has the correct format.

SYSWAPmn DD Statement

defines the TSO swap data set(s). With one possible exception, the operands should be identical to those used in the TSO procedure; the exception is that if the TSO procedure is coded DISP=(NEW,KEEP), the IMDPRDMP SYSWAPmn DD statement should be coded DISP=(OLD,KEEP). For an explanation of the values for m and n, refer to the TSO Guide.

SYSIN DD Statement

defines the data set that contains the IMDPRDMP control statements. (If you want to enter control statements from the console, omit this statement.)

Output DD Statements

PRINTER DD Statement

defines the IMDPRDMP output data set.

SYSPRINT DD Statement

defines the IMDPRDMP message data set.
SYSUT1 DD Statement (optional if input is a dump data set on tape, not used if input is an external trace data set)

defines a direct access work data set in which IMDPRDMP can collect input data. Performance improves when a SYSUT1 DD statement is included, because IMDPRDMP can reference dump information directly rather than searching for records in a sequential data set.

Required parameters are:

\[
\text{UNIT}=\text{ddd} \\
\text{SPACE}=(2052, (n,10))
\]

\(n\) is calculated as \((K/2048)+1\), where \(K\) is the number of bytes of input data.

SYSUT2 DD Statement

identifies a data set into which IMDPRDMP may transfer the contents of the SYS1.DUMP data set when time will not permit immediate formatting and printing of the SYS1.DUMP data set. For more information about this function, refer to the section "Transferring a Dump Data Set" later in this chapter.
User control statements allow you to select specific dump formatting options and control basic operation of the IMDPRDMP program.

IMDPRDMP will prompt you to supply control statements if no SYSIN data set exists, or if the supply of control statements in the SYSIN data set is exhausted before IMDPRDMP finds an END control statement.

There are two kinds of user control statements: function control statements and format control statements. All the control statements are fully described below. Figure PRDMP-2 shows the complete format of the function control statements.

### Function Control Statements

<table>
<thead>
<tr>
<th>Standard Form</th>
<th>Abbreviated Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVT={hhhhhh}</td>
<td>C={hhhhhh}</td>
</tr>
<tr>
<td>{ P</td>
<td>{ P</td>
</tr>
<tr>
<td>NEWDUMP [DDNAME={TAPE (anyname)}]</td>
<td>[DD={TAPE (anyname)}]</td>
</tr>
<tr>
<td>[FILESEQ=nn]</td>
<td>[F=nn]</td>
</tr>
<tr>
<td>[DUMPSEQ=nn]</td>
<td>[D=nn]</td>
</tr>
<tr>
<td>NEWTAPE</td>
<td>N</td>
</tr>
<tr>
<td>GO</td>
<td>G</td>
</tr>
<tr>
<td>ONGO [QCBTRACE],[LPAMAP],[FORMAT],[CVT=parm]</td>
<td>O [Q],[L],[F],[C=parm]</td>
</tr>
<tr>
<td></td>
<td>{ ,PRINT parm}</td>
</tr>
<tr>
<td></td>
<td>{ ,TSO parm}</td>
</tr>
<tr>
<td></td>
<td>{ ,EDIT parm}</td>
</tr>
<tr>
<td>TITLE text</td>
<td>T text</td>
</tr>
<tr>
<td>END</td>
<td>EN</td>
</tr>
</tbody>
</table>

### Format Control Statements

<table>
<thead>
<tr>
<th>Standard Form</th>
<th>Abbreviated Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCBTRACE</td>
<td>Q</td>
</tr>
<tr>
<td>LPAMAP</td>
<td>L</td>
</tr>
<tr>
<td>FORMAT</td>
<td>F</td>
</tr>
<tr>
<td>PRINT [ALL] [,CURRENT] [,NUCLEUS] [,STORAGE=(parm)]</td>
<td>P [A],[C],[N],[S=(parm)]</td>
</tr>
<tr>
<td></td>
<td>{ ,JOBNAME=(parm)},[F03]</td>
</tr>
<tr>
<td>TSO [SYSTEM=(YES USER NO)], [USER=(PRINT STORAGE FORMAT)]</td>
<td>TSO [S=(YES USER NO)], [U=(PRINT STORAGE FORMAT)]</td>
</tr>
<tr>
<td></td>
<td>{ ,EDIT parm}</td>
</tr>
<tr>
<td></td>
<td>E parm</td>
</tr>
</tbody>
</table>

Figure PRDMP-2. IMDPRDMP Function and Format Control Statements, Standard and Abbreviated Forms

### Function Control Statement

The function control statements allow you to control certain operations of the IMDPRDMP program, such as input tape handling, dump listing titles, job termination, etc.
CVT={\texttt{hhhhhh}}

allows you to specify the address of the communications vector table (CVT) in the main storage dump information. Use this if you think that the CVT pointer, in main storage location X'4C' of the system that was dumped, has been destroyed. If you omit this control statement, and IMDPRDMP cannot locate the CVT at location X'4C', it will scan the dump data set for unique identifiers associated with the CVT. If IMDPRDMP cannot locate the CVT by this scanning process, it will not format the input but will instead take action as specified by "$n$" in the parameter list supplied in the \texttt{PARM=} operand of the \texttt{EXEC} statement. Once the CVT has been located, it remains in effect until a \texttt{NEWDUMP} or \texttt{NEWTAPE} control statement is encountered.

\texttt{hhhhhh} is a hexadecimal address specifying the location of the CVT in the input dump information.

P specifies that the location found at X'4C' in the system on which IMDPRDMP is being executed can be used as a valid pointer to the CVT of the dumped system.

\texttt{NEWDUMP DDNAME=\{TAPE \}[,FILESEQ=n[,DUMPSEQ=n]
\{anyname\}}

defines an input data set. If you want to process more than one input data set in a single execution of IMDPRDMP you must supply a separate \texttt{NEWDUMP} control statement for each. If there is only one input data set, \texttt{NEWDUMP} is not needed.

\texttt{NEWDUMP} has three keyword parameters:

\texttt{DDNAME=}

gives the ddname of the input dump data set. This parameter is not required if the \texttt{TAPE DD} statement describes the input data set.

\texttt{FILESEQ=}

identifies the sequence number of an input data set that is one of several data sets on a single magnetic tape volume. If this parameter is omitted, IMDPRDMP assumes a default value of FILESEQ=1.

\texttt{DUMPSEQ=}

specifies the sequence number of a TSO dump that is one of several TSO dumps in a single data set. If this parameter is omitted, IMDPRDMP assumes a default value of DUMPSEQ=1.

\texttt{NEWTAPE}

has the same function as the \texttt{NEWDUMP} statement with parameters specified as \texttt{DDNAME=TAPE}, \texttt{FILESEQ=1}, and \texttt{DUMPSEQ=1}. Use it when the \texttt{TAPE DD} statement defines a single tape device on which are to be mounted multiple volumes, each containing one dump data set.
GO

specifies a predefined set of format control statements. They are: QCCTRLACE, LPAMAP, FORMAT, EDIT, and PRINT ALL. The effects of the GO control statement may be overridden by the ONGO control statement, which is described next.

ONGO [QCCTRLACE][,LPAMAP][,CVT=parm][,FORMAT][,PRINT parm]
[,EDIT parm][,TSO parm]

overrides the predefined set of format control statements requested by the GO control statement. The new set of format control statements will remain in effect for all subsequent uses of the GO control statement, until IMDRPDMP ends or a new ONGO control statement is entered. An ONGO control statement with no parameters restores the original GO functions: QCCTRLACE, LPAMAP, FORMAT, EDIT, and PRINT ALL.

NOTE: The ONGO-GO combination is not required for IMDRPDMP execution. You need not specify GO unless you want to use a predefined set of IMDRPDMP options; you need not use ONGO unless you want to change that predefined set. Each IMDRPDMP control statement may be specified directly at any time.

TITLE text

specifies a title to be printed at the top of each page in the output listing. Use this statement if you do not expect IMDRPDMP to prompt you to supply title information; that is, if you did not specify T in the PARM= field of the EXEC statement or if you are not entering control statement from the console. You can specify any title up to 62 characters in length.

END

signals IMDRPDMP to stop processing, close all data sets, and return control to the system control program. (If END is the only control statement specified, IMDRPDMP will load the data set defined by the SYSUT2 DD statement. See Example 1.)

Format Control Statements

Format control statements allow you to choose particular parts of the input to be formatted and printed.

QCCTRLACE

requests a trace of the queue control blocks (QCBs) in the input data set.

LPAMAP

causes IMDRPDMP to format and list the contents of the link pack area (MVT) or the resident reenterable load module area (MFT) in the input data set. If the input data set does not contain these areas, LPAMAP will be ignored.
causes IMDPRDMP to format and print the contents of the major system control blocks in the input data set.

PRINT [ALL][,CURRENT][,NUCLEUS][,STORAGE=(addresses)]
{,JOBNAME=(jobnames)][,F03]

indicates which parts of the input data set IMDPRDMP should print, according to several parameters.

ALL

instructs IMDPRDMP to print the nucleus, the system queue area, and all allocated regions of main storage in the input data set. This parameter also requests printing of the dumped system's registers.

CURRENT

instructs IMDPRDMP to print only the area of main storage that was associated with the current task when the input data set was created. This parameter also requests printing of the dumped system's registers.

NUCLEUS

instructs IMDPRDMP to print the nucleus portion of the input data set. If the input data set was taken from a system that was executing under MVT, the system queue area will also be printed. For the IBM System/360 Model 65 Multiprocessor, both the high and the low prefixes will be shown on the dump listing.

STORAGE=(startaddr1,endaddr1,...[,startaddrn,endaddrn])

allows you to supply beginning and ending addresses of areas in the input data set that you want printed. You may specify any number of pairs of hexadecimal addresses, so long as the beginning address in each pair is lower than the ending address. If you specify a beginning address and no ending address, IMDPRDMP prints the entire contents of main storage starting at the address you specify. If you do not specify any addresses, IMDPRDMP will print the entire contents of main storage, whether allocated or not. If you specify this parameter at all, IMDPRDMP will also print the dumped system's registers.

JOBNAME=(jobname1,jobname2...,jobname10)

allows you to limit the scope of the output listing to areas in main storage that are associated with specific jobs. You can specify up to ten jobnames. IMDPRDMP will print the areas associated with each job name in the order specified in the JOBNAME= parameter.

F03

instructs IMDPRDMP to print areas of main storage that were associated with a task terminated by the damage assessment routine (DAR).
TSO \[ \text{SYSTEM} = \{ \text{YES} \} \] \[ \text{USER} = \{ \text{PRINT} \} \] \[ \text{STORAGE} = \{ \text{FORMAT} \} \]

instructs IMDPRDMP to process the TSO dump data set and the TSO swap data sets. IMDPRDMP will not format the swap data sets unless you have defined them in SYSWAPmn DD statements.

Two parameters allow you to limit the amount of formatting that IMDPRDMP will do. If you omit a parameter, IMDPRDMP will give you maximum formatting.

\begin{itemize}
  \item \textbf{SYSTEM=}
  \begin{itemize}
    \item defines the extent of formatting for TSO system control blocks.
    \item The default value is \text{SYSTEM=\text{YES}}; it causes IMDPRDMP to format the following control blocks:
    \begin{itemize}
      \item TCB family for TSC
      \item TSCVT
      \item RCBs for each TS region
      \item Active TJBs
      \item SWAP CBs for each swap device
      \item Active TSBs
      \item User Main Storage Map.
    \end{itemize}
  \end{itemize}
  \item \textbf{USER=}
  \begin{itemize}
    \item defines the extent of formatting for the TSO user region and the TSO user control blocks. The default is \text{USER=\text{PRINT}}, which causes IMDPRDMP to format both the region and the control blocks. \text{USER=\text{STORAGE}} requests only the region, \text{USER=\text{FORMAT}} requests only the control blocks. \text{USER=\text{NO}} requests no formatting of the user region or control blocks.
  \end{itemize}
\end{itemize}

\textbf{EDIT Control Statement}

The EDIT control statement causes IMDPRDMP to obtain and process trace data created by the Generalized Trace Facility (GTF). Like other control statements, it may be specified either from the operator's console or through cards in the input stream.

\textbf{Edit Keyword Parameters}

The keywords associated with the EDIT control statement are shown in Figure PRDMP-3; they are described on the next page. All EDIT keyword parameters are optional.
EDIT  [EXIT=pgmname]  
[ [ DD ] ]
[ , DDNAME=ddname ]
[ , START=(ddd,hh.mm.ss) ]
[ , STOP=(ddd,hh.mm.ss) ]
[ , JOBNAME=(jobname1[,jobname2]...[,jobname5]) ]
[ , TCB=(address1[,address2]...[,address5]) ]
[ , SYS ]
[ , IO= (cuu1[,cuu2]...[,cuu50]) ]
[ , I=IO, SIO=SIO ]
[ , SVC= (svcnum1[,svcnum2]...[,svcnum256]) ]
[ , PI= (code1[,code2]...[,code15][,SSM]) ]
[ , EXT ]
[ , DSP ]
[ , USR= (ALL [symbol11][idvalue1][idrange1] ... [symbol12][idvalue2][idrange2] ... [symbol120][idvalue20][idrange20] ) ]

Figure PRDMP-3. Format of the EDIT Control Statement, Showing All Valid Keywords

EXIT=pgmname

defines the program name of a user-written exit routine that will inspect all trace records when IMDPRDMP gives it control. If the routine does not exist or cannot be loaded successfully, EDIT execution will terminate and the next IMDPRDMP control statement will be read.

DDNAME=ddname

specifies the name of the DD statement that defines the input trace data set. If you omit this keyword, IMDPRDMP assumes that trace data exists in buffers in a dump of main storage, and therefore will not accept any other EDIT keywords except EXIT. You must include this parameter if you want to selectively edit data management trace records.

START=(ddd,hh.mm.ss)

STOP=(ddd,hh.mm.ss)

These optional keywords specify that IMDPRDMP is to edit all trace records produced during the time of day indicated. If no START= time is specified, EDIT processing will begin at the beginning of the trace data set. If no STOP= time is specified, EDIT processing will continue to the end of the data set. If the trace data was recorded on an MFT system with no timer option, IMDPRDMP will ignore these keywords.

188 Service Aids (Release 21)
allows you to specify up to five 8-character jobnames for which EDIT will process trace data. If all the jobnames to be specified cannot fit on one line, close the first line with a right parenthesis followed by a comma; on the next line respecify the JOBNAME keyword with the additional jobnames.

This keyword is not valid if SYSM data is to be edited.

allows you to specify addresses of up to five task control blocks for which EDIT should process trace data. The addresses must be specified as 1- to 6-digit hexadecimal addresses. If all addresses cannot fit on one line, close the first line with a right parenthesis followed by a comma; on the next line respecify the TCB keyword with the additional addresses.

This keyword is not valid if SYSM data is to be edited.

This optional keyword requests EDIT to process all system event trace records -- that is, SVC, SIO, IO, PI, EXT, and DSP. If no EDIT keyword except DDNAME, EXIT, START, STOP, JOBNAME, and/or TCB is specified, EDIT will assume SYS as the default.

defines up to fifty different devices for which IO trace records, SIO trace records, or both should be formatted. If no specific devices are requested, all IO and/or SIO trace records will be formatted. If any specific devices are specified, only trace records associated with those devices will be formatted and all others will be ignored.

Devices should be specified as 3-digit device addresses. If all devices to be specified cannot fit on one line, close the first line with a right parenthesis followed by a comma; on the next line respecify the keyword with the remaining addresses.

defines up to 256 SVC trace records that EDIT is to format. svcnum is a 1- to 3-digit decimal SVC number.

If no svcnum parameters are specified or if both SVC and SVC= are specified, all SVC trace records will be formatted. If any SVC numbers are specified, only trace records associated with those SVC numbers will be formatted; all others will be ignored.

If all SVC numbers cannot fit on one line, close the first line with a right parenthesis followed by a comma; on the next line respecify the keyword with the remaining SVC numbers.
PI
PI=(code[,code2]...[code15],[SSM])

requests EDIT to format trace records associated with up to fifteen specified program interrupt codes. If no program interrupt codes are specified or if both PI and PI= are specified, all program interrupt trace records will be formatted. If any program interrupt codes are specified, only those program interrupt trace records will be formatted; all others will be ignored. If SSM is specified, EDIT will format SSM interrupt trace records for data recorded on a Model 65 Multiprocessing System.

If all codes to be specified cannot fit on one line, close the first line with a right parenthesis followed by a comma; on the next line respecify the keyword with the remaining codes.

EXT
requests that EDIT format all external interrupt trace records.

DSP
requests that EDIT format all dispatcher task-switch trace records.

USR=
USR=(symbol1[,symbol2][...symbol20])
USR=(idvalue1[,idvalue2][...idvalue20])
USR=(idrange1[,idrange2][...idrange20])

specifies which user/subsystem trace records should be formatted; (user or subsystem trace records are created by the GTF GTRACE macro.) You can specify up to 20 ID values, ranges or symbols representing single components or subsystems. Idvalue is a 3-digit hexadecimal ID specified in the GTRACE macro when the records to be formatted were created. Idrange is a pair of idvalues defining a range of records to be formatted, for example, USR=(010-040,BFD-BFF). If you want to edit data management trace records, specify USR=DMA1.

If ALL is specified alone or in combination with other parameters, all user or subsystem trace entries will be formatted. (See Figure PRDMP-4.)

If all parameters cannot fit on one line, close the first line with a right parenthesis followed by a comma, making sure that any idrange specified is complete; on the next line respecify the USR= keyword and continue with the remaining parameters.
EDIT Parameter Defaults and Priorities

All EDIT defaults depend on the presence or absence of the DDNAME= parameter.

- If it is present, the input is an external trace data set. All parameters are valid. If none except DDNAME= are specified, EDIT assumes a default of SYS.

- If it is absent, the input is a main storage dump containing trace buffers. No parameters except EXIT= are valid, since EDIT cannot select records from a dump. All records, both system and user, will be processed. If you attempt to select specific records, EDIT will prompt you to supply the missing DDNAME= parameter or terminate EDIT processing.

Figure PRDMP-4 summarizes the priority and effect of those EDIT parameters that select records by trace event type. Any keyword shown in the table can be considered to include as subsets all the parameters shown indented below it; for example, SVC=svccnum is a subset of SVC, and SVC is a subset of SYS. Any parameter can override another parameter in the same set that has a lower priority.

You should not combine any parameter with another parameter that can override it; for example, do not combine SIO with SIO=ddd. You can, however, combine parameters that are part of separate sets; for example, you can combine SIO=ddd with IO and SVC, or SYS with USR=ALL. You can also combine any parameters that have the same priority; for example, you can combine SIO=aaa with SIO=IO=bbb. In this case the effect will be IO=bbb and SIO=(aaa,bbb).

Note: START=, STOP=, JOBNAME=, and TCB= have no effect on trace event selection. They merely exercise further selectivity over records already chosen by default or by by parameters that select system trace events.
EDIT Parameter Priorities Trace Events Selected

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>SIO=10</td>
<td>All SIO, IO, SVC, PI, DSP, and EXT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIO</td>
<td>SIO</td>
<td>All SIO and IO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIO=ddd</td>
<td>SIO for device(s) ddd</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIO=IO=ddd</td>
<td>SIO and IO for device(s) ddd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>IO=ddd</td>
<td>IO for device(s) ddd</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO=SIo=ddd</td>
<td>IO and SIO for device(s) ddd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC</td>
<td>SVC=num</td>
<td>All SVCs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified SVCs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td>PI=code</td>
<td>All PIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified PI code(s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP</td>
<td>DSP</td>
<td>All DSP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXT</td>
<td>All EXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USR=ALL</td>
<td>All USR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USR=notall</td>
<td>Specified USR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure PRDMP-4. Priorities and Effects of EDIT Parameters Used to Select Records by Trace Event Type

Combining Control Statements

The following control statements may be combined freely with each other on a single card or in a single reply to a prompting message. They may be specified in any order.

CVT=parm
NEWTAPE
QCBTRACE
LPAMAP
FORMAT
EDIT (coded with no parameters)

All other control statements are restricted; that is, no more than one may be specified on a single card or in a single reply to a prompting message. If a control statement from this group is combined with any of the control statements listed above, the restricted control statement must come last.

Here are some examples of control statements combined correctly:

LPAMAP,EDIT,P N
F,QCBTRACE,EDIT DDNAME=TRACE,SVC,SIO=IO=ALL,PI
F,P F
Q,L,F,E,TSO S=YES,U=NO

192 Service Aids (Release 21)
Allocating Space for the Output Data Set

IMDPRDMP output is usually directed to a SYSOUT device; therefore in most cases its output is stored temporarily on a direct access storage device from which it is later written to the printer. This temporary storage allows the user to specify space allocation and blocking factors that will enhance IMDPRDMP's performance.

(Note that if time is not critical and the output data set is very large, the output data set may be allocated directly to a printer. Do this by specifying the UNIT parameter in the PRINTER DD statement, for example UNIT=00E.)

Specifying the Maximum Output Block Size

Since IMDPRDMP uses QSAM as the access method for the SYSOUT data set, you can improve performance by specifying the largest possible block size for the data set. The maximum block size within the limits of the track capacity of the output device can be calculated by the following method: Divide the maximum track capacity in bytes by the output record length, 121 bytes, and ignore any remainder. The quotient is the number of records per block. Multiply this number by 121 to find the maximum block size.

To illustrate: A 2311 disk storage unit has a track capacity of 3625 bytes. The IMDPRDMP output record length is 121 bytes. Thus the number of records per block is 29. This value multiplied by the output record length (121) gives the maximum block size, 3509 bytes. Code this value in the DCB= parameter of the PRINTER DD statement as follows:

DCB=(BLKSIZE=3509)

Increasing the Space Allocated to SYSOUT

Depending on the number of lines to be printed, the amount of space normally allocated to a SYSOUT data set may not be enough to contain the entire formatted dump or trace listing. To eliminate this potential problem, allocate extra direct access storage space for the SYSOUT data set via the SPACE= operand in the PRINTER DD statement that represents the data set. This extra space may be expressed in terms of bytes, tracks, or cylinders.
Use the table below to determine the approximate number of lines that will be printed in a dump listing. (The table does not include figures for the EDIT function of IMDPRDMP.)

<table>
<thead>
<tr>
<th>STORAGE SIZE</th>
<th>PRINTED LINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>16K</td>
<td>500</td>
</tr>
<tr>
<td>32K</td>
<td>1000</td>
</tr>
<tr>
<td>64K</td>
<td>2000</td>
</tr>
<tr>
<td>128K</td>
<td>4000</td>
</tr>
<tr>
<td>256K</td>
<td>8000</td>
</tr>
<tr>
<td>512K</td>
<td>16000</td>
</tr>
<tr>
<td>1024K</td>
<td>32000</td>
</tr>
</tbody>
</table>

Calculating Space Requirements by Block Size

Each printed line is represented by a 121-byte record; the space requirement can therefore be expressed in bytes as the record length multiplied by the number of records. As an example, the SPACE= operand for a 512K dump SYSOUT data set might be expressed as: SPACE=(121,(16000,100)).

If a blocking factor was specified for this SYSOUT data set (as discussed above), the space allocation can be expressed in terms of block size. For example, if the block size has been calculated as 3509 bytes (or a blocking factor of 29 records per block), the same 512K dump listing would require 552 blocks to contain all of the listing information. This block figure was calculated as follows:

\[
\text{16000 Output records} / \text{29 Records per block} = 552 \text{ Blocks}
\]

The PRINTER DD statement might then be expressed as:

```
//PRINTER DD SYSOUT=x,
// SPACE=(3509,(552,10)),
// UNIT=2311,DCB=(BLKSIZE=3509)
```

Calculating Space Requirements for EDIT Output

When GTF trace data is edited using the EDIT function of IMDPRDMP, the number of lines of output can be estimated provided the maximum GTF trace buffer size and the number of blocks to be edited are known. Figure PRDMP-5 shows the number of lines of EDIT output as a function of maximum buffer size (block size) and the type of trace.

Editing Internal Trace Data

To estimate the number of lines to be printed when GTF buffers are edited from a dump data set, use the following formula to determine the number of buffers:

\[
\text{(GTF Region Size-11K) / Buffer Size = Number of Buffers}
\]
Then multiply the number of buffers by the number of lines per buffer as shown in Figure PRDMP-5. (Note that the size of the region in which GTF was running must be known.)

<table>
<thead>
<tr>
<th>Maximum Trace Buffer Size</th>
<th>SYSM Trace</th>
<th>SYSM With User Time Stamp</th>
<th>Comprehensive Trace</th>
<th>Comprehensive Trace With User Time Stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>25</td>
<td>50</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>2048</td>
<td>50</td>
<td>100</td>
<td>60</td>
<td>120</td>
</tr>
<tr>
<td>3500</td>
<td>65</td>
<td>130</td>
<td>110</td>
<td>220</td>
</tr>
<tr>
<td>4096</td>
<td>100</td>
<td>200</td>
<td>120</td>
<td>240</td>
</tr>
</tbody>
</table>

Figure PRDMP-5. Number of Lines of EDIT Output per Buffer as a Function of Maximum Buffer Size and Trace Type

To illustrate: if a GTF internal (SYSM) trace is to be edited from a stand-alone dump taken by IMDSADMP, and GTF had been running in a 20K region, then the buffer size is 1024 bytes (implied by the specification MODE=INT); thus

Number of buffers = \((20K-11K)/1K\)

Number of buffers = 9

Figure PRDMP-5 indicates that for a SYSM trace the number of lines per buffer is 25; thus 9 (25) or 225 is the expected number of printed lines. The PRINTER DD statement in this case might be expressed as

```
//PRINTER DD SYSOUT=A,SPACE=(121,(225,10))
```

Editing an External Trace Data SET

To estimate the number of lines to be printed when GTF data is edited from the trace data set on a direct access device, determine the number of blocks per track and multiply that value by the allocated number of tracks; the resulting value is the number of blocks per data set. Multiply that value by the number of lines per block as indicated in Figure PRDMP-5.

For example: A comprehensive trace with user time stamps is to be edited from a data set that occupies 50 tracks of a device whose track capacity is 7200 bytes. The maximum blocksize for the trace (established by the IEFORDER DD statement in the GTF start procedure) is 3500 bytes. Thus the number of blocks per track (in round figures) is 2, and the number of blocks in the data set is 2(50) or 100. Figure PRDMP-5 indicates that for a comprehensive trace with user time stamps the number of lines per block is 220; thus the expected number of printed lines is 100(220) or 22000.

In this case the PRINTER DD statement might be expressed as:

```
//PRINTER DD SYSOUT=A,SPACE=(121,(22000,100))
```

If the trace data set is on a tape volume, you can estimate the maximum number of lines to be printed by calculating the number of blocks per foot of tape and multiplying by the length of the tape.

Chapter 8: IMDPRDMP 195
Cataloged Procedure

Figure PRDMP-6 shows the cataloged procedure, PRDMP, that IBM supplies for executing IMDPRDMP.

```
//PRDMP PROC
//DMP EXEC PGM=IMDPRDMP
//SYSPRINT DD SYSOUT=A
//TAPE DD DSN=SYS1.DUMP,DISP=OLD
//PRINTER DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(2052,(257,64))
```

Figure PRDMP-6. The cataloged procedure PRDMP.

The statements are explained below.

EXEC Statement

    calls for the execution of IMDPRDMP.

SYSPRINT DD Statement

    defines the IMDPRDMP message data set.

TAPE DD Statement

    defines the input data set. Unless overridden with other data set names, this statement defines SYS1.DUMP as the input data set.

PRINTER DD Statement

    defines the output data set.

SYSUT1 DD Statement

    defines the work data set.

Note that the SYSIN DD statement has been omitted. Unless this statement is supplied, IMDPRDMP will prompt the operator to enter control statements through the console.
Figures PRDMP-7 through PRDMP-15 are samples of IMDPRDMP output. The formats are explained in detail in the Programmer's Guide to Debugging, GC28-6670.

Figure PRDMP-7. Queue Control Block Trace Sample
**MODULE IMDSAMP**

**DATE** 11/12/70

**TIME** 00.15

**PAGE** 0001

---

****** LINK PACK AREA MAP ******

<table>
<thead>
<tr>
<th>NAME</th>
<th>STA</th>
<th>LGNH</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEELWAIT</td>
<td>072418</td>
<td>072418</td>
<td>0003E8 MAJOR</td>
</tr>
<tr>
<td>IGG0209Z</td>
<td>CT4000</td>
<td>CT4000</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0201Z</td>
<td>CT4C00</td>
<td>CT4C00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0201Y</td>
<td>CT5000</td>
<td>CT5000</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0200Z</td>
<td>CT5400</td>
<td>CT5400</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0200Y</td>
<td>CT5800</td>
<td>CT5800</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0200M</td>
<td>CT5C00</td>
<td>CT5C00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0200G</td>
<td>CT6000</td>
<td>CT6000</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0200F</td>
<td>CT6400</td>
<td>CT6400</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0200A</td>
<td>CT6800</td>
<td>CT6800</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0199M</td>
<td>CT6C00</td>
<td>CT6C00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0196B</td>
<td>CT7000</td>
<td>CT7000</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0196A</td>
<td>CT7400</td>
<td>CT7400</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0191D</td>
<td>CT8400</td>
<td>CT8400</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0191G</td>
<td>CT8800</td>
<td>CT8800</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0191D</td>
<td>CT8C00</td>
<td>CT8C00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0191B</td>
<td>CT9000</td>
<td>CT9000</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0191A</td>
<td>CT9400</td>
<td>CT9400</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0190S</td>
<td>CT9800</td>
<td>CT9800</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0190N</td>
<td>CT9C00</td>
<td>CT9C00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0190M</td>
<td>CTAC00</td>
<td>CTAC00</td>
<td>000400 MAJOR</td>
</tr>
<tr>
<td>IGG0190L</td>
<td>CTAD00</td>
<td>CTAD00</td>
<td>000500 MAJOR</td>
</tr>
<tr>
<td>IGG0190E</td>
<td>CTBD00</td>
<td>CTBD00</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190D</td>
<td>CTCD00</td>
<td>CTCD00</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190C</td>
<td>CTED00</td>
<td>CTED00</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190B</td>
<td>CTFD00</td>
<td>CTFD00</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190A</td>
<td>CTFE00</td>
<td>CTFE00</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190G</td>
<td>CTG000</td>
<td>CTG000</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190F</td>
<td>CTG010</td>
<td>CTG010</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190E</td>
<td>CTG020</td>
<td>CTG020</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190D</td>
<td>CTG030</td>
<td>CTG030</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190C</td>
<td>CTG040</td>
<td>CTG040</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190B</td>
<td>CTG050</td>
<td>CTG050</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190A</td>
<td>CTG060</td>
<td>CTG060</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190G</td>
<td>CTG070</td>
<td>CTG070</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190F</td>
<td>CTG080</td>
<td>CTG080</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190E</td>
<td>CTG090</td>
<td>CTG090</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190D</td>
<td>CTG100</td>
<td>CTG100</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190C</td>
<td>CTG110</td>
<td>CTG110</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190B</td>
<td>CTG120</td>
<td>CTG120</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190A</td>
<td>CTG130</td>
<td>CTG130</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190G</td>
<td>CTG140</td>
<td>CTG140</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190F</td>
<td>CTG150</td>
<td>CTG150</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190E</td>
<td>CTG160</td>
<td>CTG160</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190D</td>
<td>CTG170</td>
<td>CTG170</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190C</td>
<td>CTG180</td>
<td>CTG180</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190B</td>
<td>CTG190</td>
<td>CTG190</td>
<td>000600 MAJOR</td>
</tr>
<tr>
<td>IGG0190A</td>
<td>CTG200</td>
<td>CTG200</td>
<td>000600 MAJOR</td>
</tr>
</tbody>
</table>

---

Figure PRDMP-8. Sample MVT Link Pack Area Map
| Module IMDSADMP | Date 11/12/70 | Time 00.15 | Page 002 |

**Current Task**

<table>
<thead>
<tr>
<th>TCB 020400</th>
<th>RBP 0002E410</th>
<th>PIE 00000000</th>
<th>DEB 000240BC</th>
<th>TIM 0002E1F0</th>
<th>CMP 00000000</th>
<th>TRN 00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG 0002E770</td>
<td>PC-FLG F0000000</td>
<td>FLG 00000180</td>
<td>LLS 0002E19D</td>
<td>JLB 00000000</td>
<td>JPA 0002E1B3</td>
<td></td>
</tr>
<tr>
<td>RG 07-0</td>
<td>0000000C</td>
<td>00000006</td>
<td>00020FBC</td>
<td>00000000</td>
<td>0002D660</td>
<td>0002D1E8</td>
</tr>
<tr>
<td>RG 07-1</td>
<td>000002FA0</td>
<td>00000000</td>
<td>00020FBC</td>
<td>00000000</td>
<td>0002D660</td>
<td>0002E1B8</td>
</tr>
<tr>
<td>FSA 0006B4F0</td>
<td>TCB 00000000</td>
<td>TME 00000000</td>
<td>JST 0002D400</td>
<td>NTC 00000000</td>
<td>JTC 0002D1E8</td>
<td></td>
</tr>
<tr>
<td>LTC 00000000</td>
<td>IQE 00000000</td>
<td>ECB 00020FBC</td>
<td>TSPR 00000000</td>
<td>D-PQE 0002E770</td>
<td>SQS 0002D400</td>
<td></td>
</tr>
<tr>
<td>STA 00010E</td>
<td>C0G00000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>RES 00000000</td>
<td>JSCB 0002E33C</td>
</tr>
</tbody>
</table>

**Active RBS**

<table>
<thead>
<tr>
<th>PRB 02E410</th>
<th>RESV 00000000</th>
<th>APSW 00000000</th>
<th>WC-SZ-STAB 00040082</th>
<th>FL-CDE 0002E5E8</th>
<th>PSW FFG50009 AC05DEF9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q/TP 00000000</td>
<td>WT-LNK 0002D400</td>
<td>NM 00</td>
<td>EPA 0500E50</td>
<td>STA 0500E50</td>
<td>LN 000180</td>
</tr>
</tbody>
</table>

**Main Storage**

<table>
<thead>
<tr>
<th>D-PQE 0002E770</th>
<th>FIRST 0002E688</th>
<th>LAST 0002E688</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQE 026400</td>
<td>FF8 0005E000</td>
<td>LF8 0005E000</td>
</tr>
<tr>
<td>TCB 0002D1E0</td>
<td>RSI 0000F000</td>
<td>RAD 0005D800</td>
</tr>
</tbody>
</table>

**Load List**

<table>
<thead>
<tr>
<th>CDE 02E3E8</th>
<th>NM RETURNS</th>
<th>USE 01</th>
<th>RESP 01</th>
<th>ATR1 0B</th>
<th>EPA 0500C8</th>
<th>STA 0500C8</th>
<th>LN 000088</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDE 02EB50</td>
<td>NM 006019CC</td>
<td>USE 03</td>
<td>RESP 01</td>
<td>ATR1 0B</td>
<td>EPA 07E928</td>
<td>STA 07E928</td>
<td>LN 000088</td>
</tr>
<tr>
<td>CDE 02EB50</td>
<td>NM 006019CH</td>
<td>USE 03</td>
<td>RESP 01</td>
<td>ATR1 0B</td>
<td>EPA 07E888</td>
<td>STA 07E888</td>
<td>LN 000070</td>
</tr>
<tr>
<td>CDE 02EB74</td>
<td>NM 006019AC</td>
<td>USE 02</td>
<td>RESP 01</td>
<td>ATR1 0B</td>
<td>EPA 070848</td>
<td>STA 070848</td>
<td>LN 000088</td>
</tr>
<tr>
<td>CDE 02EB74</td>
<td>NM 006019AC</td>
<td>USE 03</td>
<td>RESP 01</td>
<td>ATR1 0B</td>
<td>EPA 070848</td>
<td>STA 070848</td>
<td>LN 000088</td>
</tr>
</tbody>
</table>

**Job Pack Queue**

<table>
<thead>
<tr>
<th>CDE 02E3E8</th>
<th>NM RETURNS</th>
<th>USE 01</th>
<th>RESP NA</th>
<th>ATR1 0B</th>
<th>EPA 0500C8</th>
<th>STA 0500C8</th>
<th>LN 000088</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDE 02E2E8</td>
<td>NM GO</td>
<td>USE 01</td>
<td>RESP NA</td>
<td>ATR1 0B</td>
<td>EPA 0500E50</td>
<td>STA 0500E50</td>
<td>LN 000180</td>
</tr>
</tbody>
</table>

**JEB 02D400**

<table>
<thead>
<tr>
<th>APPENDAGES</th>
<th>END OF EXT 07E888</th>
<th>SID 000072</th>
<th>PCI 000072</th>
<th>CH END 000072</th>
<th>AB END 000072</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFX 00000000</td>
<td>C9000006</td>
<td>00010B00</td>
<td>11000060</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCB 0402D400</td>
<td>ADEB 10000000</td>
<td>ASYN 00000000</td>
<td>SPRG 00000000</td>
<td>UPRG 0106BE18</td>
<td>PLST 18000000</td>
</tr>
<tr>
<td>AVT 0402E9A9</td>
<td>FM-UCB START END TRK5</td>
<td>580026AC</td>
<td>0002D003</td>
<td>COC00003</td>
<td>0001</td>
</tr>
</tbody>
</table>

**TIDT 02E1F0**

<table>
<thead>
<tr>
<th>JOB JOB4</th>
<th>STEP GO</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET</td>
<td>LN-STA</td>
</tr>
<tr>
<td>001B</td>
<td>002C</td>
</tr>
<tr>
<td>14040101</td>
<td>14040101</td>
</tr>
<tr>
<td>00410</td>
<td>00410</td>
</tr>
<tr>
<td>PMNG=*RA</td>
<td>00271500</td>
</tr>
<tr>
<td>00271500</td>
<td>00271500</td>
</tr>
<tr>
<td>00026AC</td>
<td>00026AC</td>
</tr>
</tbody>
</table>

---

Figure PRDMP-9. Sample MVT Major Control Block Format
Figure PRDMP-10. Sample MFT Major Control Block Format
Figure PRDMP-11. Sample TSO Control Block Format (Part 1 of 3)
Figure PRDMP-11. Sample TSO Control Block Format (Part 2 of 3)
Figure PRDMP-11. Sample TSO Control Block Format (Part 3 of 3)
Figure PRDMP-12. Sample TCB Summary for MVT or MFT With Subtasking
Figure PRDMP-13. Sample TCB Summary for MFT Without Subtasking
Figure PRDMP-15. Sample EDIT for Trace Data Set
The following examples illustrate some of the functions that IMDPRDMP can perform.

**Example 1: Using the Cataloged Procedure**

IBM supplies a cataloged procedure, called PRDMP, that defines the input and output data sets and a work data set for IMDPRDMP. This example shows how to use the cataloged procedure.

```jcl
//PROCDMP
// JOB MSGLEVEL=(1,1)
// EXEC PROC=PRDMP,PARM=DMP=T
//DMP.SYSIN DD *
GO
END
/*
```

In this example:

**EXEC Statement**

calls the cataloged procedure, and requests prompting for a dump title.

**DMP.SYSIN DD Statement**

defines the data set that contains the IMDPRDMP control statements. The data set follows immediately.

**GO Control Statement**

requests formatting and printing according the the QCBCTRACE, LPAMAP, FORMAT, EDIT, and PRINT ALL control statements.

**END Control Statement**

terminates IMDPRDMP processing.
Example 2: Transferring a Dump Data Set

If you need to clear the SYS1.DUMP data set quickly to make room for more dump information, you can use IMDPRDMP to transfer its contents to another data set. This new data set is not formatted or printed during this execution of IMDPRDMP, but it can be used as input later.

This example shows how to transfer the SYS1.DUMP data set, which ordinarily is a cataloged data set on direct access storage, to a tape volume described by the SYSUT2 DD statement.

```
//CLEAR
//EXEC
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A
//TAPE DD DSNAMES=SYS1.DUMP,DISP=OLD
//SYSUT2 DD UNIT=2400,VOL=SER=DUMP,LABEL=(NL),
//DISP=NEW
//SYSIN DD *
```

In this example:

SYSPRINT DD Statement
- defines the message data set.

PRINTER DD Statement
- defines the data set to which IMDPRDMP ordinarily directs its output. This statement must be included, even though its function is not used in this application.

TAPE DD Statement
- defines the input data set, SYS1.DUMP.

SYSUT2 DD Statement
- defines the data set to which the contents of SYS1.DUMP will be transferred

SYSIN DD Statement
- defines the data set that contains the IMDPRDMP control statements. The data set follows immediately.

END Control Statement
- terminates IMDPRDMP processing. Note that this is the only IMDPRDMP control statement needed.
Example 3: Processing Multiple Data Sets

IMDPRDMP can process any number of input data sets in a single execution, provided that each data set is properly defined by both DD statements and control statements. This example shows how to process three data sets in the same execution, two of which are on the same tape volume.

```
//NOLINK JOB MSGLEVEL=(1,1)
//EXEC PGM=IMDPRDMP,PARM=T
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A,SPACE=(121,(1600,100))
//TAPE DD UNIT=2400,VOL=SER=DPTAPE,
//      LABEL=(,NL),DISP=OLD
//TODAYDMP DD UNIT=SYSDA,VOL=SER=DPDADMP,
//      DSNAME=DMPDS,DISP=OLD
//SYSUT1 DD UNIT=SYSDA,DISP=(NEW,DELETE),
//      SPACE=(2052,(257,10))
//SYSIN DD *
ONGO Q,F,P A
GO NEWDUMP FILESEQ=2
GO NEWDUMP DDNAME=TODAYDMP
ONGO
GO END
/*
```

In this example:

EXEC Statement

invokes IMDPRDMP and requests that the operator be prompted for a dump title.

SYSPRINT DD Statement

defines the message data set.

PRINTER DD Statement

defines the output data set.

TAPE DD Statement

defines two input data sets on the same tape volume.

TODAYDMP DD Statement

identifies an input data set on a direct access volume.

SYSUT1 DD Statement

defines the IMDPRDMP work data set; it is required in this example because one of the input data sets is on a direct access volume.

SYSIN DD Statement

defines the data set containing the control statements. The data set follows immediately.

210 Service Aids (Release 21)
ONGO Control Statement with $Q$, $F$, and $P$ A parameters

alters the default parameters for all subsequent GO statements by deleting the LPAMAP and EDIT parameters.

GO Control Statement

instructs IMDPRDMP to process the first data set on the volume described by the TAPE DD statement.

NEWDUMP Control Statement with FILESEQ=2

identifies the second data set to be processed. Since no DDNAME= parameter is specified, IMDPRDMP assumes that the data set resides on the volume described by the TAPE DD statement. FILESEQ=2 specifies that the second data set on the volume should be processed.

GO Control Statement

instructs IMDPRDMP to process the data set described by the NEWDUMP control statement.

NEWDUMP Control Statement with DDNAME=TODAYDMP

identifies the third data set to be processed. DDNAME=TODAYDMP specifies that the data set is the one described by the TODAYDMP DD statement.

ONGO Control Statement with No Parameters

restores the original default parameters for the GO control statement.

GO Control Statement

instructs IMDPRDMP to process the data set described by the last NEWDUMP control statement. The original default parameters will be used.

END Statement

terminates IMDPRDMP processing.
Example 4: Processing a TSO Dump

IMDPRDMP can produce a complete dump of a TSO system by merging the system dump data set with the TSO swap data sets and formatting and printing the resulting data set. This example shows how to request a TSO dump.

```
//TSODUMP JOB MSGLEVEL=(1,1)
// EXEC PGM=IMDPRDMP
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A,SPACE=(121,(32000,100))
//TAPE DD UNIT=2400,VOL=SER=DUMP,
// LABEL=(NL),DISP=OLD
//SYSUT1 DD UNIT=SYSDA,DISP=(NEW,DELETE),
// SPACE=(2052,(513,10))
//SYSWAP00 DD DSNAME=SYS1.SWAP.D1100,UNIT=2311,
// VOL=SER=SWAP00,DISP=OLD
//SYIGIN DD *
LPAMAP
FORMAT
PRINT ALL
TSO
END
```

In this example:

**PRINTER DD Statement**

defines a very large output data set. If you prefer not to allocate so much space to SYSOUT=A, you can direct IMDPRDMP's output directly to a printer by coding this statement as:

```
//PRINTER DD UNIT=printeraddress
```

CAUTION: In a multiprogramming environment conflicts with the system writers may arise if the output data set is allocated directly to a printer.

**TAPE DD Statement**

defines the input dump data set.

**SYSUT1 DD Statement**

defines the IMDPRDMP work data set. Although it is not required in this example, it has been included to reduce IMDPRDMP processing time.

**SYSWAP00 DD Statement**

defines the swap data set for this particular TSO system. If this system had more than one swap data set, each one would have to be defined on a separate SYSWAPnn DD statement.

**SYIGIN DD Statement**

defines the data set containing the IMDPRDMP control statements. The data set follows immediately.
LPAMAP Control Statement
requests a map of the link pack area of the dumped system.

FORMAT Control Statement
requests that the major control blocks of the dumped system be formatted and printed.

PRINT Control Statement with the ALL Parameter
requests printing of the nucleus, system queue area, and all allocated regions of main storage in the dumped system.

TSO Control Statement with No Parameters
requests formatting and printing of all TSO system and user control blocks and TSO user regions.

END Control Statement
terminates processing.

Note that the GO control statement is not used in this example.

Example 5: Recording the TSO Swap Data Set

If the TSO subsystem fails and must be restarted, or if the operating system fails while TSO is being used, the TSO swap data sets must be recorded so that the failure may be diagnosed. The fastest way to do this is to restart the operating system, if necessary, and use IMDPRDMP to store the swap data set on tape before restarting TSO. Later, if the failure cannot be diagnosed solely by analyzing the main storage dump that was produce when the failure occurred, the swap data set that was stored on tape may be printed using IEBPTPCH.

This example shows how to use IMDPRDMP to store the swap data set and how to use IEBPTPCH to print it later.

```
//SWAPDUMP JOB MSGLEVEL=(1,1)
// EXEC PGM=IMDPRDMP,REGION=200K
//SYSPRINT DD SYSOUT=A
//PRINTER DD UNIT=2400,VOL=SER=SCRTCH,
 // DISP=(NEW,KEEP),LABEL=(,NL),
 // DCB=(BLKSIZE=1210,LRECL=121,RECFM=FB,BUFNO=100,OPTCD=C)
//TAPE DD UNIT=2400,VOL=SER=DUMP,
 // DISP=(OLD,KEEP),LABEL=(,NL)
//SYSUT1 DD UNIT=SYSDA,DISP=(NEW,DELETE),
 // SPACE=(2052, (513,10))
//SYSSWAP00 DD DSN=SYS1.SWAP00,VOL=SER=SWAP00,
 // DISP=(OLD,KEEP),UNIT=2314
//SYSSWAP01 DD DSN=SYS1.SWAP01,VOL=SER=SWAP01,
 // DISP=(OLD,KEEP),UNIT=2314
//SYSIN DD *

TSO
END
```
This example is actually composed of two job steps. In the IMDPRDMP step:

EXEC Statement

invokes IMDPRDMP and overrides the default region size with a value of 200K. This large figure is necessary to accommodate the large number of output buffers requested in the PRINTER DD statement.

SYSPRINT DD Statement

defines the message data set.

PRINTER DD Statement

defines the output data set. The output is directed to magnetic tape to make IMDPRDMP execution time as brief as possible; speed is further increased by the blocked records, large number of output buffers, and chain scheduling requested in the DCB operand.

PAPE DD Statement

defines an input dump data set.

SYSUT1 DD Statement

defines the IMDPRDMP work data set. Although it is not required in this example because the input data set is on tape, it is included to reduce IMDPRDMP processing time.

SYSWAP00 and SYSWAP01 DD Statements

define the TSO swap data sets. These statements are identical to those used in the cataloged procedure for starting TSO.

SYSIN DD Statement

defines the data set containing the IMDPRDMP control statements. The data set follows immediately.

TSO Control Statement

requests formatting and printing of TSO system and user control blocks and TSO user regions.

END Statement

terminates IMDPRDMP processing.

In the IEBPTPCH step:
EXEC Statement
invokes IEBPTPCH.

SYSPRINT DD Statement
defines the IEBPTPCH message data set.

SYSUT1 DD Statement
defines the input data set, which in the IMDPRDMP step was the output data set defined by the PRINTER DD statement.

SYSUT2 DD Statement
defines the IEBPTPCH output data set, which in this case is allocated directly to a printer.

SYSIN DD Statement
defines the data set containing the IEBPTPCH control statements. The data set follows immediately.

PRINT control statement with PREFORM=M
tells IEBPTPCH that each record begins with a machine control character.

Example 6: Editing GTF Trace Data from a Dump

//EDIT JOB MSGLEVEL=(1,1)
// EXEC PGM=IMDPRDMP
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A
//TAPE DD UNIT=2400,VOL=SER=DUMP,LABEL=(,NL),
// DISP=OLD UNIT=SYSDA,SPACE=(2052,(257,10))
//SYSUT1 DD EDIT
//SYSIN DD *
/*

In this example:

EXEC Statement
invokes IMDPRDMP.

SYSPRINT DD Statement
defines the message data set.

PRINTER DD Statement
defines the output data set.

TAPE DD Statement
defines the input data set.
SYSUT1 DD Statement

defines the IMDPRDMP work data set. Although it is not required unless the input data set is on direct access, it should be included to reduce IMDPRDMP processing time. When it is included, it must specify enough space to contain the entire dump.

SYSIN DD Statement

defines the data set containing the IMDPRDMP control statements. The data set follows immediately.

EDIT Control Statement with No Parameters

instructs IMDPRDMP to format and print GTF trace buffers in the input data set, according to the default options SYS and USR=ALL.

END Control Statement

terminates IMDPRDMP processing.

Example 7: Editing a GTF Trace Data Set

When GTF trace data is recorded in an external data set, you can specify editing of only selected records. This example shows how to edit trace records associated with two specific jobs.

```
//EDIT
//SYSPRINT
//PRINTER
//TRACE
//DISP=OLD,DCB=(BLKSIZE=2048,BUFNO=10)
//SYSIN DD *
EDIT DDNAME=TRACE,JOBNAME=X57A
EDIT DDNAME=TRACE,JOBNAME=X56B,
      SIO=IO=(190,191)
END
/*
```

In this example:

EXEC Statement

invokes IMDPRDMP and specifies the action that IMDPRDMP should take if a program interruption occurs in a user program.

SYSPRINT DD Statement

defines the message data set.

PRINTER DD Statement

defines the output data set.
TRACE DD Statement

defines the input trace data set. Since this data set resides on a non-labeled tape, subparameters of the DCB parameter are used to specify the same trace block size as was specified when creating the trace record, and to request that ten input buffers be used to process the trace data.

SYSIN DD Statement

defines the data set containing the IMDPRDMP control statements. The data set follows immediately.

EDIT Control Statement

instructs IMDPRDMP to edit trace records in the data set defined by the TRACE DD statement. The JOBNAME=X57A parameter requests editing for only those records associated with job X57A.

EDIT Control Statement

instructs IMDPRDMP to edit trace records from the data set defined by the TRACE DD statement; that is, the same data set referred to in the first EDIT statement. This time, however, only records associated with job X56B are to be processed; of those, only SIO and I/O interrupt traces for devices 190 and 191 are edited.

END Control Statement

terminates IMDPRDMP processing.
Chapter 9: IMAPTFLE
Generates JCL needed to apply a PTF and/or applies the PTF.
INTRODUCTION ................................................................. 223
Generate Function ......................................................... 224
Application Function ...................................................... 225
EXECUTING PTFLE ............................................................ 227
Application Function ....................................................... 227
Generate Function ........................................................... 228
CONTROL STATEMENTS ..................................................... 230
IMAPTFLE Control Statements ............................................ 230
IDENTIFY Control Statement ............................................. 231
OUTPUT ........................................................................... 233
Application Function ......................................................... 233
Generate Function .............................................................. 233
EXAMPLES ........................................................................ 236
Example 1: Generate Function ............................................ 236
Example 2: Application Function ......................................... 237
OPERATIONAL CONSIDERATIONS ........................................ 238
General Considerations ...................................................... 238
Generate Function Considerations ........................................ 239
Application Function Considerations .................................... 239

Figures

Figure PTFLE-1. The Generate Function of IMAPTFLE .................. 224
Figure PTFLE-2. The Application Function of IMAPTFLE ................ 225
Figure PTFLE-3. Minimum Main Storage Required for IMAPTFLE When Using the Application Function ......................... 227
Figure PTFLE-4. PTFLE Cataloged Procedure .............................. 227
Figure PTFLE-5. Sample JCL Needed to Execute the Generate Function of IMAPTFLE .................................................. 229
Figure PTFLE-6. Sample Linkage Editor (IEWL) Output from IMAPTFLE Generate Function (Sample #1) ............................. 234
Figure PTFLE-7. Sample Linkage Editor (IEWL) Output from IMAPTFLE Generate Function (Sample #2) ......................... 235
Figure PTFLE-8. Sample IEBCOPY Output from IMAPTFLE Generate Function ............................................................. 235
Figure PTFLE-9. Sample IEHIOSUP Output from IMAPTFLE Generate Function ............................................................. 235

Chapter 9: IMAPTFLE 221
The IMAPTFLE service aid is a problem program that is used to apply program temporary fixes (PTFs) to the IBM System/360 Operating System. You can use IMAPTFLE to:

- Generate the JCL and execution control statements needed to add PTF to an operating system in a later step, or
- Apply PTFs to an operating system by dynamically invoking the linkage editor.

Either the generate function or the application function of IMAPTFLE can be used to add PTFs to an operating system. The method is determined by the PARM operand of the EXEC statement in the execution JCL.

Both functions of IMAPTFLE require the Stage I output from sysgen as input. A brief explanation of the system generation process will clarify this requirement.

An operating system is generated in two stages. During Stage I, user-supplied macro instructions that describe both the installation's machine configuration and the desired programming options are analyzed and used to generate a job stream. The Stage I output contains the JCL that makes up this job stream. In Stage II, the job stream is processed to generate the libraries that form the user's operating system. Each member of these libraries has a certain set of attributes. When a member (load module) is to be modified by a PTF, these attributes must be maintained.

The attributes of the load module being modified by the PTF are contained in the JCL and control statements for the linkage editor and IEBCOPY utility generated during Stage I of system generation (SYSGEN). To ensure that the PTF will be correctly applied, IMAPTFLE uses the Stage I output to determine the attributes of the load module being replaced with the PTF module.
Generate Function

When using the generate function, two steps are required to apply PTFs. In the first step, IMAPTFLE generates the JCL and control statements for the linkage editor and IEBCOPY utility that are needed to apply the PTFs. In the second step, these JCL and control statements are executed to apply the PTFs to the operating system. Figure PTFLE-1 shows the generate function; the shaded area is performed after IMAPTFLE completes processing.

One control statement is provided for each module that comprises the PTF. Each control statement contains the module name and system status index (SSI) for the PTF module. (Alias names of modules that were copied by the IEBCOPY utility during system generation must be provided in additional control statements following the control statements that contain the associated module name. These additional control statements should contain only one alias each. They may not be used to add new alias names.) IMAPTFLE searches the Stage I output for the module names contained in the control statements. From this search, IMAPTFLE produces the necessary JCL and control statements needed to apply the PTFs to the operating system.

Figure PTFLE-1. The Generate Function of IMAPTFLE
Application Function

When using the application function, only one step is required to apply PTFs. One control statement is used for each module that comprises the PTF. Each control statement contains the module name and system status index (SSI) for the PTF module. (Alias names of modules that were copied by the IEBCOPY utility during system generation must be provided in additional control statements following the control statements that contain the associated module name. These additional control statements should contain only one alias each. They may not be used to add new alias names.) When preparing the input, the PTF object modules are placed immediately behind their corresponding control statement(s), as shown in Figure PTFLE-2.

IMAPTFLE reads all of the control statements and object modules into a work data set, creates a table of PTF module names, and then searches the Stage I output from the generated system being updated. When a module name from the Stage I output matches a PTF module name in the table, IMAPTFLE internally produces the information necessary to apply the PTF, and then invokes the linkage editor to update the operating system. IMAPTFLE then repeats the operation until all PTFs have been applied or the Stage I output reaches end-of-file.

![Diagram of IMAPTFLE application function](PTFLE.png)

Figure PTFLE-2. The Application Function of IMAPTFLE
Executing IMAPTFLE

The requirements for executing the IMAPTFLE service aid vary according to the desired function: generate or application.

Application Function

For execution of the application function, the main storage space is dependent on both the linkage editor and operating system, as shown in Figure PTFLE-3. Input to IMAPTFLE consists of the Stage I output from the generated system to be updated, IMAPTFLE control statements identifying the CSECTs being replaced, and the object module PTF CSECT replacements. The control statements are discussed under "IMAPTFLE Control Statement."

<table>
<thead>
<tr>
<th>Design Level of Linkage Editor</th>
<th>Minimum Main Storage Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MVT</td>
</tr>
<tr>
<td>44K(F)</td>
<td>68K</td>
</tr>
<tr>
<td>88K(F)</td>
<td>109K</td>
</tr>
<tr>
<td>128K(F)</td>
<td>149K</td>
</tr>
</tbody>
</table>

Figure PTFLE-3. Minimum Main Storage Required for IMAPTFLE When Using The Application Function.

Figure PTFLE-4 shows the cataloged procedure that IBM supplies for executing the application function of IMAPTFLE. This procedure, called PTFLE, resides in the SYS1.PROCLIB data set.

// PROC USE='IEWL',LIB1=LINKLIB,REG=68K
//PTF EXEC PGM=IMAPTFLE,PARM=&USE,REGION=&REG
//PRINT DD SYSOUT=A
//PCHF DD VOL=SER=STAGE1,DCB=(BLKSIZE=80)
//OUTF DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSUT2 DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSPRINT DD SYSOUT=A
//SYSLMOD DD DSNNAME=SYS1.&LIB1,DISP=OLD

Figure PTFLE-4. PTFLE Cataloged Procedure.

The statements in the cataloged procedures and their meanings are:

PROC Statement

defines values for the symbolic parameters in the PTFLE cataloged procedure. The default values are designated by USE, LIB, and REG in the parameter field of this statement.

Chapter 9: IMAPTFLE 227
EXEC Statement

specifies the program to be executed, in this case IMAPTFLE. The
PARM= field contains the symbolic parameter &USE that will be
assigned the default value of 'IEWL' in the PROC statement; if IEWL
is not the linkage editor to be used, override &USE with the name of
another linkage editor.

The default value for the symbolic region size (&REG) is 68K;
this value assumes that MVT is being used with the 44K linkage
editor. If these assumptions do not apply, replace the PROC
statement with one that contains the appropriate region size.

PRINT DD Statement

defines the message data set for IMAPTFLE.

PCHF DD Statement

defines the Stage I output from the generated system to be updated.
This data set is input to IMAPTFLE. If the data set resides on an
unlabeled tape, add a DCB parameter specifying the logical record
length (80 bytes) and the blocksize.

OUTF DD Statement

defines a temporary sequential data set used by IMAPTFLE and the
linkage editor. This data set may reside on magnetic tape or a
direct access device. Do not specify the blocksize.

SYSUT1 DD Statement

defines a work data set for the linkage editor. This data set must
reside on a direct access device.

SYSUT2 DD Statement

defines a work data set for IMAPTFLE. This data set must reside on
a direct access device. Do not specify the blocksize.

SYSPRINT DD Statement

defines the message data set for the linkage editor.

SYSLMOD DD Statement

defines the output module library for the PTF being added to the
system the DSNAME keyword contains the symbolic parameter &LIB1.
The &LIB1 parameter is assigned the value LINKLIB from the PROC
statement when the procedure is invoked. Before overriding the
LINKLIB data set name, see the publication IBM System/360 Operating
System: Job Control Language Reference, GC28-6704.

Generate Function

For execution of the generate function, IMAPTFLE requires at least a 46K
region or partition. Input to IMAPTFLE must consist of the Stage I
output from SYSGEN and control statements identifying the modules for
which JCL output is to be produced.
Figure PTFLE-5 illustrates the JCL needed to execute the generate function.

```
//JOB      JOB MSGLEVEL=(1,1),REGION=46K
//STEP     EXEC PGM=IMAPTFLE
//PRINT    DD SYSOUT=A
//OUTF     DD UNIT=2400,LABEL=(,NL),
//         DISP=(,KEEP),VOL=SER=OUTPUT
//PCHF     DD UNIT=2400,LABEL=(,NL),
//         DISP=OLD,VOL=SER=SYSGEN,DCB=(BLKSIZE=80)
//MODF     DD
/*
 control statements */
```

Figure PTFLE-5. Sample JCL Needed to Execute the Generate Function of IMAPTFLE

**JOB Statement**

initiates the job, and specifies a region size of 46K.

**EXEC Statement**

invokes IMAPTFLE. Do not specify any other parameters on this statement.

**PRINT DD Statement**

defines the IMAPTFLE message data set.

**OUTF DD Statement**

defines a sequential data set to which IMAPTFLE will direct its output. This data set may reside on a direct access device or a magnetic tape, or it may be directed to a SYSOUT data set. Do not specify a block size.

**PCHF DD Statement**

defines the Stage I output from SYSGEN to be used as input to IMAPTFLE. If an unlabeled tape is used, the DCB parameter specifying logical record length (80 bytes) and blocksize must be specified.

**MODF DD Statement**

defines the input stream that contains the IMAPTFLE control statements.
Control Statements

Two types of control statement are valid in IMAPTFLE: the IMAPTFLE control statement and the linkage editor IDENTIFY statement. When using the application function, each IMAPTFLE control statement must be followed by the PTF object module named in the control statement, which in turn must be followed by the corresponding IDENTIFY control statement. When using the generate function, the IDENTIFY control statement is optional; if used it must follow the corresponding IMAPTFLE control statement.

The following sections describe the IMAPTFLE control statement and the IDENTIFY control statement.

IMAPTFLE Control Statement

The IMAPTFLE control statement has the following general format:

module name  SSI number  comments

module name

identifies the name of the module for which JCL is to be created. The length of this name can vary, but it must not exceed eight characters. If an input module can be specified by either of two names (component library name or system library name), the component library name must be used. Statements containing duplicate module names will be ignored by IMAPTFLE. JCL will be produced for the module the first time the name is encountered.

SSI number

reflects the bit settings that are to be placed in the library directory entry for a load module after the PTF has been applied. The SSI information consists of indicators that reflect the status of the load module. The SSI must be updated to show that a module has been modified. The number must begin in column 10 and be exactly eight characters long. To determine the exact bit settings of the SSI before the PTF is applied, the utility program IEHLIST may be used to obtain the current SSI information for all the members of a library.

comments

any user data.

The coding specifications for this statement are:

- Each control statement must contain only one module name and its 8-character System Status Index (SSI) number. (As mentioned, when a user applies a PTF to a module, he is responsible for making sure that the SSI is updated to reflect these changes. For information on the SSI see the discussion "Updating System Status Information" in the IMASPZAP chapter of this publication, and the publication IBM System/360 Operating System: Maintenance Program, GC27-6918.)
The module name must begin in column 1 of the control card. If the module name is less than eight characters, leave blanks between the end of the module name and column 9.

The SSI number must begin in column 10 of the control card.

Comments are permitted through and including columns 19 and 80 of the control card.

Columns 9 and 18 may contain delimiting blanks or commas.

When using the application function, each control statement must be followed by the PTF object module named in the control statement.

Directory entries for existing alias names of modules that were copied by the IECOPY utility during system generation will be updated properly only if such alias names are provided in control statements that follow the control statements for associated module. These additional control statements need not contain SSI information. (Note: The alias names in additional control statements must be only those that appear in the same copy step as the true name of the module in the Stage I output from system generation.)

IMAPTFLE control statements are included in the input stream following the MODF DD statement, as previously described. A /* record denotes the end of input for the execution of IMAPTFLE.

Multiple control statements can be used in any execution of IMAPTFLE, but the total number of control statements must not exceed 150. After the limit has been reached, error message IMA001I will be issued.

The IMAPTFLE control statements may be entered in any order. Any module named in a control statement must exist on the Stage I output tape. Any module names that cannot be found on this tape will be listed by an error message. Duplicate module names detected will also be flagged by the message.

IDENTIFY Control Statement

An IDENTIFY statement for use by the Linkage Editor may also be included in the input defined by the MODF DD statement.

The IDENTIFY statement is not a control statement for IMAPTFLE, but for the linkage editor. IMAPTFLE will copy it (exactly as it appears in the MODF input stream) into the SYSLIN input stream that it creates for the linkage editor.

The IDENTIFY statement is required for the application function and optional for the generate function. For the application function each PTF object module must be followed immediately by an IDENTIFY statement; if the IDENTIFY statement is absent, IMAPTFLE will terminate processing and issue message IMA010I. For the generate function the IDENTIFY statement must follow the IMAPTFLE module name control statement that it is associated with. Only 150 IDENTIFY statements, including continuation statements, are permitted in a job step. If this limit is exceeded, IMAPTFLE will terminate processing with a return code of 16 and issue message IMA011I.
The format of the statement must be identical to that of the Linkage Editor IDENTIFY control statement, as follows:

IDENTIFY (csectname('data')...,csectname('data'))
csectname('data')

csectname

is the symbolic name of the control section that is to be identified. If the CSECT name is changed at system generation by a CHANGE statement, the resulting name should be used.

data

is the identifying information (maximum of 40 characters) that is used to identify the CSECT. This must be enclosed in quotes.

Column one of the statement must be blank. The outer parentheses may be deleted if only one control section is identified in the operand field.
IMAPTFLE produces two different types of output, as described below.

Application Function

The final result of running the IMAPTFLE application function is the updated load module. Because the application function is a self-contained operation, it produces no physical printed output.

Generate Function

The final result of running the IMAPTFLE generate function is a data set that consists of the job control language statements, linkage editor control statements, and the IEBCOPY control statements needed to add the PTFs to the generated operating system in a later run. Three types of JCL statements are produced:

- Linkage Editor (IEWL) JCL: This type of JCL is produced if the load module requested for processing was originally link edited into the system during system generation.
- IEBCOPY JCL: This type of JCL is produced if the member was originally copied into the system.
- IEHIOSUP JCL: This type of JCL is produced in addition to LINK EDIT and/or IEBCOPY JCL. The IEHIOSUP statements are used to execute the IEHIOSUP utility. This program updates any TTR entries in the transfer control tables of the supervisor call library (SVC library) that may require a change as a result of applying a PTF.

Figures PTFLE-6, 7, 8, and 9 show sample output from the generate function of IMAPTFLE. All of these samples were derived by using the IMAPTFLE JCL and control statements illustrated in Figure PTFLE-5. For a more detailed explanation of the JCL statements and their parameters, refer to the publication IBM System/360 Operating System: Job Control User's Guide, GC28-6703.

Note: The generate function IMAPTFLE will produce a JOB statement to precede any other JCL produced.
Figure PTFLE-6. Sample Linkage Editor (IEWL) Output from IMAPTFLE Generate Function (Sample #1)

Figure PTFLE-7. Sample Linkage Editor (IEWL) Output from IMAPTFLE Generate Function (Sample #2)

234 Service Aids (Release 21)
Figure PTFLE-8. Sample IEBCOPY Output from IMAPTFLE Generate Function

Figure PTFLE-9. Sample IEHIOSUP Output from IMAPTFLE Generate Function
Examples

Example 1: Generate Function

This example shows the JCL and control statements needed to execute the generate function of IMAPTFLE. In this case, the input data set from sysgen resides on a magnetic tape.

```
//JOB
//STEP
//PRINT
//OUTF
//PCHF
//MODF

IDENTIFY IEX51000('PTF20191')
IGE0000A 03144004
IGE0000D 02155123
IGE0000G 05194025
/*

In this example:

JOB Statement
  initiates the job.

EXEC Statement
  invokes IMAPTFLE.

PRINT DD Statement
  defines the message data set.

OUTF DD Statement
  defines the output data set, in this case residing on a direct access volume.

PCHF DD Statement
  defines the input data set containing the Stage I SYSGEN output.

MODF DD Statement
  defines the input stream that contains the IMAPTFLE control statements.
```
Example 2: Application Function

This example illustrates the JCL needed to execute the Application function of IMAPTFLE using the cataloged procedure PTFLE.

```
//PTFPROC JOB MSGLEVEL=(1,1)
//STEP EXEC PTFLE
//PTF.MODF DD *
IEFS082 01117251
  Insert PTF Object Deck
  Insert Linkage Editor IDENTIFY Statement
IEFS085 01117251
  Insert PTF Object Deck
  Insert Linkage Editor IDENTIFY Statement

/*

JOB Statement

  initiates the job.

EXEC Statement

  invokes the PTFLE cataloged procedure, which executes the application function of IMAPTFLE. When PTFLE is invoked, these statements merge with the JCL statements in the cataloged procedure.

PTF.MODF DD Statement

  defines the input stream, which contains the IMAPTFLE control statements.

IMAPTFLE Control Statements

  identify the module to be updated with the PTF, and supplies the SSI information to be placed in the library directory entry for the module once the PTF has been successfully applied.

IDENTIFY Control Statements

  identify the CSECT within the module identified by the IMAPTFLE control statement that is to be updated with a PTF, and supplies information needed to identify that CSECT once the PTF application is successful.
```
Operational Considerations

Before attempting to use IMAPTFLE, the following considerations should be examined.

General Considerations

- IMAPTFLE will not accept more than 150 module names as input. If the number of names exceeds this limit, the job must be divided into more than one job of no more than 150 module names each. If control statements are provided for alias names of modules that were copied during system generation, these additional names must be counted toward the total of 150 when the generate function is being invoked.

- The Stage I output must be from the generated system of the operating system being updated with the PTFs.

- If Stage I output is an unlabeled tape, the DCB parameter containing the logical record length and blocksize must be added to the PCHF DD statement.

- The Stage I output must not contain control characters (i.e., printer or punch).

- If an input module name can be specified by either of two names (component library name or system library name), the component library name must be used. For example, IEAATM02 is a component library name; its system name is IGC0201C. If JCL were required for this module, IEAATM02 would have to be specified as the input module name.

- It is the user's responsibility to ensure that the SSI is correctly updated when the module is applied to the system. The user, therefore, should make sure that the correct SSI information is placed on each control card. (The correct SSI data appears on the cover letter for the PTF.) Absence of the SSI on the control card will cause the SSI in the module's directory entry to be set to zeros.

- If an input load module was created from multiple load modules in the distribution library, the user should make sure that a linkage editor ENTRY statement exists for that module in the Stage I output from system generation. If no such statement is present, IMAPTFLE should not be used, since it may cause the module to be updated with an incorrect entry point.

- IMAPTFLE should not be used to apply a PTF to a module if the module name in the distribution library is different from the CSECT name in the module, and if the module's overlay structure was defined during system generation by INCLUDE statements rather than by INSERT statements. An example of such a module is the FORTRAN H compiler.
• IMAPTFLE should not be used to apply a PTF to a module that is a member of a library copied totally from the distribution library at system generation. Libraries containing modules to be processed by IMAPTFLE should have been copied selectively by the IEBCOPY utility during system generation (that is, the SELECT statement must have been used.)

Generate Function Considerations

• IMAPTFLE will not produce JCL for either an INASPZAP PTF (discussed in Section III of this publication) or a PTF that requires some degree of system generation for its application.

• IMAPTFLE requires Stage I output from a system generation of Release 19 or later. Output from earlier system generations cause the IMAPTFLE program to be terminated with an error message.

• The system library being updated by a PTF must not be used as a driver to run the JCL job stream created by IMAPTFLE. It is recommended that the STARTER SYSTEM be used instead.

• The user should verify that both the component libraries and the four utility data sets are cataloged on the driver system before the PTF is applied. (For more complete information on utility data sets, refer to the publication IBM System/360 Operating System: Utilities, GC28-6670. System data sets are cataloged, and successful application of the PTF therefore depends on their being cataloged as described.

• IMAPTFLE does not produce JCL to apply PTFs to the Distribution Libraries (DLIBs). The JCL produced by IMAPTFLE is designed to be used in updating the system by using the DLIBs. Therefore, before running the JCL produced by IMAPTFLE, the user must apply the PTFs to the DLIBs to ensure a successful update of the system when the JCL stream is run.

• The IMAPTFLE generate function will not accept more than 150 IDENTIFY cards, including continuation cards. If the number of cards exceeds this limit, the job should be divided into more than one step of no more than 150 IDENTIFY cards and continuation cards each.

Application Function Considerations

• PTFs containing multiple CSECTs can only be applied to load modules residing on the same system library.

• TTR entries in the transfer control tables of the supervisor call library (SVCLIB) are updated for PTFs applied to SYS1.SVCLIB. It is not necessary to run the IEHIOSUP utility.
Chapter 10: IMDSADMP

Operates as a stand-alone program to produce a high-speed or low-speed dump of main storage.
Introduction

When a system goes into a disabled wait state or an unending loop, a stand-alone dump program is needed to dump the contents of main storage so that the condition can be analyzed. Optimally, this dump program should be high-speed so that the system is inoperative for as short a period of time as possible. IBM provides IMDSADMP for this purpose. IMDSADMP is a macro instruction that allows a user to generate a stand-alone dump program specifically tailored to his installation's needs.

IMDSADMP can generate two types of dump program: a high-speed version that can quickly write the contents of main storage to a tape volume in large blocks, and a low-speed version in which the contents of main storage are written to either a printer or a tape volume in unblocked, printable format.

The high-speed version of the dump program may reside on either a tape or direct access volume; the low-speed version may reside only on a direct access volume. See the IMDPRDMP service aid for instructions on processing the high-speed output of IMDSADMP.

Creation and usage of the dump program is simple. The user employs the IMDSADMP macro instruction to define the type of dump program he wants (see the topic "Specifying the Dump"). The dump creation process includes a specification step and an initialization step. In the specification step, the macro instruction is assembled with the IBM-provided IMDSADMP macro definition. This specification step produces:

- IPL text necessary to make the dump program loadable for execution.
- Code that allows the IPL text and the dump program module to be stored on a selected tape or direct access volume.
- The dump program itself.

In the initialization step, the IPL text and the dump program module are placed on the specified device. To execute the dump program, the user loads it into main storage from the device by means of standard IPL procedure. The main storage dump information is written to either a tape or printer device based upon user-specified operands of the IMDSADMP macro instruction. During execution of the direct access resident version of the dump program, the operator can override the device address which was specified as a result of the expansion of the macro instruction.

The two steps required to create an executable dump program and a discussion of dump program execution follow the detailed descriptions of the high and low-speed versions.

Multiprocessing: In multiprocessing systems, IMDSADMP can dump the contents of the registers in both CPUs when the direct control feature is operational, and can dump all of addressable main storage. This is accomplished by an optional parameter of the IMDSADMP macro instruction.
Size of SADMP

The size of the assembled IMDSADMP program depends on the output option selected and whether or not IMDSADMP will be on a multiprocessing system; see Figure SADMP-1. The size of IMDSADMP is the same both in main storage and on the resident volume; see Figure SADMP-2.

<table>
<thead>
<tr>
<th>Output Option</th>
<th>Without Multiprocessing</th>
<th>With Multiprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Speed</td>
<td>1024</td>
<td>1088</td>
</tr>
<tr>
<td>Low-Speed Printer</td>
<td>1088</td>
<td>1344</td>
</tr>
<tr>
<td>Low-Speed Tape</td>
<td>1280</td>
<td>1472</td>
</tr>
</tbody>
</table>

Figure SADMP 1. Size of the IMDSADMP Program in Bytes
Figure SADMP 2. Format of Cylinder 0, Track 0 for Disk Resident IMDSADMP
The High-Speed Dump Program

This version of the IMDSADMP generated dump program (hereinafter referred to as the dump program) dumps the contents of main storage to a tape volume. Each dump record is 2052 bytes long. To further expedite the dump and conserve program storage requirements, the main storage information is written to a nonlabeled tape volume in an untranslated, hexadecimal form. Formatting, converting and printing of the information is performed by the IMDPRDM service aid.

Loading the High-Speed Dump Program

The high-speed dump program may reside on either a tape or direct access volume. In either case, the user loads the program from the device into main storage by means of the IPL procedure. The high-speed dump program is loaded into the CPU Log Out Area or into a storage specified by the user through an operand of the IMDSADMP macro instruction. If IMDSADMP is loaded into the CPU logout area, IMDSADMP destroys the contents of the logout area. In case of hardware errors, or when requested by the system, it may be necessary to display the contents of the CPU Log Out Area before invoking the dump program. This can be done by executing the System Environment Recording, Edit and Print routine, SEREP, which is discussed in the publication IBM System/360 Operating System: Operator's Reference, GC28-6691.

Output of the High-Speed Dump Program

If the user selects the high-speed version of the dump program during the specification step, he must select a tape device as the output medium, even though the dump program itself may reside on either tape or disk. The input device type selected has an effect on output retrieval.
If the dump program resides on a tape volume, the dump information is written to the nonlabeled tape that contains the program. The information, in untranslated, hexadecimal form, follows the IPL text and the dump program module records (see Figure SADMP-3, format 1). Each dump information record is 2052 bytes long (see Figure SADMP-3, formats 2 and 3).

Figure SADMP 3. Output Tape Formats for the High Speed Version of the Dump Program

If the dump program resides on a direct access device, the 2052-byte dump information records are written to the nonlabeled output tape volume (see Figure SADMP-3, formats 2 and 3). The IPL text and dump program records and work record are contained on cylinder 0, track 0 of the volume on which the dump program resides (see Figure SADMP-2). The work record is used to temporarily record the main storage information from the area into which the dump program is to be loaded.
The Low-Speed Dump Program

The low-speed version of the dump program writes the contents of main storage to either a printer or a tape device. If output is to tape, the information may be subsequently printed by a program such as the IEBGENER utility program, as discussed in the publication IBM System/360 Operating System: Utilities, GC28-6586, or by IMDPRDMP.

Loading the Low-Speed Dump Program

The low-speed dump program must reside on disk. To execute the program, the user performs the IPL procedure to load the dump program from its resident device. The IPL statements and dump program reside on cylinder 0, track 0 (see Figure SADMP-2).

During the specification step, the user may either select an address at which to begin loading, or use the default value. If the user selects his own starting address, the value he specifies must be at least 128 decimal or 80 hexadecimal.

Output of the Low-Speed Dump Program

The low-speed version of the dump program writes dump information to either a tape volume or a printer. The format of the main storage information is the same, regardless of the output device type to which it is being written. Each dump record contains 120 characters of formatted dump information. An output sample is shown in Figure SADMP-4. The contents of the general purpose registers are printed first, followed by the remainder of main storage. (Note that for low-speed dumps of a Model 65 Multiprocessing System, IMDSADMP shows both sets of general purpose registers; see Figure SADMP-5.) A storage location field containing the address of the first byte is printed to the left of each line. A character translation field, showing the EBCDIC translation of the hexadecimal contents, is displayed to the right of each line. Only alphabetic or numeric representations of hexadecimal information are given in the character translation field; all other bytes are represented by a period. If a line duplicates the contents of the previous line, it is not printed; instead the duplicate line is left blank.

If the output of the dump program is directed to a tape volume, each dump information record is preceded by a one-byte ASA character that is used by the subsequent printing program to control printer spacing. This results in a total record length of 121 bytes. This tape volume may be printed by using the IEBGENER utility program or IMDPRDMP.
Figure SADMP-4. IMDSADMP Low-Speed Dump Output Sample
Figure SADMP 5. IMDSADMP Low-Speed Dump Output Sample of a Model 65 Multiprocessing System
Specifying the Dump

The particular version of the IMDSADMP dump program to be generated is specified by the operands entered in the IMDSADMP macro instruction. Depending upon the operands coded, the program will be generated as shown in Figure SADMP-6. The IMDSADMP macro instruction statement is coded as shown in Figure SADMP-7.

Figure SADMP-6. IMDSADMP Parameter-Dependent Program Generation
**Figure SADMP-7. The IMDSADMP Macro Instruction Statement**

<table>
<thead>
<tr>
<th>symbol</th>
<th>IMDSADMP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[PL=TAPE] [CPU=360</td>
</tr>
<tr>
<td></td>
<td>[TYPE=HI, OUTPUT=TAPE] [PROTECT= YES] NO]</td>
</tr>
<tr>
<td></td>
<td>[START=X'380' or X'100'] [CPU=360</td>
</tr>
<tr>
<td></td>
<td>[CONSOLE=016</td>
</tr>
</tbody>
</table>

*Note: The default value of the START= operand is dependent upon the value specified for the CPU= operand.

X'380' is the default if CPU=360 is specified or if CPU= is omitted.
X'100' is the default if CPU=370 is specified.

**symbol**

any symbol may be associated with the IMDSADMP macro instruction. However, this symbol should not be referenced by any other assembler input statement, such as the END statement.

**IMDSADMP**

The name of the macro instruction is coded as shown.

**IPL=**

describes the device upon which the dump program resides. As such, it dictates the operation of the initialization step of the dump creation procedure. The allowable options for the IPL= operand and their meanings are:

**TAPE**

specifies that the dump module is stored on a tape device. If this option is coded, all keywords except CPU= and PROTECT= are ignored. TYPE=HI is assumed. When the dump is executed, output is written to the same tape device on which the dump program resides, immediately following the IPL and dump program records.

**c uu**

specifies a direct access device address where "c" indicates the channel address, and "uu" indicates the device address. The direct access volume that is to contain the dump program must be mounted on this device during the initialization phase. After initialization, the volume may be moved to any other direct access device.

If the IPL= operand is not specified, a default value of direct access device 191 is assumed.
TYPE=

specifies the version of the dump program to be generated for direct access residence. The allowable options for this operand and their meanings are:

HI

specifies the high-speed version of the dump program that will write unformatted core-image output to a tape volume in 2052-byte blocks. Note: The resultant output tape must be used as input to the Print Dump service aid (IMDPREDMP) to format and print the dump information.) If this option is coded and the dump program resides on a direct access device, OUTPUT=Tc uu is required. (See the options for the OUTPUT=keyword operand.)

LO

specifies the low-speed version of the dump program that will produce formatted EBCDIC output to either a tape device or the printer. If no options are entered for the TYPE= operand, TYPE=LO is assumed.

OUTPUT=

specifies the output device type. It also specifies the default output device address to which the dump is to be written if the operator chooses to use the default address rather than specify an address through a console reply in response to a message. The allowable options for this operand and their meanings are:

Tc uu

specifies the channel and unit address of a tape output device where "T" indicates tape, "c" indicates the channel address, and "uu" indicates the unit address. This is the only valid option for this operand when TYPE=HI is specified. If TYPE=HI is specified and this option is not specified or the entire OUTPUT= operand is omitted, TYPE=HI will be changed to TYPE=LO and the default value of P00E will be used as the OUTPUT= operand.

Pc uu

specifies the channel and unit address of a printer where "P" indicates printer output, "c" indicates the channel address, and "uu" indicates the unit address.

If the OUTPUT= operand is not specified and TYPE=LO is coded, a default value of printer 00E (P00E) is assumed.

PROTECT=

applicable only if TYPE=HI is selected. This operand specifies whether or not the storage protection feature is available on the CPU. The allowable options for this operand and their meanings are:

YES

This value must not be coded if the storage protection feature is not available on the CPU on which the dump program is intended to be executed, as the dump program will not work. If it is coded or assumed, it specifies that the feature is
implemented. The storage protection key field in the output of the high-speed version of the dump program will contain the storage protection key associated with the block of storage being dumped (see Figure SADMP-3, format 3).

NO

If the storage protection feature is not available on the CPU, or if it is not to be used, the NO value must be coded for the PROTECT= operand. If NO is coded, the storage protection key field in the output of the high-speed version of the dump program will contain zero (see Figure SADMP-3, format 3).

If this operand is not coded, PROTECT=YES is assumed.

START=

specifies the storage location into which the CCW's for loading the direct access resident dump program will be read. 43 bytes of storage are required for the load CCW's and, with the 24 bytes of storage starting at location 0 that are required for the IPL procedure, represent the only storage destroyed by execution of the dump program.

The START parameter is valid for both high and low speed options of the direct access resident dump program. Allowable values for this operand and their meanings are:

address

specifies the starting address of the CCW loading area expressed as a decimal number. The storage address must be greater than or equal to 128 and be aligned on a doubleword boundary. If the value specified is less than 128, it is ignored and 128 is used; if the value is not a multiple of eight, the next higher multiple of eight is used. The maximum allowable address must be at least 48 bytes less than the maximum main storage address of the CPU on which the dump program is to be executed.

X'address'

specifies the starting address of the CCW loading area expressed as a hexadecimal number. The address specified in this operand must be X'80' or greater, and be aligned on a doubleword boundary. If the value specified is less than X'80', it is ignored and X'80' is used; if the value is not a multiple of eight, the next higher multiple of eight is used. The maximum allowable address must be at least X'30' bytes less than the maximum main storage address of the CPU on which the dump program is to be executed.

The default value for the START parameter is dependent on the values of the CPU parameter. If CPU=360 is specified, or if the CPU parameter is omitted, the default value used for the START parameter will be X'80'. If CPU=370 is specified, a default of X'100' will be used for the START parameter. Adjusting the START value in this way is done to ensure that the storage overlayed by the dump program will be contained in the log-out area of the CPU on which the dump program is to be executed.
CPU=
defines the IBM computer system that IMDSADMP will dump. There are
two possible subparameters:

1. The system subparameter -- 360 or 370 for the IBM System/360
   and IBM System/370 respectively.

2. The multiprocessing subparameter -- NOMP for
   non-multiprocessing systems and MP for multiprocessing systems.

Implicit in the system subparameter is the location of the log-out
area (sometimes called the diagnostic scan-out area). For the IBM
System/360, the log-out area is located at X'80'; for System/370, it
it located at X'100'.

When IMDSADMP is loaded from magnetic tape (IPL=TAPE), the IPL
procedure overlays the first 24 bytes of main storage and the entire
256 bytes of log-out area.

When IMDSADMP is loaded from a direct access device (IPL=cuu), the
log-out area is used as the default value for the START parameter
(refer to START).

One version of IMDSADMP is used for a non-multiprocessing system,
and another version is used for a multiprocessing system; CPU=(,NOMP)
and CPU=(,MP) specify the different systems. When
applied to a multiprocessing system, IMDSADMP must be resident on a
direct access device; in that case, therefore, define IPL=cuu. At
the present time, the only multiprocessing system that IMDSADMP can
be used with is the IBM System/360 Model 65 Multiprocessing system;
for this system define CPU=(360,MP).

CONSOLE=
specifies the address and type of the console through which commands
will be entered. Valid values and their meanings are:

{009} {cuu}

The console address. If you omit the CONSOLE= parameter,
IMDSADMP assumes a default address of 009.

{1052} {3066} {3210} {3215} {5450}

The console device type. If you omit the CONSOLE= parameter,
assumes 1052 as the default device type. (Model 65
Multiprocessing only).

{01F} {cuu}

The address of the second console in a multiprocessing system.
This value is not valid for non-multiprocessing systems. If
this value is omitted, IMDSADMP assumes a default address of
01F for the second console.
Retrieving and Creating the Dump Program

The dump program is created in two steps: a specification step and an initialization step. The specification step involves the creation of a dump initialization deck that will be used as input to the initialization step. These two steps are discussed below:

The Specification Step

Before commencing operation on the specification step, the user must have made two decisions. First, he must have decided which version and options of the dump program he wishes to be in effect, as detailed in the previous discussion. Second, he must also have decided whether he wants the macro definition to be in a library (and, if so, which one) or in card image form.

Before the IMDSADMP macro definition can be assembled into a stand-alone program, the macro definition statements must be available on a media from which they can be assembled. Figure SADMP-8 shows five media from which IMDSADMP can be assembled.
Figure SADMP-8. Availability of IMDSADMP Macro Definition Statements
If the MACLIB macro instruction was specified during system generation, the macro definition for IMDSADMP is transferred from the SYS1.MACLIB component data set in the distribution library to the SYS1.MACLIB system data set. The IMDSADMP stand-alone program can then be assembled in the same manner as any other program in macro definition form. If MACLIB was not specified, use one of the following techniques to obtain the IMDSADMP macro definition:

Distribution Library as a Private Library: The distribution library can be used as a private library for the assembling of the IMDSADMP stand-alone program, see Figure SADMP-9. This example assumes that the distribution libraries are cataloged; if not, add the UNIT and VOL=SER operands to the ASM.SYSLIB data definition statement.

```assembler
//ASMSAD    JOB   MSGLEVEL=(1,1)
//          EXEC   ASMFC
//ASMSYSLIB  DD   DSN=SYS1.MACLIB,DISP=OLD
//ASMSYSIN   DD   *
  . (include the IMDSADMP macro instruction here)
  .
  END
/*
```

Figure SADMP-9. An Example of IMDSADMP JCL Statements for Designating SYSLIB

Copying to a Private Library: The IMDSADMP member of the SYS1.MACLIB component of the distribution library can be copied to a user-defined library. The IMDSADMP stand-alone program can then be assembled from the user-defined library.

Punching the Definition Statements: The IMDSADMP member of the SYS1.MACLIB component of the distribution library can be punched into cards using a utility program. With the macro definition statements on cards, the IMDSADMP stand-alone program can be assembled using these cards as input. Figure SADMP-10 shows the specification step when the macro definition statements are in punched card form.
Prior to executing the specification step, the user should ensure that he has all the required elements:

- The Assembler job control cards.
- The IMDSADMP macro definition, in either card image form or in a library as discussed above.
- The IMDSADMP macro instruction containing the operands that defines the version of the dump program that the user wishes to generate. The macro instruction may be included only once per assembly.

The specification step, then, is an assembly that creates a dump initialization deck, to be used as input to the initialization step. This dump initialization deck consists of:

- Code that allows the remainder of the dump initialization deck to be stored on the selected tape or direct access device.
- IPL text necessary to make the dump program loadable for execution.
The Initialization Step

The specification step provides input to the initialization step. The output of the initialization step is an executable dump program, stored on an I/O device from which it is loaded by the IPL procedure into main storage for execution. Initialization may be performed in one of two ways, depending upon the device type specified in the IMDSADMP macro instruction IPL= operand. The different initialization step procedures are illustrated by Figure SADMP-11.

![Diagram of Initialization Procedures]

Figure SADMP-11. IMDSADMP Initialization Procedures

Tape Initialization (High-Speed Only)

If the user has specified that the high-speed version of the dump program is to reside on tape, the specification step will have provided three types of statements to the initialization step: job control statements, IPL text and the dump program object module. The JCL statements invoke and control the operation of the IEBGENER utility program (as discussed in the publication IBM System/360 Operating System; Utilities, GC28-6586) that copies the remainder of the dump initialization deck to the specified tape volume. The IPL text allows the dump program to be loaded from the tape volume into main storage for execution. The dump program object module consists of the actual machine instructions that perform the desired dump function. The initialization process for a dump program that is loadable from a tape volume is under control of the operating system, and is performed in the same manner as for any other job. During execution of the job, the operator will receive message IEF233A from the job scheduler, asking for tape serial number "DUMP". At this time, a non-labeled scratch tape is mounted to be initialized.

Note: The output of IEBGENER describes the tape to be initialized as is shown in Figure SADMP-12. If a 7-track unit is to be used to initialize a dump tape, the UNIT= parameter must be changed to describe a 7-track tape device; that is, UNIT=2400-2. If the data conversion feature is not present on the 7-track unit, the TRTCH=C parameter should be removed.
The JCL statements that head the dump initialization deck invoke the
IEBGENER utility program, which in turn copies the remainder of the dump
initialization deck onto the selected tape volume. When a user wishes to
obtain a high-speed dump of main storage, the necessary program and
storage space are available to him on the volume he has initialized.

Direct Access Initialization

A direct access device must be used to store the low-speed dump program,
whereas the high-speed version may be stored on either direct access or
tape devices. When the user specifies a direct access device, the
specification step will have passed a loader and initialization program,
IPL text, and the dump program object module to the initialization step.
The initialization program transfers the IPL and program statements to
cylinder 0, track 0, of the volume on the specified direct access device.

The volume on which the assembled IMDSADMP service aid resides must
have a standard 80-character label located at cylinder 0, track 0. There
may be up to seven 80-character labels on that track. The number of
labels depends upon the IMDSADMP options selected and the track capacity
of the device; see Figure SADMP-13.

<table>
<thead>
<tr>
<th>Dump Residence Device</th>
<th>IMDSADMP Options</th>
<th>Maximum Number of User Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>2301</td>
<td>All Options</td>
<td>7</td>
</tr>
<tr>
<td>2303</td>
<td>Multiprocessing Low-Speed Tape Output</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>All Other Options</td>
<td>7</td>
</tr>
<tr>
<td>2305</td>
<td>All Options</td>
<td>7</td>
</tr>
<tr>
<td>2311</td>
<td>High-Speed Tape Output</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Low-Speed Tape Output</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Low-Speed Printer Output</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Multiprocessing High-Speed Tape Output</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Multiprocessing Low-Speed Tape Output</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Multiprocessing Low-Speed Printer Output</td>
<td>2</td>
</tr>
<tr>
<td>2314</td>
<td>All Options</td>
<td>7</td>
</tr>
<tr>
<td>2319</td>
<td>All Options</td>
<td>7</td>
</tr>
<tr>
<td>3330</td>
<td>All Options</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure SADMP-13. Maximum Number of User Labels Depending on Device and Options Selected
The IPL text is then used to load the dump program from the direct access device into main storage for execution. The dump program object module consists of actual machine instructions that perform the specified dump function. Initialization of a dump program that is loadable from a direct access volume is a stand-alone process and proceeds as follows:

- Ready the desired direct access volume on the device specified by the IPL= operand of the IMDSADMP macro instruction.
- Place the dump initialization deck in the card reader.
- Set the Load Unit dials to the address of the card reader.
- Press the Load key on the operator's console.

When the initialization program has successfully transferred the IPL text and the dump program module to the direct access volume, a completion code of X'01' appears in the instruction address register (IAR). After the initialization step has been completed, the direct access volume containing the dump program may be moved to any device. The direct access volume may be repeatedly dumped and restored without reinitialization of the dump program. If the user keeps the dump program direct access volume permanently mounted, the dump program is immediately available when the user wishes to obtain a stand-alone dump of main storage.

If the direct access initialization process is not successful, an error code is set in the IAR. This code indicates the cause of the initialization failure:

X'04'

The VTOC of the volume being initialized begins on cylinder 0, track 0; hence the record containing the dump program cannot be written on this track. Such a direct access volume cannot contain the IMDSADMP program.

X'08'

The unused space on cylinder 0, track 0 is not sufficient to hold the dump program. Only standard IPL records, the 80-character volume label, and one to seven user labels can reside on cylinder 0, track 0.

X'0F0F0F'

A permanent I/O error (the condition persisted after 16 retries) occurred on the direct access device being initialized. This condition is usually caused by cylinder 0, track 0 being defective. A direct access volume with a defective cylinder 0, track 0 is not suitable for use as an IPL volume. The volume should be analyzed, using either utility program IEHDASDR or IBCDASDI, and the initialization process repeated.
Executing the Dump Program

The operating procedures for the tape resident version of the dump program vary slightly from those of the direct access resident version. Console operation procedures for the execution of the tape resident version of the dump program are as follows:

- Ensure that the initialized tape volume containing the dump program has the write ring in place.
- Mount the initialized dump tape volume (discussed under Creating the Dump) on an appropriate tape device.
- Set the Load Unit dials to the address of the tape device containing the initialized dump tape volume.
- Press the Load key on the operator's console.

The contents of main storage are written to the same tape volume that contains the dump program. The dump information is written to the tape volume immediately behind the dump program records (see Figure SADMP-3). Successful completion of the dump is indicated by the appearance of X'01' in the instruction address register. At this point, the user must perform the OS/360 IPL procedure in order to restart the operating system. The tape containing the dump information must then be used as input to the Print Dump service aid to format and print the information. After the information contained on the tape volume has been printed, the same initialized volume may be used to perform another dump. The IPL text and the program module heading the initialized tape volume are not destroyed in the dump process.

A direct access resident dump program is executed as follows:

- Mount the initialized direct access volume (discussed under "Creating the Dump") on any suitable direct access device and bring the device to ready status. (Usually, the dump program would be stored on a permanently mounted direct access volume, so that it would always be available.)
- Set the Load Unit dials to the address of the direct access device containing the initialized volume.
- Press the Load key on the operator's console.
- Message IMD001A will be issued to the console at the address specified by the CONSOLE= parameter. This message asks for the address of the device to which the dump output is to be written. When message IMD001A is issued, the operator should ready the desired output device and enter the address of that device or signal end-of-block if the default output device is to be used. If the operator responds with end-of-block, or if an error occurs during an I/O console operation, the output device specified by the OUTPUT= parameter when the IMDSADMP macro was assembled will be used. If the OUTPUT= parameter had not been specified, the default value of P0OE will be used.

The device address specified in response to message IMD001A must be that of a device whose type agrees with the device type specified by message IMD001A. If Tuu was specified for the OUTPUT= parameter as the device type, message IMD001A TAPE= will be issued,
indicating that a tape device is desired. If Puu was specified for
the OUTPUT= parameter as the device type, or if the OUTPUT=
parameter was allowed to default, the message IMD001A PTR= will be
issued, indicating that a printer device is desired. When output is
to a tape device, the volume mounted on the specified device is
checked for standard labels before the dump is written. Standard
labels are checked by comparing the first four bytes of the first
record for VOL1. (The VOL1 identifier is checked against both EBCDIC
and ASCII encodings.) If such a label is found, or if an I/O error
occurs during the label checking procedure, the volume is unloaded
and the message IMD002I LBL ERR is issued. Message IMD001A is
reissued and the operator must ready and specify the output device
again. The operator can mount a non-labeled scratch tape and enter
the device address again, or he can enter the address of a different
device on which a non-labeled tape has been previously mounted.

The contents of main storage are written to the specified output
device. Successful completion of the dump is indicated by the appearance
of message IMD005I. At this point, the user must perform the IPL
procedure in order to restart the operating system. If the dump
information is written to tape, it must be printed by a subsequent
program. In the case of the low-speed version of the dump program, the
tape output may be printed by the IEBGENER utility program, as discussed
in IBM System/360 Operating System: Utilities, GC28-6586, or by
IMDPRDMP. Tape output produced by the high-speed version of the dump
program must be formatted and printed by IMDPRDMP.

Note 1: If the printer runs out of paper during the execution of the
dump program, insert more paper and start the printer. IMDSADMP will
continue normally.

Note 2: Neither version of the dump program issues a mode set command
to the tape output device. If output is to a 7-track tape, additional
JCL parameters are required on the input DD statement for programs which
read the dump tape. When the dump has been written to a 7-track tape,
the following must be coded as subparameters of the DCB parameter:
DEN=2,TRTCH=C. If the data conversion feature is not included on the
7-track device, the TRTCH= keyword must be omitted.
Operational Considerations

Following are points to which careful consideration should be given when using the stand-alone dump service aid (IMDSADMP):

- If IMDSADMP output is to tape, the tape volume mounted must be non-labeled. If the output volume has standard labels, or if an I/O error occurs during this checking procedure, the tape volume is unloaded and message IMD002I is issued. A non-labeled scratch tape (e.g., one with a tapemark as the first record) must then be mounted and IMDSADMP reloaded.

- Non-labeled scratch tapes on 7-track devices may not be accepted by IMDSADMP. The volume on a 7-track unit will be unloaded unless it is one of the following types:
  1. A scratch tape with a tapemark as the first record, or
  2. A non-labeled tape with data recorded in the mode: 800 BPI, odd parity, translator off. For example, a dump tape previously produced by IMDSADMP.

- If the user specifies the disk resident version of IMDSADMP he must consider the direct access space requirements. The IPL text, dump program records, and work record are contained on cylinder 0, track 0 of the volume on which the dump program resides (see Figure SADMP-2). This direct access volume must have the standard 80-character volume label, and may have one to seven 80-character user labels, on cylinder 0, track 0. The number of user labels possible is dependent upon the dump program output option specified by the user.

<table>
<thead>
<tr>
<th>Option Specified</th>
<th>Number of User Labels Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Speed</td>
<td>1 to 7</td>
</tr>
<tr>
<td>Low-Speed to Printer</td>
<td>1 to 6</td>
</tr>
<tr>
<td>Low-Speed to Tape</td>
<td>1 to 5</td>
</tr>
</tbody>
</table>

- Depending on the track capacity of the IMDSADMP resident direct access device, the user may need to limit the number of user labels written on that track; see Figure SADMP-10.1.

- When specifying the IMDSADMP macro instruction operands, PROTECT=YES must not be coded if the storage protect feature is not implemented, as the dump program will not execute.

- If the dump program resides on a direct access volume, the IPL text and dump program records are contained on cylinder 0, track 0, of that volume. The resident volume must have a standard 80-character label on cylinder 0, track 0. With the IBM System/360 Disk Operating System, the volume table of contents for that volume must begin at some location other than cylinder 0, track 0.

- If IPL=ceu or IPL=191 is specified or implied, the direct access volume that contains the dump program must be mounted on the specified direct access device during the initialization step. After initialization the volume may be moved to any other applicable device.
• If the IMDSADMP macro definition resides in either the component library or a private library, the user should not attempt to concatenate either library to SYS1.MACLIB unless the attributes and device type are identical.

• Neither version of the dump program issues a mode set command. Therefore, output to a 7-track tape may produce a volume that cannot be read by other programs. If output is to a 7-track tape, additional JCL parameters are required on the input DD statement for programs which read the dump tape. When the dump has been written to a 7-track tape, the following must be coded as subparameters of the DCB parameter: DEN=2,TRTCH=C. If the data conversion feature is not included on the 7-track device, the TRTCH= keyword must be omitted.

• If the user specifies the START= parameter for the disk resident version of IDSADMP, the address he specifies must be equal to or greater than 128 or X'80'. The address specified must also be at least 48 bytes (X'30') less than the maximum main storage address of the CPU on which the dump program is to be executed.

• The low-speed version of the dump program must reside on a direct access device. The high-speed version may reside on either a tape or direct access device.

• Initialization of a disk-resident dump program must be performed on a System/360 model 40, or higher.

• The output tape produced by the high-speed version of the dump program must be printed by IMDPRDMP.

• The output tape produced by the low-speed version of the dump program may be printed by the IEBGENER utility program or IMDPRDMP.

• Error recovery during dump execution: If output is to tape, a failing I/O operation is retried indefinitely. Before the operation is retried, the tape volume is backspaced and a record gap is erased.

• Occurrence of a Unit Check or Unit Exception condition on the printer as the result of an I/O operation will cause the WRITE Operation to be retried until the condition is cleared. If the Unit Check condition exists when the I/O operation is initiated, the program will enter a two instruction loop. When the Unit Check condition is cleared (that is, when the device is made ready), the dump operation will continue.

• Occurrence of the Unit Check condition on the first I/O operation to the console causes the dump to be written to the device specified by OUTPUT=.

• IMDSADMP supports only the following devices:

1. Printer - 1403, 3211
2. TAPE - 2400 series, 3400 series
3. DASD - 2311,2312,2313,2314,2318,
   2319,2301,2303,2305,3330
4. Card reader - 2540
5. Console - 1052,3066,3210,3215,5450
Note: IMDSADMP uses data chaining when writing a high-speed dump. Therefore, when running IMDSADMP on a System/360 Model 30, do not direct output to a tape device with a high data transfer rate.

- Location X'10' is used by the system to locate the CVT. The IPL procedure used to load the dump program when using IMDSADMP destroys this location. Therefore, if there is reason to believe that this location has been overlaid during processing by the system, its value must be manually displayed and recorded prior to taking the stand-alone dump.

When using IMDSADMP on a multiprocessing (MP) system, the following additional points should be considered:

- IMDSADMP should be permanently resident on a shared volume to permit IMDSADMP to be loaded by either CPU.
- IMDSADMP must be loaded by the CPU whose prefix switch is set to disable. If the CPU that is not prefixing has had a hardware malfunction, set the prefix switch on the other CPU to disable and load IMDSADMP from that CPU.
- For IMDSADMP to dump the registers of both CPUs, both CPUs must be in multiprocessing mode when IMDSADMP is executed.
Error Conditions

This section describes various error conditions which can occur during execution of IMDSADMP. During such execution, it is imperative that the data in core to be dumped remain unaltered. Error recovery is consequently limited to providing attempted retries of I/O operations and presenting an indication of the error. If an error occurs, the system operator should note the error indication (IAR content, wait state, loop, load light on, or incomplete output), and execute the program again. If the problem recurs, call IBM for programming support.

Error Handling

All operations of IMDSADMP are executed with machine check disable. A machine check during IMDSADMP execution will remain pending so that the dumping function can continue to completion. When dump execution is complete, a wait PSW is loaded by IMDSADMP to enable machine checks. Any pending machine check interrupts will be presented at this time.

All I/O operations in IMDSADMP are done with the system mask disabled for I/O interrupts. I/O status is received by IMDSADMP through use of the TIO instruction.

Errors During Initialization of Direct Access Resident Version

1. Loading of Initialization Program
   a. If, during IPL, an I/O error occurs on the card reader, the CPU will enter a wait state with the console load light on.
   b. Loading of the initialization program is done by IMDSADMP, executing within the CPU. Each I/O operation to the card reader is checked for unit check, unit exception or any condition indicated in the second status byte in the CSW. If any of these conditions is present, a one-instruction loop is entered.

2. Once the initialization program is loaded, I/O errors can occur only on the direct access device being initialized. I/O errors on this device are indicated by light settings in the IAR. Possible indications and remedial actions are described in the initialization discussion in this section.

3. A program interrupt during initialization will result in the program entering a WAIT condition with X'03' set in the IAR.

Errors During Dump Execution

If an I/O error should occur on the load device during loading of the dump program from tape or direct access, the CPU will enter a wait state with console load light on.
The program check new PSW is modified after the storage location containing that PSW is written. A program interrupt before this PSW has been initialized cannot be indicated, since to do so would overlay the data to be dumped. Therefore, the result of a program check at this time is unpredictable. If a program check occurs after this PSW has been modified, the dump will be terminated normally with message IMD005I, but output will be incomplete.

1. Direct Access Resident Dump Program

IMDSADMP tests console availability by issuing a TIO instruction to the console device. If the resulting condition code is zero, the console is assumed to be operational. All other condition codes indicate to IMDSADMP that the console is not operational; therefore, the dump is written to the output device specified in the OUTPUT parameter of the macro instruction.

Before each I/O operation, a TIO instruction is issued to the device to be used. If the device is not available the TIO instruction is repeated until the device is ready. When an I/O operation completes, the CSW is checked for the following conditions:

- Channel program check.
- Protection violation.
- Channel data check.
- Interface control check.
- Chaining check.

If any of these conditions occurs, it is indicated by message IMD003I CHAN ERR and execution is terminated. If unit check is indicated in the CSW and the I/O operation was not being performed on the dump output device, the operation is retried until the unit check condition is cleared. Unit exception conditions on devices other than the output device are ignored.

   a. Printer Output

   Condition code zero is the only status accepted on the SIO instruction. The SIO is repeated until the condition code becomes zero (that is, when the device is made ready). If an I/O operation completes with either unit check or unit exception, then the write operation (not the spacing command) is retried until the condition is cleared. The dump then continues.

   b. Tape Output

   Condition Code 1 following the SIO instruction is interpreted as an error condition and the error recovery procedure for that device is entered. If unit exception is indicated when an operation to tape completes, message IMD004I EOR is issued and execution is terminated. A unit check condition will initiate a recovery channel program which will be retried repeatedly until the unit check condition is cleared.
2. Tape Resident Dump Program

I/O operation are issued only to the tape device from which the dump program was loaded. If condition codes 2 or 3 occur as a result of the SIO instruction, the SIO instruction is repeated until the condition is cleared. A unit exception condition is ignored. If a unit check condition occurs, a recovery channel program will be initiated and repeated until the condition is cleared. A condition code of 1 occurring as a result of an instruction also causes the recovery channel program to be executed.

Macro Expansion Messages

During the expansion of the IMDSADMP macro definition, the operands of the IMDSADMP macro instruction statement are examined for validity. If an invalid operand value is detected, a diagnostic error message is issued, indicating the error and showing what corrective action was taken, or what assumption has been made. The message texts, their severity codes, and their meanings are shown in the table below:

ALTERNATE CONSOLE AVAILABLE ONLY FOR CPU=(360,MP). ALTERNATE IGNORED

Explanation: The user specified two consoles in the CONSOLE= parameter, but did not specify CPU=(,MP). A second console can be specified only for a multiprocessing system (MP). The assembly continues, and the alternate console definition is ignored.

Severity Code: 4.

CONSOLE DEVICE TYPE XXXX NOT SUPPORTED, 1052 CONSOLE ASSUMED

Explanation: IMDSADMP does not support the device specified in the CONSOLE= parameter. A 1052 console is assumed.

Severity code: 4

CPU VALUE ERROR, S/360 ASSUMED

Explanation: The value specified by the user for the CPU= parameter was not one of the valid values, 360 or 370. The default value of 360 is assumed.

Severity Code: 4.

CPU=xxx INVALID, CPU=(yyy,NOMP) IS ASSUMED

Explanation: The second subparameter of the CPU parameter is invalid. The first parameter has already been tested. The second subparameter must be either MP or NOMP. xxx is the entry made by the user; yyy is the CPU type, either 360 or 370, entered as the first parameter of CPU=. The assembly continues, and NOMP is assumed.

Severity Code: 4.

CPU=xxx INVALID, CPU=(360,MP) IS ASSUMED

Explanation: The user has specified CPU=(360,MP), which is invalid. The MP option of IMDSADMP is available only for the M65MP system, for which the CPU= parameter must specify CPU=(360,MP). IMDSADMP assembly continues with CPU=(360,MP) assumed.

Severity Code: 4.
HIGH-SPEED MEMORY DUMP REQUIRES TAPE OUTPUT, TYPE = HI IGNORED

Explanation: The user has attempted to generate the high-speed version of the dump program with the output directed to a printer. The output must be assigned to a tape device. The TYPE=HI operand has been ignored and output has been assigned to the printer. Check the specifications of the OUTPUT= operand.

Severity Code: 4.

IMDSADMP MACRO ALLOWED ONLY ONCE PER ASSEMBLY

Explanation: The use has attempted to issue the IMDSADMP macro instruction more than once within this assembly.

Severity Code: 8.

INVALID CHARACTER IN DECIMAL PARAMETER, START= xxx INVALID, X '80' USED

Explanation: The value specified for the START= operand was not decimal. The value was coded as xxx. Review the description of the START= operand. The two parts of the message in this discussion may be issued independently.

Severity Code: 4.

MP OPTION NOT AVAILABLE FOR IPL=TAPE. IPL=TAPE IS ASSUMED

Explanation: Parameter conflict. The user specified both the MP and IPL=TAPE parameters. MP requires the direct access resident option. The assembly continues, and the MP option is ignored.

Severity Code: 4.

OUTPUT = xyyy IS INVALID. OUTPUT = POOE USED

Explanation: The channel and unit address (yyy) specified for the output device is invalid. (x indicates the device type, 'P' for a printer and 'T' for a tape device.) The printer (POOE) will be used for output if possible.

Severity Code: 4.

x DENOTES INVALID OUTPUT DEVICE, A PRINTER IS ASSUMED

Explanation: The device type indicator of the OUTPUT= operand was specified as other than 'P' (for a printer) or 'T' (for a tape device); x is the character that was coded. POOE is assumed.

Severity Code: 4.
Chapter 11: IMASPZAP

Verifies and/or replaces instructions and/or data in a load module.
IMASPZAP is a service aid program that operates under control of the System/360 Operating System. This program is designed to enable authorized personnel to:

- Inspect and modify instructions and data in any load module that exists as a member of a partitioned data set.
- Inspect and modify data in a specific data record that exists in a direct access data set.
- Dump an entire data set, a specific member of a partitioned data set, or any portion of a data set residing on a direct access device.
- Update the System Status Index (SSI) in the directory entry for any load module.

Capabilities of SPZAP

The functions of IMASPZAP provide the user with many capabilities. Three of these are suggested below.

- By using the inspect and modify functions of IMASPZAP, programming errors that require only the replacement of instructions in a load module can be fixed on the spot, thus eliminating the need for immediate recompilation of the program.
- In another instance, the user may want to obtain a storage dump for the purpose of diagnosing a problem. The modify function of IMASPZAP could be used to alter an instruction in the problem program and cause the execution of the job to terminate at a precise location. A dump of storage would automatically be given at the forced termination of the program.
- Since IMASPZAP can replace data directly on a direct access device, it could also be used to reconstruct VTOCs or data records that may have been destroyed as the result of a device I/O error or a programming error.

Monitoring the Use of SPZAP

Because IMASPZAP provides the ability to modify data on a direct access storage device, misuse of this program could result in serious damage to both user and system load modules or data sets. To protect against the occurrence of such damage by IMASPZAP, two means of controlling its use are suggested below:

- One means of exercising control is provided by IBM under MFT II and MVT. The System Management Facility (SMF) provides a system interface with user exit routines for the purpose of monitoring the job stream. Essentially, this facility, when incorporated into the system, affords an internal means of checking to see whether a particular user is authorized to execute the program specified on the EXEC job control language statement. (For further information on the SMF facility, refer to the publication Data Management for System Programmers, GC28-6550.)
A second means of protecting against unauthorized use of IMASPZAP is to store IMASPZAP in a "password protected" private library. If IMASPZAP is located in such a library, any person trying to execute this program would be required to include in his JCL statements a JOBLIB DD statement defining the library, and at initiation time he would be required to give the password associated with the library. Only personnel knowing the password would then be able to execute IMASPZAP. Password protected libraries are discussed in the publication Data Management for System Programmers, GC28-6550.
IMASPZAP can be used to inspect and modify data in either a specific record of a direct access data set or a load module that is part of a partitioned data set. The specific functions performed are governed by the use of control statements.

The modification of data is implemented through the REP control statement. The REP operation allows the user to replace instructions or data at a specific location in a load module or physical record.

The inspection of data is implemented through the VERIFY statement. This operation is provided to protect the user against erroneous replacement of data and to allow him to conditionally modify data. The VERIFY function should be used to check the contents of a specific location in a load module or physical record prior to replacing it. If the contents at the specified location do not agree with the contents as specified in the VERIFY statement, subsequent REP operations will not be performed.

Note: Although it is not required that the VERIFY function be employed prior to the REP function, it is strongly recommended that this control function be utilized to avoid possible errors in the replacement of data.

Inspecting and Modifying a Load Module

To reference data in a load module, the user must supply IMASPZAP with the member name of the load module through the use of a NAME control statement. The load module must be a member of the partitioned data set identified by the SYSLIB DD statement included in the execution JCL.

If the load module being inspected or modified contains more than one control section (CSECT), the user must also supply IMASPZAP with the name of the CSECT that is to be involved in the operations of the program. If no CSECT name is given in the NAME statement, IMASPZAP will assume that the control section to be referenced is the first one encountered in searching the load module.

IMASPZAP will place descriptive maintenance data in the IMASPZAP CSECT Identification Record (IDR) of the load module whenever a REP operation associated with a NAME statement is performed on a control section contained in that module. This function will be performed automatically after all REP statements associated with the NAME statement have been processed; any optional user data that has to be placed in the IDR will come from the IDRDATA statement (See "IMASPZAP Control Statements" for an explanation of the IDRDATA statement).

Accessing a Load Module

Once the CSECT has been found, IMASPZAP must locate the data that is to be verified and replaced. This is accomplished through the use of offset parameters in the VERIFY and REP statements. These parameters are specified in hexadecimal notation, and define the displacement of the data relative to the beginning of the CSECT. For example, if a hexadecimal offset of X'40' is specified in a VERIFY statement, IMASPZAP will find the location that is 64 bytes beyond the beginning of the CSECT identified by the NAME statement, and begin verifying the data from that point.
Normally, the assembly listing address associated with the instruction to be inspected or modified can be used as the offset value in the VERIFY or REP statement. However, if a CSECT has been assembled with other CSECTs so that its origin is not at assembly location zero, then the locations in the assembly listing do not reflect the correct displacements of data in the CSECT. The proper displacements must be computed by subtracting the assembly listing address delimiting the start of the CSECT from the assembly listing address of the data to be referenced.

To eliminate the need for such calculations and allow the user to use the assembly listing locations, IMASPZAP provides a means of adjusting the offset values on VERIFY and REP statements. This is achieved through the use of the BASE control statement. This statement should be included in the input to IMASPZAP immediately following the NAME statement that identifies the CSECT. The parameter in the BASE statement must be the assembly listing address (in hexadecimal) at which the CSECT begins. IMASPZAP will then subtract this value from the offset specified on any VERIFY or REP statement that follows the BASE statement, and use the difference as the displacement of the data.

The usage of the control statements mentioned in the above discussion is explained in detail in the section entitled "IMASPZAP Control Statements."

Figure SPZAP-1 exemplifies an assembly listing showing more than one control section. If a user were to reference the second CSECT (IEFCVOL2), he could include in the input to IMASPZAP a BASE statement with a location of 0398. Then, to refer to the subsequent LOAD instruction (L R2, LCTJCTAD), he could use an offset of 039A in the VERIFY or REP statements that follow in the IMASPZAP input stream.

```
Figure SPZAP-1. Sample Assembly Listing Showing Multiple Control Sections
```

282 Service Aids (Release 21)
Inspecting and Modifying a Data Record

To reference a specific data record, the user must specify the actual cylinder, track and record numbers that comprise the direct access address associated with it. The CCHHR control statement used by IMASPZAP relates this information to the program. This CCHHR address must be within the limits of the direct access data set defined in the SYSLIB DD control statement.

To provide a record of modifications to potentially sensitive data records, a REP operation associated with a CCHHR statement will cause IMASPZAP to write message IMA121I to the operator.

Accessing a Data Record

When this type of reference is used, IMASPZAP is able to read directly the physical record the user wants to inspect or modify. The offset parameters specified in subsequent VERIFY and REP statements are then used to locate the data that is to be verified or replaced within the record. These offsets must be specified in hexadecimal notation and define the displacement of data relative to the beginning of the record. Also, the user must include the length of any key data field in the calculation of his offset values. This is because IMASPZAP considers the key associated with a direct access record to be part of it.
Dumping Data

The dumping options provided by IMASPZAP constitute a very necessary facility for the user. By providing a visual picture of the load module or data record that has been changed, the dump feature allows the user to double check the modifications he has made.

There are two formats in which the data may be dumped. In the first (formatted hexadecimal dump) the data requested for the dump is printed in hexadecimal. The second format (translated dump) includes the hexadecimal data, a translation of all printable characters, and, where applicable, an indication of mnemonic operation code equivalents. (Refer to "IMASPZAP Output" for figures showing these two kinds of dumps.)

The DUMP and ABS_DUMP statements are the control statements used to specify the options described above. A user may also indicate the portion of the data he wants IMASPZAP to dump. The operation code in the DUMP and ABS_DUMP statements indicates the kind of dump wanted; the parameters identify the portion of the data to be dumped. (Use of the DUMP and ABS_DUMP statements is discussed in detail under the topic "IMASPZAP Control Statements.")
Updating System Status Information

The system status index (SSI) is a 4-byte field created (upon request) in the directory entry of a load module at linkage editor processing time. Its primary function is to retain information pertaining to the status of the load module. This index is useful for keeping track of any modifications that are performed on a load module. The IMASPZAP program will, as part of its normal function, update the system status index (when there is one) to reflect local modification when a replacement of data in a module is effected. The user can also, by means of the SETSSI control statement, insert his own 4-byte information field into the SSI, overlaying the information originally stored there. However, for purposes of maintaining an accurate record of the status of a load module, the SETSSI statement should be used with caution.

To ensure proper use of the SETSSI statement, an explanation of the location, significance, and format of the system status index is provided here. For more detailed information regarding the SSI, refer to the publication IBM System/360 Operating System: Maintenance Program, GC27-6918.

The System Status Index (if it exists) is located in the last four bytes of the user data field in the directory entry for a load module. Figure SPZAP-2 shows the position of the SSI in load module directory entries.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>TRR</th>
<th>C</th>
<th>User Data Field</th>
<th>SSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13 to 70 maximum</td>
<td>variable</td>
</tr>
</tbody>
</table>

Figure SPZAP-2. SSI Bytes in a Load Module Directory Entry

Figure SPZAP-3 gives a breakdown of the System Status Index field and the flag bits used to indicate the types of changes made to the corresponding load module program. A detailed explanation of this field and its applicability to the IMASPZAP program follows.

As shown in Figure SPZAP-3, the first byte of SSI information contains the member's change level. When a load module is initially released by IBM, its change level is set at one. Thereafter, the change level is incremented by one for each release that includes a new version of that program. If a user makes a change to the SSI for any of the IBM-released programs, he should take care not to destroy this maintenance level indicator unless he purposely means to do so. To keep the change level byte at its original value, he should find out what information is contained in the SSI before using the SETSSI function of IMASPZAP.
Figure SPZAP-3. Flag Bits in the System Status Index Field

Note: IMBLIST can be used to determine the SSI setting prior to making any modification to this status indicator.

The second byte of the SSI is termed the flag byte. Bits within the flag byte contain information reflecting the member's maintenance status. Using IMASPZAP, a user need only be concerned with two of the eight bits:

- The local fix flag contained in the third bit (bit 2) is used to indicate that the user has modified a particular member. (This is opposed to IBM PTF changes.) IMASPZAP sets this local fix flag bit on after successful completion of a modify operation to a load module.

- The program temporary fix flag (relative bit 3) is set on when an IBM-authorized program temporary fix (PTF) is applied to a system library to correct an erroneous IBM module.

All other bits in the flag byte should be retained in the SSI as they appeared before the SETSSI operation was enacted, so as not to interfere with the normal system maintenance procedures.

The third and fourth bytes of the system status index are used to store a serial number that identifies the first digit and the last three digits of a PTF number. These bytes are not altered by IMASPZAP unless the user deliberately changes them with a SETSSI statement.
Operational Considerations

Technical considerations for the use of the IMASPZAP service aid program are listed below:

- IMASPZAP utilizes system OPEN, and therefore cannot modify "read-only" or inspect "write-only" password protected data sets unless the correct password is provided at OPEN.

- Unexpired data sets such as system libraries cannot be modified unless the operator replies '00,'U' to the expiration message that occurs during OPEN.

- If IMASPZAP is used to modify an operating system module that is loaded only at IPL time, an additional IPL is required to invoke the new version of the altered module.

- The SYSLIB DD statement cannot define a concatenated data set.

- IMASPZAP supports only the following direct access devices: 2311, 2312, 2313, 2314, 2318, 2319, 2301, 2302, 2303, 2305, 2321, and 3330. One of these devices must be specified in the unit parameter of the SYSLIB DD statement.

- IMASPZAP is a non-reusable module.

- When modifying a system data set, such as SYS1.LINKLIB, DISP=OLD should be specified on the SYSLIB DD statement.
Executing IMASPZAP

Both JCL and control statements are required to execute IMASPZAP. The following sections describe the required statements.

JCL Statements

SPZAP can be executed using the following job control statements. The minimum region for execution is 13K plus the larger of 3K or the blocksize in bytes for the data set specified on the SYSLIB DD statement.

JOB Statement

marks the beginning of the job.

EXEC Statement

invokes the program IMASPZAP.

SYSPRINT DD Statement

defines a sequential output message data set, that can be written on a system printer, a magnetic tape volume, or a direct access volume. This statement is required for each execution of IMASPZAP.

SYSLIB DD Statement (required for each execution)

defines the direct access data set that will be accessed by IMASPZAP when performing the operations specified on the control statements. The DSNAME parameter and DISP=OLD or DISP=SHR must always be defined. The VOLUME and UNIT parameters are necessary only of the data set is not cataloged. When this data set is the VTOC, DSNAME=FORMAT4.DSCB must be specified. This statement cannot define a concatenated data set.

SYSABEND DD Statement

defines a sequential output data set to be used in case IMASPZAP terminates abnormally. This data set can be written to a printer, a magnetic tape volume, or a direct access volume.

SYSIN DD Statement

defines the input stream data set that contains IMASPZAP control statements.

IMASPZAP Control Statements

The IMASPZAP control statements (entered either through the user's input stream or through the system console) are used to define the processing functions to be enacted during a particular execution of IMASPZAP. The statements may be grouped into three categories depending upon the program's usage of them: those that are used to reference load modules (NAME, DUMP, DUMPT, IDRDATA, SETSSI, BASE), those that refer to specific records within a data set (CCHHR, ABSDUMP, ABSDUMPT), and those that can be used to specify processing control for either type of input mentioned in the first two categories (VERIFY, REP, CONSOLE).
IMASPZAP control statements must be coded according to the following rules:

- IMASPZAP control statements may begin in any column, but the operation name must precede the parameters.
- There must be at least one blank between the specified operation name and the first parameter.
- All parameters must also be separated by at least one blank space.
- Data field parameters may be formatted with commas for easier visual check, but imbedded blanks within data fields are not permitted.
- Data and offset parameter values must be specified as a multiple of two hexadecimal digits.
- The size of an IMASPZAP control statement is 80 bytes.
- Following the last required parameter and its blank delimiter, the rest of the control statement space can be used for comments. Exceptions to this are the NAME and DUMP control statements. If the CSECT parameter is omitted from either of these statements, the space following the load module parameter should not be used for comments.
- A record beginning with an asterisk and a blank is considered to be a comment statement.

The control statements are the following:

NAME member [csect]

used to identify a CSECT in a load module that is to be the object of subsequent VERIFY, REP, or SETSSI operations. The parameters are:

member

the member name of the load module that contains the control section in which the data to be inspected and/or modified is resident. The load module must be a member of the partitioned data set defined by the SYSLIB DD statement.

csect

the name of the particular control section that contains the data to be verified or replaced. When this parameter is omitted, it is assumed that the first CSECT contained in the load module (if there is more than one) is the one to be referenced. If there is only one CSECT in the load module, this parameter is not necessary.

Note: More than one NAME statement can be defined in the input to IMASPZAP. However, the VERIFY, REP and SETSSI statements associated with each NAME statement must immediately follow the NAME statement to which they apply.

CCHR record address

used to identify a physical record on a direct access device that is to be modified or verified. The record must be in the data set defined by the SYSLIB DD statement. Any immediately following REP or VERIFY statements will reference the data in the specified record. The parameter is:
record address

the actual direct access device address of the record containing the data to be replaced or verified. It must be specified as a 10-position hexadecimal number in the form cccchhhhr. For all direct access devices other than the 2321 data cell, cccc is the cylinder, hhhh is the track, and rr is the record number. For example, 0001000A01 addresses record 1 of cylinder 1, track 10.

In the case of the 2321 data cell, cccc indicates the subcell and strip; hhhh indicates the cylinder and track; rr indicates the record number. The bin number to which the CCHHR applies is determined by the UNIT parameter in the SYSLIB DD statement. For example, if the SYSLIB DD specifies UNIT=2321/2 and the CCHHR statement specifies 0102000103, then record 3 of subcell 1, strip 2, cylinder 0, track 1 in bin 2 will be retrieved.

In both cases a zero record number is invalid and will default to 1.

Note: More than one CCHHR statement can be defined in the input to IMASPZAP. However, the VERIFY, REP and SETSSI statements associated with each CCHHR statement must immediately follow the specific CCHHR statement to which they apply.

```
VERIFY offset expected content
```

causes the contents at a specified location within a control section or physical record to be compared with the data the user supplies in the statement. If the two fields being compared are not in agreement, no succeeding REP or SETSSI operations, pertinent to the NAME or CCHHR statement in effect, will be performed. The parameters are:

offset

the hexadecimal displacement of the data to be inspected in a CSECT or record. This displacement does not have to be aligned on a fullword boundary, but it must be specified as a multiple of two hexadecimal digits (00, 021C, 014682, etc.). If this offset value is outside the limits of the CSECT or data record defined by the preceding NAME or CCHHR statement, the VERIFY statement will be rejected. When inspecting a record with a key, the length of the key should be considered in the calculation of the displacement; i.e., offset zero is the first byte of the key.

expected content

defines the bytes of data that are expected at the specified location. As with the offset parameter, the number of bytes of data defined must be specified as a multiple of two hexadecimal digits. If desired, the data within the parameters may be separated by commas (never blanks), but again, the number of digits between commas must also be a multiple of two. For example, the data may look like this:

```
5840C032 (without commas),
```

or like this:

```
5840,C032 (with commas)
```
If all the data will not fit into one VERIFY statement (80-byte logical record), then another VERIFY statement must be defined.

A formatted dump of the CSECT or data record is automatically provided following each rejected VERIFY, so that the cause of the rejection can be determined. Subsequent REP (replacement) or SETSSI operations will not be performed if a verification is rejected, but other VERIFY operations will be performed, permitting complete verification in one execution of IMASPZAP. The error condition caused by the VERIFY reject will be in effect only until another NAME or CCHHR statement is encountered. Any subsequent VERIFY or REP statements can then be processed.

REP offset data

used to modify data at a specified location in a CSECT or physical record that has been previously defined by a NAME or CCHHR statement. The data specified on the REP statement will replace the data at the record or CSECT location stipulated in the offset parameter field. This replacement is on a "one for one" basis; that is, one byte of data defined in the statement replaces one byte of data at the specified location. The parameters are:

offset

is the hexadecimal displacement of the data to be replaced in a CSECT or data record. This displacement need not address a fullword boundary, but it must be specified as a multiple of two hexadecimal digits (00, 02C8, 001C52). If this offset value is outside the limits of the data record (physical block) or CSECT being modified, the replacement operation will not be performed. When replacing data in a record with a key, the length of the key should be considered in the calculation of the displacement; i.e., offset zero is the first byte of the key.

data

defines the bytes of data that are to be inserted at the specified location. As with the offset parameter, the number of bytes of data defined must be specified as a multiple of two hexadecimal digits. If desired, the data within the parameter may be separated by commas (never blanks), but again, the number of digits between commas must also be a multiple of two. For example, a REP data parameter may look like this:

4160B820 (without commas)

or like this:

4160,B820 (with commas).

If all the data to be modified will not fit into one REP statement (80-byte logical record), then another REP statement must be defined.

NOTE: ALTHOUGH IMASPZAP DOES NOT REQUIRE THE USER TO VERIFY A LOCATION BEFORE PERFORMING A REP OPERATION, IT IS ADVISABLE TO CHECK THE CONTENTS TO MAKE SURE THAT THE DATA BEING CHANGED IS, IN FACT, WHAT THE USER EXPECTS IT TO BE.

The user should also keep in mind the fact that IMASPZAP, as a part of its normal function, updates the system status index (SSI) for the specified module upon successful completion of the last REP operation performed on a control section of that particular module.
For a more complete explanation of the value of the SSI to the maintenance of a load module, refer to "Updating System Status Information" in this chapter.

Two programming notes that are pertinent to this discussion of the REP statement are listed below:

- If multiple VERIFY and REP operations are to be performed on a CSECT, then all the VERIFY statements should precede all the REP statements. This procedure will ensure that all the REP operations are ignored if a VERIFY reject occurs.

- When a record in the VTOC (i.e., a DSCB) is accessed for modification, message IMAl7D is written to the console. No message is issued, however, when an ABSDUMPT operation is performed on the VTOC.

**IDRDATA xxxxxxxx**

causes IMASPZAP to place up to eight bytes of user data into the IMASPZAP CSECT Identification Record of the load module; this is only done if a REP operation associated with a NAME statement is performed and the load module has been processed by the Linkage Editor to include CSECT Identification Records. The parameter is:

```
xxxxxxx
```

is the eight (or less) bytes of user data (with no imbedded blanks) that is to be placed in user data field of the IMASPZAP IDR of the load module. If more than eight characters are in the parameter field only the first eight characters will be used.

The IDRDATA statement is valid only when used in conjunction with the NAME statement. It must follow its associated NAME statement and precede any DUMP or ABSDUMP statement. IDRDATA statements associated with CCHHR statements will be ignored.

**SETSSI xxyynnnn**

places user-supplied system status information in the PDS (partitioned data set) directory entry for the library member specified in the preceding NAME statement. The SSI, however, must have been created when the load module was link edited. The parameter is:

```
xxynnnn
```

represents the 4 bytes of system status information the user wishes to place in the SSI field for this member. Each byte is supplied as two hexadecimal digits signifying the following:

- xx - change level
- yy - flag byte
- nnnn - modification serial number

If an error has been detected in any previous VERIFY or REP operation, the SETSSI function will not be performed.
Note: Since all bits in the SSI entry are set (or reset) by the SETSSI statement, extreme care should be exercised in its use to avoid altering information vital to the depiction of the maintenance status of the program being changed. (See the discussion in this chapter entitled "Updating System Status Information.")

\{DUMP\} \{member\} \{csect\} \{DUMPT\} \{ALL\}

used to dump a specific control section or all control sections in a load module. The format of the output of this dump is hexadecimal (see the discussion in this chapter entitled "IMASPZAP Output"). The DUMPT statement differs from the DUMP statement in that it also gives the user an EBCDIC and instruction mnemonic translation of the hexadecimal data. The parameters are:

- **member**
  - the member name of the load module that contains the control section(s) to be dumped. (Note: This load module must be a member of a partitioned data set that is defined by the SYSLIB DD statement.)

- **csect**
  - defines the name of the particular control section that is to be dumped. To dump all the CSECTs of a load module, code "ALL" instead of the CSECT name; if the CSECT parameter is omitted entirely, it is assumed that the user means to dump only the first control section contained in the load module.

\{ABSDUMP\} \{startaddr\} \{stopaddr\}
\{ABSDUMPT\} \{membername\} \{ALL\}

These statements are used to dump a group of data records, a member of a partitioned data set, or an entire data set, as defined in the SYSLIB DD statement. If the key associated with each record is to be formatted, DCB=(KEYLEN=nn), where "nn" is the length of the record key, must also be specified by the SYSLIB DD statement. Note that when dumping a VTOC, DCB=(KEYLEN=44) should be specified; when dumping a PDS directory, DCB=(KEYLEN=8) should be specified. ABSDUMP produces a hexadecimal printout only, while ABSDUMPT prints the hexadecimal data, the EBCDIC translation, and the mnemonic equivalent of the data (see "IMASPZAP Output"). The parameters are:

- **startaddr**
  - is the absolute direct access device address of the first record to be dumped. This address must be specified in hexadecimal in the form cccchhhhrr (cylinder, track and record numbers).

- **stopaddr**
  - is the absolute direct access device address of the last record to be dumped, and it must be in the same format as the start address.
Note: Both addresses must be specified when this method of dumping records is used, and both addresses must be within the limits of the data set defined by the SYSLIB DD statement. The record number specified in the start address must be a valid record number. The record number specified as the stop address need not be a valid record number, but if it is not, the dump will continue until the last record on the track specified in the stop address has been dumped.

**membername**

is the name of a member of a partitioned data set. The member can be a group of data records or a load module. In either case, the entire member is dumped when this parameter is specified.

**ALL**

specifies that the entire data set defined by the SYSLIB DD statement is to be dumped.

How much of the space allocated to the data set is dumped depends on how the data set is organized:

- For sequential data set, IMASPZAP dumps until it reaches end of file.
- For indexed sequential and direct access data sets, IMASPZAP dumps all extents.
- For partitioned data sets, IMASPZAP dumps all extents, including all linkage editor control records, if any exist.

**BASE xxxxxx**

used by IMASPZAP to adjust offset values that are to be specified in any subsequent VERIFY and REP statements. This statement should be used when the offsets given in the VERIFY and REP statements for a CSECT are to be obtained from an assembly listing in which the starting address of the CSECT is not location zero.

For example, assume that CSECT ABC begins at assembly listing location X'000400', and that the data to be replaced in this CSECT is at location X'000408'. The actual displacement of the data in the CSECT is X'08'. However, an offset of X'0408' (obtained from the assembly listing location X'000408') can be specified in the REP statement if a BASE statement specifying X'000400' is included prior to the REP statement in the IMASPZAP input stream. When IMASPZAP processes the REP statement, the base value X'000400' will be subtracted from the offset X'0408' to determine the proper displacement of data within the CSECT. The parameter is:

**xxxxxxxx**

is a 6-character hexadecimal offset that is to be used as a base for subsequent VERIFY and REP operations. This value should reflect the starting assembly listing address of the CSECT being inspected or modified.

The **BASE** statement should be included in the IMASPZAP input stream immediately following the **NAME** statement that identifies the control section that is to be involved in the IMASPZAP operations. The specified base value remains in effect until all VERIFY, REP, and SETSSI operations for the CSECT have been processed.
CONSOLE

indicates that IMASPZAP control statements are to be entered through the system console.

When this statement is encountered in the input stream, the following message is written to the operator:

IMA116A ENTER IMASPZAP CONTROL STATEMENT OR END

The operator may then key in any valid IMASPZAP control statement conforming to the specifications described at the beginning of this control statement discussion. After each operator entry through the console is read, validated, and processed, the message is reissued, and additional input is accepted from the console until "END" is replied. IMASPZAP will then continue processing control statements from the input stream until an end-of-file condition is detected.

Note: The control statements can be entered through the console in either upper or lower case letters.

* (Comment)

can be used to annotate the IMASPZAP input stream and output listing. Any number of comment statements can be included in the input stream. When such a statement is encountered, IMASPZAP writes the entire statement to the data set specified for SYSPRINT.

The asterisk (*) can be specified in any position of the statement, but it must be followed by at least one blank space as a delimiter.
IMASPZAP Output

IMASPZAP provides two different dump formats for the purpose of checking the data that has been verified and/or replaced. These dumps (written to the SYSPRINT data set specified by the user) may be of the formatted hexadecimal type or the translated form. Both formats are discussed below in detail with examples showing how each type will look.

The Formatted Hexadecimal Dump

When DUMP or ARSDUMP is the control statement used, the resulting printout will be a hexadecimal representation of the requested data. Figure SPZAP-4 gives a sample of the formatted hexadecimal dump. A heading line is printed at the beginning of each block. This heading consists of the hexadecimal direct access address of the block, a two-byte record length field, and the names of the member and the control section that contain the data being printed (if the dump is for a specific CSECT or load module). Each printed line thereafter has a three-byte displacement address at the left, followed by eight groups of four data bytes each. The following message:

IMA1131 COMPLETED DUMP REQUIREMENTS

is printed directly under the last line of the dump printout.
Figure SPZAP-4. Sample Formatted Hexadecimal Dump
The Translated Dump

The control statements DUMPT and ABSDUMPT also provide an operation code translation and an EBCDIC representation of the data contained in the dump. Figure SPZAP-5 shows the format of the translated dump. The first byte of each halfword of data is translated into its mnemonic operation code equivalent, provided such a translation is possible. If there is no equivalent mnemonic representational value to be given, the space is left blank. This translated line of codes and blanks is printed directly under the corresponding hexadecimal line. An EBCDIC representation of each byte of data is printed on two lines to the right of the corresponding line of text with periods (.) substituted for those bytes that do not translate to valid printable characters.
Figure SPZAP-5. Sample Translated Dump
IMASPZAP Examples

Example 1: Inspecting and Modifying a Load Module Containing a Single CSECT

This example shows how to inspect and modify a load module containing a single CSECT.

```
//ZAPCSECT
//STEP
//SYSPRINT
//SYSLIB
//SYSIN

NAME       IEEVLNKT
VERIFY     0018   C9C8,D2D9,D1C2,C7D5
REP        0018   E5C6,D3D6,E6F0,4040
IDRDATA    711144
DUMP       IEEVLNKT

* / *
```

In this example:

**JOB Statement**

initiates the job

**EXEC Statement**

invokes IMASPZAP.

**SYSPRINT DD Statement**

defines the message data set.

**SYSLIB DD Statement**

defines the system library SYS1.LINKLIB containing the module IEEVLNKT that SPZAP is to process.

**SYSIN DD Statement**

defines the input stream.

**NAME Control Statement**

instructs IMASPZAP that the operations defined by the control statements that follow are to be performed on the module IEEVLNKT.

**VERIFY Control Statement**

requests that IMASPZAP check the hexadecimal data at the location that is offset X'0018' from the start of the module IEEVLNKT to make sure that it is the same as the hexadecimal data specified in this statement. If the data is the same, IMASPZAP continues processing the subsequent statements sequentially. If the data is not identical, IMASPZAP dumps a hexadecimal image of the module IEEVLNKT to the SYSPRINT data set. As a result of this "VERIFY REJECT", IMASPZAP will not perform the REP and SETSSI operations requested for the module. It will, however, perform the DUMP operation.
requested before discontinuing the processing.

REP Control Statement
causes IMASPZAP to replace the data at hexadecimal offset 0018 from
the start of module IEEVLNKT with the data given in this control
statement, provided the VERIFY statement was successful.

SETSSI Control Statement
instructs IMASPZAP that it is to replace the system status
information in the directory entry for module IEEVLNKT with the SSI
data given in the statement, provided the VERIFY statement was
successful. The new SSI is to contain:
1. A change level of 01,
2. A flag byte of 21,
3. A serial number of 1234.

IDRDAIA Control Statement
causes IMASPZAP to update the IDR in module IEEVLNKT with the data
71144, if the REP operation is successful.

DUMP Control Statement
requests that a hexadecimal image of module IEEVLNKT be dumped to
the SYSPRINT data set. Since the DUMP statement follows the REP
statement, the image will reflect the changes made by IMASPZAP
(provided the control statements were successfully verified).

Example 2: Inspecting and Modifying a CSECT in a Multiple-CSECT Load Module

This example show how to apply an IBM-supplied PTF in the form of an
IMASPZAP fix, rather than a module replacement PTF.

```plaintext
//PTF40228 JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=IMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNNAME=SYS1.NUCLEUS,DISP=OLD
//SYSP DD *
NAME IEANUC01 IEWFETCH
IDRDATA LOCFIX01
VERIFY 01F0 47F0C018
VERIFY 0210 5830C8F4
REP 01F0 4780C072
REP 0210 4130C8F4
SETSSI 02114228
DUMPT IEANUC01 IEWFETCH
```

JOB Statement
initiates the job.

EXEC Statement
invokes IMASPZAP.

SYSPRINT DD Statement
defines the message data set.

**SYSLIB DD Statement**

defines the library (SYS1.NUCLEUS) that contains input module IEANUC01.

**SYSIN DD Statement**

defines the input stream that contains the SPZAP control statements.

**NAME Control Statement**

instructs IMASZAP that the operations defined by the control statements that immediately follow this statement are to be performed on the CSECTIEWFETCH contained in the load module IEANUC01.

**IDRDATA Control Statement**

causes IMASZAP to update the IDR in module IEANUC01 for CSECTIEWFETCH with the data LOCIX01, if either of the REP operations is successful.

**VERIFY Control Statements**

request that IMASZAP compare the contents of the locations X'01F0' and X'0210' in the control section IEWFETCH with the data given in the VERIFY control statements. If the comparisons are equal, IMASZAP will continue processing subsequent control statements in the order in which they are encountered. However, if the data at the locations does not compare identically to the data given in the VERIFY control statements, IMASZAP will dump a hexadecimal image of CSECTIEWFETCH to the SYSPRINT data set; the subsequent REP and SETSSI statements will be ignored. The DUMPT function specified will be performed before IMASZAP terminates processing.

**REP Control Statements**

cause IMASZAP to replace the data at hexadecimal offsets X'01F0' and X'0210' from the start of CSECTIEWFETCH with the hexadecimal data specified on the corresponding REP statements.

**SETSSI Control Statement**

requests that IMASZAP replace the system status information in the directory for module IEANUC01 with the SSI data given in the SETSSI statement after the replacement operations have been effected. The new SSI will contain:

1. A change level of 02,
2. A flag byte of 11,
3. A serial number of 4228.

**DUMPT Control Statement**

causes IMASZAP to perform the DUMPT function for CSECTIEWFETCH of load module IEANUC01.
Example 3: Inspecting and Modifying Two CSECTs in the Same Load Module

This example shows how to inspect and modify two control sections in the same module.

```
//CHANGIT
//STEP   EXEC   PGM=IMASPZAP
//SYSPRINT DD   SYSOUT=A
//SYSLIB   DD   DSNAMESYS1.LINKLIB,DISP=OLD
//SYSIN   DD
   NAME   IEFX5000 IEFQMSSS
   VERIFY 0284 4780,C096
   REP    0284 4770,C096
   IDRDATA PTF01483
   SETSSI 01212448
   DUMPT IEFX5000 IEFQMSSS
   NAME   IEFX5000 IEFQMRAW
   VERIFY 0154 4780,C042
   REP    0514 4770,C042
   IDRDATA PTF01483
   SETSSI 01212448
   DUMPT IEFX5000 IEFQMRAW
/*

JOB Statement

initiates the job.

EXEC Statement

invokes IMASPZAP.

SYSPRINT DD Statement

defines the message data set.

SYSLIB DD Statement

defines the data set to be accessed by IMASPZAP while performing the
operations specified by the control statements. In this case, it
defines the system library SYS1.LINKLIB containing the load module
IEFX5000 that is to be changed by IMASPZAP.

NAME Control Statement #1

instructs IMASPZAP that the operations requested via the control
statements immediately following it are to be performed on CSECT
IEFQMSSS in load module IEFX5000 that resides in the data set
defined by the SYSLIB DD statement.

VERIFY Control Statement #1

requests that IMASPZAP check the hexadecimal data at offset X'0284'
from the beginning of CSECT IEFQMSSS to make sure it is the same as
the data specified in this control statement. If the two data fields
match, IMASPZAP continues processing the control statements that
follow sequentially. If the data is not identical, IMASPZAP dumps a
formatted hexadecimal image of CSECT IEFQMSSS to the SYSPRINT data
set. If a "VERIFY REJECT" occurred, IMASPZAP would not perform the
REP or SETSSI functions for CSECT IEFQMSSS, but it would implement
the DUMPT function specified for this CSECT and continue to process
the control statements that follow in the same job step.

Chapter 11: IMASPZAP 303
REP Control Statement #1

causes IMASPZAP to replace the data at hexadecimal displacement 0284 from the beginning of CSECT IEFQMSSS with the hexadecimal data given in this control statement.

IDRDATA Control Statement #1

causes IMASPZAP to update the IDR in module IEFX5000 for CSECT IEFQMSSS with the data PTF01483, if the first REP operation is successful.

SETSSI Control Statement #1

instructs IMASPZAP that it is to replace the system status information in the directory entry for module IEFX5000 with the SSI data given. The new SSI will contain:

1. A change level of 01,
2. A flag byte of 21,
3. A serial number of 2448.

DUMPT Control Statement #1

causes IMASPZAP to perform the DUMPT operation on CSECT IEFQMRAW, and nullifies any previous "VERIFY REJECTS" that may have been encountered.

NAME Control Statement #2

indicates that the operations defined by the control statements that immediately follow this statement are to be performed on CSECT IEFQMRAW in the load module IEFX5000.

VERIFY Control Statement #2

requests that IMASPZAP perform the VERIFY function at offset X'0154' from the start of CSECT IEFQMRAW. If the VERIFY operation is successful, IMASPZAP will continue processing the subsequent control statements sequentially. If the VERIFY is rejected, however, IMASPZAP will not perform the following REP or SETSSI operations, but it will dump a hexadecimal image of CSECT IEFQMRAW to the SYSPRINT data set and perform the DUMPT operation as requested.

REP Control Statement #2

causes IMASPZAP to replace the data at hexadecimal offset X'0154' from the start of CSECT IEFQMRAW with the hexadecimal data that is specified in this control statement.

IDRDATA Control Statement #2

causes IMASPZAP to update the IDR in module IEFX5000 for CSECT IEFQMRAW with the data PTF01483, if the second REP operation is successful.

SETSSI Control Statement #2

causes IMASPZAP to perform the same function as the previous SETSSI, but it is performed only if the second VERIFY is not rejected.
DUMPT Control Statement #2
causes IMASPZAP to perform the DUMPT function on control section IEFQMRAW.

Example 4: Inspecting and Modifying a Data Record
In this example, the data set to be modified is a volume table of contents.

```
//ZAPIT JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=IMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNNAME=FORMAT4.DSCB,DISP=OLD,
// UNIT=2311,VOLUME=SER=111111,DCB=(KEYLEN=44)
//SYSIN DD*
 CCHHR 005000001
VERIFY 2C 0504
REP 2C 0A08
REP 2E 0001,03000102
ABSDUMPT ALL
/*
```

JOB Statement
initiates the job.

EXEC Statement
invokes IMASPZAP.

SYSPRINT DD Statement
defines the message data set.

SYSLIB DD Statement
defines the data set to be accessed by IMASPZAP in performing the operations specified by the control statements. In this example, it defines the VTOC (a Format 4 DSCB) on a 2311 volume with a serial number of 111111. DCB=(KEYLEN=44) is specified so that the dump produced by the ABSDUMPT control statement will show the dsname which is a 44 byte key. Note that this is not necessary for the VERIFY and REP control statements.

CCHHR Control Statement
indicates that IMASPZAP is to access the direct access record address "0005000001" in the data set defined by the SYSLIB DD statement while performing the operations specified by the following control statements.

VERIFY Control Statement
requests that IMASPZAP check the data at hexadecimal displacement X'2C' from the start of the data record defined in the CCHHR statement to make sure it is the same as the hexadecimal data specified in this control statement. If the data is the same, IMASPZAP continues processing the following control statements sequentially. If the data is not identical, IMASPZAP dumps a formatted hexadecimal image of the data record defined by the CCHHR
statement to the SYSPRINT data set. If a "VERIFY REJECT" occurred, IMASPZAP would not perform the REP functions requested, but it would give the dump specified by the ABSDUMPT statement.

REP Control Statements
cause the eight bytes of data starting at displacement 2C from the beginning of the record to be replaced with the hexadecimal data in the REP control statements. The 2C displacement value allows for a 44-byte key at the beginning of the record.

ABSDUMPT Control Statement
causes IMASPZAP to dump the entire data set to the SYSPRINT data set. Since DCB=(KEYLEN=44) is specified on the SYSLIB DD statement, the 44 byte dsname will also be dumped.

Note: If the VTOC is to be modified, message IMA117D will be issued to the operator, requesting permission for the modification.

Example 5: Entering SPZAP Control Statements Through the Console
This example shows how to enter IMASPZAP control statement through the console.

//CONSLIN JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=IMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN DD *

CONSOLE
/*

JOB Statement
initiates the job.
EXEC Statement
invokes IMASPZAP.
SYSPRINT DD Statement
defines the message data set.
SYSLIB DD Statement
defines the data set that contains the module to be updated.
SYSIN DD Statement
defines the input stream.
CONSOLE Control Statement
indicates that IMASPZAP control statements are to be entered through the console.
Example 6: Using the BASE Control Statement

This example shows how to inspect and modify a CSECT whose starting address does not coincide with assembly listing location zero.

```
//MODIFY JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=IMASPZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
//SYasin DD
  NAME IEFMCVOL IEFCVOL2
  BASE 0398
  IDRDATA MOD04
  VERIFY 039A 5820C010
  REP 039A 47000000
  DUMP IEFMCVOL IEFCVOL2
/*

JOB Statement

initiates the job.

EXEC Statement

invokes IMASPZAP.

SYSPRINT DD Statement

defines the message data set.

SYSLIB DD Statement

defines the data set to be accessed by IMASPZAP when performing the operations requested via the control statements. In this case, it defines the system library, SYS1.LINKLIB, that contains the module IEFMCVOL in which the CSECT to be changed, IEFCVOL2, resides.)

SYSIN DD Statement

defines the input stream that contains the IMASPZAP control statements.

NAME Control Statement

instructs IMASPZAP that the operations defined by the control statements that immediately follow it are to be performed on CSECT IEFCVOL2 in the load module IEFMCVOL.

BASE Control Statement

provides IMASPZAP with a base value that is to be used to readjust the offsets on the VERIFY and REP statements that follow it.

IDRDATA Control Statement

causes IMASPZAP to update the IDR in module IEFMCVOL for CSECT IEFCVOL2 with the data MOD04, the the REP operation is successful.
VERIFY Control Statement

requests that IMASPZAP inspect the data at offset X'039A'. The base value X'0398' given in the previous BASE statement is subtracted from this offset to determine the proper displacement of the data within CSECT IEFCVOL2. Therefore, IMASPZAP checks the data at the location that is actually displaced X'0002' bytes from the beginning of CSECT IEFCVOL2 to ensure that it is the same as the hexadecimal data specified in this control statement.

If the data is the same, IMASPZAP continues processing the following statements in the order in which they are encountered. If the data is not identical, IMASPZAP dumps a hexadecimal image of CSECT IEFCVOL2 to the SYSPRINT data set.

If a "VERIFY REJECT" occurs, IMASPZAP will not perform the REP, SETSSI, and IDRDATA functions, but it will perform the DUMP function requested for CSECT IEFCVOL2.

REP Control Statement

causes IMASPZAP to replace the data at displacement X'0002' (offset 039A minus base value 0398) into CSECT IEFCVOL2 with the hexadecimal data specified in this control statement.

DUMP Control Statement

requests that IMASPZAP dump a hexadecimal image of CSECT IEFCVOL2 to the SYSPRINT data set. Since the DUMP statement follows the REP statement, the image will reflect the changes made by IMASPZAP (assuming no verification has been rejected).
Appendix: Writing EDIT User Programs
Tells how to write and use EDIT user programs.
INTRODUCTION .......................................................... 313

USER PROGRAM INTERFACES ........................................... 314
Gaining Control ......................................................... 314
Using the Parameter List ............................................ 314
Input Record ........................................................... 315
GTF Option Word ....................................................... 316
Returning to EDIT ..................................................... 316
Exit Routine Return Codes ......................................... 317
Format Appendage Return Codes ................................... 317
Handling Errors ....................................................... 317
Errors in Finding or Loading a User Program .................. 318
Invalid Return Codes and Program Checks ....................... 318

AVOIDING UNRECOVERABLE ERRORS ................................. 319

SAMPLE USER EXIT ROUTINE ....................................... 321

SAMPLE FORMAT APPENDAGE ........................................ 324

DEBUGGING A USER PROGRAM ....................................... 329

JCL AND CONTROL STATEMENT EXAMPLES .......................... 331
Example 1: Link Editing a User Exit Routine into a Library .... 331
Example 2: Testing a User Exit Routine ............................ 332
Example 3: Testing a User Format Appendage .................... 333

Figures

Figure APNDX-1. EDIT Parameter List and Related Fields ........ 315
Figure APNDX-2. Contents of GTF Option Word, Showing GTF Options in Effect ........ 316
Figure APNDX-3. PRDMP/EDIT Actions in Response to Errors in Finding or Loading User Programs .......... 318
Figure APNDX-4. Sample Exit Routine (Part 1 of 3) .............. 321
Figure APNDX-4. Sample Exit Routine (Part 2 of 3) .............. 322
Figure APNDX-4. Sample Exit Routine (Part 3 of 3) .............. 323
Figure APNDX-5. Sample Format Appendage (Part 1 of 2) ....... 324
Figure APNDX-5. Sample Format Appendage (Part 2 of 2) ....... 325
Figure APNDX-6. Sample ABEND Dump Showing Fields Needed for Debugging User Exit Routine ABENDXIT (Part 1 of 3) ........ 326
Figure APNDX-6. Sample ABEND Dump Showing Fields Needed for Debugging User Exit Routine ABENDXIT (Part 2 of 3) ........ 327
Figure APNDX-6. Sample ABEND Dump Showing Fields Needed for Debugging User Exit Routine ABENDXIT (Part 3 of 3) ........ 328
You may want to code special programs to supplement GTF and IMDPRDMP/EDIT operation. EDIT allows for two types of user programs: exit routines and format appendages. Neither type may occupy more than 10K bytes of main storage.

- An exit routine allows you to inspect each input trace record before EDIT begins processing it; on the basis of the inspection you must decide whether EDIT should process the record normally or take special action.

- A format appendage allows you to format all user trace records of a specified type. A format appendage must be named IMDUSRxx, where xx is the hexadecimal form of the format identifier (FID) specified in the GTRACE macro when the record was created.

This appendix is designed to help you write efficient, helpful user programs.
User Program Interfaces

A user program interfaces with the EDIT function of IMDPRDMP in the following ways:

Gaining Control

Until EDIT calls them, user programs reside in SYS1.LINKLIB or in a data set defined by the JOBLIB or STEPLIB DD statement. Once a user program is loaded into main storage, it remains there until EDIT processing is complete, or until it is deleted due to a need for space.

An exit routine is named in the EXIT= parameter of the EDIT control statement. It gets control every time EDIT reads an input trace record, and always completes its examination of the record before EDIT processes it.

A format appendage is invoked only when EDIT encounters a record that contains an FID field corresponding to the name of the format appendage. It remains in main storage until deleted, but only gets control when EDIT encounters a record with the corresponding FID.

Using the Parameter List

When EDIT passes control to a user program, register 1 contains the address of a parameter list. The contents of that parameter list, and its related fields are shown in figure APNDX-1. The exit routine or format appendage uses the parameter list to find the record it is to process, determine how to process it, and decide where to put the processed record.
Input record

As shown in Figure APNDX-1, the first four bytes of the parameter list give the address of the input record. Four-byte fields at offset 12 and 16, respectively, point to the event identifier (EID) field and the data area in the input record.

For a complete description of the input record format, see Figure GTF-8 in Chapter 3: GTF (Generalized Trace Facility).

Figure APNDX-1. EDIT Parameter List and Related Fields
GTF Option Word

A four-byte field at offset 8 in the parameter list gives the address of the GTF option word, a four-byte table that summarizes the GTF options in effect when the input trace records were produced. Figure APNDX-2 lists the contents of the GTF option word.

<table>
<thead>
<tr>
<th>BYTES</th>
<th>BITS</th>
<th>OPTIONS IN EFFECT DURING TRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1</td>
<td>1...</td>
<td>SYSM-- minimal tracing for system events</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>SYSP-- maximum tracing, prompting requested.</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>SYS-- maximum tracing for system events</td>
</tr>
<tr>
<td></td>
<td>...1</td>
<td>USR-- all GTRACE-generated interrupts traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>TRC-- all GTF interrupts traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>DSP-- all task-switches traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>SSM-- all SSM interrupts traced (MP only)</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>PCI-- program-controlled interrupts traced</td>
</tr>
<tr>
<td>Byte 2</td>
<td>1...</td>
<td>SVC-- all SVC interrupts traced</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>SVCP-- SVC interrupts selected by prompting</td>
</tr>
<tr>
<td></td>
<td>...1</td>
<td>SIO-- all SIO events traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>SIOP-- SIO events selected by prompting</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>PI-- all program interrupts traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>PIP-- program interrupts selected by prompting</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>IO-- all I/O interrupts traced</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>IOP-- I/O interrupts selected by prompting</td>
</tr>
<tr>
<td>Byte 3</td>
<td>1...</td>
<td>EXT-- external interrupts traced</td>
</tr>
<tr>
<td></td>
<td>.xxx</td>
<td>reserved bits</td>
</tr>
<tr>
<td></td>
<td>1...1</td>
<td>IO=SIO-- identical devices selected for I/O &amp; SIO</td>
</tr>
<tr>
<td>Byte 4</td>
<td>1...</td>
<td>tracing system - MFT</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>tracing system - MFT with ATTACH</td>
</tr>
<tr>
<td></td>
<td>..1...</td>
<td>tracing system - MVT</td>
</tr>
<tr>
<td></td>
<td>...1</td>
<td>tracing system - Model 65 Multiprocessing</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>real Monitor Call instruction</td>
</tr>
<tr>
<td></td>
<td>....0</td>
<td>simulated monitor call instruction</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>no timer option selected at SYSGEN</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>Tracing system has time-of-day clock</td>
</tr>
<tr>
<td></td>
<td>....1</td>
<td>user timestamp requested</td>
</tr>
</tbody>
</table>

Figure APNDX-2. Contents of GTF Option Word, showing GTF Options in Effect During Trace

For more information about any of the GTF options, refer to Chapter 3, GTF (Generalized Trace Facility).

Returning to EDIT

A user program must return to EDIT with one of the return codes listed below. If EDIT receives an invalid return code from a user program, it takes action as specified by the ER= subparameter of the PARM= parameter of the EXEC statement that invokes IMDPRDMP. This parameter, its values and their meanings are described in Chapter 8: PRDMP in the section "Job Control Language Statements".

316 Service Aids (Release 21)
Exit Routine Return Codes

An exit routine must return to EDIT with one of the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EDIT should print the contents of the output area, clear the area, and return immediately to the exit routine. This allows the exit routine to print more than one line of output. (Note that the output buffer may be in a different location when the format appendage receives control again.)</td>
</tr>
<tr>
<td>4</td>
<td>EDIT should print the contents of the output area and obtain the next logical record.</td>
</tr>
<tr>
<td>8</td>
<td>EDIT should format and print the trace record according to the selectivity specified in the EDIT control statement.</td>
</tr>
<tr>
<td>12</td>
<td>EDIT should obtain the next logical input trace record without printing the contents of the output buffer.</td>
</tr>
<tr>
<td>16</td>
<td>EDIT should print the contents of the output buffer and no longer invoke the exit routine, which is no longer needed.</td>
</tr>
<tr>
<td>20</td>
<td>EDIT should format and print the trace record according to the selectivity specified in the EDIT control statement, and should no longer invoke the exit routine, which is no longer needed.</td>
</tr>
<tr>
<td>24</td>
<td>EDIT should terminate processing and return control to IMDPRDMP so that the next IMDPRDMP control statement may be processed.</td>
</tr>
<tr>
<td>28</td>
<td>EDIT should format and print this record as though no selectivity had been specified in the EDIT control statement.</td>
</tr>
</tbody>
</table>

Format Appendage Return Codes

A format appendage must return to EDIT with one of the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EDIT should print the contents of the output buffer and return immediately to the format appendage. (Note that the output buffer may be in a different location when the format appendage receives control again.)</td>
</tr>
<tr>
<td>4</td>
<td>EDIT should print the contents of the output buffer and obtain the next logical input trace record.</td>
</tr>
<tr>
<td>8</td>
<td>EDIT should obtain the next logical input trace record without printing the contents of the output buffer.</td>
</tr>
</tbody>
</table>

Handling Errors

EDIT is prepared to handle two types of errors: invalid return codes and program checks. Other types of errors and their consequences are discussed later in this appendix, in the section "Avoiding Unrecoverable Errors".
Errors in Finding or Loading a User Program

If EDIT cannot find or load a user program, it takes action as shown in Figure APNDX-3.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Exit Routine</th>
<th>Format Appendage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Found</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Dump</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Trace Data Set</td>
<td>A</td>
<td>B A</td>
</tr>
</tbody>
</table>

Action A: EDIT terminates processing and returns control to IMDPRDMP, which obtains the next IMDPRDMP control statement.

Action B: EDIT dumps the associated record in hexadecimal and obtains the next input trace record. Any subsequent records that have the same FID will be dumped in hexadecimal.

Figure APNDX-3. IMDPRDMP/EDIT Actions in Response to Errors in Finding or Loading User Programs.

Invalid Return Codes and Program Checks

EDIT's action in response to invalid return codes and program checks depends on the value for ER= that you specify in the PARM= parameter of the EXEC statement that invokes IMDPRDMP. For an explanation of the valid values for ER=, refer to the section "Job Control Language Statements" in Chapter 10: IMDPRDMP.
Avoiding Unrecoverable Errors

As shown in the previous sections, EDIT can recover from two kinds of errors in a user program: invalid return codes and program checks. EDIT cannot protect you, however, against errors that you may generate, for example by performing I/O operations or issuing GETMAIN macro instructions. In fact, you should avoid issuing any SVCs in your user program. Ordinarily this is not difficult, since EDIT provides you with the ability to examine records, manipulate data, and request formatted output to be printed. If you must issue an SVC, EDIT will permit you to do so; you should be prepared, however, for possibly unpredictable results if an error occurs during an operation that you have requested by issuing an SVC.

Another error condition that EDIT cannot handle, but which you can avoid, arises when you assign IMDPRDMP too small a region. You must specify a region large enough to accommodate all of IMDPRDMP's work areas and buffers plus all format appendages that can be called plus any exit routine. If you do not do so, IMDPRDMP may delete one or more user programs already in main storage to make room for a new one.

Deletion is critical if the deleted program issues an OPEN because the reinitialization that is necessary when the program is reloaded can cause two DCBs to be open at the same time. Deletion is also critical if the deleted program is an exit routine that sets a switch before relinquishing control and tests the same switch when it gets control again. Resulting errors may not cause abnormal termination, but they can prevent successful operation of the exit routine.

If none of your user programs will be damaged by deletion, you need not allow extra space for them in IMDPRDMP's region. IMDPRDMP's minimum region size includes 10K for use by system format appendages and user programs.

If your user program must issue a GETMAIN macro, be sure to specify a region large enough to include the amount of main storage requested in the user program. Also be sure to reserve that amount of storage for your own use by means of the FREEnnn subparameter of the PARM parameter in the EXEC statement. If you do not reserve it, IMDPRDMP will make available to your program only a limited amount of storage and your GETMAIN may fail. For more information about the FREEnnn parameter, refer to the section "Job Control Language Statements" in Chapter 10: IMDPRDMP.

On completion of your user program, be sure to issue a FREEMAIN macro for all storage that you reserved for your own use. If you do not do so, and your program is deleted, the storage you reserved will remain allocated to you and thus unavailable to subsequent users.
A few examples may further clarify the areas in which EDIT does not provide error recovery:

• A user program, known as module A, issues the LINK SVC for module B. A program check occurs in module B. EDIT will attempt error recovery, since the error is a program check, but it knows nothing about module B. Therefore when it produces diagnostic information it will give the entry point of module A as the entry point of the failing module, and attribute the registers at the time of the program check to module A.

• A user program issues the OPEN SVC (SVC X'13') unsuccessfully and is posted with a system completion code of 213. EDIT cannot recover, so EDIT, the user program and IMDPRDMP will all be terminated.

• A user program opens a DCB. Before it can close the DCB, the program is deleted to make room for another user program. When the deleted program is reloaded, it creates a new DCB and opens it. Thus there are two open DCBs with the same name in storage at the same time. The operating system will not tolerate this situation, so the user program is abnormally terminated.

• A user program issues the SPIE SVC, thereby nullifying EDIT's SPIE routine. As a result any program checks in the user program that EDIT would normally handle will go through the user's own SPIE routine, perhaps with unpredictable results.
Sample User Exit Routine

Figure APNDX-4 shows a sample exit routine. This routine, named ABENDXIT, was written to aid diagnosis of an abnormal termination condition in a particular job. It scans each input trace record, suppressing printing until it finds a record with the specified jobname. When it finds such a record, ABENDXIT signals IMDPRDMP to print that record. All subsequent records will be printed until ABENDXIT encounters an SVC 13 record for the specified jobname; then ABENDXIT instructs IMDPRDMP to print that record and terminate.

Note that this program decides how to treat each new record on the basis of the way it treated previous records. To do this it must maintain certain switches intact between records, and as a result this program is not serially reusable. To guarantee the integrity of the switches in the program, therefore, it is necessary to specify a region large enough to hold both IMDPRDMP and the user exit routine contiguously. This is the only way to make sure that the exit routine will not be deleted if EDIT needs more room to execute.

**********************************************************************
* ABENDXIT IS AN EDIT USER EXIT ROUTINE DESIGNED TO CONTROL PRINTING
* OF ALL GTF RECORDS ASSOCIATED WITH A PROGRAM THAT HAS
* PROGRAM CHECKED AND ABENDED
**********************************************************************

ABENDXIT CSECT
* EQUATE STATEMENTS
FRSTREG EQU 0
PARMREG EQU 1
EIDREG EQU 2
DATEREG EQU 3
WORKREG EQU 4
CHAINREG EQU 9
BASE EQU 12
SAVEPTR EQU 13
RETPTR EQU 14
CODEREG EQU 15
STM RETPTR,BASE,12(SAVEPTR)  STORE REGISTERS
BALR BASE,0        ESTABLISH ADDRESSABILITY
USING *,BASE        USING REGISTER 12
ST SAVEPTR,SAVE+4   BACKWARD CHAINING
LA CHAINREG,SAVE    MY SAVE AREA POINTER
ST CHAINREG,8(SAVEPTR)  FORWARD CHAINING
LR SAVEPTR,CHAINREG REG 13 ADDRESSES SAVE AREA

Figure APNDX-4. Sample Exit Routine.  (Part 1 of 3)
IMDMEDIT  SYMBOLIC EID MACRO

*/* THE IMDMEDIT MACRO MAPS THE EID VALUES ASSOCIATED WITH IBM */
*/* SYSTEM AND SUBSYSTEM EVENTS. THE STORAGE FOR ANY OR ALL OF */
*/* THE MAPPED VALUES MUST BE CONTAINED IN THE MODULE REFERENCING */
*/* THE DESIRED EIDS. IMDMEDIT IS DESIGNED TO BE USED BY IBM-*/
*/* SUPPLIED FORMAT APPENDAGES, AND USER-SUPPLIED USER EXIT */
*/* MODULES. */

+IMDMPCI EQU X'2FDF' PCI I/O INTERRUPT
+IMDMsvc EQU X'3FFF' SVC INTERRUPT
+IMDMDSP EQU X'4FE7' TASK SWITCH
+IMDMIO1 EQU X'5FE7' I/O INTERRUPT
+IMDMIO2 EQU X'5FFF' I/O INTERRUPT
+IMDMSIO EQU X'5FF0' SIO OPERATION
+IMDMSSM EQU X'5FFC' SSM INTERRUPT
+IMDMPI EQU X'6FDF' PROGRAM INTERRUPT
+IMDMEXT EQU X'6FE7' EXTERNAL INTERRUPT
+IMDMMA1 EQU X'SFEE' OPEN/CLOSE/EOV

TM TERMSW,X'01' Q/HAS TERMINATION BEEN REQSTD
BC 1,FNISH YES,TELL EDIT TO TERMINATE
L EIDREG,12(PARMREG) GET POINTER TO EID
L DATAREG,16(PARMREG) GET POINTER TO DATA(JOBNAME)
TM PRINTSW,X'01' Q/HAS JOBN ALREADY BEEN FOUND
BC 1,PRINTALL YES, SO PRINT THIS RECORD
LA WORKREG,0 GET ZERO CONSTANT
C WORKREG,ECBl Q/HAS THIS ECB BEEN POSTED
BC 7,MJOBLAB YES, CHECK IF JOBN FOUND

*WTR 'SPECIFY 8-CHARACTER JOBNNAME OF ABBENDING PROGRAM',
  MYJOBN,8,ECB1

WAIT ECB=ECBl
LA WORKREG,MYJOBN ADDRESS OF JOBNNAME SELECTED
OC 0(8,WORKREG),BLANKS CONVERT LOWER-CASE CHARS TO
  UPPER CASE
MYJOBLAB CLC 0(8,DATAREG),MYJOBN Q/IS THIS MY JOBNNAME
BC 7,NOPRINT NO -- JUST RETURN

* ONCE JOBNNAME FOUND| SET SWITCH AND PRINT ALL RECORDS UNTIL
* ENCOUNTER AN SVC 13 (ABEND) CONTAINING THIS JOBNNAME
OI PRINTSW,X'01' TURN ON JOBNNAME FOUND SWITCH
PRINTALL CLC 0(2,EIDREG),SVCEID Q/IS THIS AN SVC RECORD
BC 7,PRINREC NO, SO PRINT AND CONTINUE
CLI 15(DATAREG),X'0D' Q/IS THIS AN SVC 13 (ABEND)
BC 7,PRINREC NO, SO PRINT AND CONTINUE
CLC 0(6,DATAREG),MYJOBN Q/IS THIS MY JOBNNAME
BC 7,PRINREC NO, SO PRINT AND CONTINUE
EXIT OI TERMSW,X'01' INDICATE THAT THIS IS LAST

* RECORD TO BE PRINTED
PRINTREC LA CODEREG,8 FORMAT AND PRINT THIS RECORD
L SAVEPTR,4(SAVEPTR) RESTORE SAVE AREA POINTER
L RETPTR,12(SAVEPTR) RESTORE REGISTER 14
LM FRSTREG,BASE,20(SAVEPTR) RESTORE OTHER REGS EXCEPT 15
BCR 15,RETPTR RETURN TO EDIT

Figure APNDX-4. Sample Exit Routine (Part 2 of 3)
FINISH LA CODEREG,24 TERMINATE EDIT PROCESSING
* B RETURN SINCE SVC 13 WAS LAST RECORD
NOPRINT LA CODEREG,12 RESTORE REGISTERS AND RETURN
B RETURN IGNORE RECORD
SAVE DC 18'0' RESTORE REGISTERS AND RETURN
SVCEID DC AL2(IMDMSVC) SAVE AREA
TERMSW DC X'00' ESTABLISH REAL AREA FOR EID FROM IMDMEDIT MAP MACRO
PRINTSW DC X'00' TERM ED, 13 WAS LAST RECO R D
ECB1 DC F'0' JOB N FOUND, SO PRINT REC IND
MYJOBN DC C' PLACE FOR OPR TO PUT JOBNAM E
BLANKS DC C' TO CONVERT LOWER TO UPPER CASE
END

Figure APNDX-4. Sample Exit Routine. (Part 3 of 3)

Some instructions in the sample exit routine require special attention. These are shaded in Figure APNDX-4, and they are discussed below.

IMDMDIT

This mapping macro expands, as shown, into a list of equate statements that supply symbolic names for the event identifiers (EIDs). You should use the symbolic name in your program; this is your protection against program failure, if for any reason, the EID values are later changed.

TM TERMSW,X'01'

This instruction tests a switch to determine a course of action. Because of instructions like these, which any user exit is likely to use, you should always make sure your region is large enough so that the user exit need not be deleted at any time during EDIT execution.

L EIDREG,12(PARMREG)
L DATAREG,16(PARMREG)

These two instructions access the EDIT parameter list. (See Figure APNDX-1.)

WTOR 'SPECIFY 8-CHARACTER JOBNAM E OF ABENDING PROGRAM', MYJOBN,8,ECB1

This instruction requests information that cannot be obtained from the EDIT parameter list. You can use a WTOR to request any information that the operator is likely to have, such as the EDIT options in effect. Note, however, that when you issue an SVC in a user program you risk abnormal termination if an error occurs during the SVC operation. For more information about this point, refer to the section "Avoiding Unrecoverable Errors" earlier in this chapter.

SVCEID DC AL2(IMDMSVC)

This establishes a main storage location for the value equated to IMDMSVC in the expansion of the IMDMEDIT mapping macro.
Figure APNDX-5 shows how to use the EDIT parameter list and how to handle multiple EIDs. It consists of excerpts from a sample format appendage named IMDUSR01, which formats three different types of user records. For each record IMDUSR01 produces two lines of output. The first line varies according to the record type. The second line is the same for all records.

********************************************************************
IMDUSR01 IS AN EDIT USER FORMAT APPENDAGE MODULE THAT PROCESSES
THREE DIFFERENT TYPES OF INPUT RECORDS, THUS, THREE DIFFERENT EIDS.
LINE ONE OF THE FORMATTED OUTPUT VARIES ACCORDING TO THE EID. LINE
TWO OF THE FORMATTED OUTPUT IS THE SAME FOR ALL EIDS, AND IS
PRODUCED IN COMMON CODE.
********************************************************************
IMDUSR01 CSECT
* EQUATE STATEMENTS
FIRSTREG EQU 0
PARMREG EQU 1
EIDREG EQU 2
DATAREG EQU 3
CHAINREG EQU 9
BASE EQU 12
SAVEPTR EQU 13
RETPTR EQU 14
CODEREG EQU 15
STM RETPTR,BASE,12(SAVEPTR) STORE REGISTERS
BALR BASE,0 ESTABLISH ADDRESSABILITY
* Base using register 12
ST SAVEPTR,SAVE+4 BACKWARD CHAINING
LA CHAINREG,SAVE MY SAVE AREA POINTER
ST CHAINREG,8(SAVEPTR) FORWARD CHAINING
LA SAVEPTR,CHAINREG REG 13 ADDRESSES SAVE AREA
L EIDREG,12(PARMREG) GET POINTER TO EID
L DATAREG,16(PARMREG) GET POINTER TO FIRST LINE DATA
TM SWITCH,X'01'
BC 1,LINETWO Q/ HAS FIRST LINE BEEN OUTPUTTED
* WHICH IS COMMON TO ALL THREE EID RTNS
CLC 0(2,EIDREG),EID1 NO--Q/ IS THIS A RECORD WITH EID1
BC 8,RTN1 YES--FORMAT LINE ONE
CLC 0(2,EIDREG),EID2 Q/ IS THIS A RECORD WITH EID2
BC 8,RTN2 YES--FORMAT LINE ONE
CLC 0(2,EIDREG),EID3 Q/ IS THIS A RECORD WITH EID3
BC 8,RTN3 YES--FORMAT LINE ONE
LA CODEREG,8
BC RETURN NO--IF NONE OF THESE EIDS, IGNORE
....
RTN1
....
B ZEROCODE SET ZERO RETURN CODE

Figure APNDX-5. Sample Format Appendage (Part 1 of 2)
RTN2
   ...
   B ZEROCODE

RTN3
   ...
   *
   B RETURN

ZEROCODE OI SWITCH,X'01'
SR CODEREG,CODEREG
   FIRST LINE COMPLETE INDICATOR
   OUTPUT THIS LINE AND RETURN
*   IMMEDIATELY TO THIS
   FORMAT APPENDAGE
   RESTORE REGISTERS AND RETURN
LINETWO
   ...
   NI SWITCH,X'FE'
   TURN OFF LINE 2 INDICATOR
   OUTPUT THIS LINE--COMPLETE
   CODEREG,4 OUTPUT THIS LINE AND
   COMPLETE INDICATOR
   RETURN RESTORE REGISTERS AND
   RETURN
RETURN L SAVEPTR,4(SAVEPTR)
   RESTORE SAVE AREA POINTER
L RETPTR,12(SAVEPTR)
   RESTORE REGISTER 14
LM FRSTREG,BASE,20(SAVEPTR)
   RESTORE OTHER REGS EXCEPT 15
BCR 15,RETPTR
   RETURN TO EDIT
SAVE DC 18F'0'
SWITCH DC X'00'
EID1 DC X'E001'
EID2 DC X'E002'
EID3 DC X'E003'
END
/*

Figure APNDX-5. Sample Format Appendage (Part 2 of 2)
Figure APPX-6. Sample ABEND Dump Showing Fields Needed for Debugging
User Exit Routine ABENDXIT (Part 1 of 3)
Figure APNDX-6. Sample ABEND Dump Showing Fields Needed for Debugging User Exit Routine ABENDXIT (Part 2 of 3)
Figure APNDX-6. Sample ABEND Dump Showing Fields Needed for Debugging
User Exit Routine ABENDXIT (Part 3 of 3)
Debugging a User Program

Figure APNDX-6 shows a sample ABEND dump of the user exit routine ABENDXIT, shown in Figure APNDX-5. Certain important fields are highlighted in the figure and marked with numbers; the numbers refer to the explanations below:

1. **PSW** for the abnormally terminating program. The address in the second half of the PSW is an address in the abnormally terminated program. To find the entry point and name of the program, compare this address to the entry point addresses in the contents directory entry list. The abnormally terminating program is the one whose entry point address is closest to and greater than the address in the PSW.

   **NOTE:** If the address in the PSW does not immediately indicate the entry point address of the failing program, you can locate the beginning address of the abnormally terminating program by tracing IMDPRDMP's save area chain. See point 4, below.

2. Part of a contents directory entry (CDE). This shows the name of the abnormally terminating program, ABENDXIT, its entry point, X'05D080', and the pointer to the appropriate entry in the extent list.

3. An extent list entry. This shows the beginning address (not necessarily the entry point) of the abnormally terminating program. Subtract this address from the address in the PSW to find the address of the instruction following the instruction that failed.

   For example, in this case:

   \[
   \text{address in PSW} - \text{beginning address} = \text{offset (hex)}
   \]

   \[
   5D092 - 5D080 = 12
   \]

   The failing instruction in ABENDXIT can be found at offset X'12' in the program. (See part 2 of Figure APNDX-6, number 3.)

4. The first save area in the save area trace table (system save area) is chained to the following IMDPRDMP module save areas:

   - **IMDPRCTL** - IMDPRDMP control routine
   - **IMDPRMSC** - IMDPRDMP scan routine
   - **IMDPRFRM** - EDIT control routine
   - **IMDPRFLT** - EDIT trace record selection routine
   - **IMDPREXT** (or **IMDPRAPP**) - EDIT user program selection routine.

5. The user program's registers are stored in IMDPREXT's or IMDPRAPP's save area. Add the contents of register 12 to X'6AC' to get the address of a fullword that points to an EDIT communication table. At offset X'1D0' into this table are the following:

   A. The 8-byte EBCDIC name of the current user program (the failing program).

   B. The entry point address of the current user program (the failing program).
These fields are shown in part 3 of Figure APNDX-6.

6. Register 1 in IMDPREXT's or IMDPRAPP's save area points to the parameter list that EDIT passes to the user program. (See Figure APNDX-1.)
The following examples show how to test a user program.

Example 1: Link Editing a User Exit Routine into a Library

This example shows how to make a user exit routine available to IMDPRDMP by link-editing it into a system library.

```plaintext
//LKUSRPGM JOB
MSGLEVEL=(1,1)
// EXEC
PGM=IEWL,PARM='XREF,LET,LIST,NCAL',
// REGION=96K
//SYSPRINT DD
SYSOUT=A
//SYSLMOD DD
DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN DD
object deck
NAME EXITNAME
/*
In this example:
EXEC Statement
invokes the linkage editor and requests maximum diagnostic listings.
SYSPRINT DD Statement
defines the message data set.
SYSLMOD DD Statement
defines the output data set, in this case the linkage library, SYS1.LINKLIB. The output data set can also be a permanent library to be invoked later by a JOBLIB or STEPLIB DD statement; in that case the SYSLMOD DD statement should be coded as follows:

```plaintext
//SYSLMOD DD
DSNAME=MYLIB,UNIT=2314,VOL=SER=231400,
DISP=(NEW,KEEP),SPACE=(1024,(20,2,1))
```

SYSLIN DD Statement
defines the input data set, in this case, the object deck for the user program.

NAME Control Statement
specifies the member name, and thus the program name, to be assigned to the user program. In this case, the member name is EXITNAME; to invoke this program in a later execution of IMDPRDMP, you would have to specify EXIT=EXITNAME on the EDIT control statement.
Example 2: Testing a User Exit Routine

This example shows how to link edit a user exit routine into a library for testing.

```
//TSEXTRTN JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IEWL,PARM='XREF,LET,LIST,NCAL',
    // REGION=96K
//SYSPRINT DD SYSOUT=A
//SYSLMOD DD DSNAME=MYLIB,UNIT=2314,VOL=SER=231400,
    // DISP=(NEW,KEEP),SPACE=(1024,(20,2,1))
//SYSLIB DD *
   object deck
   NAME MYEXIT
/*
//STEP2 EXEC PGM=IMDPRDMP,PARM='ER=1'
//STEPLIB DD DSNAME=MYLIB,UNIT=2314,VOL=SER=231400,
   // DISP=OLD
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A
//TRACEDD DD DSNAME=TRACE2,UNIT=2400,VOL=SER=TRC2TP,
   // LABEL=(,NL),DISP=OLD
//SYSIN DD *
   EDIT DDNAME=TRACEDD,SYS,EXIT=MYEXIT
*/
```

This example consists of two steps. In the first step:

**EXEC Statement**

invokes the linkage editor and requests diagnostic information.

**SYSPRINT DD Statement**

defines the message data set.

**SYSLMOD DD Statement**

defines the output data set, in this case a permanent job or step library named MYLIB.

**SYSLIN DD Statement**

defines the input data set, in this case an object deck containing the user program.

**NAME Control Statement**

specifies a member name (program name) to be assigned to the user program. Specify this program name on the EDIT control statement (EXIT=MYEXIT) when you need the exit routine for a particular IMDPRDMP execution.

In the second step:

**EXEC Statement**

invokes IMDPRDMP and specifies that, if an error occurs in the exit routine, EDIT should print the record associated with the error and delete the exit routine. (See the discussion of the EXEC statement in the section "Job Control Language Statements" earlier in this chapter.)
STEPLIB DD Statement

defines the data set that contains the exit routine, which, in this case, is MYLIB, a data set defined in STEPLIB by the SYSLMOD DD statement.

SYSPRINT DD Statement

defines the message data set.

PRINTER DD Statement

defines the data set to which IMDPRDMP output will be directed.

TRACEDD DD Statement

defines the data set containing trace records to be processed by the exit routine.

SYSIN DD Statement

defines the data set that contains the IMDPRDMP control statement. The data set follows immediately.

EDIT Control Statement

invokes the EDIT function of IMDPRDMP, specifies that the trace data exists as an external trace data set, and supplies the name of the exit routine. Note that this name is the same as the membername specified in the NAME control statement in STEPLIB.

Example 3: Testing a User Format Appendage

This example shows how to add a user format appendage to a temporary data set for testing.

```plaintext
//TSTPFMT JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IEWL,PARM='XREF,LET,LIST,NCAL',
//   REGION=96K
//SYSPRINT DD SYSOUT=A
//SYSLMOD DD DSNAME=&TEMPLIB,UNIT=SSYSDA,
//   SPACE=(1024,(20,2,1)),DISP=(NEW,PASS)
//SYSLIN DD
   * object deck
   NAME IMDUSR01
   *
//STEP2 EXEC PGM=IMDPRDMP,PARM='ER=3'
//STEPLIB DD DSNAME=&TEMPLIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//PRINTER DD SYSOUT=A
//TRACEDD DD DSNAME=TRACE,UNIT=2400,VOL=SER=TRCTPE,
//   LABEL=(,NL),DISP=OLD
//SYSIN DD *
   EDIT DDNAME=TRACEDD,USR=ALL
   */
```

This example consists of two steps. In the first step:

EXEC Statement

invokes the linkage editor.

Appendix - Writing EDIT User Programs 333
SYSPRINT DD Statement
defines the message data set.

SYSLMOD DD Statement
defines a temporary data set that contains the format appendage.

SYSLIN DD Statement
defines the input data set, in this case the object deck containing the format appendage.

NAME Control Statement
specifies a member name (program name) for the format appendage. 
Note that the name shown in this example conforms to the convention for naming format appendages; that is, it is formed from the prefix IMDUSR concatenated with the format identifier (FID) to be specified in the GTRACE macro when user records are created.

In the second step:

EXEC Statement
invokes IMDPRDMP and specifies that ABEND processing should not be suppressed if a program check occurs in the format appendage. (See the discussion of the EXEC statement in the section "Job Control Language Statements" earlier in this chapter.)

STEPLIB DD Statement
defines the data set where the format appendage resides.

SYSPRINT DD Statement
defines the message data set.

PRINTER DD Statement
defines the data set to which the format appendage will direct its output.

TRACEDD DD Statement
defines the trace data set containing the records that the format appendage will process. In this case, the trace data set is on tape.

SYSIN DD Statement
defines the data set containing IMDPRDMP control statements. The data set follows immediately.

EDIT Control Statement
invokes the EDIT function of IMDPRDMP, specifies that the trace data exists as an external trace data set, and specifies that EDIT is to process all user-created records.
Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, Order No. C28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

@USE parameter
- of PTFLE EXEC statement 227,228
- control statement 295
  - used in SPZAP
    - function 295
    - parameter 295

abbreviations for PRDMP control statements 183
ABS_DUMP control statement 293-294
  - used in SPZAP
    - format 293
    - function 293
    - parameters 293-294
ABS_DUMP control statement 293-294
  - used in SPZAP
    - example 305
    - format 293
    - function 293
    - parameters 293-294
AID (application identifier)
  - in GTF output record 70,76,77
ALL parameter
- (see PRINT control statement)
allocated main storage
- (see PRINT control statement)
application identifier
- (see AID)

BASE control statement 294
  - used in SPZAP
    - example 307
    - format 294
    - function 294
    - parameters 294
CCHHR control statement 289-290
  - used in SPZAP
    - example 305
    - format 289
    - function 289
    - parameters 290
changing space allocation for SYS1.LOGREC 23
choosing a service aid 13-14
combining PRDMP control statements 192
comment control statement
- (see * control statement)
communications vector table
  - address of, how to specify 184
  - (see also CVT control statement)

comprehensive trace
  - how to request 60
CONSOLE= parameter
  - (see IMDGADMP macro instruction)
console communications
  - in GTF 63
  - in JQDMP 91-93
  - in OSJQD 161
  - in PRDMP 183
  - in SADMP 265-266
CONSOLE control statement 295
  - used in SPZAP
    - examples 306
    - format 295
    - function 295
    - parameters 295
consoles supported by SADMP 257-258,268
control block formatting by PRDMP 177,186
  - (see also FORMAT control statement)
control section, description of 131
  - (see also CSECT Identification Record)
control statements
  - for DIP00 23
  - for EREP0 32
  - for LIST 109-111
  - for OSJQD 162
  - for PRDMP 183-192
  - for PTFLE 230-232
  - for SPZAP 289-295
CPU= parameter
  - (see IMDGADMP macro instruction)
cross-reference listing
  - output of LIST
    - contents of 115-116
    - how to obtain 109
CSECT identification record
  - how to create (PTFLE) 231
  - how to print (LIST) 111
  - how to update (SPZAP) 292
CURRENT parameter
  - (see PRINT control statement)
current task's main storage, printing of
  - (see PRINT control statement)
C VT address, how to specify 184
CVT (communications vector table)
  - (see CVT= control statement)
CVT control statement 184
  - used in PRDMP
    - function 184
    - format 184
damage assessment routine
  - (see F03)
DAR (damage assessment routine)
  - (see F03)
DATA= parameter
(see GTRACE macro instruction)
data management records, printing of 188
DD statements
in DIPOO 23
  SEREORDS 23
in EREPO 32
  ACCIN 32
  ACCDEM 32
  EREP 32
  JOOIB 32
  SERLOG 32
in LIST 119-124
  input 119-124
  output 119-124
in SYSPRINT 119-124
in MDMAP 140
  input 140
  SNAPDUMP 140
  SYSABEND 140
  SYSPRINT 140
  SYSUDUMP 140
in OSJQD 159
  OSJQDIN 159
  OSJQDOUT 159
  SYSIN 159
  SYSPRINT 159
in PRDMP 180-182
  anyname 180
  PRINTER 181
  SYSIN 181
  SYSPRINT 181
  SYSUT1 182
  SYSUT2 182
  SYSWAMM 181
  TAPE 180
in PTFLE 228,229
  OUTF 228,229
  PCHF 228,229
  PRINT 228,229
  PTF.MODF 237
  SYSLMOD 228
  SYSPRINT 228
  SYSUT1 228
  SYSUT2 228
in SPZAP 288
  SYSABEND 288
  SYSIN 288
  SYsLIB 288
  SYSPRINT 288
DDN= parameter
(see LISTIDR control statement;
  LISTLOAD control statement;
  LISTOBJ control statement)
DDNAME= parameter
(see NEWDUMP control statement;
  EDIT control statement)
DEBUG= parameter
(see GTF START command parameters)
device identification command (JQDMP) 91
differences between JQDMP and OSJQD 157
DIPOO service aid 17
control statements 22
JCL statements 22-24
how to run 23-24
input 22
DMA1 190

DSP trace option in GTF 62
DSP parameter
(see EDIT control statement)
DUMP control statement 293
used in SPZAP
eample 300
  format 293
  function 293
  parameters 293
dump title, how to specify
in LIST
  in LISTIDR control statement 111
  in LISTLOAD control statement 109
  in LISTOBJ control statement 110
in PRDMP
  in TITLE control statement 185
dumping main storage 245
dumping SYS1.LOGREC 29
DUMPT control statement 293
used in SPZAP
eample 301
  format 293
  function 293
  parameters 293

EDIT control statement 187-191
used in PRDMP
  format 188
  function 187
  parameters 188-190
    DGP 190
    EXIT= 188
    EXT 190
    DDNNAME= 188
    IO= 189
    IO=1 189
    IO=IO= 189
    JOBNAME= 189
    PI 190
    PI= 190
    SIO 189
    SIO= 189
    SIO=IO 189
    SIO=IO= 189
    START= 188
    STOP= 188
    SVC 189
    SVC= 189
    SYS 189
    SYS= 189
    TCB= 189
    USR= 190

EDIT function
  control statement format 188
  defaults 191-192
  error recovery 180
  examples 215-217
  JCL 179-182
  output 207
  output space requirements 194-195
  parameters 188-190
  storage requirements 179
  EDIT parameter defaults 191

336 Service Aids (Release 21)
EDIT parameter priorities 191
editing GTF trace data 187-191
from buffers in a dump 215
from external trace data set 216
EID
as field in GTF output 70-71
extracted by IMDEDIT macro 322,323
END control statement
used in PRDMP 185
environment records
as input to EREP0 31
ER= parameter
of PRDMP EXEC statement 180
EREP0 service aid 25
capabilities 29,30
control statements 32-35
examples 36-40
how to execute 32,36-40
input 31
JCL 32
output 41-49
ESD definition 132
event identifier
(see EID)
examples
DIP00 23,24
EREP0 36-40
GTF 68
LIST 119-123
MDMAP 146-151
OSJQD 169
PRDMP 208-216
PTFLE 236-237
SADMP 260,261,263
SPZAP 300-308
EXEC statement parameters
used in GTF cataloged procedure 59
used in MDMAP 140-142
used in PRDMP 179-180
used in PTFLE 238
EXIT= parameter
(see EDIT control statement)
exit routines
function 313
sample exit routine 321-323
EXT parameter
(see EDIT control statement)
EXT trace option in GTF 61
FID
as field in GTF output 70,77,78
as parameter in GTRACE macro 67,68
used in naming format appendage 67
FID= parameter
(see GTRACE macro instruction)
format appendages
function 313
sample format appendage 324
FORMAT control statement 186
used in PRDMP
example 212
format 186
function 186
format control statements in
PRDMP 185-190
EDIT 187-190
FORMAT 186
LPAMAP 185
PRINT 186
QCFTRACE 185
TSO 187
FREMenn parameter
of PRDMP EXEC statement 179
function control statements
in PRDMP 183-185
CVT= 184
END 185
GO 185
NEWDUMP 184
NEWTAPE 184
ONGO 185
TITLE 185
functions of service aids, summary of 13
F03 parameter
(see PRINT control statement)
GO control statement 185
used in PRDMP
format 185
function 185
use with ONGO control statement 185
GO option
(see GO control statement)
GTF service aid 51
calculating storage requirements 65
error recovery handling 69
output 70-77
control record format 76-77
trace record format 70
recording user data 67-68
coding the GTRACE macro 67
printing user data 67
starting GTF 57-64
cataloged procedure 59
prompting 62
specifying trace options 60-61
START command 57-58
storing trace options in SYS1.PARMLIB 63-64
GTF START command parameters 57
devaddr 57
keyword=option 59
parmvalue 57-58
MODE= 57
TIME= 58
DEBUG= 58
procname.identifier 57
REG= 59
volser 57
GTF trace options 60-62
DSP 62
EXT 61
I0,IOP 61
PI,PIP 61
PCI 62
SIO,SIOP 61
SSM 62
SVC,SVCP 61

Index 337
SYS, SYSM, SYSP 60
TRC 62
USR 62

GTRACE macro instruction in GTF 67
effect on EDIT user programs 67
how to code 67
parameters 67-68

DATA= 67
LNG= 67
ID= 68
FID= 68

high-speed dump
as output of SADMP
how to print 245
how to specify 248, 249, 258-265

ID= parameter
(see GTRACE macro instruction)

IDR
(see CSECT identification record)

IDRDATA control statement
used in SPZAP 292

IFCDIP00
(see DIP00 service aid)

IFCEREPO
(see EREPO service aid)

IHLGTF
(see GTF)

I1MaPTFLE
(see PTFLE service aid)

IMASPBAP
(see SPZAP service aid)

IMBLIST
(see LIST service aid)

IMBMDMAP
(see MDMAP service aid)

IMCJQDMP
(see JQDMP service aid)

IMCOSJQD
(see OSJQD service aid)

IMDPDMP
(see PRDMP service aid)

IMDSADMP
(see SADMP service aid)

IMDSADMP macro instruction
format 254
function 253
parameters 254-258

CONSOLE= 257-258
CPU= 257
IPL= 254
OUTPUT= 255
PROTECT= 256
START= 256
TYPE= 255

input address parameter in JQDMP 92

IO parameter (PRDMP)
(see EDIT control statement)

IOP trace option in GTF 61

IPL= parameter
(see IMDSADMP macro instruction)

JCL
(see job control language statements)

job control language statements
DIP00 22-24
EREPO 32
LIST 119-123
MDMAP 140
OSJQD 159
PRDMP 179-182
PTFLE 238, 239
SADMP
for initializing dump program 261
for retrieving macro instruction 260

SPZAP 288

job queue data set
JQDMP 88-90
OSJQD 162

JOBNAME= parameter
in JQDMP 93
in OSJQD 163
in PRDMP
in PRINT control statement 186
in EDIT control statement 189

JQDMP service aid 79
device identification command 91-93
error handling 95
job queue format 88
operational considerations 102
output 97-101
retrieval 15

LINECNT= parameter
of PRDMP EXEC statement 179

link pack area formatting
(see LPAMAP control statement)

link pack area maps
MDMAP 137, 146
PRDMP 185

LINKPACK parameter
of MDMAP EXEC statement 141

LIST service aid 103
control statements 109-111
LISTIDR 111
LISSLING 109
LISTOBJ 110
examples 119-123
executing LIST 109-111
listing a load module 109-110
listing an object module 110
listing CSECT identification records 111
features 108
JCL 119-123
output 112-118

LISTIDR control statement 111
used in LIST
example 121
format 111
function 111
parameters 111

listing local fixes 111
listing PTFs 111

338 Service Aids (Release 21)
LISTLOAD control statement 109
   used in LIST
      example 119
      format 109
      function 109
      parameters 109
LISTOBJ control statement 110
   used in LIST
      example 120
      format 110
      function 110
      parameters 110
LNG= parameter
   (see GTRACE macro instruction)
load module attribute definitions 135-136
   linkage-editor assigned 135
   programmer assigned 135
load module, definition of 131
load module listing
   output of LIST
      contents of 115-116
      how to obtain 109
load module map
   output of MDMAP
      contents of 143-145
      how to obtain 140
low speed dumps
   output of SADMP
      printing 250
      specifying 255
LPAMAP control statement
   used in PRDMP 185
LPA maps
   (see link pack area maps)

MDMAP service aid 125
   definitions
      of input 131-138
      of output 143-145
   examples 146-151
   EXEC statement parameters 140
      BASIC 141
      DEBUG 141
      hhhhhh 141
      LINKPACK 141
   input types
      load modules 131
      MFT resident reenterable load module area 137
      MVT link pack area 137
      nucleus 138
      JCL 140
   operational considerations 152
   output 143-145
   messages
      (see output comments; output error indicators)
   minimal trace
      how to request 61
   MODE= parameter
      (see GTF START command parameters)
   MODF DD statement
   used in PTFLE 229
   modifying data (SPZAP) 283
module definition
   (see load module definition, object module definition)
   multiple dump processing (PRDMP) 210

N parameter
   of PRDMP EXEC statement 179
NAME control statement 289
   used in SPZAP
      example 300
      format 289
      function 289
      parameters 289
NEWDUMP control statement 184
   used in PRDMP
      example 210
      format 184
      function 184
      parameters
         DDNAME= 184
         FILESEQ= 184
         DUMPSEQ= 184
NEWTAPE control statement 184
   used in PRDMP
      format 184
      function 184
      parameters
      NUCLEUS parameter
         (see PRINT control statement)
nucleus maps 146
   output of MDMAP
      contents of 146
      how to obtain 146

macro expansion messages (SADMP) 272-273
main storage, printing of by PRDMP
   allocated storage 186
   current task 186
   specific addresses 186
   jobnames, by 186
   DAR terminated task 186
   F03 186
main storage requirements
   EDIT user programs 313
   GTF 59
   OSGJQD 160
   PRDMP 179
   PTFLE 227
major control blocks, formatting of
   by PRDMP 186
maps
   link pack area 137,185
   load modules 131
   main storage
      MVT 186
      TSO UMSM 187
      nucleus 138

Index 339
object module, definition of (MDMAP) 131

ONGO control statement 185
used in PRDMP
example 210
format 185
function 185
parameters 185
CVT= 185
EDIT 185
FORMAT 185
LPAMAP 185
PRINT 185
QCBTRACE 185
TSO 185
relationship to GO control statement 185
OSJQD service aid 153
control statements 161,162-163
ALL 163
END 161
JOBNAME= 163
QCR= 162
JCL 159
output 164,168
OUTF DD statement
used in PTFLE 228,229
output address parameter in JQDMP 91
OUTPUT= parameter
(see IMDSADMP macro instruction)
output comments
OSJQD 167
object module listing
output of LIST
contents of 117
how to obtain 110
output
of DIP00 23,24
of EREPO 41-49
of GTF 70-78
of JQDMP 97-101
of LIST 112-118
of MDMAP 143-145,147,149,151
of OSJQD 164-168
of PRDMP 197-207
of PTFLE 233-235
of SADMP 248-252
of SPZAP 296-299
output space requirements (PRDMP) 193-19

P control statement
(see PRINT control statement (PRDMP))
parameters
of control statements
(see DIP00 service aid;
EREPO service aid; LIST
service aid; OSJQD service aid;
PRDMP service aid; and
SPZAP service aid)
of EXEC statement
in GTF cataloged procedure 59
in MDMAP 140-142
in PRDMP 179-180
in PTFLE 227
in GTF START command 57-59
in IMDSADMP macro instruction 254-258

PARM= parameter in EXEC statement
in EREPO 33-35
ACC= 35
CUA= 34
DATE= 34
DEV= 34
HIST= 35
MESS= 34
MOD= 34
M67= 35
PRINT= 35
RESUM= 35
TERMIN= 35
TYPE= 33
VOLID= 34
ZERO= 34
in GTF cataloged procedure 57-58
DEBUG= 57
MODE= 58
TIME= 58
in MDMAP 140-142
base address 141
BASIC 141
DEBUG 141-142
LINKPACK 141
in PRDMP 179-180
ER=x 180
FREEnnn 179
LINECNT 179
n 179
S 179
T 179
in PTFLE 228
&USE 228
PCHF DD statement
used in PTFLE 228,229
PCI trace option in GTF 62
PI parameter
(see EDIT control statement)
PI trace option in GTF 61
PIP trace option in GTF 61
(see also prompting, how to request)
PRDMP cataloged procedure 196
PRDMP service aid 171
cataloged procedure 196
control statements 183
CVT= 184
END 185
FORMAT 186
GO 185
LPAMAP 185
NEWDUMP 184
NEWSAPPE 184
ONSO 185
PRINT 186
QCBTRACE 185
TITLE 185
TSO 187
EDIT function 187-191
control statement 187-191
defaults 191
error recovery 180
examples 215-216
JCL 179-182
output 207
storage requirements 179

340 Service Aids (Release 21)
SSM trace option in GTF 62
START= parameter
in PRDMP
(see EDIT control statement)
in SADMP 256
STOP= parameter
(see EDIT control statement)
storage requirements
(see main storage requirements)
STORAGE= parameter
(see PRINT control statement)
SVC parameter
(see EDIT control statement)
SVC trace option in GTF 61
SVCP trace option in GTF 61
(see also prompting, how to request)
SWAP data sets, how to print 213
SYS parameter
(see EDIT control statement)
SYS trace option in GTF 60
SYSABEND DD statement
used in MDMAP 140,142
used in SPZAP 288
SYSIN DD statement
used in PRDMP 181
used in SPZAP 288
SYSLIB DD statement
used in SPZAP 288
SYSLIB trace option in GTF
function 61
How to request 60
SYSGUT space, allocation of by
PRDMP 193-195
SYSP trace option in GTF 61
(see also prompting, how to request)
SYSPRINT DD statement
used in PRDMP 181
used in SPZAP 288
system events (GTF) 56
SYSTEM= parameter
(see TSO control statement)
SYSSDUMP DD statement
used by MDMAP 140,142
SYSUT1 DD statement
used in PRDMP 182
used in PTFLE 228
SYSUT2 DD statement
used in PRDMP 182
used in PTFLE 228
SYSSWAPmn DD statement
used in PRDMP 181
SYS1.DUMP data set
as input to PRDMP
printing the dump data set 175
clearing the dump data set 209
SYS1.LOGREC data set
changing space allocation 23
dumping 29
initializing 23
processing selected records 29
accumulating 29,36-40
editing and writing 29,36-37
summarizing 30
T parameter
of PRDMP EXEC statement 179
TAPE DD statement
used in PRDMP 180
TIME= parameter
(see GTF START command parameters)
timestamp
how to request 58
TITLE control statement 185
used in PRDMP
format 185
function 185
title, how to specify
(see dump title, how to specify)
trace options
(see GTF trace options)
tracing with prompting 62
tracing without prompting 60-62
TRC trace option in GTF 62
TSO control statement 187
used in PRDMP
format 187
function 187
parameters
SYSTEM= 187
USER= 187
TSO dumps, how to print 187
TYPE= parameter
(see IMDSADMP macro instruction)
TYPE=HI option 255
TYPE=LO option 255
user programs 309
error handling 317-318
exit routines 313,321-323
interfaces with EDIT 314-318
parameter list 314
return codes 317
USER= parameter
(see TSO control statement)
USR= parameter
(see EDIT control statement)
USR trace option in GTF 62
(see also GTRACE macro)
VER control statement
(see VERIFY control statement)
VERIFY control statement 290-291
used in SPZAP
example 303
format 290
function 290
parameters 290-291
work data set, use of in PRDMP 182
READER'S COMMENT FORM

IBM System/360 Operating System: Service Aids

Order No. GC28-6719-2

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

• Please indicate your occupation: ________________________________

• How did you use this publication?
   □ Frequently for reference in my work.
   □ As an introduction to the subject.
   □ As a textbook in a course.
   □ For specific information on one or two subjects.

• Comments (Please include page numbers and give examples.):

• Thank you for your comments. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

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)
READER'S COMMENT FORM

IBM System/360 Operating System: Service Aids

Order No. GC28-6719-2

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

• Please indicate your occupation: ___________________________________________

• How did you use this publication?
  □ Frequently for reference in my work.
  □ As an introduction to the subject.
  □ As a textbook in a course.
  □ For specific information on one or two subjects.

• Comments (Please include page numbers and give examples.):

• Thank you for your comments. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.