z/OS
2.5

MVS Diagnosis: Tools and Service Aids

IBM

GA32-0905-50

Note

Before using this information and the product it supports, read the information in "Notices" on page 691.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-06-26

Contents

Figures

Tables

About this information

This information covers the tools and service aids that IBM® provides for use in diagnosing MVS™ problems.

Chapter 1, "Selecting tools and service aids," on page 1 contains a guide on how to select the appropriate tool or service aid for your purposes. It also provides an overview of all the tools and service aids available,

Each subsequent chapter covers one of the tools or service aids. While topics vary, the following topics are typically covered for each tool or service aid:

- Customizing and planning information
- Starting and stopping the tool or service aid
- Receiving, formatting, and reading the output from the tool or service aid.

At the beginning of each chapter, there is a short editorial style comment that is intended to characterize the tool or service aid that is covered in the chapter.

Who should use this information

This information is for anyone who diagnoses software problems that occur on the operating system. This person is usually a system programmer for the installation. This information is also for application programmers who are testing their programs.

This information assumes that the reader:

- Understands basic system concepts and the use of system services
- Codes in Assembler language, and reads Assembler and linkage editor output
- Codes JCL statements for batch jobs and cataloged procedures
- Understands the commonly used diagnostic tasks and aids, such as message logs, dumps, and the interactive problem control system (IPCS)
- Understands how to search the problem reporting databases
- Understands the techniques for reporting problems to IBM

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to IBM Documentation (www.ibm.com/docs/en/zos).

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxxi.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS MVS Diagnosis: Tools and Service Aids, GA32-0905-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 5 (V2R5)

The following content is new, changed, or no longer included in V2R5.

New

The following content is new.

May 2023 refresh

- Support for Validated Boot for z/OS is added. (APAR OA63507)
  - SADMP support for list-directed IPL is added. See “Syntax of the AMDSADMP macro” on page 68 and “Procedure A: Initialize and run stand-alone dump” on page 96.
  - Chapter 25, “IEAVBPRT: Validated Boot for z/OS print utility,” on page 683 is added.
  - Chapter 18, “IEWSIGN: Sign, unsign, and report load modules ,” on page 627 (APAR OA63377)

Prior to May 2023 refresh

- None

Changed

The following content is changed.

May 2023 refresh

- “Setting up a cataloged procedure” on page 214 statement “EXEC” on page 214 now has parameter “NOWRAP|NW” on page 215.
- “Obtaining an SVC dump” on page 16 is updated for when you specify the AUXMGMT=ON parameter. (APAR OA51098)

March 2023 refresh

- The description of the MODE parameter is updated in “Setting up a cataloged procedure” on page 214.

Prior to March 2023 refresh

- For APAR OA56723, the trace record format and field descriptions are updated in “SYNS and SYNE trace records” on page 283.
- “Initializing the logrec data set” on page 518 is updated for clarification.
- The Comprehensive Trace Record Format is updated in “PGM and PI trace records” on page 268.
- The record format is updated in “PI comprehensive trace record” on page 308.
- The SYSOMVS component trace section is updated because both the minimum and maximum buffer size were changed. For more information, see “SYSOMVS component trace” on page 450.

Deleted

The following content was deleted.

- None

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 4

The following changes are made for z/OS Version 2 Release 4 (V2R4).

New

The following new information is added in this publication:

January 2021 refresh

- Chapter 24, "Data Privacy for Diagnostics (DPfD)," on page 681 is updated to include the ANALYZER function via APAR OA58114.

September 2020 refresh

- "SSRV trace entries" on page 187 is updated to support the IARV64 CHANGEATTRIBUTE request via APAR OA58289.

July 2020 refresh

- For the "BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162, the descriptions of pc-addr- and pr-addr- were updated.

June 2020 refresh

- SSRV IDs 1000 and 1001 are added in "SSRV trace entries" on page 187.
- For APAR OA57889, the RUCSAFLT event trace option for SYSRSM component trace is added in "OPTIONS parameter" on page 474.

Prior to June 2020 refresh

- New component trace event options are added for SAF in SYSRSM. See "OPTIONS parameter" on page 474.
- New component trace SYSIEAVX is added. See "SYSIEAVX component trace" on page 415.
- A new INORIGIN flag is added to the GETSTOR GETSHARED Request flags section in "SSRV trace entries" on page 187.
- For APAR OA53860, in Chapter 11, "The generic tracker facility," on page 319, a new section, "JES2 PUT-UPDATE tracking" on page 346, is added.
- "SYSjes2 component trace" on page 435 is updated to now include sublevel traces SYSOUT API (SAPI) request trace, checkpoint activity trace, and QGET processing trace.
- New component trace SYSGLZ added see "SYSGLZ component trace" on page 405.

Changed

The following information is changed in this publication:

June 2021 refresh

- The FTPCMDS keyword is updated. See "JCL statements for PDUU" on page 646.

September 2020 refresh

- The table summarizing information for requesting a SYSIEAVX component trace is updated. See "SYSIEAVX component trace" on page 415.

July 2020 refresh

- The unformatted GFS Trace Output is updated to add bit settings. See: "Unformatted GFS trace output" on page 514.

June 2020 refresh

- The description of HTTPS_LOCALIPADDR is updated in "JCL statements for PDUU" on page 646.
- The description of TARGET_DSN is updated in "JCL statements for PDUU" on page 646.

Prior to June 2020 refresh

- Examples are updated in Chapter 20, "AMAPDUPL: Problem Documentation Upload Utility," on page 645.

Summary of changes for z/OS MVS Diagnosis: Tools and Service Aids for Version 2 Release 3

The following changes are made for z/OS Version 2 Release 3 (V2R3).

New and changed

- For APAR OA55959, Chapter 20, "AMAPDUPL: Problem Documentation Upload Utility," on page 645 has been updated.
- For APAR OA54807, in Chapter 8, "System trace," on page 153, information about ssid 0151 has been added to "SSRV trace entries" on page 187.
- For APAR OA54086, the IBM z/OS Problem Documentation Upload Utility (PDUU) is updated to allow AMAPDUPL to accept a CASE number in place of a PMR number. See "JCL statements for PDUU" on page 646.
- Updates for APAR OA50653: z/OS zHyperLinks Support in the following sections:
  - System trace: "Summary of system trace entry identifiers" on page 156 and "SYNS and SYNE trace entries" on page 200.
  - Generalized trace facility (GTF) trace:
    - "GTF trace options" on page 227
    - "Prompting keywords" on page 231
    - "Reading GTF output" on page 241
    - "Trace record identifiers" on page 243
    - "SYNS and SYNE trace records" on page 283
    - "SYNCH I/O trace records" on page 285
- Updates to "Stopping GTF" on page 225.
- The IOSQTim field has been added to the EOS, IO, IOCS, and PCI records in "EOS, INTG, IO, IOCS, and PCI trace records" on page 253.
- A PRECHECK option was added to the PARM parameter for SPZAP to check the SYSIN input for errors before any updates are made to the target dataset. For more information, see "Running SPZAP" on page 606
- Chapter 11, "The generic tracker facility," on page 319 was updated to include:
  - "JES2 control statement tracking" on page 344.
  - "TSO/E tracking" on page 347.

  Updates were made to:

  - "MVS Allocation tracking" on page 346.
  - "SDSF tracking" on page 347.

- The syntax and examples have been updated in “Using the AMDSADDD utility” on page 79.
- The CP column of the system trace table was changed from 2 characters to 4 characters. For more information, see Chapter 8, “System trace,” on page 153.

Chapter 1. Selecting tools and service aids

This topic introduces the tools and service aids that MVS provides for diagnosis. For the purposes of this document, **tools** includes dumps and traces, while **service aids** include the other facilities provided for diagnosis. For example:

- SVC dump and system trace are tools
- Logrec data set and AMBLIST are service aids.

There are major two topics:

- "How do I know which tool or service aid to select?" on page 1 - This topic lists problem types and matches them with the appropriate service aid or the appropriate tool. Use this topic to select the tool or service aid you need for a particular problem.
- "What tools and service aids are available?" on page 2 - This topic describes each tools and service aids, including when to use it for diagnosis. Use this topic when you need an overview of tools and service aids available or to find the appropriate time to use a particular tool or service aid.

How do I know which tool or service aid to select?

This topic contains tables that provide criterion for selecting a tool or service aid, depending on the problem or need. The tables show the problem or need, the corresponding tool or service aid, and the topic or document that covers it in complete detail. (Most of the detailed information on tools and service aids is in this document.) Use these tables to quickly find a tool or service aid.

Table 1 on page 1 provides guidance on how to select the type of dump to use for a specific problem.

Table 1. Selecting a dump

What is the problem or need?	Type of dump to use
Testing of an authorized program or a problem program while it is running, especially for 64-bit applications	Transaction dump (see Chapter 3, "Transaction dump," on page 49)
Testing of a problem program while it is running	SNAP dump (see Chapter 6, "SNAP dump," on page 141)
Abnormal end of an authorized program or a problem program	ABEND dump (see Chapter 5, "ABEND dump," on page 121)
System problem when the system continues processing	SVC dump (see Chapter 2, "SVC dump," on page 7)
System problem when the system stops processing or is stopped by the operator because of slowdown or looping	Stand-alone dump (see Chapter 4, "Stand-alone dump," on page 59)

Table 2 on page 1 provides guidance on how to select the type of trace to use for a specific problem.

Table 2. Selecting a trace

What is the problem or need?	Type of trace to use
System problem: diagnosis requires checking of component events	Component trace (see Chapter 12, "Component trace," on page 349)
System problem: diagnosis requires detailed checking of one or two system events	Generalized trace facility (GTF) trace (see Chapter 10, "The Generalized Trace Facility (GTF)," on page 211)
System or authorized program problem: diagnosis requires the messages related to a dump	Master trace (see Chapter 9, "Master trace," on page 205)

Table 2. Selecting a trace (continued)	
What is the problem or need?	**Type of trace to use**
System problem: diagnosis requires checking many system events	System trace (see Chapter 8, “System trace,” on page 153)
System or problem program: diagnosis requires information about allocation of virtual storage.	GETMAIN, FREEMAIN, STORAGE (GFS) trace (see Chapter 14, “GETMAIN, FREEMAIN, STORAGE (GFS) trace,” on page 511)

Table 3 on page 2 provides guidance on how to select the service aid to use for a specific problem.

Table 3. Selecting a service aid	
What is the problem or need?	**Type of service aid to use**
System or hardware problem: need a starting point for diagnosis or when diagnosis requires an overview of system and hardware events in chronological order.	Logrec data set (see Chapter 15, “Recording logrec error records,” on page 517)
Information about the content of load modules and program objects or problem with modules on system.	AMBLIST (see Chapter 16, “AMBLIST: Map load modules and program objects,” on page 541)
Diagnosis requires dynamic change to a program, such as fixing program errors, inserting a SLIP trap match, or altering a program to start component trace.	SPZAP (see Chapter 17, “SPZAP: Modify data in programs and VTOCs,” on page 595)
Need to pack the diagnostic materials for transmission to another site, and create similar data sets at the receiving site.	AMATERSE (see Chapter 19, “AMATERSE: Pack and unpack a data set,” on page 639)
Need to eliminate duplicate or unneeded dumps.	DAE (see Chapter 21, “Dump suppression,” on page 661)
Diagnosis requires a trap to catch problem data while a program is running.	SLIP (see *z/OS MVS System Commands*)
Diagnosis requires formatted output of problem data, such as a dump or trace.	IPCS (see *z/OS MVS IPCS User's Guide*)
Process sensitive data in system dumps	Data Privacy for Diagnostics, DPfD (see Chapter 24, “Data Privacy for Diagnostics (DPfD),” on page 681)

What tools and service aids are available?

This topic provides an overview of the tools and service aids in more detail. The tables that follow contain a brief description of each tool or service aid, some reasons why you would use it, and a reference to the topic or document that covers the tool or service aid in detail. (Most of the detailed information on tools and service aids is in this document.) The tools and service aids are covered in three tables; the dumps, traces, or service aids are listed in order by frequency of use.

Table 4 on page 3 lists each type of dump and gives an overview of how they can be used.

Table 4. Description of dumps

Type of dump	Description
ABEND Dump	Use an ABEND dump when ending an authorized program or a problem program because of an uncorrectable error. These dumps show: • The virtual storage for the program requesting the dump. • System data associated with the program. The system can produce three types of ABEND dumps, SYSABEND, SYSMDUMP, and SYSUDUMP. Each one dumps different areas. Select the dump that gives the areas needed for diagnosing your problem. The IBM supplied defaults for each dump are: • SYSABEND dumps - The largest of the ABEND dumps, containing a summary dump for the failing program plus many other areas useful for analyzing processing in the failing program. • SYSMDUMP dumps - Contains a summary dump for the failing program, plus some system data for the failing task. SYSMDUMP dumps are the only ABEND dumps that you can format with IPCS. • SYSUDUMP dumps - The smallest of the ABEND dumps, containing data and areas only about the failing program. **Reference:** See Chapter 5, "ABEND dump," on page 121 for detailed information.
Transaction Dump (IEATDUMP)	Similar to SNAP dumps, an application can issue an IEATDUMP macro to dump virtual storage areas of interest if the application is running. However, the result is an unformatted dump that must be analyzed using IPCS. See Chapter 3, "Transaction dump," on page 49 for details.
SNAP Dump	Use a SNAP dump when testing a problem program. A SNAP dump shows one or more areas of virtual storage that a program, while running, requests the system to dump. A series of SNAP dumps can show an area at different stages in order to picture a program's processing, dumping one or more fields repeatedly to let the programmer check intermediate steps in calculations. SNAP dumps are preformatted, you cannot use IPCS to format them. Note that a SNAP dump is written while a program runs, rather than during abnormal end. **Reference:** See Chapter 6, "SNAP dump," on page 141 for detailed information.
Stand-Alone Dump	Use a stand-alone dump when: • The system stops processing. • The system enters a wait state with or without a wait state code. • The system enters an instruction loop. • The system is processing slowly. These dumps show central storage and some paged-out virtual storage occupied by the system or stand-alone dump program that failed. Stand-alone dumps can be analyzed using IPCS. **Reference:** See Chapter 4, "Stand-alone dump," on page 59 for detailed information.
SVC Dumps	SVC dumps can be used in two different ways: • Most commonly, a system component requests an SVC dump when an unexpected system error occurs, but the system can continue processing. • An authorized program or the operator can also request an SVC dump when they need diagnostic data to solve a problem. SVC dumps contain a summary dump, control blocks and other system code, but the exact areas dumped depend on whether the dump was requested by a macro, command, or SLIP trap. SVC dumps can be analyzed using IPCS. **Reference:** See Chapter 2, "SVC dump," on page 7 for detailed information.

Table 5 on page 4 lists each type of trace and gives an overview of how they can be used.

Table 5. Description of traces

Trace	Description
Component Trace	Use a component trace when you need trace data to report an MVS component problem to the IBM Support Center. Component tracing shows processing within an MVS component. Typically, you might use component tracing while recreating a problem. The installation, with advice from the IBM Support Center, controls which events are traced for a component. **Reference:** See Chapter 12, “Component trace,” on page 349 for detailed information.
GFS Trace	Use GFS trace to collect information about requests for virtual storage through the GETMAIN, FREEMAIN, and STORAGE macro. **Reference:** See Chapter 14, “GETMAIN, FREEMAIN, STORAGE (GFS) trace,” on page 511 for detailed information.
GTF Trace	Use a GTF trace to show system processing through events occurring in the system over time. The installation controls which events are traced. GTF tracing uses more resources and processor time than a system trace. Use GTF when you are familiar enough with the problem to pinpoint the one or two events required to diagnose your system problem. GTF can be run to an external data set as well as a buffer. **Reference:** See Chapter 10, “The Generalized Trace Facility (GTF),” on page 211 for detailed information.
Master Trace	Use the master trace to show the messages most recently issued. Master trace is useful because it provides a log of these messages in a dump. These can be more pertinent to your problem than the messages accompanying the dump itself. **Reference:** See Chapter 9, “Master trace,” on page 205 for detailed information.
System Trace	Use system trace to see system processing through events occurring in the system over time. System tracing is activated at initialization and, typically, runs continuously. It records many system events, with minimal detail about each. The events traced are predetermined, except for branch tracing. This trace uses fewer resources and is faster than a GTF trace. **Reference:** See Chapter 8, “System trace,” on page 153 for detailed information.

Table 6 on page 4 describes the service aids and how they can be used.

Table 6. Description of service aids

Service Aid	Description
AMATERSE	Use the AMATERSE service aid to create a compact image of diagnostic data sets. The compact image helps to use less space while retaining materials and prepare for efficient transmission of materials from one site to another, such as to send the materials to IBM support.
AMBLIST	Use AMBLIST when you need information about the content of load modules and program objects or you have a problem related to the modules on your system. AMBLIST is a program that provides lots of data about modules in the system, such as a listing of the load modules, map of the CSECTs in a load module or program object, list of modifications in a CSECT, map of modules in the LPA (link pack area), and a map of the contents of the DAT-on nucleus. **Reference:** See Chapter 16, “AMBLIST: Map load modules and program objects,” on page 541 for detailed information.

Table 6. Description of service aids (continued)	
Service Aid	**Description**
Common Storage Tracking	Use common storage tracking to collect data about requests to obtain or free storage in CSA, ECSA, SQA, and ESQA. This is useful to identify jobs or address spaces using an excessive amount of common storage or ending without freeing storage. Use RMF or the IPCS VERBEXIT VSMDATA subcommand to display common storage tracking data. **References:** • See *z/OS MVS Initialization and Tuning Guide* for detailed information on requesting common storage tracking. • See the VSM chapter of *z/OS MVS Diagnosis: Reference* for information on the IPCS VERBEXIT VSMDATA subcommand.
DAE	Use dump analysis and elimination (DAE) to eliminate duplicate or unneeded dumps. This can help save system resources and improve system performance. **Reference:** See Chapter 21, “Dump suppression,” on page 661 for detailed information.
IPCS	Use IPCS to format and analyze dumps, traces, and other data. IPCS produces reports that can help in diagnosing a problem. Some dumps, such as SNAP and SYSABEND and SYSUDUMP ABEND dumps, are preformatted, and are not formatted using IPCS. **Reference:** See *z/OS MVS IPCS User's Guide* for detailed information.
Logrec Data Set	Use the logrec data set as a starting point for problem determination. The system records hardware errors, selected software errors, and selected system conditions in the logrec data set. Logrec information gives you an idea of where to look for a problem, supplies symptom data about the failure, and shows the order in which the errors occurred. **Reference:** See Chapter 15, “Recording logrec error records,” on page 517 for detailed information.
SLIP Traps	Use serviceability level indication processing (SLIP) to set a trap to catch problem data. SLIP can intercept program event recording (PER) or error events. When an event that matches a trap occurs, SLIP performs the problem determination action that you specify: • Requesting or suppressing a dump. • Writing a trace or a logrec data set record. • Giving control to a recovery routine. • Putting the system in a wait state. • Issuing system commands **Reference:** See the SLIP command in *z/OS MVS System Commands* for detailed information.
SPZAP	Use the SPZAP service aid to dynamically update and maintain programs and data sets. For problem determination, you can use SPZAP to: • Fix program errors by replacing a few instructions in a load module or member of a partitioned data set (PDS). • Insert an incorrect instruction in a program to force an ABEND or make a SLIP trap work. • Alter instructions in a load module to start component trace. • Replace data directly on a direct access device to reconstruct a volume table of contents (VTOC) or data records that were damaged by an input/output (I/O) error or program error. **Reference:** See Chapter 17, “SPZAP: Modify data in programs and VTOCs,” on page 595 for detailed information.
Process Sensitive data in system dumps	Data Privacy for Diagnostics (DPfD). See Chapter 24, “Data Privacy for Diagnostics (DPfD),” on page 681

Chapter 2. SVC dump

An SVC dump provides a representation of the virtual storage for the system when an error occurs. Typically, a system component requests the dump from a recovery routine when an unexpected error occurs. However, an authorized program or the operator can also request an SVC dump when diagnostic dump data is needed to solve a problem.

An SVC dump comes in the following types, depending on how it was requested. Note that the type of dump requested determines its contents.

- **Asynchronous SVC dump (scheduled SVC dump):**

  The system issues an instruction or the caller uses a combination of parameters on the SVC dump macro invocation. SVC dump captures all of the dump data into a set of data spaces then writes the dump data from the data spaces into a dump data set. The system is available for another SVC dump upon completion of the capture phase of the dump. In an asynchronous SVC dump, the summary dump data is captured first and can be considered more useful for diagnosis.
- **Synchronous SVC dump:**

  The requester's SVC dump macro invocation issues an instruction to obtain the dump under the current task. The system returns control to the requester once the dump data has been captured into a set of data spaces. SVC dump processing then writes the dump data from the data spaces into a dump data set. The system is available for another SVC dump upon completion of the capture phase of the dump. In a synchronous SVC dump, the summary dump data is captured last.

Each SVC dump also contains a summary dump, if requested. Because dumps requested from disabled, locked, or SRB-mode routines cannot be handled by SVC dump immediately, system activity overwrites much useful diagnostic data. The summary dump supplies copies of selected data areas taken at the time of the request. Specifying a summary dump also provides a means of dumping many predefined data areas simply by specifying one option. Summary dump data is dumped using ASID(X'aaaa') SUMDUMP records and ASID(X'aaaa') DSPNAME(dddddddd) SUMDUMP records. The IPCS user has the option of causing storage dumped in these records also to be mapped as ASID(X'aaaa') or ASID(X'aaaa') DSPNAME(dddddddd) storage. Message BLS18160D is displayed during dump initialization when TSO prompting and IPCS confirmation options permit. If the TSO prompting and the IPCS confirmation options don't permit, the additional mapping is performed. Selective display of ASID(X'aaaa') SUMDUMP or ASID(X'aaaa') DSPNAME(dddddddd) SUMDUMP storage might be requested by referring to those address spaces.

This section includes information system programmers need to know about SVC dump and SVC dump processing:

- “Using automatically allocated dump data sets” on page 8
- “Using pre-allocated dump data sets” on page 12
- “Choosing SVC dump data sets” on page 13
- “Obtaining an SVC dump” on page 16
- “Printing, viewing, copying, and clearing a pre-allocated or SYS1.DUMPxx data set” on page 22
- “Contents of SVC dumps” on page 23
- “Analyzing summary SVC dumps” on page 33
- “Analyzing an SVC dump” on page 36

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information any programmer needs to know about programming the SDUMP or SDUMPX macros to obtain an SVC dump:

- Deciding when to request an SVC dump
- Understanding the types of SVC dumps that MVS produces

- Designing an application program to handle a specific type of SVC dump
- Identifying the data set to contain the dump
- Defining the contents of the dump
- Suppressing duplicate SVC dumps using dump analysis and elimination (DAE)

Planning data set management for SVC dumps

SVC dump processing stores data in dump data sets that the system allocates automatically, as needed, or that you pre-allocate manually. IBM recommends the use of automatically allocated dump data sets whenever possible. Only the space required for the dump being written is allocated. The dump is written using a system-determined block size, so write time is reduced. SMS extended attributes, such as compression and striping, can be assigned to further reduce the amount of space required and the time to write.

IBM recommends using extended format sequential data sets as dump data sets for SVC Dumps. For the reasons why, see "Choosing SVC dump data sets" on page 13.

Use pre-allocated dump data sets only as a back up, in case the system is not able to automatically allocate a data set. Otherwise, the dump can become truncated, making error diagnosis difficult.

Using automatically allocated dump data sets

SVC dump processing supports automatic allocation of dump data sets at the time the system writes the dump to DASD. Automatically allocated dumps will be written using the system-determined block size. The dump data sets can be allocated as SMS-managed or non-SMS-managed, depending on the VOLSER or SMS classes defined on the DUMPDS ADD command. When the system captures a dump, it allocates a data set of the correct size from the resources you specify. See "Choosing SVC dump data sets" on page 13 for DFSMS support of extended format sequential data sets. Using Extended Format Sequential data sets, the maximum size of the dump can exceed the size allowed for non-SMS managed data sets.

If automatic allocation fails, preallocated dump data sets are used. If no preallocated SYS1.DUMPnn data sets are available, message IEA793A is issued, and the dump remains in virtual storage. SVC Dump periodically retries both automatic allocation and writing to a preallocated dump dataset until successful or until the captured dump is deleted either by operator intervention or by the expiration of the CHNGDUMP MSGTIME parameter governing message IEA793A. If you set the MSGTIME value to 0, the system will not issue the message, and it deletes the captured dump immediately.

Naming automatically allocated dump data sets

The installation has control of the name of the data sets created by the automatic allocation function, and you can select a name-pattern to allow for dump data set organization according to your needs. The name is determined through an installation-supplied pattern on the DUMPDS command. A set of symbols is available so that you can include the following kinds of information in the names of your automatically allocated dump data sets:

- System name
- Sysplex name
- Job name
- Local and GMT time and date
- Sequence number

You can specify a name-pattern to generate any name acceptable under normal MVS data set name standards. The only requirement is that you include the sequence number symbol to guarantee each automatically allocated dump data set has a unique name.

Using automatic allocation of SVC dump data sets

You can specify the command instructions to enable or disable automatic allocation either in the COMMNDxx parmlib member, to take effect at IPL, or from the operator console at any time after the IPL, to dynamically modify automatic allocation settings. The DUMPDS command provides the following flexibility:

- Activate automatic allocation of dump data sets
- Add or delete allocation resources
- Direct automatic allocation to SMS or non-SMS managed storage
- Deactivate automatic allocation of dump data sets
- Reactivate automatic allocation of dump data sets
- Change the dump data set naming convention

Set up automatic allocation with the following steps:

- Set up allocation authority
- Establish a name pattern for the data sets
- Define resources for storing the data sets
- Activate automatic allocation

After automatic allocation of these SVC dump data sets is active, allocation to a DASD volume is done starting with the first resource allocated via the DUMPDS ADD command. When allocation to that volume is no longer successful, the next resource is then used.

SVC Dump data sets can be SMS-managed or non-SMS-managed. If the DUMPDS ADD command defined SMS classes, then the allocation will first pass these classes to the ACS routines to try to allocate the SVC dump data set as SMS-managed. If this allocation is not successful for any reason, or if no SMS classes are defined, then the data set allocation will use the DASD volumes that were defined on the DUMPDS ADD command, and the SVC Dump data set will be allocated as non-SMS-managed.

SVC Dump data sets allocated as non-SMS-managed must be single volume; they can have multiple extents but they cannot span multiple volumes. Non-SMS-managed DASD does not support striping. SVC Dump data sets allocated as SMS-managed can be multi-volume only if they are allocated as striped data sets. Striping is an attribute that must be defined in the SMS classes. Striping and compression, another SMS attribute, can be used to allocate datasets that are larger than those allowed for a pre-allocated or non-SMS managed dataset.

Note: You must update automatic class selection (ACS) routines to route the intended data set into SMS-management so that it is assigned a storage class.

Setting up allocation authority

To allocate dump data sets automatically, the DUMPSRV address space must have authority to allocate new data sets. Do the following:

1. **Associate the DUMPSRV address space with a user ID.**

   Use the RACF® STARTED general resource class or the RACF Started Procedures Table, ICHRIN03, to associate DUMPSRV with a user id.

2. **Authorize DUMPSRV's user ID to create new dump data sets using the naming convention in the following topic.**

   With the high-level qualifier of SYS1, the data sets are considered *group* data sets. You can assign CREATE group authority to the DUMPSRV user ID within that group.

See *z/OS Security Server RACF System Programmer's Guide* for information about the RACF STARTED general resource class, and the RACF Started Procedures Table. See *z/OS Security Server RACF Security Administrator's Guide* for information on using the RACF STARTED general resource class, and on controlling creation of new data sets.

Establishing a name pattern

Establishing the name pattern for the dump data sets is accomplished by the DUMPDS NAME= command. Names must conform to standard data set naming conventions and are limited to 44 characters, including periods used as delimiters between qualifiers. For a complete description, see *z/OS DFSMS Using Data Sets*. To allow meaningful names for the dump data sets, several symbols are provided that are resolved when the dump data is captured in virtual storage. For a complete list of the symbols you can use, see the explanation of DUMPDS NAME= in *z/OS MVS System Commands*.

When determining the pattern for the dump data set names, consider any automation tools you may have at your installation that work on dump data sets. Also, the automatic allocation function requires you to include the &SEQ. sequence number symbol in your data set name pattern to guarantee unique data set names. If you do not use the sequence number, the system rejects the name pattern with message IEE855I and the previous name pattern remains in effect.

By default, the system uses one of three name patterns. The system typically uses the normal pattern **SYS1.DUMP.D&DATE..T&TIME..&SYSNAME..S&SEQ;**. The system automatically uses S&SYSNAME convention when the system name begins with numeric and is less than 8 characters long. For example:

- **SYS1.DUMP.D&YYMMDD..T&HHMMSS..S&SYSNAME..S&SEQ.** or
- **SYS1.DUMP.D&YYMMDD..T&HHMMSS..S&SYSNAME(2&colon.8)..S&SEQ** when the system name begins with numeric and is 8 characters long.

Figure 1 on page 10 describes the default name pattern.

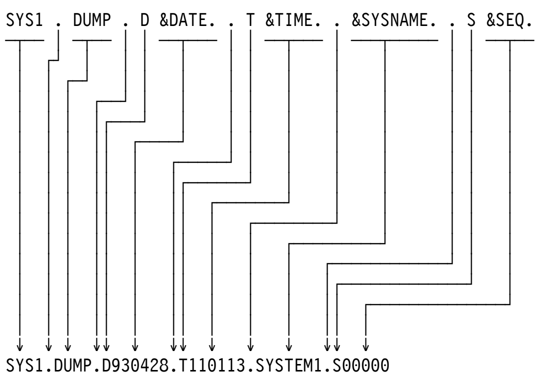


Figure 1. Default name pattern for automatically allocated dump data set

Note: While the default data sets begin with a high-level qualifier of SYS1, this convention is no longer a requirement for data sets named by your installation.

Notice that the symbols are resolved into date, time, and sequence numerics, so they are preceded by an alphabetic character to conform to MVS data set name requirements. Also, the symbol starts with an ampersand (&) and ends with a period (.), resulting in a name pattern that has double periods when a symbol finishes a qualifier. One period ends the symbol, and the second serves as the delimiter between qualifiers of the generated data set name.

Defining resources for dump data sets

If allocation is active, SVC dump data sets can be automatically allocated as soon as resources are defined to store them. If you have not changed the name pattern, then the system default is used. See "Establishing a name pattern" on page 10. You can define dump data set resources using the DUMPDS ADD,VOL=volser (for DASD volumes) and DUMPDS ADD,SMS=class (for SMS classes) commands. You can remove resources using the DUMPDS DEL,VOL=volser and DUMPDS DEL,SMS=class commands. Automatic allocation is directed to SMS classes in preference to DASD volumes.

When automatic allocation is inactive, dumps are written to pre-allocated SYS1.DUMPxx data sets. Deactivating automatic allocation does not result in the loss of resource definitions, however. So, if automatic allocation is reactivated, all the previous resources remain available for receiving automatically allocated dump data sets. Similarly, removing the last allocation resource will not cause automatic allocation to be inactive. Removing the last allocation resource *effectively* ‘turns off’ the function, though, just as if all the defined resources were full. In both cases the system responds with message IEA799I and dumps are written to pre-allocated SYS1.DUMPxx data sets if they exist. Otherwise the dump remains captured until:

- You create a place for it
- The established time limit, as indicated by the CHNGDUMP MSGTIME parameter, expires
- The operator deletes the dump.

Activating automatic allocation

By default, automatic allocation is inactive after IPLing the system. However, you can add to your COMMNDxx parmlib member the DUMPDS NAME= command, any DUMPDS ADD commands, and the DUMPDS ALLOC=ACTIVE command to activate automatic allocation during IPL.

If you have turned off automatic allocation using ALLOC=INACTIVE, reactivate it by entering the DUMPDS ALLOC=ACTIVE operator command.

Verifying dump status

To verify dump status issue the DISPLAY DUMP,STATUS command. For example, after IPLing SYSTEM1 specifying DUMP=NO as a system parameter, and without requesting any dumps or specifying any DUMPDS or CHNGDUMP commands, the output shown in Figure 2 on page 11 would be expected as a result of the DISPLAY DUMP,STATUS command:

```
IEE852I 10.56.03 SYS1.DUMP STATUS
SYS1.DUMP DATA SETS AVAILABLE=000 AND FULL=000
CAPTURED DUMPS=0000, SPACE USED=00000000M, SPACE FREE=00000500M
AUTOMATIC ALLOCATION IS: INACTIVE
   NO SMS CLASSES DEFINED
   NO DASD VOLUMES DEFINED
   NAME=SYS1.DUMP.D&DATE..T&TIME..&SYSNAME..S&SEQ.
      EXAMPLE=SYS1.DUMP.D930324.T105603.SYSTEM1.S00000
```

Figure 2. Example: verifying dump status

Now assume that the following steps are performed to establish the automatic allocation function:

1. Set up your installation data set name pattern using the DUMPDS command:

   ```
   DUMPDS NAME=&SYSNAME..&JOBNAME..Y&YR4.M&MON..D&DAY.T&HR.&MIN..S&SEQ.
   ```

   Note: This step is only required if you are not using the default name pattern as shown in Figure 1 on page 10.
2. Add dump data set resources that can be used by the automatic allocation function:

   ```
   DUMPDS ADD,VOL=(SCRTH1,HSM111)
   DUMPDS ADD,SMS=(DUMPDA)
   ```
3. Activate automatic dump data set allocation using the DUMPDS command:

   ```
   DUMPDS ALLOC=ACTIVE
   ```

Note: These steps can be performed after IPL using the DUMPDS command from an operator console, or early in IPL by putting the commands in the COMMNDxx parmlib member and pointing to the member from the IEASYSxx parmlib member using CMD=xx.

If you use COMMNDxx, you may want to specify DUMP=NO in the IEASYSxx parmlib member to prevent dumps taken during IPL from being written to SYS1.DUMPxx data sets.

After issuing the DUMPDS commands shown in steps “1” on page 11 through “3” on page 11, requesting dump status would be as shown Figure 3 on page 12.

```
SYSTEM1  IEE852I 12.34.18 SYS1.DUMP STATUS 886
SYS1.DUMP DATA SETS AVAILABLE=000 AND FULL=000
CAPTURED DUMPS=0000, SPACE USED=00000000M, SPACE FREE=00000500M
AUTOMATIC ALLOCATION IS: ACTIVE
   AVAILABLE SMS CLASSES: DUMPDA
   AVAILABLE DASD VOLUMES: SCRTH1,HSM111
   NAME=&SYSNAME..&JOBNAME..Y&YR4.M&MON..D&DAY.T&HR.&MIN..S&SEQ.
      EXAMPLE=SYSTEM1.#MASTER#.Y1994M01.D26T1634.S00000
```

Figure 3. Example of dump status

Managing automatically allocated dump data sets

Automatic allocation of dump data sets is managed through the DUMPDS command. Placing appropriate commands into the COMMNDxx parmlib member allows the function to be established at IPL.

The DISPLAY DUMP command can display information about the last 100 data sets that were automatically allocated during the current IPL. Typical dump inventory management should be done using the Sysplex Dump Directory. The System Dump Directory provides access to all of the cataloged and added dump data sets created across system IPLs. Details about using the User and Sysplex Dump Directory can be found in *z/OS MVS IPCS User's Guide*.

The installation must manage the space allocated to dump data sets by limiting the volumes (non-SMS) or the classes (SMS) available for automatic allocation of dump data sets. *z/OS MVS System Commands* contains the syntax of the DUMPDS ADD, DEL, and ALLOC=ACTIVE commands.

For more information about SMS, see *z/OS DFSMSdfp Storage Administration*.

Using pre-allocated dump data sets

Pre-allocated dump data sets should be used as a backup method to automatic allocation. Like the automatically allocated dump data sets, pre-allocated dump data sets will hold SVC dump information for later review and analysis, but have size and performance limitations that automatically allocated dump data sets do not have. This section describes how to set up pre-allocated data sets for SVC dump, including:

- "Allocating SYS1.DUMPxx data sets with secondary extents" on page 12
- "Specifying SYS1.DUMPxx data sets" on page 15
- "Controlling SYS1.DUMPxx data sets" on page 15

Allocating SYS1.DUMPxx data sets with secondary extents

Allocate SYS1.DUMPxx data sets using the following requirements:

- Name the data set SYS1.DUMPxx, where *xx* is decimal 00 through 99.
- Select a device with a track size of 4160 bytes. The system writes the dump in blocked records of 4160 bytes.
- Initialize with an end-of file (EOF) record as the first record. (If you use ISPF 3.2 to allocate SYS1.DUMPxx data sets, the EOF record will be automatically written on the first track.)
- Allocate the data set before requesting a dump. Allocation requirements are:
  - UNIT: A permanently resident volume on a direct access device.
  - DISP: Catalog the data set (CATLG). Do not specify SHR.
  - VOLUME: Place the data set on only one volume. Allocating the dump data set on the same volume as the page data set could cause contention problems during dumping, as pages for the dumped address space are read from the page data set and written to the dump data set.

– SPACE: An installation must consider the size of the page data set that will contain the dump data. The data set must be large enough to hold the amount of data as defined by the MAXSPACE parameter on the CHNGDUMP command, VIO pages, and pageable private area pages.

  SVC dump processing improves service by allowing secondary extents to be specified when large dump data sets are too large for the amount of DASD previously allocated. An installation can protect itself against truncated dumps by specifying secondary extents and by leaving sufficient space on volumes to allow for the expansion of the dump data sets.

  For the SPACE keyword, you can specify CONTIG to make reading and writing the data set faster. Request enough space in the primary extent to hold the smallest SVC dump expected. Request enough space in the secondary extent so that the primary plus the secondary extents can hold the largest SVC dump. The maximum size of a data set is 65,535 tracks. For a 3390 this is 4369 cylinders, and will hold about 2.8 gigabytes of data. The actual size of the dump depends on the dump options in effect when the system writes the dump.

  Estimate the largest dump size as follows:

```
Bytes of SDATA options + bytes in largest region size = Result1
```

```
Result1 * number of address spaces in dump             = Result2
PLPA * 20%                                             = Result3
Bytes of requested data space storage                  = Result4

Result2 + Result3 + Result4                  = Bytes in SVC dump
```

  Where:

  - `Result1, Result2, Result3, Result 4`: Intermediate results
  - `SDATA options`: Described in "Contents of SVC dumps" on page 23
  - PLPA: Pageable link pack area

For the size of the smallest dump, use the default options for the SDUMPX macro. The difference between the largest dump and the smallest dump will be the size of the secondary extent.

For example, to calculate the largest amount of storage required for a 3390 DASD, assume that, from the preceding calculations, the records needed for the SVC dump amount to 43200 kilobytes. There are 11 records per track and 15 tracks per cylinder. To determine the number of cylinders needed to allocate a data set of this size, do the following:

- For 43200 kilobytes of storage, you will need space for 10800 SVC dump records (43200 / 4 kilobytes per record).
- With 11 records per track, you will require 982 (10800 / 11 records) tracks.
- Therefore, the data set would require 66 cylinders (982 / 15 tracks per cylinder) for allocation.

  Note: If you are not receiving the dump data you require, increase the size of the dump data set. You will receive system message IEA911E.

The system writes only one dump in each SYS1.DUMPxx data set. Before the data set can be used for another dump, clear it using the DUMPDS command with the CLEAR keyword.

See *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU* for information about the default dump options of the SDUMPX macro. See *z/OS MVS System Commands* for information about using the DUMPDS command.

Choosing SVC dump data sets

IBM recommends using extended format sequential data sets as dump data sets for SVC Dumps. Extended format sequential data sets:

- Have a greater capacity than sequential data sets

- Support striping
- Support compression

Greater capacity: Some dump data sets are quite large compared with other data sets generated by a system. An extended format sequential data set can hold the largest SVC dumps, as much as 128 gigabytes.

Support for striping: Striping spreads sections, or *stripes*, of a data set across multiple volumes and uses independent paths, if available, to those volumes. The multiple volumes and independent paths accelerate sequential reading and writing of the data set, reducing the time during which dump I/O competes with production I/O.

It is recommended that the number of stripes match the number of volumes you use. This combination will yield the best performance because MVS data management allows random access to any record as though it appeared on a single volume. This is particularly useful during an IPCS analysis of a dump. The savings when loading the data set are real but smaller, the result of reducing the number of times end of volume processing comes into play.

In a striped data set, when the last volume receives a stripe, the next stripes are placed on the first volume, the second volume, the third, and so on to the last volume, then back to the first volume.

If you use more than six dozen stripes, the performance benefit of each additional stripe is much less than the performance benefit of adding the earlier stripes. Keep in mind that this is talking about the original data set definition. You can not add stripes to an existing striped data set. You must plan ahead. The faster processing speeds up moving dump data from relatively expensive data space storage to less expensive DASD.

Support for compression: Compression allows dump data sets to use less DASD space. Before using compression, consider the following:

- Compression and decompression trade off processing cycles for more efficient use of DASD. If hardware compression is not available, the number of processing cycles is significantly higher.

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for SVC dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. SVC dump fully supports placement of dump data sets in cylinder-managed space.

Finding automatically allocated dump data sets

The AUTODSN= parameter of the DISPLAY DUMP,TITLE operator command enables you to list up to 100 of the most recent dump data sets that were automatically allocated during this IPL. No information is preserved about data sets that were automatically allocated before the last 100. As an example, if you wanted to see the titles of the last 5 automatically allocated dump data sets, you would issue:

```
DISPLAY DUMP,TITLE,AUTODSN=5
```

For complete information on the use of the DISPLAY DUMP command, see DISPLAY DUMP in *z/OS MVS System Commands*.

Communication from the system

The system communicates about automatic allocation of dump data sets using two messages:

- IEA611I is issued when a complete or partial dump is taken to an automatically allocated dump dataset. IEA611I is an informational message, it will not be issued highlighted.

- IEA799I is issued once per captured dump when automatic allocation fails; it will not be re-issued as a result of automatic allocation failing for subsequent attempts to allocate the same dump data set unless the reason text is different.

Specifying SYS1.DUMPxx data sets

When planning SYS1.DUMPxx data sets, remember that the data sets frequently contain sensitive data (user or installation confidential information, logon passwords, encryption keys, etc.). Protect these data sets with RACF to limit access to them.

The installation can specify SYS1.DUMPxx data sets in two ways:

- IBM recommends that you use the DUMPDS operator command through the COMMNDxx parmlib member. Use the DUMPDS ADD command within the COMMNDxx parmlib member to ensure that all interaction with the dump data set occurs through the DUMPDS command.

  For example, to specifically add data set SYS1.DUMP05, enter:

  ```
  COM='DUMPDS ADD,DSN=05'
  ```

- During system initialization, in the DUMP parameter in the IEASYSxx parmlib member.

  Specify DUMP=NO in the IEASYSxx parmlib member. Otherwise, all available data sets will be allocated before the COMMNDxx parmlib member is processed.

The data sets are on direct access only. The maximum number of SYS1.DUMPxx data sets an installation can have is 100. The direct access data set must be on a permanently resident volume; that is, the data set must be allocated and cataloged. These dump data sets cannot be shared by more than one system.

All dump data sets should not be on the same pack. A pack should contain enough storage to allow the dump data sets to allocate secondary extent space, if needed.

For more information, see the following references:

- See *z/OS MVS Initialization and Tuning Reference* for information about the IEACMDxx and IEASYSxx parmlib member.
- See *z/OS MVS System Commands* for information about using the DUMPDS command.

Controlling SYS1.DUMPxx data sets

After system initialization, use the following to change and control these data sets:

- Copy the dump from the SYS1.DUMPxx data set to another data set; then clear the SYS1.DUMPxx data set, so that it can be sued for another dump. You can use IPCS to format and view or print the copied dump, as described in the following topic.
- Use the DUMPDS operator command to:
  - Add more SYS1.DUMPxx data sets on direct access for SVC dumps.
  - Delete SYS1.DUMPxx data sets for SVC dumps.
  - Clear a SYS1.DUMPxx data set containing a dump by writing an EOF mark as the first record. An EOF mark as the first record makes the data set available for another dump.

  A reIPL is not necessary when adding, deleting, or clearing a data set with the DUMPDS operator command.
- Use the REPLY command to system message IEA793A to cancel a dump.
- Use a post dump exit routine to copy the dump to another data set. IEAVTSEL is an SVC dump post dump exit name list that lists the module names of installation exit routines to be given control when dump processing ends.

For more information, see the following references:

- See *z/OS MVS Installation Exits* for more information about the IEAVTSEL post dump exit name list.

- See *z/OS MVS IPCS Commands* for the IPCS COPYDUMP subcommand.
- See *z/OS MVS System Commands* for the DUMPDS and REPLY operator commands.

Obtaining an SVC dump

Obtain an SVC dump by:

Using a SDUMP or SDUMPX macro in an authorized program.
Entering the DUMP or SLIP operator command
Setting a SLIP in the IEASLPxx parmlib member.

When z/OS takes an SVC dump, it copies data into the DUMPSRV owned data spaces and high virtual storage. The collection of data can introduce an unusually heavy burden on storage resources. The virtual storage load remains until the dump is written out to a target data set on DASD. The data set can be:

SVC dump data set that is specified on the DCB parameter of the SDUMP or SDUMPX macro.
Pre-allocated SYS1.DUMPxx data set, or automatically allocated dump data set.

Use the DUMPDS operator command to manage the pre-allocated and automatically allocated data sets.

When an SVC dump occurs, if normal auxiliary storage use rises above 30%, the system might experience severe performance problems. The system might also experience an 03C wait state, which indicates that the system ran out of available paging slots. You can use the MAXSPACE and AUXMGMT options on the CHNGDUMP SET, SDUMP command to manage the burden of taking SVC dumps on a system. Using the options might not be sufficient to eliminate the problems that are associated with the restricted auxiliary (paging) storage.

- The MAXSPACE value restricts the virtual storage available to the DUMPSRV address space. When you use MAXSPACE, the installation must tune for the worst case usage of real and auxiliary storage. The following rules apply:

  If the installation does not have a history that can be drawn upon, see "Allocating SYS1.DUMPxx data sets with secondary extents" on page 12 for help in determining a maximum data set size. Use a multiple of the data set size to determine a MAXSPACE value. The installation must predict the size of the largest dump that it can configure.

  The paging resources for the affected systems must also be increased to accommodate the additional load that is represented by the MAXSPACE value. The minimum value for defining the additional auxiliary storage capacity must be three times the MAXSPACE value. Adhering to the MAXSPACE guideline can maintain utilization within 30%.

- When you specify the AUXMGMT=ON parameter, the installation disregards first failure data capture (FFDC) to maintain system availability. New dumps are not allowed when SRM (System Resource Manager) determines that a System storage shortage exists. New dumps are allowed only after the storage shortage has been relieved. Current dump data capture stops when SRM determines that a critical storage shortage exists, resulting in a truncated dump. Note that auxiliary and real storage resources are taken into account by SRM when determining the system storage consumption.

For more information about setting the MAXSPACE or AUXMGMT value, see the CHNGDUMP command in *z/OS MVS System Commands*

Requesting dumps from multiple systems

In a sysplex, you probably need dumps from more than one system to collect all of the problem data. These dumps must be requested at the same time. To request multiple dumps, use the following procedures on any of the systems that might be part of the problem:

- Enter a DUMP command with a REMOTE parameter.
- Issue a SDUMPX macro with a REMOTE parameter.
- Create a SLIP trap in an IEASLPxx parmlib member in the shared SYS1.PARMLIB or in the parmlib on each system.

  - Sometimes you cannot predict which system has the problem. Use a ROUTE operator command to activate the traps on all systems with similar configurations. Each trap must include a REMOTE parameter to dump all the other systems that might be involved.

To help you set the requests, the commands and macro can contain the wildcard characters * and ?. Use wildcard characters when an installation has names that form patterns to the systems in the sysplex and to the jobs for associated work.

For example, use wildcard characters * and ? to specify job names. Use TRANS? for the job names TRANS1, TRANS2, and TRANS3 and TRANS* for TRANS1, TRANS12, and TRANS123.

Issuing a macro for SVC dump

To request an SVC dump iIn an authorized program, use an SDUMP or SDUMPX macro. The system writes the dump in a SYS1.DUMPxx data set or, if specified in the macro, in a user-supplied data set.

For example, to dump the default contents listed in "Contents of SVC dumps" on page 23 to a SYS1.DUMPxx data set, enter the following command:

```
SDUMPX
```

If the dump is written to a user-supplied SVC dump data set, the program provides a data control block (DCB) for the data set, opens the DCB before issuing the SDUMP or SDUMPX macro, and closes the DCB after the dump is written. For a synchronous dump, the close should occur when the system returns control to the requester. For a scheduled dump, the close should occur when the ECB is posted or when the SRB is scheduled.

As another example, to write a synchronous dump to a data set whose DCB address is in register 3, you would specify the following command:

```
SDUMPX DCB=(3)
```

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about requesting a scheduled SVC dump and a synchronous SVC dump.

Operator activities

From a console with master authority, the operator can enter either of the following commands:

- DUMP operator command.

  The following DUMP operator command will write an SVC dump:

  ```
  DUMP COMM=(MYDUMP1 5-9-88)
  ```

  - To a SYS1.DUMPxx data set
  - With a dump title of "MYDUMP1 5-9-88"
  - With the default contents listed in "Contents of SVC dumps" on page 23
  - For a job named MYJOB1

  The system will respond with the message:

  ```
  * 23 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
  ```

  Ask the operator to reply:

  ```
  REPLY 23,JOBNAME=MYJOB1
  ```

  Note that if the operator replies `REPLY 23,U` to IEE094D, the system dumps the current address space, which is the master scheduler address space. The operator must use an ASID, JOBNAME, or TSONAME parameter in the reply to obtain other dumps.

  Use the DUMPDS command to produce a scheduled SVC dump.

- SLIP operator command with an ACTION option of STDUMP, SVCD, SYNCSVCD, or TRDUMP.

  For example, the following SLIP operator command will write an SVC dump:

  – To a SYS1.DUMPxx data set
  – When a program check interruption occurs in a job named MYJOB1
  – With the default contents shown in "Contents of SVC dumps" on page 23

```
SLIP SET,ACTION=SVCD,ERRTYP=PROG,JOBNAME=MYJOB1,END
```

  The SLIP command produces a scheduled SVC dump.

Operator command in an IEASLPxx parmlib member

The installation can also place SLIP operator commands in IEASLPxx parmlib members to produce an SVC dump. When a command is needed, the operator dynamically sets the IEASLPxx member containing the needed SLIP command. The installation can place SLIP commands that request different types of SLIP traps in different IEASLPxx members.

See *z/OS MVS System Commands* for details about the DUMP and SLIP operator commands. See *z/OS MVS Initialization and Tuning Reference* for information aoutt the IEASLPxx member.

Operator command in an IEADMCxx parmlib member

IEADMCxx enables you to supply DUMP command parameters through a parmlib member. IEADMCxx enables the operator to specify the collection of dump data by issuing a DUMP command, indicating the name of the parmlib member and any symbolic substitution variables.

Since z/OS Release 2, a number of sample DUMP command parmlib members are delivered in SYS1.SAMPLIB. Modifications to, for example, system names, address space names, and so on, are potentially required to deal with each installation's specific requirements. In some cases, substitution variables are supplied to provide an example of how an installation could use them.

Note: The length of the sample substitution variable name might not suffice for the values that are actually used at a particular site. For example, &job in IEADMCAS can only accommodate job names that are 4 characters or less. To accommodate up to eight characters, change the variable to something like &thisjob.

Therefore, to take advantage of these members, save a modified copy into a data set in your parmlib concatenation.

Table 7 on page 18 summarizes the sample dump commands that z/OS supplies in SYS1.SAMPLIB.

Table 7. Sample operator DUMP command members in SYS1.SAMPLIB

Member Name	Suspected Problem Area	Areas Dumped	Symbolics Used	Remote Option Used
IEADMCAR	APPC	APPC transaction environment, including RRS		
IEADMCAS	Shared Tape	Allocation Autoswitch and XCF, with affected job	&job	Y
IEADMCCA	Catalog	Catalog address space and associated areas		
IEADMCCN	Console	CONSOLE address space and its data spaces		
IEADMCCP	CP/SM	CICSplex SM environment on all systems in the sysplex. This includes the CAS, CMAS and EYU address spaces.		Y
IEADMCD2	DB2® distributed transactions	DB2/RRS environment		Y

Table 7. Sample operator DUMP command members in SYS1.SAMPLIB (continued)				
Member Name	**Suspected Problem Area**	**Areas Dumped**	**Symbolics Used**	**Remote Option Used**
IEADMCJ2	JES2	JES2/XCF environment on current and specified system	&SYSTM	Y
IEADMCLC	Logger/CICS	System Logger, RLS and CICS®		
IEADMCLG	Logger/GRS	System Logger and GRS on all systems in the sysplex		Y
IEADMCLS	General Logger Problem	Logger, XCF, ALLOC, CATALOG, GRS, DFHSM, and SMS along with specified structure, on all systems in the sysplex.	&STRNAME &STRNAME2	Y
IEADMCLX	Logger/XCF	System Logger and XCF on all systems in the sysplex		Y
IEADMCOE	OMVS	OMVS		Y
IEADMCRL	RRS	RRS and the System Logger on all systems in the sysplex		Y
IEADMCRR	RRS	RRS and its data spaces		
IEADMCSQ	IMS	IMS Shared Queues environment (IMS Control region, CL/I SAS Region, DBRC Region, and all of the CQS address spaces connected to the shared queues		Y
IEADMCTA	TCP/IP	TCP/IP, along with the specified application	&tcp &appl	
IEADMCTC	TCP/IP	TCP/IP, along with the Comm Server address space	&tcp	
IEADMCTI	TCP/IP	TCP/IP and its data space	&tcp	
IEADMCTO	TCP/IP	TCP/IP and OMVS	&tcp	
IEADMCVC	Comm Server	VTAM® and TCP/IP, with the TCPIP and VTAM data spaces	&tcp &net	
IEADMCVG	VTAM GR	VTAM Generic Resources environment, with its CF structure		Y
IEADMCVT	Comm Server	VTAM and TCP/IP (address spaces only)	&tcp &net	
IEADMCVV	VTAM	VTAM and the VIT data space	&net	
IEADMCWL	WLM	WLM on all systems in the sysplex		Y
IEADMCWS	Web server	HTTP web server with OMVS		
IEADMCWT	Web server	HTTP web server and TCP/IP		
IEADMCXI	IRLM	XCF and IRLM		
IEADMCX1	IRLM	XCF and IRLM on all systems in the sysplex		Y
IEADMCZM	CEA	CEA		

Note:

1. When specifying parmlib members containing symbolic parameters, you must specify the symbolic and substitution value using the SYMDEF keyword.
2. The dump command indicated by each row with a “Y” in the “Remote Option Used” column results in a multi-system dump.

Making a dump data set available

An SVC dump is taken to an SVC dump data set, either specified on the DCB parameter of the SDUMP or SDUMPX macro, available as SYS1.DUMPxx, or automatically allocated. SVC dump processing issues message IEA793A when the dump has been captured but there are no available dump data sets. When a SYS1.DUMPxx data set is not available, the operator has the option either of deleting the captured dump by replying D or making another dump data set available to SVC dump processing. To make another dump data set available, the operator uses the DUMPDS command.

For example, you can use a DUMPDS command to make a dump data set available to SVC dump.

- System message:

```
   * 16 IEA793A NO SVC DUMP DATA SETS AVAILABLE FOR DUMPID=dumpid FOR JOB
(*MASTER*).
   * 16 IEA793A USE THE DUMPDS COMMAND OR REPLY D TO DELETE THE CAPTURED DUMP
```

- Operator reply:

```
DUMPDS ADD,DSN=02
```

See *z/OS MVS System Commands* for information about the DUMPDS command. See *z/OS MVS System Messages, Vol 6 (GOS-IEA)* for information about message IEA793A.

Determining current SVC dump options and status

An operator can determine the current dump options and the SYS1.DUMPxx data sets that contain SVC dumps.

Dump mode and options

Use a DISPLAY DUMP operator command to get the dump mode and options in effect for SVC dumps and SYSABEND, SYSMDUMP, and SYSUDUMP dumps. The system displays the mode and options in message IEE857I.

For example, to determine the mode and options, enter:

```
DISPLAY DUMP,OPTIONS
```

If the options listed are not the ones desired, have the operator use a CHNGDUMP operator command to change them.

See *z/OS MVS System Commands* for the DISPLAY for more information about dump modes. See *z/OS MVS System Messages, Vol 7 (IEB-IEE)* for more information about IEE857I.

Status of SYS1.DUMPxx data sets

Use a DISPLAY DUMP operator command to get the status of all defined SYS1.DUMPxx data sets on direct access. The system displays the status in message IEE852I or IEE856I. The message indicates the full and available data sets.

For example, to determine the status of SYS1.DUMPxx data sets, enter:

```
DISPLAY DUMP,STATUS
```

See *z/OS MVS System Commands* for information aout the CHNGDUMP and DISPLAY commands. For a description of these messages, see *MVS System Messages*.

Finding SVC dumps

An operator can search the current SYS1.DUMPxx data sets for the SVC dump for a particular problem. To select the dump, use the title and time or use the dump symptoms. The operator can also find a dump that has been captured in virtual storage but has not been written to a data set.

Title and time of SVC dump(s)

Use one of the following to get the titles and times for SVC dumps:

- A DISPLAY DUMP operator command. The system displays the titles and times in message IEE853I.

  You can use the DISPLAY command to find title and time information.

  - To see the titles and times for the dumps in SYS1.DUMP08 and SYS1.DUMP23, without displaying any automatically allocated dump data sets, enter:

    ```
    DISPLAY DUMP,TITLE,DSN=(08,23)
    ```

  - To display the titles of the most recently automatically allocated dump data set and all pre-allocated dump data sets, enter:

    ```
    DISPLAY DUMP,TITLE
    ```

    or:

    ```
    DISPLAY DUMP,TITLE,DSN=ALL
    ```

  - To display the titles of the last 5 most recently allocated dump data sets, enter:

    ```
    DISPLAY DUMP,TITLE,AUTODSN=5
    ```

  - To see the dump titles for all captured dumps, enter:

    ```
    DISPLAY DUMP,TITLE,DUMPID=ALL
    ```

- An IPCS SYSDSCAN command entered at a terminal by a TSO/E user. IPCS displays the titles and times at the terminal.

  For example, to use IPCS to find the dump titles and times for the dumps in SYS1.DUMP08 and SYS1.DUMP23, enter the following IPCS command:

  ```
  SYSDSCAN 08
  SYSDSCAN 23
  ```

  See *z/OS MVS IPCS Commands* for information about SYSDSCAN.

If a data set listed in either command is empty or undefined, the system issues a message to tell why the title is not available.

Symptoms from SVC dumps

Use a DISPLAY DUMP operator command to get the symptoms from SVC dumps in SYS1.DUMPxx data sets on direct access or from SVC dumps that have been captured in virtual storage. The system displays the following symptoms in message IEE854I:

- Dump title or a message telling why the title is not available
- Error id consisting of a sequence number, the processor id, the ASID for the failing task, and the time stamp
- System abend code
- User abend code
- Reason code
- Module name
- Failing CSECT name
- Program status word (PSW) at the time of the error
- Interrupt length code in the system diagnostic work area (SDWA)
- Interrupt code in the SDWA

- Translation exception address in the SDWA
- Address of the failing program in the SDWA
- Address of the recovery routine in the SDWA
- Registers at the time of the error saved in the SDWA

For example, you can issue DISPLAY DUMP to view various types of symptoms:

- To see symptoms from the dump in the SYS1.DUMP03 data set without displaying any automatically allocated data sets, enter:

```
DISPLAY DUMP,ERRDATA,DSN=03
```

- To see symptoms from the most recently automatically allocated dump data set and all pre-allocated dump data sets, enter:

```
DISPLAY DUMP,ERRDATA
```

  or:

```
DISPLAY DUMP,ERRDATA,DSN=ALL
```

- To see symptoms from the last 5 most recently allocated dump data sets, enter:

```
DISPLAY DUMP,ERRDATA,AUTODSN=5
```

- To see symptoms from the captured dump identified by DUMPID=005, enter:

```
DISPLAY DUMP,ERRDATA,DUMPID=005
```

Using the DISPLAY DUMP,ERRDATA command (see Figure 4 on page 22), you can retrieve basic information about the dump without having to format the dump or read through the system log. Message IEE854I displays the error data information, including the PSW at the time of the error, the system abend code and reason code, and the module and CSECT involved.

```
d d,errdata
IEE854I 13.01.25 SYS1.DUMP ERRDATA 745
SYS1.DUMP DATA SETS AVAILABLE=001 AND FULL=001
CAPTURED DUMPS=0000, SPACE USED=00000000M, SPACE FREE=00000500M
     DUMP00 TITLE=ABDUMP ERROR,COMPON=ABDUMP,COMPID=5752-SCDMP,
                  ISSUER=IEAVTABD
         DUMP TAKEN TIME=13.01.02 DATE=09/27/1996
         ERRORID=SEQ00010 CPU0000 ASID0010 TIME=13.00.55
         SYSTEM ABEND CODE=0C1  REASON CODE=00000001
         MODULE=IGC0101C CSECT=IEAVTABD
         PSW AT TIME OF ERROR=070C0000 80000000 00000000 023FE0F6
         ILC=2 INT=01
         TRANSLATION EXCEPTION ADDR=008E1094
         ABENDING PROGRAM ADDR=******** RECOVERY ROUTINE=ADRECOV
         GPR 0-3   00000000  7F6EBAE4  023FFFF6  023F2BFF
         GPR 4-7   008FD088  023FEFF7  823FDFF8  7F6EC090
         GPR 8-11  00000048  7F6EC938  7F6EBA68  7F6EBA68
         GPR12-15  7F6EB538  7F6EB538  00000000  00000000
NO DUMP DATA AVAILABLE FOR THE FOLLOWING EMPTY SYS1.DUMP DATA SETS: 01
```

Figure 4. Example: Output from DISPLAY,DUMP ERRDATA command

See *z/OS MVS System Commands* for information about the DISPLAY operator command.

Printing, viewing, copying, and clearing a pre-allocated or SYS1.DUMPxx data set

SVC dumps are unformatted when created. Use IPCS to format a dump and then view it at a terminal or print it.

After the dump has been copied to a permanent data set, use a DUMPDS operator command to clear the data set so that the system can use the data set for another dump. Then use IPCS to view the copy.

You can copy a dump that was written to tape so that you can view the dump through IPCS more efficiently.

For a pre-allocated data set or a SYS1.DUMPxx data set, the JCL shown in Figure 5 on page 23 does the following:

- Uses the SVC dump in the SYS1.DUMP00 data set. The IPCSDUMP DD statement identifies this data set.
- Copies the dump from the SYS1.DUMP00 data set to the data set identified in the DUMPOUT DD statement. To use this example, change the DUMPOUT DD statement to give the DSNAME for the desired location.
- Clears the SYS1.DUMP00 data set so that it can be used for a new dump.
- Deletes the IPCS dump directory in the DELETE(DDIR) statement. This statement uses the USERID of the batch job in the directory identification.
- Allocates the dump directory through the BLSCDDIR statement. The default is volume VSAM01. The example shows VSAM11. Override the default volume with the desired volume.
- Formats the dump using the IPCS subcommands in LIST 0. To use this example, replace the LIST 0 command with the desired IPCS subcommands or a CLIST. See *z/OS MVS IPCS User's Guide* for CLISTs.

```
//IPCSJOB  JOB
//IPCS     EXEC PGM=IKJEFT01,DYNAMNBR=75,REGION=1500K
//SYSPROC  DD   DSN=SYS1.SBLSCLI0,DISP=SHR
//IPCSDUMP DD   DSN=SYS1.DUMP00,DISP=SHR
//DUMPOUT  DD   DSN=GDG.DATA.SET(+1),DISP=SHR
//SYSUDUMP DD   SYSOUT=*
//IPCSTOC  DD   SYSOUT=*
//IPCSPRNT DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
DELETE(DDIR) PURGE CLUSTER
BLSCDDIR VOLUME(VSAM11)
IPCS NOPARM
SETDEF DD(IPCSDUMP) LIST NOCONFIRM
LIST 0
COPYDUMP INFILE(IPCSDUMP) OUTFILE(DUMPOUT) CLEAR NOPRINT NOCONFIRM
END
/*
```

Figure 5. Example: JCL to print, copy, and clear an SVC dump data set

Contents of SVC dumps

Unlike ABEND dumps, SVC dumps do not have a parmlib member that establishes the dump options list at system initialization. The IBM-supplied IEACMD00 parmlib member contains a CHNGDUMP operator command that adds the local system queue area (LSQA) and trace data (TRT) to every SVC dump requested by an SDUMP or SDUMPX macro or a DUMP operator command, but not for SVC dumps requested by SLIP operator commands.

The contents of areas in an SVC dump depend on the dump type:

- Scheduled SVC dump: The current task control block (TCB) and request block (RB) in the dump are for the dump task, rather than for the failing task. For additional address spaces in the dump, the TCB and RB are for the dump task.
- Synchronous SVC dump: The current TCB and RB in the dump are for the failing task.

See *z/OS MVS IPCS Commands* for examples of IPCS output formatted from SVC dumps.

Customizing SVC dump contents

You can customize the contents of an SVC dump to meet the needs of your installation. For example, you might want to add areas to be dumped, reduce the dump size, or dump Hiperspaces. In most cases, you will customize the contents of an SVC dump or summary dump through the SDATA parameter of the SDUMP or SDUMPX macro or through operator commands.

Reducing dump size: To obtain a smaller dump that does not have all the usual defaults, code a NODEFAULTS option in the SDATA parameter of the SDUMP or SDUMPX macro. With the NODEFAULTS option, the dump contains:

- Certain default system areas needed by IPCS for dump analysis
- Areas requested on the SDUMP or SDUMPX macro

Hiperspaces: SVC dumps do not include Hiperspaces. To include Hiperspace data in an SVC dump, you have to write a program to copy data from the Hiperspace into address space storage that is being dumped.

Adding areas: If the dump, as requested, will not contain all the needed areas, see one of the following for ways to add the areas:

- "Customized contents using the SDATA parameter" on page 24
- "Contents of summary dumps in SVC dumps" on page 28
- "Customizing contents through operator commands" on page 31

Customized contents using the SDATA parameter

The IBM-supplied default contents and the contents available through customization are detailed in Table 8 on page 24. The tables show dump contents alphabetically by the parameters that specify the areas in the dumps. Before requesting a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump. The symbols are:

C
: Available on the command that requests the dump

D
: IBM-supplied default contents

M
: Available on the macro that requests the dump

P
: Available in the parmlib member that controls the dump options

X
: Available on the CHNGDUMP operator command that changes the options for the dump type

blank
: No symbol indicates that the area cannot be obtained.

Note: System operator commands and assembler macros use the parameters in the table to specify dump contents.

The order of the symbols in the following table is not important.

Table 8. Customizing SVC dump contents through the SDATA parameter

SDATA Parameter Option	Dump Contents	SVC Dump for:					
		SDUMPX Macro or DUMP Command with SDATA Parameter	DUMP Command without SDATA Parameter	SLIP Command ACTION= SVCD or SYNCSVCD	SLIP Command ACTION= STDUMP	SLIP Command ACTION= TRDUMP	DUMP Command SVCDUMPRGN =YES
ALLNUC	The DAT-on and DAT-off nucleuses	M C X	X	C	C	C	
ALLPSA	Prefixed save area (PSA) for all processors	D M X	D X	D C	C	C	

Table 8. Customizing SVC dump contents through the SDATA parameter (continued)

SDATA Parameter Option	Dump Contents	SVC Dump for:					
		SDUMPX Macro or DUMP Command with SDATA Parameter	**DUMP Command without SDATA Parameter**	**SLIP Command ACTION= SVCD or SYNCSVCD**	**SLIP Command ACTION= STDUMP**	**SLIP Command ACTION= TRDUMP**	**DUMP Command SVCDUMPRGN =YES**
COUPLE	Data on cross-system coupling **Note:** COUPLE cannot be specified on an SDUMP macro. It can, however, be specified on an SDUMPX macro.	M C X		C	C	C	
CSA	Common service area (CSA) (that is, subpools 227, 228, 231, 241) and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETCOMMON,DUMP=LIKECSA` • `IARCP64 COMMON=YES,DUMP=LIKECSA` • `IARST64 COMMON=YES,TYPE=PAGEABLE` **Note:** When CSA is specified without any high CSA parameters, all the CSA storage, including high virtual CSA, is included in the SVC dump.	M C X	D	D C	C	C	
DEFAULTS	Default areas	M X					
GRSQ	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks	M C X	X	C	C	C	
HCAS (HCSAByASID)	High CSA area by ASID	M C X		C	C	C	
HCNO (HCSANoOwner)	High CSA area for which the owner has ended	M C X		C	C	C	
HCSY (HCSASysOwner)	High CSA area that belongs to the SYSTEM	M C X		C	C	C	
IO	Input/output supervisor (IOS) control blocks for the task being dumped: • EXCPD • UCB	D					
LPA	Active link pack area (LPA): module names and contents	M C X	X	D C	C	C	
LSQA	Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255), and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETSTOR,DUMP=LIKELSQA` • `IARCP64 COMMON=NO,DUMP=LIKELSQA` • `IARST64 COMMON=NO`	D M C X	D X	D C	C	C	
NOALL	No ALLPSA	M X	X	C	C	C	
NOALLPSA	No ALLPSA	M X	X	C	D C	D C	

Table 8. Customizing SVC dump contents through the SDATA parameter (continued)							
SDATA Parameter Option	**Dump Contents**	**SVC Dump for:**					
		SDUMPX Macro or DUMP Command with SDATA Parameter	**DUMP Command without SDATA Parameter**	**SLIP Command ACTION= SVCD or SYNCSVCD**	**SLIP Command ACTION= STDUMP**	**SLIP Command ACTION= TRDUMP**	**DUMP Command SVCDUMPRGN =YES**
NODEFAULTS	• Minimum default areas needed for IPCS dump analysis • Areas requested on the SDUMP or SDUMPX macro	M					
NOPSA	No PSA	C					
NOSQA	No SQA	M C X	X	C	D C	D C	
NOSUM	No SUM	M C X	X	C	D C	D C	
NUC	Read/write portion of the control program nucleus (that is, only the non-page -protected areas of the DAT-on nucleus), including: • CVT • LSQA • PSA • SQA	M C X	X	D C	C	C	
PSA	Prefixed save areas (PSA) for the processor at the time of the error or the processor at the time of the dump	D M C X	D X	D C	C	C	
RGN	Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 230, 236, 237, 244, 249, 251 - 255, and virtual storage for 64-bit addressable memory objects created using the following services: • `IARV64 REQUEST=GETSTOR,DUMP=LIKERGN` • `IARV64 REQUEST=GETSTOR,SVCDUMPRGN=YES` • `IARCP64 COMMON=NO,DUMP=LIKERGN` • `IARST64 COMMON=NO` .	M C X	X	D C	C	C	C
SERVERS	Areas added by IEASDUMP. SERVERS exits	M C X					
SQA	System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETCOMMON,DUMP=LIKESQA` • `IARCP64 COMMON=YES,DUMP=LIKESQA` • `IARST64 COMMON=YES,TYPE=FIXED` • `IARST64 COMMON=YES,TYPE=DREF`	D M C X	D X	D C	C	C	
SUM	Summary dump (See "Contents of summary dumps in SVC dumps" on page 28.)	D M C X	D X	D C	C	C	

Table 8. Customizing SVC dump contents through the SDATA parameter (continued)

SDATA Parameter Option	Dump Contents	SVC Dump for:					
		SDUMPX Macro or DUMP Command with SDATA Parameter	DUMP Command without SDATA Parameter	SLIP Command ACTION= SVCD or SYNCSVCD	SLIP Command ACTION= STDUMP	SLIP Command ACTION= TRDUMP	DUMP Command SVCDUMPRGN =YES
SWA	Scheduler work area (SWA) (that is, subpools 236 and 237)	M C X	D X	C	C	C	
TRT	System trace, generalized trace facility (GTF) trace, and master trace, as available	D M C X	D X	D C	D C	D C	
Default system data	Instruction address trace, if available	D	D	D	D	D	
Default system data	Nucleus map and system control blocks, including: • ASCB for each address space being dumped • ASVT • Authoriza- tion table for each address space • CVT, CVT prefix, and secondary CVT (SCVT) • Entry tables for each address space • GDA • JSAB of each address space being dumped • Linkage stack • Linkage table for each address space • PCCA and the PCCA vector table • TOT • TRVT • UCB	D					
Default system data	DFP problem data, if DFP Release 3.1.0 or a later release is installed	D	D	D	D	D	
Default system data	Storage for the task being dumped and program data for all of its subtasks	D	D	D			
Default system data	Storage: 4 kilobytes before and 4 kilobytes after the address in the PSW at the time of the error	D					
Default system data	SUBTASKS: Storage for the task being dumped and program data for all of its subtasks		D	D			

The following table describes how HCSAByASID, HCSANoOwner, and HCSASysOwner affect the CSA storage that is captured in an SVC dump:

Table 9. Effects on the CSA storage captured in an SVC dump

Specified SDATA option or options	CSA storage that is included in the dump
CSA	All above the bar and below the bar CSA storage

Table 9. Effects on the CSA storage captured in an SVC dump (continued)	
Specified SDATA option or options	**CSA storage that is included in the dump**
CSA, HCSAByASID, HCSANoOwner, HCSASysOwner	All below the bar CSA storage, high virtual CSA storage that is owned by the ASIDs that are included in the dump, high virtual CSA storage for which the owner has ended, and high virtual CSA storage that belongs to the SYSTEM. The dump does not include high virtual CSA storage that is owned by the ASIDs that are excluded from the dump.
HCSAByASID, HCSANoOwner, HCSASysOwner	All high virtual CSA storage that is owned by the ASIDs that are included in the dump, high virtual CSA storage for which the owner has ended, and high virtual CSA storage that belongs to the SYSTEM No below the bar CSA storage is included in the dump.
(Neither CSA nor any of the HCSA*xxxx* options)	None of the CSA storage is included in the dump.

Contents of summary dumps in SVC dumps

Request a summary dump for two reasons:

1. The SUM or SUMDUMP parameters request many useful, predefined areas with one parameter.
2. The system does not write dumps immediately for requests from disabled, locked, or SRB-mode programs. Therefore, system activity destroys much needed diagnostic data. When SUM or SUMDUMP is specified, the system saves copies of selected data areas at the time of the request, then includes the areas in the SVC dump when it is written.

Use SDUMP or SDUMPX macro parameters to request different types of summary dumps, as follows:

- **Disabled summary dump**: This summary dump saves data that is subject to rapid and frequent change before returning control to the scheduled dump requester. Because the system is disabled for this dump, the dump includes only data that is paged in or in DREF storage. Specify BRANCH=YES and SUSPEND=NO on an SDUMP or SDUMPX macro to obtain a disabled summary dump.
- **Suspend summary dump**: This summary dump also saves data that is subject to rapid and frequent change before returning control to the scheduled dump requester. This dump, however, can save pageable data. To obtain a suspend summary dump, do the following:
  - For an SDUMP or SDUMPX macro, specify BRANCH=YES and SUSPEND=YES
  - For an SDUMPX macro, specify BRANCH=NO for a scheduled dump with SUMLSTL parameter
- **Enabled summary dump**: This summary dump does not contain volatile system information. The system writes this summary dump before returning control to the dump requester; the summary information is saved for each address space being dumped. To obtain an enabled summary dump, do the following:
  - For an SDUMP or SDUMPX macro, specify BRANCH=NO.
  - For a SLIP operator command, do not specify an SDATA parameter or specify SUM in an SDATA parameter.
  - For a DUMP operator command, do not specify an SDATA parameter or specify SUM in an SDATA parameter. Note that this dump does not contain data that the system creates when it detects a problem; for example, this dump would not contain a system diagnostic work area (SDWA).

In Table 10 on page 29, an S indicates that the area is included in the summary dump for the dump type.

Table 10. Customizing SVC dump contents through summary dumps

Summary dump contents	Disabled	Suspend	Enabled
Address space identifier (ASID) record for the address space of the dump task			S
Control blocks for the failing task, including:			
• For task-mode dump requesters: – Address space control block (ASCB) – Request blocks (RB) – System diagnostic work areas (SDWA) pointed to by the recovery termination management 2 work areas (RTM2WA) associated with the task control block (TCB) – TCB – Extended status block (XSB)	S		
• For service request block (SRB)-mode dump requesters: – ASCB – Suspended SRB save area (SSRB) – SDWA used for dump – XSB		S	
Control blocks for the recovery termination manager (RTM):			
• **RTM2WA** associated with all TCBs in the dumped address space			S
• **RTM2WA** associated with the TCB for the dump requester		S	
Cross memory status record and, if the dump requester held a cross memory local (CML) lock, the address of the ASCB for the address space whose local lock is held	S	S	
Dump header, mapped by AMDDATA For the AMDDATA mapping, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).	S	S	S
Functional recovery routine (FRR) stack for the current processor	S		
Interrupt handler save area (IHSA) for the home address space or, if a CML is held, for the address space whose local lock is held	S	S	
Logical communication area (LCCA) for each active processor In dumps requested by AR-mode callers, the LCCA includes the AR mode control blocks	S	S	
Physical configuration communication area (PCCA) for each active processor	S	S	
Program call link stack elements (PCLINK) stack elements:			
• Pointed to by PSASEL	S		
• Pointed to by the XSB associated with the IHSA in the dump	S	S	
• Pointed to by the SSRB and XSB for the SRB-mode dump requester		S	
• Associated with the suspended unit of work		S	
Prefixed save area (PSA) for each active processor	S	S	

Table 10. Customizing SVC dump contents through summary dumps (continued)

Summary dump contents	Disabled	Suspend	Enabled
Save areas of register contents		S	
SDWA associated with the failure of a system routine	S		
Storage: The storage ranges and ASIDs requested in parameters on the SDUMP or SDUMPX macro	S	S	S
Storage: 4 kilobytes before and 4 kilobytes after: • The address in the program status word (PSW) • All valid unique addresses in the registers saved in the IHSA shown in the dump • All valid unique addresses in the registers saved in the SDWA shown in the dump • Instruction counter values of the external old PSW, program check old PSW, I/O old PSW, and restart old PSW saved in the PSAs of each active processor	S		
Storage: 4 kilobytes before and 4 kilobytes after: • All valid unique addresses in the registers saved in the SDWA shown in the dump • All valid unique addresses in the registers in the dump requester's register save area • All valid unique addresses in the PSWs in all SDWAs shown in the dump		S	
Storage: 4 kilobytes before and 4 kilobytes after: • All valid unique addresses in the PSWs in the RTM2WAs shown in the dump • All valid unique addresses in the registers in the RTM2WAs shown in the dump			S
Storage: When a PSWREGS parameter is specified on the SDUMP or SDUMPX macro, 4 kilobytes before and 4 kilobytes after: • The address in the PSW, if supplied in the PSWREGS parameter • The address in the general purpose registers, if supplied in the PSWREGS parameter The storage dumped is from the primary and secondary address spaces of the program issuing the SDUMP or SDUMPX macro. The control registers, if supplied in the PSWREGS parameter, are used to determine the primary and second address spaces. If access registers are also provided and the PSW indicates AR ASC mode, the access registers will also be used to locate the data.	S	S	S
Supervisor control blocks: • Current linkage stack • Primary address space number (PASN) access list • Work unit access list	S	S	S
Vector Facility control blocks: Global, CPU, and local work/save area vector tables (WSAVTG, WSAVTC, and WSAVTL) and work/save areas pointed to by addresses in the tables	S		
XSB associated with the IHSA in the dump	S	S	

For information about control blocks listed in the above table, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Customizing contents through operator commands

The dump options list for SVC dumps can be customized through a DUMP operator command by all the ways shown in Table 11 on page 31.

Note: The contents of SVC dumps requested by SLIP operator commands are controlled only by the SLIP operator command. They are not affected by the IEACMD00 parmlib member or the CHNGDUMP command.

Nucleus areas in dumps: Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used:

- SVC dump for a SLIP operator command with ACTION=SVCD contains the read/write DAT-on nucleus
- SVC dump for an SDUMP or SDUMPX macro contains the nucleus map and certain control blocks

If no nucleus changes have been made, an installation should obtain one copy of the DAT-off nucleus to use with all dumps. To obtain this nucleus, enter a DUMP operator command with SDATA=ALLNUC and no other SDATA options. The nucleus does not change from one IPL to another, so one dump can be used again and again.

DAT, dynamic address translation, is the hardware feature that enables virtual storage. In the DAT-on part of the nucleus, the addresses are in virtual storage; in the DAT-off part of the nucleus, the addresses are in central storage.

Table 11. Customizing SVC dump contents through operator commands

Customization	Effect	Example
Use SDATA=NODEFAULTS on SDUMP or SDUMPX macro	**Change occurs:** At dump request **What changes:** Excludes the following SDATA default options currently in effect: • ALLPSA • SQA • SUMDUMP • IO • From all CHNGDUMP operator commands entered through the IEACMD00 parmlib member or through the console Exclusion is only for the dump being requested. Note that certain default system areas are not excluded; these areas are required for IPCS dump analysis. The CHNGDUMP operator command can override the NODEFAULTS option.	To minimize the amount of default data in the dump, code in the program: `SDUMPX SDATA=NODEFAULTS`
Replacing CHNGDUMP operator command in IEACMD00 parmlib member	**Change occurs**: At system initialization **What changes**: This command establishes the IBM-supplied dump options for SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands; see “Contents of SVC dumps” on page 23 for the list.	To add the link pack area (LPA) to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands, while keeping the local system queue area (LSQA) and trace data, add the following command to IEACMD00: `CHNGDUMP SET,SDUMP=(LPA)`

<table>
<caption>Table 11. Customizing SVC dump contents through operator commands (continued)</caption>
<tr><th>Customization</th><th>Effect</th><th>Example</th></tr>
<tr><td>Entering CHNGDUMP operator command with SDUMP parameter on a console with master authority</td><td>Change occurs: Immediately when command is processed

What changes:

For the ADD mode: CHNGDUMP options are added to the current SVC dump options list and to any options specified in the macro or operator command that requested the dump. The options are added to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until another CHNGDUMP SDUMPX operator command is entered.

For the OVER mode: CHNGDUMP options are added to the current SVC dump options list. The system ignores any options specified in the macro or operator command that requested the dump. The options override all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until a CHNGDUMP SDUMP,ADD operator command is entered.

For the DEL option: CHNGDUMP options are deleted from the SVC dump options list.

When more than one CHNGDUMP operator command with SDUMPX is entered, the effect is cumulative.</td><td>To add the LPA to all SVC dumps for SDUMP or SDUMPX macros and DUMP operator commands until changed by another CHNGDUMP SDUMP, enter:

<code>CHNGDUMP SET,SDUMP=(LPA)</code>

To add the CHNGDUMP SDUMPX options list to all SVC dumps:

<code>CHNGDUMP SET,SDUMP,ADD</code>

To override all SVC dumps with the CHNGDUMP SDUMPX options list:

<code>CHNGDUMP SET,SDUMP,OVER</code>

To remove LPA from the SDUMPX options list:

<code>CHNGDUMP DEL,SDUMP=(LPA)</code></td></tr>
<tr><td>Using an operator command parameter.

Parameters on the DUMP operator command specify the contents for the dump being requested.</td><td>Change occurs: At dump request

What changes: The DUMP operator command parameter options are added to the dump options list, but only for the dump being requested.</td><td>To add ALLNUC to this SVC dump, enter:

<code>DUMP COMM=(MYDUMP1 5-9-88)</code>

The system issues a message:

<code>* 23 IEE094D SPECIFY
OPERAND(S)
FOR DUMP COMMAND</code>

Enter in reply:

<code>REPLY 23,JOBNAME=MYJOB1,
SDATA=(ALLNUC),END</code></td></tr>
</table>

Tailoring SVC dumps

Sometimes servers retain client-related data in address spaces and dataspaces other than the client's, which means this data will not be in the dump. For this reason, server code can modify the contents of an SVC dump to provide additional problem determination data by creating a tailored SVC dump exit. This feature allows a requestor to specify a dump request without identifying related server address, dataspaces, and storage areas, which could be unknown and dynamic in nature.

The server code provider can create these SVC dump exits without modifying module IEAVTSXT. The CSVDYNEX macro identifies the exit load module and associates it with the IEASDUMP.SERVER resource.

The exit is allowed to scan the current dump request and determines if data should be added to the dump. The data is added to the dump by identifying it in the appropriate SDMSE_OUTPUT area. For additional details, see IEASDUMP.SERVER dynamic exit processing in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The tailored SVC dump exits are not called in any particular order. To ensure that the current requests are presented to an exit, the dump request is updated between exit invocations. If an exit adds data to the dump, every exit is re-invoked until no additional changes are made. Because of the additional processing required, tailored SVC dump exits do not receive control by default for SDUMPX macro requests. To cause the exit processing to take place, you must specify SDATA=SERVERS in the SDUMPX macro.

SDATA=SERVERS is in force for all operator Dump and SLIP SVC dump requests.

Analyzing summary SVC dumps

The SUMDUMP or SUM option on the SDUMP or SDUMPX macro causes SVC dump to capture a summary dump. Two types of information are captured in summary dumps. First, index data for storage is captured. This index data can be formatted using the IPCS VERBX SUMDUMP command. The second type of information captured is the storage itself. Storage captured by summary dump processing can be viewed using IPCS by specifying the SUMDUMP option (for example, IPCS LIST 00003000 SUMDUMP). IBM strongly recommends that you view the SUMDUMP output prior to investigating the usual portions of the dump. The SUMDUMP option provides different output to SDUMPX branch entries and SVC entries to SDUMP. For example, data included for branch entries to SDUMPX include PSA, LCCA, and PCCA control blocks, and data recorded for SVC entries to SDUMPX include RTM2WA control blocks. Each summary dump index record, when formatted using the IPCS VERBX SUMDUMP command, is displayed as “----tttt---- range-start range-end range-asid range-attributes”. The range-attributes include a value of INCOMP, which means that some or all of the areas represented by the specified range may not be in the dump.

Figure 6 on page 33 is an example format using the IPCS VERBX SUMDUMP command. The summary dump is formatted by the IPCS VERBEXIT SUMDUMP subcommand and has an index which describes what the summary contains. Summary dumps are not created for dumps taken with the DUMP command. Only dumps created by the SDUMP or SDUMPX macro contain summary dumps.

```
STORAGE TYPE       RANGE START       RANGE END      ASID ATTRIBUTES
REGISTER AREA--          0135F000          01363FFF 001E (COMMON)
REGISTER AREA-- 00000001_7F5AD000 00000001_7F5B0FFF 001E
```

Figure 6. Example: Format of IPCS VERBX SUMDUMP command

Note: During SVC dump processing, the system sets some tasks in the requested address space non-dispatchable; non-dispatchable tasks in the dump may have been dispatchable at the time of the problem.

Figure 7 on page 34 is a partial example of a summary dump using the IPCS VERBX SUMDUMP command.

```
STORAGE TYPE         RANGE START        RANGE END      ASID ATTRIBUTES
                          023BCD70           023BCD7F 001E (COMMON)
SUMLSTA RANGE --          017E8000           017E8FFF 0001 (COMMON)
SUMLSTA RANGE --          01F9B000           01F9CFFF 0001 (COMMON)
SUMLSTA RANGE --          02166000           02167FFF 0001 (COMMON)
PSA ------------          00000000           00001FFF 001E (COMMON)
PCCA -----------          00F43008           00F4324F 001E (COMMON)
LCCA -----------          00F82000           00F82A47 001E (COMMON)
LCCX -----------          021C7000           021C771F 001E (COMMON)
INT HANDLER DUCT          02232FC0           02232FFF 001E (COMMON)
I.H. LINKAGE STK          02262000           0226202F 001E (COMMON)
REGISTER AREA --          0000E000           00010FFF 001E
REGISTER AREA --          00FC4000           00FC6FFF 001E (COMMON)
REGISTER AREA --  00000001_7F5AD000   00000001_7F5B0FFF 001E
REGISTER AREA --          7FFFE000           7FFFEFFF 001E
```

Figure 7. Example: IPCS VERBX SUMDUMP command

To examine the storage shown in Figure 7 on page 34, invoke the IPCS LIST command, as shown in Figure 8 on page 34.

```
IPCS LIST 00FC4000. SUMDUMP LEN(256) DISPLAY

****************************** TOP OF DATA ***********************************


LIST 00FC4000 ASID(X'001E') SUMDUMP LENGTH(X'0100')
AREA
ASID(X'001E') SUMDUMP ADDRESS(FC4000.) KEY(00)
00FC4000. 7F6BFFD0 7F6BFFD0 02259010 00000000 |",.}",.}........|
00FC4010. 0225D000 02259010 00000004 00000001 |..}.............|
00FC4020. 00000000 00FC3E10 00000000 00000000 |................|
00FC4030. 00000000 00000000 00000000 00800000 |................|
00FC4040. 06102000 00000000 00000000 00000000 |................|
00FC4050 LENGTH(X'10')==>All bytes contain X'00'
00FC4060. 00000000 02247CB8 00000000 00000000 |......@.........|
00FC4070. 02258040 0000164E 00008000 00000000 |... ...+........|
00FC4080 LENGTH(X'80')==>All bytes contain X'00'
****************************** END OF DATA ***********************************
```

Figure 8. Example: Examining storage

For more information about the record ID values, see the SMDLR and SMDXR control blocks in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

SUMDUMP output for SVC-Entry SDUMPX

For an SVC entry, the storage captured in a summary dump can contain information that is not available in the remainder of the SVC dump if options such as region, LSQA, nucleus, and LPA were not specified in the dump parameters.

For each address space dumped, a summary dump index record is written with the ASID, plus the jobname and stepname for the last task created in the address space. The SUMDUMP output contains RTM2 work areas for tasks in address spaces that are dumped. Many of the fields in the RTM2WA provide valuable debugging information.

The summary dump data is dumped in the following sequence:

1. The ASID record is dumped for the address space.
2. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 ranges and the PSWREGS, parameter list and data ID, data are dumped next. These contain information that is helpful in debugging the problem, and should be examined carefully.
3. All RTM2 work areas pointed to by all TCBs in this address space are dumped.
4. An address range table is built containing the following ranges, pointed to by the RTM2WA:
   - 4K before and after the PSW at the time of error (RTM2NXT1)

- 4K before and after each register at the time of error (RTM2EREG).

Duplicate storage is eliminated from this address range table to reduce the amount of storage dumped.

SUMDUMP output for branch-entry SDUMPX

For branch entry to SDUMP, there are two types of summary dumps:

- Disabled summary dump - which performs the summary dump with the system disabled for interruptions. This means that all data to be dumped must be paged in at the time of the summary dump.
- Suspend summary dump - which is taken in two parts. The first part is similar to the disabled summary dump and dumps some of the global system control blocks. The second part runs with the system enabled for interruptions. This allows data to be dumped that is currently paged out, but was going to be modified by the recovery routine that requested SVC dump processing.

The SUMDUMP output for a branch entry to SVC dump might not match the data that is at the same address in the remainder of the dump. The reason for this is that SUMDUMP is taken at the entry to SVC dump while the processor is disabled for interruptions. The system data in the remainder of the dump is often changed because other system activity occurs before the dump is complete. The SUMDUMP output follows a header that contains the ASID of the address space from which the data was obtained.

The following conditions can occur that prevent SDUMPX from taking a disabled or suspend summary dump.

- The system is not able to obtain the necessary locks to serialize the real storage buffer (RSB).
- The system is in the process of modifying the storage queues and cannot satisfy the request for a RSB.
- No frames are available for a RSB.
- SVC dump encounters an error while holding serialization for the RSB.
- A critical frame shortage causes the system to steal the pages of the RSB.
- The SVC dump timer disabled interruption exit determines that SVC dump has failed and frees the RSB.

Analyzing disabled summary dumps

For **disabled summary dumps**, records are dumped in the following order:

1. If a suspend summary dump was requested but could not be taken, the system attempts to obtain a disabled summary dump. If this occurs, an error record is written to that effect. If the system is unable to obtain a suspend summary dump and a disabled summary dump, then no summary data is available for the dump.
2. The XMEM ASID record is written that gives the ASID that is home, primary, secondary, and CML (if the CML lock is held).
3. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 address ranges and the PSWREGS data are dumped.
4. The PSA, PCCA, LCCA, and LCCX for each processor are dumped.
5. The current PCLINK stack (pointed to by PSASEL) is dumped (if it exists).
6. If this is a SLIP request for a dump (ACTION=SVCD), then the SLIP reg/PSW area (pointed to by the SUMDUMP parameter list SDURGPSA) is dumped.

   The following address ranges are added to the address range table:

   - 4K before and after the PSW address at the time of the SLIP trap.
   - 4K before and after each address in the registers at the time of the SLIP trap.

   Duplicate storage is eliminated from this address range table to reduce the amount of data written to the dump data set.

   Note that if the primary and secondary ASIDs are different, the above address ranges are added to the table for both ASIDs.

7. The IHSA is dumped along with its associated XSB and PCLINK stack. The PSW and register addresses from the IHSA are added to the range table. This causes 4K of storage to be dumped around each address.
8. The caller's SDWA is dumped, if one exists. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address.
9. The addresses in the address range table are dumped.
10. The super FRR stacks are dumped.
11. The global, local, and CPU work save area (WSA) vector tables are dumped. The save areas pointed to by each of these WSA vector tables are also dumped.
12. 4K of storage on either side of the address portion of the I/O old PSW, the program check old PSW, the external old PSW, and the restart old PSW saved in the PSA for all processors, are dumped.

Analyzing suspend summary dumps

For **suspend summary dumps**, records are dumped in the following order:

1. The ASID: the PSA, PCCA, LCCA records, the IHSA, XSB, and the PCLINK stack, are all dumped with the system disabled in the same way they are dumped in steps 2, 4, and 5 for the disabled summary dump.

   At this point, an SRB is scheduled to the DUMPSRV address space and the current unit of work (SDUMP's caller) is suspended by using the STOP service. Data dumped at this point does not have to be paged in because the system is enabled. Cross memory functions are used to gain access to data in the caller's address space.
2. The SUMLIST/SUMLSTA/SUMLSTL/SUMLIST64 address ranges and the PSWREGS data are dumped.
3. The caller's ASCB is dumped.
4. The suspended unit of work (SVC dump's caller) is dumped. This is either a TCB or an SSRB. The related PCLINK stacks are also dumped.
5. For TCB mode callers, the caller's SDWA is dumped. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address. All RTM2 work areas pointed to by this TCB and any associated SDWAs are all dumped.

   For SRB mode callers, the SDWA is dumped. The PSW and register addresses from the SDWA are added to the range table. This causes 4K of storage to be dumped around each address. Also, the caller's register save area is added to the range table and the storage dumped.

   Duplicate storage is eliminated from the address range table to reduce the amount of storage dumped.
6. After all the storage is saved in a virtual buffer in the DUMPSRV address space, the caller's unit of work is reset by using the RESET service. This allows SVC dump to complete and return to the caller. When SVC dump processing completes in the address space to be dumped, whatever processing was taking place in that address space when it was interrupted by SVC dump resumes. The rest of the dump is then scheduled from the DUMPSRV address space.

Analyzing an SVC dump

This section shows you how to use IPCS to analyze an SVC dump. You would analyze an SVC dump because of one of the following:

- Dump output from the IPCS STATUS FAILDATA subcommand did not contain data for the abend being diagnosed.
- The problem involved multiple abends.
- The dump was taken but does not contain abend-related information.

This section contains the following topics, which, if followed in order, represent the procedure for analyzing an SVC dump:

- “Formatting the SVC dump header” on page 37

- "Looking at the dump title" on page 38
- "Displaying the incident token, time and type of dump" on page 39
- "Locating error information" on page 40
- "Analyze TCB structure" on page 42
- "Examining the LOGREC buffer" on page 44
- "Examining the system trace" on page 45
- "Looking at the registers" on page 46
- "Other useful reports for SVC dump analysis" on page 47
- "Reading the SDUMPX 4K SQA buffer" on page 47

Specifying the source of the dump

The first step in analyzing the dump is to specify the source of the dump that IPCS should format. In the IPCS dialog (see Figure 9 on page 37), choose option 0 (DEFAULTS) and specify the name of the SVC dump data set on the "Source" line.

```
------------------------- IPCS Default Values --------------------------------
COMMAND ===> 0


  You may change any of the defaults listed below.
  If you change the Source default, IPCS will display the current default
  Address Space for the new source and will ignore any data entered in
  the Address Space field.

  Source  ==> DSN('D46IPCS.SVC.CSVLLA.DUMP002')
  Address Space   ==>                        Ignored if Source is changed.
  Message Routing ==> NOPRINT TERMINAL
  Message Control ==> FLAG(WARNING)
  Display Content ==> NOMACHINE REMARK REQUEST NOSTORAGE SYMBOL

Press ENTER to update defaults.
Use the END command to exit without an update.
```

Figure 9. IPCS Default Values menu

Press Enter to register the new default source name. Then, press PF3 to exit the panel.

You can also use the SETDEF subcommand to specify the source. For the dump in the preceding example, enter:

```
SETDEF DSNAME('D46IPCS.SVC.CSVLLA.DUMP002')
```

IPCS does not initialize the dump until you enter the first subcommand or IPCS dialog option that performs formatting or analysis. At that time IPCS issues message BLS18160D to ask you if summary dump data can be used by IPCS. The summary dump data should always be used for an SVC dump because it is the data captured closest to the time of the failure. If you do not allow IPCS to use summary dump data, other data captured later for the same locations will be displayed, if available. Such data is less likely to be representative of the actual data at these storage locations at the time of the failure.

Formatting the SVC dump header

The SVC dump header contains the following information:

- SDWA or SLIP data
- Dump title, error identifier, and time of the dump
- Requestor of dump

This information describes the type of SVC dump and can tell you if the dump is a CONSOLE dump or a dump caused by the SLIP command. You would analyze these dumps differently.

Format data in the header of an SVC dump using the following IPCS subcommands:

- LIST TITLE
- STATUS FAILDATA
- STATUS REGISTERS
- STATUS WORKSHEET

The following sections give examples of how to use these IPCS subcommands (or IPCS dialog options, where applicable) to obtain the desired information.

Looking at the dump title

The dump title tells you the component name, component identifier and module name. You can find the dump title using the following IPCS subcommands:

- LIST TITLE
- STATUS WORKSHEET

You can also obtain the STATUS WORKSHEET report through option 2.3 of the IPCS dialog. First, choose option 2 (ANALYSIS) from the primary option menu, as shown in Figure 10 on page 38.

```
-------------------- z/OS 01.02.00 IPCS PRIMARY OPTION MENU ----------------
OPTION ===> 2

                                                    ********************
 0DEFAULTS  - Specify default dump and options      * USERID   - IPCSU1
 1BROWSE    - Browse dump data set                  * DATE     - 84/06/08
 2ANALYSIS  - Analyze dump contents                 * JULIAN   - 84.160
 3SUBMIT    - Submit problem analysis job to batch  * TIME     - 16:43
 4COMMAND   - Enter subcommand, CLIST or REXX exec  * PREFIX   - IPCSU1
 5UTILITY   - Perform utility functions             * TERMINAL - 3278
 6DUMPS     - Manage dump inventory                 * PF KEYS  - 24
 TTUTORIAL  - Learn how to use the IPCS dialog      ********************
 XEXIT      - Terminate using log and list defaults

Enter END command to end the IPCS dialog.
```

Figure 10. IPCS primary option menu

Then, choose option 3 (WORKSHEET) from the analysis of dump contents menu, as shown in Figure 11 on page 38.

```
-------------------- IPCS MVS ANALYSIS OF DUMP CONTENTS ---------------------
OPTION  ===> 3
To display information, specify the corresponding option number.

  1SYMPTOMS    - Symptoms                             *****************
  2STATUS      - System environment summary           * USERID  - IPCSU1
  3WORKSHEET   - System environment worksheet         * DATE    - 84/06/08
  4SUMMARY     - Address spaces and tasks             * JULIAN  - 84.160
  5CONTENTION  - Resource contention                  * TIME    - 16:44
  6COMPONENT   - MVS component data                   * PREFIX  - IPCSU1
  7TRACE       - Trace formatting                     * TERMINAL- 3278
  8STRDATA     - Coupling Facility structure data     * PF KEYS - 24
                                                        ******************
 Enter END command to terminate MVS dump analysis.
```

Figure 11. IPCS MVS analysis of dump contents menu

IPCS displays a new panel with information similar to that in Figure 12 on page 39. The dump title is labelled at the top of the STATUS WORKSHEET report. The dump title is “Compon=Program Manager Library-Lookaside, Compid=SC1CJ, Issuer=CSVLLBLD.” See *z/OS MVS Diagnosis: Reference* for an explanation of dump titles.

```
IPCS OUTPUT STREAM ------------------------------------------ LINE 0 COL
COMMAND ===>                                                  SCROLL ===
 ****************************** TOP OF DATA ****************************


                             MVS Diagnostic Worksheet

Dump Title: COMPON=PROGRAM MANAGER LIBRARY-LOOKASIDE,COMPID=SC1CJ,
             ISSUER=CSVLLBLD

CPU Model 2064 Version FF Serial no. 131512 Address 01
Date: 02/15/2001   Time: 20:33:34.680912  Local

Original dump dataset: SYS1.DUMP06

Information at time of entry to SVCDUMP:

HASID 001B  PASID 001B  SASID 001B  PSW 070C1000 8001AE5A

CML ASCB address 00000000  Trace Table Control Header address 7FFE3000
Dump ID: 001
Error ID: Seq 00019  CPU 0041  ASID X'001E'  Time 20:33:33.5
```

Figure 12. STATUS WORKSHEET subcommand sample output — dump title

STATUS WORKSHEET also displays the error ID. In Figure 12 on page 39, the dump ID is 001, error ID is sequence number 00051, ASID=X'001B', and processor 0000. Use this dump ID to match messages in SYSLOG and LOGREC records to the dump.

Displaying the incident token, time and type of dump

The IPCS subcommand STATUS SYSTEM identifies the following types if information. The IPCS dialog does not have a menu option for STATUS SYSTEM. Instead you must enter the subcommand.

- The time of the dump
- The program requesting the dump
- An incident token that associates one or more SVC dumps requested for a problem on a single system or on several systems in a sysplex

Figure 13 on page 40 is an example of a STATUS SYSTEM report. For a scheduled SVC dump, the following identifies the dump:

```
Program Producing Dump: SVCDUMP
Program Requesting Dump: IEAVTSDT
```

A dump requested by a SLIP or DUMP operator command is always a scheduled SVC dump.

For a synchronous SVC dump, the following identifies the dump:

```
Program Producing Dump: SVCDUMP
Program Requesting Dump: cccccccc
```

Where *cccccccc* is one of the following:

- The name of the program running when the system detected the problem
- SVCDUMP, if the system could not determine the failing task

A SYSMDUMP ABEND dump is always a synchronous SVC dump.

```
SYSTEM STATUS:
  Nucleus member name: IEANUC01
   I/O configuration data:
     IODF data set name: SYS0.IODF43
     IODF configuration ID: CONGIG00
     EDT ID: 00
  Sysplex name: SYSPL1
  TIME OF DAY CLOCK: B566EA85 A0750707 02/15/2001 20:33:34.680912 local
  TIME OF DAY CLOCK: B567202A 89750707 02/16/2001 00:33:34.680912 GMT
  Program Producing Dump: SVCDUMP
  Program Requesting Dump: IEAVTSDT
  Incident token: SYSPL1   S4       06/23/1993 12:43:54.697367 GMT
```

Figure 13. Sample output from the STATUS SYSTEM subcommand

SYSTEM STATUS for an SVC dump contains an incident token. The request for the dump specifies the incident token or the system requesting the dumps provides it. The incident token consists of:

- The name of the sysplex
- The name of the system requesting the multiple dumps
- The date in Greenwich Mean Time (GMT)
- The time in GMT

Locating error information

Use the IPCS subcommand STATUS FAILDATA to locate the specific instruction that failed and to format all the data in an SVC dump related to the software failure. This report gives information about the CSECT involved in the failure, the component identifier, and the PSW address at the time of the error.

Note: For SLIP dumps or CONSOLE dumps, use SUMMARY FORMAT or VERBEXIT LOGDATA instead of STATUS FAILDATA.

Choose option 4 (COMMAND) from the IPCS primary option menu (see Figure 14 on page 40) and enter the following command. Use the PF keys to scroll up and down through the report. The following sections describe parts of the report.

```
------------------------- IPCS Subcommand Entry -------------------------------
Enter a free-form IPCS subcommand, CLIST, or REXX exec invocation below:

===> STATUS FAILDATA
```

Figure 14. IPCS Subcommand Entry menu

Identifying the abend and reason codes: As Figure 15 on page 40 shows, under the heading “SEARCH ARGUMENT ABSTRACT”, you will find the abend code and, if provided, an abend reason code.

```
⋮
SEARCH ARGUMENT ABSTRACT

 PIDS/5752SC1CJ RIDS/CSVLLCRE#L RIDS/CSVLLBLD AB/S0FF0 REGS/09560 REGS/ 06026
  RIDS/CSVLLBLD#R

  Symptom             Description
  -------             -----------
  PIDS/5752SC1CJ      Program id: 5752SC1CJ
  RIDS/CSVLLCRE#L     Load module name: CSVLLCRE
  RIDS/CSVLLBLD       Csect name: CSVLLBLD
  AB/S0FF0            System abend code: 0FF0
  REGS/09560          Register/PSW difference for R09: 560
  REGS/06026          Register/PSW difference for R06: 026
  RIDS/CSVLLBLD#R     Recovery routine csect name: CSVLLBLD
⋮
```

Figure 15. Search argument abstract in the STATUS FAILDATA report

In Figure 15 on page 40, the abend code is X'FF0' with no reason code. See *z/OS MVS System Codes* for a description of the abend code and reason code.

The following IPCS reports also provide the abend and reason codes:

- VERBEXIT LOGDATA
- STATUS WORKSHEET
- VERBEXIT SYMPTOMS

Finding the system mode: As Figure 16 on page 41 shows, below the “SEARCH ARGUMENT ABSTRACT“ section is information describing the system mode at the time of the error.

```
 ⋮

Home ASID: 001B    Primary ASID: 001B    Secondary ASID: 001B
PKM: 8000          AX: 0001              EAX: 0000

RTM was entered because a task requested ABEND via SVC 13.
The error occurred while: an SRB was in control.
No locks were held.
No super bits were set.
 ⋮
```

Figure 16. System mode information in the STATUS FAILDATA report

The line that starts with “The error occurred...” tells you if the failure occurred in an SRB or TCB. In the example in Figure 16 on page 41, the error occurred while an SRB was in control, which means you need to look under the heading SEARCH ARGUMENT ABSTRACT (see Figure 15 on page 40) to find the CSECT and load module names. This is the module in which the abend occurred.

If an SRB service routine was in control, look under the heading SEARCH ARGUMENT ABSTRACT for the CSECT and load module names. This is the failing module.

In output from a SUMMARY FORMAT subcommand, look for the RB for the abending program. The RB has an RTPSW1 field that is nonzero.

In a dump requested by a SLIP operator command, use a STATUS CPU REGISTERS subcommand to see data from the time of the problem.

If the error had occurred while a TCB was in control, you would find the failing TCB by formatting the dump using the IPCS subcommand SUMMARY TCBERROR. See “Analyze TCB structure” on page 42.

Identifying the failing instruction: The STATUS FAILDATA report also helps you find the exact instruction that failed. This report provides the PSW address at the time of the error and the failing instruction text. Note that the text on this screen is not always the failing instruction text. Sometimes the PSW points to the place where the dump was taken and not the place where the error occurred.

In Figure 17 on page 42, the PSW at the time of the error is X'11E6A3C' and the instruction length is 4-bytes; therefore, the failing instruction address is X'11E6A38'. The failing instruction is 927670FB.

```
 ⋮
OTHER SERVICEABILITY INFORMATION

  Recovery Routine Label:  CSVLEBLD
  Date Assembled:          00245
  Module Level:            HBB7705
  Subfunction:             LIBRARY-LOOKASIDE

Time of Error Information

  PSW: 070C0000 811E6A3C   Instruction length: 04   Interrupt code: 0004
  Failing instruction text: E0009276 70FB5030 70F8D7F7

  Registers 0-7
  GR: 0002A017 00FBE800 00000000 00000076  00000C60 00FBE600 0002A016 00000017
  AR: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
  Registers 8-15
  GR: 012221A8 811E69F8 00000001 30000000  00FD82C8 811CAD90 011E69D0 011E69D0
  AR: 00000000 00000000 00000000 00000000  00000000 00000000 00005F60 00000000
 ⋮
```

Figure 17. Time of error information in the STATUS FAILDATA report

The failing instruction text displayed in this report is always 12 bytes, 6 bytes before and 6 bytes after the PSW address. In this example, the failing instruction, 927670FB, is an MVI of X'76' to the location specified by register 7 + X'FB'.

Register 7 at the time of the error, shown under `Registers 0-7` above, contained a X'00000017'. The attempted move was to storage location X'112'. The first 512 bytes of storage are hardware protected. Any software program that tries to store into that area without authorization will receive a protection exception error and a storage protection exception error.

See *z/Architecture Principles of Operation* for information about machine language operation codes, operands, and interruption codes.

To find the module that abnormally terminated and the offset to the failing instruction, use the WHERE command. WHERE can identify the module or CSECT that the failing PSW points to.

Analyze TCB structure

If a TCB was in control at the time of the error, use the IPCS subcommand SUMMARY TCBERROR to look at the TCB information and find the failing component. SUMMARY TCBERROR summarizes the control blocks for the failing address space. (To see all the fields in the control blocks, use SUMMARY FORMAT.) Scan the completion codes (field CMP) for each TCB to find the correct TCB. This report displays RBs from newest to oldest.

Figure 18 on page 43 is an example of SUMMARY TCBERROR output. In this example the TCB at address 008E9A18 has a completion code of X'0C1.' The error occurred under this TCB. Once you have identified the failing TCB, you can follow the RB chain to the failing program.

```
  ⋮
     NAME..... IEFSD060  ENTPT.... 00DA6308

  PRB: 008FFED0
     WLIC..... 00020006  FLCE.... 00C534A0  OPSW..... 070C2000 00DC766A
     LINK..... 008FFA10

  CDE: 00C534A0
     NAME..... IEESB605  ENTPT.... 00DC7000

  TCB:  008FF0D0
     CMP...... 00000000  PKF...... 80        LMP...... FF        DSP...... FF
     TSFLG.... 00        STAB..... 008FD210  NDSP..... 00002000
     JSCB..... 008FF4FC  BITS..... 00000000  DAR...... 00
     RTWA..... 00000000  FBYT1.... 00
     Task non-dispatchability flags from TCBFLGS4:
      Top RB is in a wait
     Task non-dispatchability flags from TCBFLGS5:
      Secondary non-dispatchability indicator
     Task non-dispatchability flags from TCBNDSP2:
      SVC Dump is executing for another task
  PRB: 008E9F20
     WLIC..... 00020001  FLCDE.... 00C4CA38  OPSW..... 070C1000  00DAC66E
     LINK..... 018FF0D0

  CDE: 00C4CA38
     NAME..... IEFIIC    ENTPT.... 00DA6000  TCB: 008E9A18
     CMP...... 940C1000  PKF...... 80        LMP...... FF        DSP...... FF
     TSFLG.... 20        STAB..... 008FD180  NDSP..... 00000000
     JSCB..... 008FF33C  BITS..... 00000000  DAR...... 01
     RTWA..... 7FFE3090  FBYT1.... 08


 SVRB: 008FD7A8
     WLIC..... 00020000  FLCDE.... 00000000  OPSW..... 070C1000  82569B38
     LINK..... 008FD638

  PRB: 008E9750
     WLIC..... 00020033  FLCDE.... 14000000  OPSW..... 070C1000  80CE9AEE
     LINK..... 008FD638

  SVRB: 008FD638
     WLIC..... 0002000C  FLCDE.... 00000000  OPSW..... 070C1000  825E9768
     LINK..... 008FD4C8

  SVRB: 008FD4C8
     WLIC..... 00020001  FLCDE.... 00000000  OPSW..... 070C0000  00C47D52
     LINK..... 008FD358

  SVRB: 008FD358
     WLIC..... 00020053  FLCDE.... 00000000  OPSW..... 075C0000  00D64E0C
     LINK..... 008FF4D8

  PRB: 008FF4D8
     WLIC..... 00020014  FLCDE.... 008FF3D8  OPSW..... 078D0000  00006EF2
     LINK..... 008E9A18

  CDE: 008FF3D8
     NAME..... SMFWT     ENTPT.... 00006EB0
```

Figure 18. Example: the SUMMARY TCBERROR report

In this example, the most current RB is the SVRB at address 008FD7A8. This is the SVC dump's RB. The ESTAE's RB is the PRB at 008E9750. The ESTAE issued an SVC 33. The RB for the recovery termination manager (RTM) is the SVRB at 008FD638. RTM issued an SVC C to attach the ESTAE. The X'0C1' abend occurred under the SVRB at 008FD4C8. The last interrupt was a 1 at the address indicated in the old PSW field (OPSW). The next RB in the chain shows an SVC X'53' (SMFWTM) had been issued. This is the code the X'0C1' occurred in.

For a scheduled dump, the abnormally terminating TCB can generally be found by scanning for a nonzero completion code. If there is no code, scan the system trace for the abend. The trace identifies the ASID number and TCB address for each entry. See “Examining the system trace” on page 45.

Use the STATUS or the STATUS REGS subcommand to find the data set name and the module name of the SVC dump requester.

Examining the LOGREC buffer

Use the IPCS subcommand VERBEXIT LOGDATA to view the LOGREC buffer in a dump. This report might repeat much of the information contained in the STATUS FAILDATA report, but it helps to identify occasions when multiple error events caused the software failure.

The following sample output, from the VERBEXIT LOGDATA subcommand, shows how multiple errors can appear in the LOGREC buffer. Abend X'0D5' is the first abend and X'058' is the second. Always check for multiple errors in the VERBEXIT LOGDATA report that are in the same address space or a related address space and are coincident with or precede the SVC dump.

```
 TYPE:  SOFTWARE RECORD      REPORT:  SOFTWARE EDIT REPORT           DAY.YEAR
        (PROGRAM INTERRUPT)                                 REPORT DATE: 320.17
 SCP:   VS 2 REL 3                                           ERROR DATE: 229.17
                             MODEL:   2964                               HH:MM:SS.TH
                             SERIAL:  01F167                       TIME: 13:27:59.86

 JOBNAME: LSCMSTR

ERRORID: SEQ=01196  CPU=0042  ASID=000C  TIME=13:27:59.6

 SEARCH ARGUMENT ABSTRACT

   PIDS/####SC1C5 RIDS/NUCLEUS#L RIDS/IEAVEDS0 AB/S00D5 PRCS/00000021 REGS/0F120
   RIDS/IEAVEDSR#R

   SYMPTOM             DESCRIPTION
   -------             -----------
   PIDS/####SC1C5      PROGRAM ID: ####SC1C5
   RIDS/NUCLEUS#L      LOAD MODULE NAME: NUCLEUS
   RIDS/IEAVEDS0       CSECT NAME: IEAVEDS0
   AB/S00D5            SYSTEM ABEND CODE: 00D5
   PRCS/00000021       ABEND REASON CODE: 00000021
   REGS/0F120          REGISTER/PSW DIFFERENCE FOR R0F: 120
   RIDS/IEAVEDSR#R     RECOVERY ROUTINE CSECT NAME: IEAVEDSR

 OTHER SERVICEABILITY INFORMATION

   RECOVERY ROUTINE LABEL:  IEAVEDSR
   DATE ASSEMBLED:          08/23/89
   MODULE LEVEL:             UY41669
   SUBFUNCTION:             DISPATCHER

 TIME OF ERROR INFORMATION

   PSW: 44040000 80000000 00000000 00FEFC56
   INSTRUCTION LENGTH: 04   INTERRUPT CODE: 0021
   FAILING INSTRUCTION TEXT: 1008B777 1008B225 000007FE
   TRANSLATION EXCEPTION IDENTIFICATION: 00000041
   BREAKING EVENT ADDRESS: 00000000_00FF650E
   AR/GR 0-1    00000000/00000000_00000041  00000000/00000000_00F9A0C0
   AR/GR 2-3    00000000/00000000_00000000  00000000/00000000_00000000
   AR/GR 4-5    00000000/00000000_00000000  00000000/00000000_008DE188
   AR/GR 6-7    00000000/00000000_008E8C78  00000000/00000000_00000001
   AR/GR 8-9    00000000/00000000_00F97280  00000000/00000000_0103AB6A
   AR/GR 10-11  00000000/00000000_00FF1B08  00000000/00000000_008DE188
   AR/GR 12-13  00000000/00000000_0000000C  00000000/00000000_000C0041
   AR/GR 14-15  00000000/00000000_80FF6510  00000000/00000000_00FEFB36

   HOME ASID: 000C    PRIMARY ASID: 000C    SECONDARY ASID: 000C
   PKM: 8000          AX: 0001

   RTM WAS ENTERED BECAUSE OF A PROGRAM CHECK INTERRUPT.
   THE ERROR OCCURRED WHILE A LOCKED OR DISABLED ROUTINE WAS IN CONTROL.
   NO LOCKS WERE HELD.
   SUPER BITS SET: PSADISP  - DISPATCHER
⋮
 TYPE:  SOFTWARE RECORD      REPORT:  SOFTWARE EDIT REPORT         DAY.YEAR
        (SVC 13)                                            REPORT DAT 320.17
 SCP:   VS 2 REL 3                                           ERROR DAT 229.17
                             MODEL:   2964                               HH:MM:SS.TH
                             SERIAL:  01F167                       TIME: 13:27:59.94

 JOBNAME: LSCMSTR
 ERRORID: SEQ=01197  CPU=0000  ASID=000C  TIME=13:27:59.6

 SEARCH ARGUMENT ABSTRACT
```

```
    AB/S0058

    SYMPTOM             DESCRIPTION
    -------             -----------
    AB/S0058            SYSTEM ABEND CODE: 0058

 SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE

   PROGRAM ID
   LOAD MODULE NAME
   CSECT NAME
   RECOVERY ROUTINE CSECT NAME
   RECOVERY ROUTINE LABEL
   DATE ASSEMBLED
   MODULE LEVEL
   SUBFUNCTION

 TIME OF ERROR INFORMATION

   PSW: 47048000 00000000 00000000 00FDC266
   INSTRUCTION LENGTH: 02   INTERRUPT CODE: 000D
   FAILING INSTRUCTION TEXT: 00000000 0A0D0A06 00000000

   BREAKING EVENT ADDRESS: 00000000_001B192
   AR/GR 0-1    00000000/00000000_00A5D7A8  00000000/00000000_80058000
   AR/GR 2-3    00000000/00000000_00000041  00000000/00000000_022DC780
   AR/GR 4-5    00000000/00000000_008EDF00  00000000/00000000_008FBC7C
   AR/GR 6-7    00000000/00000000_00F86A00  00000000/00000000_026E01F0
   AR/GR 8-9    00000000/00000000_026E0160  00000000/00000000_00000000
   AR/GR 10-11  00000000/00000000_8001AC96  00000000/00000000_0001BC96
   AR/GR 12-13  00000000/00000000_00000000  00000000/00000000_000188F0
   AR/GR 14-15  00000000/00000000_8001B194  00000000/00000000_00000020

   HOME ASID: 000C    PRIMARY ASID: 000C    SECONDARY ASID: 000C
   PKM: 8000          AX: 0001

   RTM WAS ENTERED BECAUSE AN SVC WAS ISSUED IN AN IMPROPER MODE.
   THE ERROR OCCURRED WHILE AN ENABLED RB WAS IN CONTROL.
   NO LOCKS WERE HELD.
   NO SUPER BITS WERE SET.
```

This sample output was created on a currently supported z/OS release and at least one high half (bits 0-31) of the 64-bit GR from GRs 0-15 is not 0. The VERBEXIT LOGDATA report might show less data under different conditions.

When viewing the VERBEXIT LOGDATA report, skip the hardware records to view the software records. Search for the first software record.

The field “ERRORID=” gives the error identifier for the software failure. The error identifier consists of the sequence number, ASID, and time of the abend. By matching this identifier with error identifiers from other reports, you can tell if this is the same abend you have been analyzing or if it is a different abend. See “Interpreting software records” on page 532 for more information.

Examining the system trace

The system trace table describes the events in the system leading up to the error. The trace table is helpful when the PSW does not point to the failing instruction, and to indicate what sequence of events preceded the abend.

IPCS option 2.7.4 formats the system trace. The report is long. IBM recommends scrolling to the end of the report, then backing up to find the trace entry for the abend. Type an M on the command line and press F8 to scroll to the bottom of the report.

After you find the entry for the abend, start at the PSW where the dump was taken and track the events in the table to find where the failing instruction is in the code.

The system trace report marks important or significant entries with an asterisk. In Figure 19 on page 46, `*SVC D` in the `Ident CD/D` column identifies the PSW where the program took the dump. Prior to the `SVC D` are three `PGM` (program check) entries. `PGM 001` has an asterisk next to it, indicating that the program check was unresolved. The next entry, `RCVY PROG`, identifies a recovery program that failed because it issued the `SVC D` a few entries later. See Chapter 8, “System trace,” on page 153 to recognize significant entries in the system trace table.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3
                                                  Unique-4 Unique-5 Unique-6

⋮

0001 0094 00AF7D18 DSP         070C0000 81EA7000  00000000 00000000 0000800C
0001 0094 00AF7D18 SVC      78 070C0000 81EA7048  00000002 00000278 00000000
0001 0094 00AF7D18 SVCR     78 070C0000 81EA7048  00000000 00000278 03300D88
0001 0094 00AF7D18 PGM     010 070C0000 81EA704A  00040010 03300D8C
0001 0094 00AF7D18 PGM     011 070C0000 81EA704A  00040011 03300D8C
0001 0094 00AF7D18 SVC      77 070C2000 81EA7088  81EA7000 00000000 00050000
0001 0094 00AF7D18 SVCR     77 070C2000 81EA7088  00000000 00000000 40000000
0001 0094 00AF7D18 *PGM    001 070C0000 83300FAA  00020001 03300D8C
0001 0094 00AF7D18 *RCVY  PROG                    940C1000 00000001 00000000
0002 0001 00000000 I/O     1A2 070E0000 00000000  0080000E 060246C0 0C000001
0002 0054 00AD7300 SRB         070C0000 810537E0  00000054 00F3C9F8 00F3CA40
0001 0054 00AF7D18 SSRV    12D          810B9CEE  00AF7D18 000C0000 00000000
0001 0094 00AF7D18 SSRV    12D          810B9D0E  00AF7D18 000B0000 00000000
0001 0094 00AF7D18 DSP         070C0000 810BF664  00000000 00000000 40000000
0001 0094 00AF7D18 *SVC      D 070C0000 810BF666  00000040 00000000 40000000
0001 0054 00000000 SSRV    10F          00000000  00F83E80 00AD7300 00AC5040
```

Figure 19. Example: output from the IPCS subcommand SYSTRACE

Looking at the registers

Use the IPCS subcommand STATUS REGISTERS to display the registers for the TCBs and RBs. SUMMARY REGS gives the same information in a different format. This report identifies the PSW, ASID and register values just as the STATUS FAILDATA report, but STATUS REGISTERS also gives the control register values.

```
CPU STATUS:

 PSW=070C1000  80FE5CFC  (RUNNING IN PRIMARY, KEY 0,  AMODE 31, DAT ON)
     DISABLED FOR PER
   ASID(X'001B') 00FE5CFC. IEANUC01.IEAVESVC+05FC IN READ ONLY NUCLEUS
  ASCB27 at F3FA00, JOB(LLA), for the home ASID
  ASXB27 at 9FDF00 for the home ASID. No block is dispatched
  HOME ASID: 001B PRIMARY ASID: 001B SECONDARY ASID: 001B

  GPR VALUES
      0-3  80000000  80FF0000  009FF5A0  00FC4E88
      4-7  009F8E88  009FD358  80FE5CD6  00F3FA00
      8-11 00000000  80FE579C  009FD418  7FFFE2C0
     12-15 7FFE0000  00006730  00FE6200  80014910

  ACCESS REGISTER VALUES
      0-3  7FFEA5CC  00000000  00000000  00000000
      4-7  00000000  00000000  00000000  00000000
      8-11 00000000  00000000  00000000  00000000
     12-15 00000000  00000000  00005F60  8210532A

ALET TRANSLATION
AR 00    Not translatable
AR 14    Not translatable
AR 15    Not translatable

  CONTROL REGISTER VALUES
      0-3  5EB1EE40  00A2007F  007CCDC0  8000001B
      4-7  0001001B  00C506C0  FE000000  00A2007F
      8-11 00000000  00000000  00000000  00000000
     12-15 0082E07B  00A2007F  DF880C71  7FFE7008
```

Figure 20. Sample of the STATUS REGISTERS report

The example output in Figure 20 on page 46 shows the address in the PSW is X'0FE5CFC', the ASID is X'1B', and the failing instruction is located in offset X'5FC' in the CSECT IEAVESVC in the module IEANUC01 in the nucleus. You can now browse the dump at this location and look at the specific failing instruction. You could also use the information about the registers to find out more about the error if the address in the PSW does not point to the failing instruction.

This report identifies the PSW, ASID and register values just as the STATUS FAILDATA report. However, as Figure 21 on page 47 shows, STATUS REGISTERS also gives the control register values.

```
CPU STATUS:
 PSW=070C4000 00FC5C96
 (Running in AR, key 0, AMODE 24, DAT ON)
 DISABLED FOR PER
 ASID(X'001E') FC5C96. STRUCTURE(Cvt)+D6 IN READ/WRITE NUCLEUS
 ASID(X'001E') FC5C96. IEANUC01.IEAVCVT+0116 IN READ/WRITE NUCLEUS
 ASID(X'001E') FC5C96. STRUCTURE(Dcb)+0152 IN READ/WRITE NUCLEUS
 ASID(X'001E') FC5C96. STRUCTURE(Dcb)+015A IN READ/WRITE NUCLEUS
 ASCB30 at F90B80, JOB(ORANGE), for the home ASID
 ASXB30 at 6FDE90 and TCB30D at 6E7A68 for the home ASID
 HOME ASID: 001E PRIMARY ASID: 001E SECONDARY ASID: 001E

 General purpose register values
 0-1 00000000_00000020 00000000_84058000
 2-3 00000000_00000000 00000001_00004000
 4-5 00000000_01F9B9A8 00000000_01F9B9A8
 6-7 00000000_00000000 00000000_01F9BE10
 8-9 00000000_00000000 00000000_FFFFFFFC
 10-11 00000000_00000000 00000000_00FDAC58
 12-13 00000000_01560410 00000000_01F9BB08
 14-15 00000000_8155E5A8 00000000_0000003C

 Access register values
 0-3 00000000 00000000 00000000 00000000
 4-7 00000000 00000000 00000000 00000000
 8-11 00000000 00000000 00000000 00000000
 12-15 00000000 00000000 00000000 00000000

 Control register values
 Left halves of all registers contain zeros
 0-3 5F29EE40 0374C007 008D0A40 00C0001E
 4-7 0000001E 02A30780 FE000000 0374C007
 8-11 00020000 00000000 00000000 00000000
 12-15 0294EE43 0374C007 DF882A2F 7F5CD4B0
```

Figure 21. Sample of the STATUS REGISTERS report run in z/Architecture mode

Other useful reports for SVC dump analysis

To collect further SVC dump data, use any of the following commands.

IPCS subcommand	Information in the report
STATUS CPU REGISTERS DATA CONTENTION	Data about the abend, current ASID, and task.
SUMMARY FORMAT	All fields in the TCBs and the current ASID.
TCBEXIT IEAVTFMT 21C.%	The current FRR stack.
LPAMAP	The entry points in the active LPA and PLPA.
VERBEXIT NUCMAP	A map of the modules in the nucleus when the dump was taken.
VERBEXIT SUMDUMP	The data dumped by the SUMDUMP option on the SDUMPX macro.
VERBEXIT MTRACE	The master trace table.
VERBEXIT SYMPTOMS	The primary and secondary symptoms if available.

Note: Use the VERBEXIT SYMPTOMS subcommand last in your SVC dump analysis. Other subcommands can add symptoms to the dump header record. This ensures VERBEXIT SYMPTOMS provides all symptoms available from the dump.

Reading the SDUMPX 4K SQA buffer

The following SVC dumps contain problem data in an SDUMPX 4K system queue area (SQA) buffer:

- An SVC dump requested by a SLIP operator command
- Other SVC dumps, when indicated in the explanation of the dump title.
- An SVC dump requested by an SDUMP or SDUMPX macro with a BUFFER=YES parameter

To obtain the buffer, use the following IPCS subcommand:

```
LIST 0 DOMAIN(SDUMPBUFFER) LENGTH(4096)
```

Table 12 on page 48 describes the fields in the SQA buffer and should be used for diagnosis.

Table 12. Fields in SQA buffer

Offset	Length	Content
0 (0)	4	The characters, TYPE
4 (4)	4	RTM/SLIP processing environment indicator: • X'00000001': RTM1 • X'00000002': RTM2 • X'00000003': MEMTERM • X'00000004': PER
8 (8)	4	The characters, CPU
12 (C)	4	Logical processor identifier (CPUID)
16 (10)	4	The characters, REGS
20 (14)	64	General purpose registers 0 through 15 at the time of the event
84 (54)	4	The characters, PSW
88 (58)	8	The program status word (PSW) at the time of the event
96 (60)	4	The characters, PASD
100 (64)	2	The primary address space identifier (ASID) at the time of the event
102 (66)	4	The characters, SASD
106 (6A)	2	The secondary ASID at the time of the event
108 (6C)	4	The characters, ARS
112 (70)	64	Access registers 0 through 15 at the time of the event.
176 (B0)	4	The characters, G64H
180 (B4)	64	High halves of general purpose registers 0 through 15 at the time of the event
244 (F4)	variable	One of the following, as indicated by the RTM/SLIP processing environment indicator at offset 4 of the buffer: • The system diagnostic work area (SDWA), if offset 4 is 1 (RTM1) • The recovery termination manager 2 (RTM2) work area (RTM2WA), if offset 4 is 2 (RTM2) • The address space control block (ASCB), if offset 4 is 3 (MEMTERM) • The PER interrupt code and PER address, if offset 4 is 4 (PER)
4076 (FEC)	4	The characters, P16
4080 (FF0)	16	The 16-byte program status word (PSW) at the time of the event.

Chapter 3. Transaction dump

A transaction dump provides a representation of the virtual storage for an address space when an error occurs. Typically, an application requests the dump from a recovery routine when an unexpected error occurs. Transaction dumps are requested as follows:

- **Synchronous transaction dump:**

  The requester's IEATDUMP macro invocation issues an instruction to obtain the dump under the current task. IEATDUMP returns control to the requester and is available once the dump data has been written into a dump data set.

Each Transaction dump also contains a summary dump, if requested. The summary dump supplies copies of selected data areas taken at the time of the request. Specifying a summary dump also provides a means of dumping many predefined data areas simply by specifying one option. This summary dump data is not mixed with the Transaction dump because in most cases it is chronologically out of step. Instead, each data area selected in the summary dump is separately formatted and identified. IBM recommends that you request summary dump data.

This section includes information system programmers need to know about Transaction dump and Transaction dump processing:

- "Choices for IEATDUMP Data Sets" on page 50
- "Obtaining transaction dumps" on page 52
- "Printing, viewing, copying, and clearing a dump data set" on page 52
- "Contents of transaction dumps" on page 52

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information any programmer needs to know about programming the IEATDUMP macro to obtain a Transaction Dump:

- Deciding when to request a Transaction dump
- Understanding the types of Transaction Dumps that MVS produces
- Designing an application program to handle a specific type of Transaction dump
- Identifying the data set to contain the dump
- Defining the contents of the dump
- Suppressing duplicate Transaction dumps using dump analysis and elimination (DAE)

Planning data sets for transaction dumps

Transaction dump processing stores data in dump data sets that you preallocate manually, or that are allocated automatically, as needed. The output dump data set has the attributes of RECFM=FB and LRECL=4160.

Planning data set management for transaction dumps

For transaction dumps, use extended format sequential data sets because they have the following characteristics:

- Greater capacity than sequential data sets
- Striping and compression support.

For more information on extended format sequential data sets, see "Choosing SVC dump data sets" on page 13.

Sequential data sets can use large format data sets (DSNTYPE=LARGE).

Extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. Transaction dump fully supports placing dump data sets in cylinder-managed space.

Using preallocated dump data sets

To specify a pre-allocated data set, specify the DDNAME parameter that identifies a data set. The data set must contain sufficient space in one for more extents for the entire dump to be written. DDNAME does not have a 2GB size restriction for the size of the dump. If the data set does not contain sufficient space, a partial dump is returned.

Setting up allocation authority

To allocate dump data sets automatically, the caller's and/or DUMPSRV address space must have authority to allocate new data sets. Do the following:

1. **Associate the caller's and/or DUMPSRV address space with a user ID.**

   If you have RACF Version 2 Release 1 installed, use the STARTED general resource class to associate the caller or DUMPSRV with a user ID. For this step, the RACF started procedures table, ICHRIN03, must have a generic entry.

   If you have an earlier version of RACF, use the RACF started procedures table, ICHRIN03.
2. **Authorize caller's or DUMPSRV user ID to create new dump data sets using the naming convention in the following topic.**

   With the high-level qualifier of SYS1, the data sets are considered *group* data sets. You can assign CREATE group authority to the caller's user ID within that group.

See the following references for more information:

- *z/OS Security Server RACF System Programmer's Guide* for information about the RACF started procedures table.
- *z/OS Security Server RACF Security Administrator's Guide* for information on using the STARTED general resource class and on controlling creation of new data sets.

Choices for IEATDUMP Data Sets

Transaction dump processing supports both pre-allocated and automatically allocated dump data sets. The dump is allocated from the generic resource SYSALLDA. IEATDUMP processes the dump data sets in the following ways:

- For pre-allocated data sets, IEATDUMP writes to the data set without first capturing the dump into a data space. The dump can contain more than 2 GB if the data set capacity permits.
- For automatically allocated data sets, IEATDUMP processes the dump data sets depending on whether the dump section number symbol `&DS.` is used on the end of the data set name pattern:
  - If `&DS.` is not used on the end of the data set name pattern, IEATDUMP captures the dump and stores it in a data space; the data set is then allocated with the space required to contain the captured data; and the dump is written to disk. The dump cannot exceed 2 GB.

    If dynamic allocation fails, message IEA820I is issued, and the dump is deleted.
  - If `&DS.` is used on the end of the data set name pattern, IEATDUMP does not first capture the dump to a data space. Instead, IEATDUMP writes the dump directly to disk. If the size limit of the date set is reached, IEATDUMP allocates another dump with a higher value for `&DS.`. Each data set has an extent size of 500 M that can be changed using ACS routines. These extents are written to until the disk runs out of space or no more extents can be created. At that time, a new data set in the sequence is created. Multi-data set IEATDUMPs utilize up to 999 data sets. The maximum size depends on the amount of space on the volumes where these data sets get allocated. Before IPCS can process the data, you must combine all the data sets into one data set using IPCS COPYDUMP.

Naming automatically allocated dump data sets

The application has control of the name of the data sets created by the automatic allocation function, and you can select a name-pattern to allow for dump data set organization according to your needs. The name is determined through an installation-supplied pattern on the DSN(AD) keyword in the IEATDUMP macro.

Names must conform to standard MVS data set naming conventions and are limited to 44 characters, including periods used as delimiters between qualifiers. A set of symbols is available so that you can include the following kinds of information in the names of your automatically allocated dump data sets:

- System name
- Sysplex name
- Job name
- Local and GMT time and date
- Dump section number

For a complete list of the symbols you can use, see the explanation of DUMPDS NAME= in *z/OS MVS System Commands*.

Note:

1. The `&SEQ.` symbol is not supported for IEATDUMPs.
2. You can use the `&DS.` symbol for splitting the dump between several data sets. When the `&DS.` symbol is added to the end of the DSN name pattern, the transaction dump data can be placed into as many as 999 automatically-allocated 500M-extent data sets. Note that you must combine all the data sets into one data set using IPCS COPYDUMP before IPCS can process the data.

When determining the pattern for the dump data set names, consider any automation tools you may have at your installation that work on dump data sets.

Figure 22 on page 51 describes a SPFUSER name pattern. Note that the symbols are resolved into date and time, so they are preceded by an alphabetic character to conform to MVS data set name requirements. Also, the symbol starts with an ampersand (&) and ends with a period (.), resulting in a name pattern that has double periods when a symbol finishes a qualifier. One period ends the symbol, and the second serves as the delimiter between qualifiers of the generated data set name.

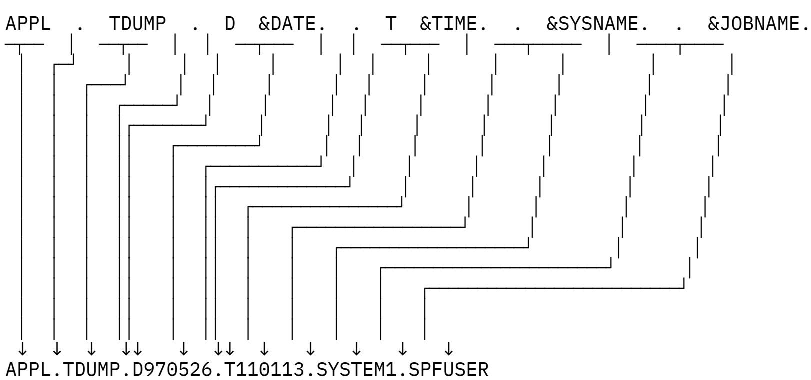


Figure 22. SPFUSER name pattern for automatically allocated dump data set

Automatically allocated dump data sets are not added to the system's sysplex dump directory, as it is for SVC dumps.

Communication from the system

The system communicates about automatic allocation of dump data sets using three messages:

- IEA827I is issued when a complete or partial dump multi-data set dump is taken. IEA827I is an informational message, it will not be issued highlighted.
- IEA822I is issued when a complete or partial dump is taken. IEA822I is an informational message, it is not issued highlighted.

- IEA820I is issued once per Transaction dump when the dump cannot be taken or allocation fails. IEA820I is an informational message, it will not be issued highlighted.

Obtaining transaction dumps

Obtain a Transaction dump by issuing a IEATDUMP macro in an authorized or unauthorized program.

In a sysplex, authorized applications might need dumps from more than one address space to collect all of the problem data. These dumps need to be requested at the same time. To request these multiple dumps, issue a IEATDUMP macro with a REMOTE parameter specifying the other address spaces involved in the problem. To help you set up these requests, the parameter can contain wildcards. If the installation gives names that form patterns to the systems in the sysplex and to jobs for associated work, you can use wildcards, * and ?, to specify the names. For example, use the name TRANS? for the jobnames TRANS1, TRANS2, and TRANS3 and the name TRANS* for TRANS1, TRANS12, and TRANS123.

Note: If a Transaction dump uses the REMOTE parameter to dump one or more address spaces on a pre-release 4 system, the result will be a single SVC dump containing the requested data, instead of one or more Transaction dumps written to data set names specified with the DSN parameter. Issue the DISPLAY DUMP,STATUS command to determine the name of this SVC dump.

Printing, viewing, copying, and clearing a dump data set

Transaction Dumps are unformatted when created. Use IPCS to format a dump and then view it at a terminal or print it. For example, for a pre-allocated data set or a dump data set, the JCL shown in Figure 23 on page 52does the following:

- Uses the Transaction dump in the APPL.TDUMP00 data set. The IPCSTDMP DD statement identifies this data set.
- Deletes the IPCS dump directory in the DELETE(DDIR) statement. This statement uses the USERID of the batch job in the directory identification.
- Allocates the dump directory through the BLSCDDIR statement. The default is volume VSAM01. The example shows VSAM11. Override the default volume with the desired volume.
- Formats the dump using the IPCS subcommands in LIST 0. To use this example, replace the LIST 0 command with the desired IPCS subcommands or a CLIST. See *z/OS MVS IPCS User's Guide* for CLISTs.

```
//IPCSJOB  JOB
//IPCS     EXEC PGM=IKJEFT01,DYNAMNBR=75,REGION=1500K
//SYSPROC  DD   DSN=SYS1.SBLSCLI0,DISP=SHR
//IPCSTDMP DD   DSN=APPL.TDUMP00,DISP=SHR
//SYSUDUMP DD   SYSOUT=*
//IPCSTOC  DD   SYSOUT=*
//IPCSPRNT DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
DELETE(DDIR) PURGE CLUSTER
BLSCDDIR VOLUME(VSAM11)
IPCS NOPARM
SETDEF DD(IPCSTDMP) LIST NOCONFIRM
LIST 0
END
/*
```

Figure 23. Example: JCL to Print, Copy, and Clear the Dump Data Set

Contents of transaction dumps

Transaction Dumps share parmlib member IEADMR00 to establish the dump options list at system initialization. The IBM-supplied IEADMR00 parmlib member specifies dump options NUC, SQA, LSQA, SWA, TRT, RGN, and SUM.

See *z/OS MVS IPCS Commands* for examples of IPCS output formatted from Transaction Dumps.

Customizing transaction dump contents

You can customize the contents of a Transaction dump to meet the needs of your installation. For example, you might want to add areas to be dumped, reduce the dump size, or dump Hiperspaces. In most cases, you will customize the contents of a Transaction dump through the SDATA parameter of the IEATDUMP macro.

Hiperspaces: Transaction Dumps do not include Hiperspaces. To include Hiperspace data in a Transaction Dump, you have to write a program to copy data from the Hiperspace into address space storage that is being dumped.

Adding areas: If the dump, as requested, will not contain all the needed areas, see one of the following for ways to add the areas:

- "Customized contents using the SDATA parameter" on page 53
- "Contents of summary dumps in transaction dumps" on page 55

Customized contents using the SDATA parameter

The IBM-supplied default contents and the contents available through customization are detailed in Table 13 on page 53. The tables show dump contents alphabetically by the parameters that specify the areas in the dumps. Before requesting a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump; the order of the symbols is not important.

D
IBM-supplied default contents

M
Available on the macro that requests the dump

P
Available in the parmlib member that controls the dump options

X
Available on the CHNGDUMP operator command that changes the options for the dump type

blank
No symbol indicates that the area cannot be obtained.

Note: System operator commands and assembler macros use the parameters in the table to specify dump contents.

Table 13. Customizing transaction dump contents through the SDATA Parameter

SDATA Parameter Option	Dump Contents	Transaction dump for IEATDUMP Macro
ALLNUC	The DAT-on and DAT-off nucleuses	M P X
CSA	Common service area (CSA) (that is, subpools 227, 228, 231, 241) and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETCOMMON,DUMP=LIKECSA` • `IARCP64 COMMON=YES,DUMP=LIKECSA` • `IARST64 COMMON=YES,TYPE=PAGEABLE`	M P X
DEFS	Default areas LSQA, NUC, PSA, RGN, SQA, SUM, SWA, TRT	M
ALL	CSA, GRSQ, LPA, NUC, RGN, SQA, SUM, SWA, TRT	X

Table 13. Customizing transaction dump contents through the SDATA Parameter (continued)

SDATA Parameter Option	Dump Contents	Transaction dump for IEATDUMP Macro
GRSQ	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks	M P X
IO	Input/output supervisor (IOS) control blocks for the task being dumped: • EXCPD • UCB	D
LPA	Active link pack area (LPA): module names and contents	M P X
LSQA	Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255), and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETSTOR,DUMP=LIKELSQA` • `IARCP64 COMMON=NO,DUMP=LIKELSQA` • `IARST64 COMMON=NO`	D M P X
NUC	Read/write portion of the control program nucleus (that is, only the non-page-protected areas of the DAT-on nucleus), including: • CVT • LSQA • PSA • SQA	M P X
PSA	Prefixed save areas (PSA) for the processor at the time of the error or the processor at the time of the dump	D M P
RGN	Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 230, 236, 237, 244, 249, 251 - 255, and virtual storage for 64-bit addressable memory objects created using the following services: • `IARV64 REQUEST=GETSTOR,DUMP=LIKERGN` • `IARV64 REQUEST=GETSTOR,SVCDUMPRGN=YES` • `IARCP64 COMMON=NO,DUMP=LIKERGN` • `IARST64 COMMON=NO` .	M P X
SQA	System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) and virtual storage for 64-bit addressable memory objects created using one of the following services: • `IARV64 REQUEST=GETCOMMON,DUMP=LIKESQA` • `IARCP64 COMMON=YES,DUMP=LIKESQA` • `IARST64 COMMON=YES,TYPE=FIXED` • `IARST64 COMMON=YES,TYPE=DREF`	D M P X
SUM	Summary dump (See “Contents of summary dumps in transaction dumps” on page 55.)	D M P X
SWA	Scheduler work area (SWA) (that is, subpools 236 and 237)	M P X

Table 13. Customizing transaction dump contents through the SDATA Parameter (continued)

SDATA Parameter Option	Dump Contents	Transaction dump for IEATDUMP Macro
TRT	System trace, generalized trace facility (GTF) trace, and master trace, as available	D M P X
Default system data	Instruction address trace, if available	D
Default system data	Nucleus map and system control blocks, including: • ASCB for each address space being dumped • ASVT • Authorization table for each address space • CVT, CVT prefix, and secondary CVT (SCVT) • Entry tables for each address space • GDA • JSAB of each address space being dumped • Linkage stack • Linkage table for each address space • PCCA and the PCCA vector table • TOT • TRVT • UCB	D
Default system data	DFP problem data, if DFP Release 3.1.0 or a later release is installed	D
Default system data	Storage for the task being dumped and program data for all of its subtasks	D
Default system data	Storage: 4 kilobytes before and 4 kilobytes after the address in the PSW at the time of the error	D

Contents of summary dumps in transaction dumps

When you request a summary dump, the **SUM** parameter requests many useful, predefined areas with one parameter.

Summary dump does not contain volatile system information. The system writes the summary dump before it returns control to the dump requester; the summary information is saved for each address space that is being dumped.

The Summary Dump contains:

1. ASID record for the address space of the dump task
2. Control blocks for the recovery termination manager (RTM):

   RTM2WA associated with all TCBs in the dumped address space
3. Dump header, which is mapped by AMDDATA.

   For the AMDDATA mapping, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
4. 4 kb before and 4 kb after:

   All valid unique addresses in the PSWs in the RTM2WAs shown in the dump
   All valid unique addresses in the registers in the RTM2WAs shown in the dump
5. Supervisor control blocks:
   - Current linkage stack
   - Primary address space number (PASN) access list

- Work unit access list

For information about control blocks that are listed in Table 13 on page 53, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Customizing contents through operator commands

The dump options list for Transaction Dumps can be customized through a CHNGDUMP operator command by all the ways shown in Table 14 on page 56.

Nucleus areas in dumps: Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used, Transaction dump for an IEATDUMP macro contains the nucleus map and certain control blocks.

Most problems can be debugged without dumping the nucleus. If a problem arises that requires the nucleus be dumped, use the CHNGDUMP operator command to add the NUC SDATA option to all IEATDUMPs. This also applies to other options.

DAT, dynamic address translation, is the hardware feature that enables virtual storage. In the DAT-on part of the nucleus, the addresses are in virtual storage; in the DAT-off part of the nucleus, the addresses are in central storage.

Table 14. Customizing transaction dump contents through operator commands

Customization	Effect	Example
Updating IEADMR00 parmlib member	**Change occurs**: At system initialization **What changes**: This parmlib member establishes the dump options for Transaction dumps for IEATDUMP macros and SDATA. See “Contents of transaction dumps” on page 52 for the list.	To add the link pack area (LPA) to all Transaction dumps for IEATDUMP macros and SDATA, while keeping the local system queue area (LSQA) and trace data, change the line in IEADMR00: `SDATA=(LSQA,TRT,LPA)`
Adding the CHNGDUMP operator command in IEACMD00 parmlib member	**Change occurs**: At system initialization **What changes**: This command establishes the dump options for Transaction dumps for IEATDUMP macros and SYSMDUMPS. See “Contents of transaction dumps” on page 52 for the list.	To add the link pack area (LPA) to all Transaction dumps for IEATDUMP macros and SYSMDUMP, while keeping the local system queue area (LSQA) and trace data, add the following command to IEACMD00: `CHNGDUMP SET,SYSMDUMP=(LPA)`

Table 14. Customizing transaction dump contents through operator commands (continued)

<table>
<tr><th>Customization</th><th>Effect</th><th>Example</th></tr>
<tr>
<td>Entering CHNGDUMP operator command with SYSMDUMP parameter on a console with master authority</td>
<td>Change occurs: Immediately when command is processed

What changes:
For the ADD mode: CHNGDUMP options are added to the current Transaction dump options list and to any options specified in the macro or operator command that requested the dump. The options are added to all Transaction dumps for IEATDUMP macros and SYSMDUMP, until another CHNGDUMP SYSMDUMP operator command is entered.
For the OVER mode: CHNGDUMP options are added to the current Transaction dump options list. The system ignores any options specified in the macro or operator command that requested the dump. The options override all Transaction dumps for the IEATDUMP macro and SYSMDUMP, until a CHNGDUMP SYSMDUMP,ADD operator command is entered.
For the DEL option: CHNGDUMP options are deleted from the Transaction dump options list.
When more than one CHNGDUMP operator command with IEATDUMP is entered, the effect is cumulative.</td>
<td>To add the LPA to all Transaction dumps for the IEATDUMP macro and SYSMDUMP, until changed by another CHNGDUMP SYSMDUMP, enter:<pre><code>CHNGDUMP SET,SYSMDUMP=(LPA)</code></pre>
To add the CHNGDUMP IEATDUMP options list to all Transaction dumps:<pre><code>CHNGDUMP SET,SYSMDUMP,ADD</code></pre>
To override all Transaction dumps with the CHNGDUMP IEATDUMP options list:<pre><code>CHNGDUMP SET,SYSMDUMP,OVER</code></pre>
To remove LPA from the IEATDUMP options list:<pre><code>CHNGDUMP
DEL,SYSMDUMP=(LPA)</code></pre></td>
</tr>
</table>

Chapter 4. Stand-alone dump

The stand-alone dump program (SADMP) produces a stand-alone dump of storage that is occupied by one of the following:

- A system that failed.
- A stand-alone dump program that failed.

  Either the stand-alone dump program dumped itself — a **self-dump** —, or the operator loaded another stand-alone dump program to dump the failed stand-alone dump program.

The stand-alone dump program and the stand-alone dump together form what is known as the stand-alone dump service aid. The term stand-alone means that the dump is performed separately from normal system operations and does not require the system to be in a condition for normal operation.

The stand-alone dump program produces a high-speed, unformatted dump of all of central storage, plus it can include user tailorable parts of paged-out virtual storage. The user generated stand-alone dump program must reside on a storage device that can be used to IPL from.

Produce a stand-alone dump when the failure symptom is a wait state with a wait state code, a wait state with no processing, an instruction loop, or slow processing.

You create the stand-alone dump program that dumps the storage. Use the AMDSADMP macro to produce the following:

- A stand-alone dump program that resides on **DASD**, with output directed to a tape volume or to a DASD dump data set
- A stand-alone dump program that resides on **tape**, with output directed to a tape volume or to a DASD dump data set.

A stand-alone dump supplies information that is needed to determine why the system or the stand-alone dump program failed.

You can create different versions of the stand-alone dump program to dump different types and amounts of storage. To create the different versions, code several AMDSADMP macros by varying the values of keywords on the macros.

Before you begin, also consider reading the following topics:

- For a set of best practices for optimizing stand-alone dump (SADMP) data capture, optimizing problem analysis time, and ensuring that the stand-alone dump is successful at capturing the necessary information for use by IBM Support, see the topic about Best practices for large stand-alone dump in *z/OS Problem Management*.
- To enable your operators and the system to respond appropriately to disabled wait states, consider activating the AutoIPL function, see the topic about Using the automatic IPL function in *z/OS MVS Planning: Operations*.

This information covers the following topics, which describe how to use stand-alone dump:

Planning for stand-alone dump

There are several decisions you need make when planning for a stand-alone dump. You implement most of these decisions when you create the stand-alone dump program, either when you code the AMDSADMP macro, when you assemble the macro, or when you use the SADMP option on the IPCS Dialog. Some typical questions follow.

Should I take a stand-alone dump to DASD or to tape?

When choosing an output device for stand-alone dump, consider the need for operator intervention, the amount of operator intervention involved, and the amount of time the system will be unavailable.

You can reduce the level of operator intervention during stand-alone dump processing by dumping to DASD. With an automation package set up to IPL the stand-alone dump program from DASD, stand-alone dump can be run from a remote site. When you dump to tape, an operator is required to handle other aspects of dumping, such as mounting or changing tapes, unless the tape is in an IBM Virtual Tape Server (VTS).

The system is unavailable when a stand-alone dump is taken. The amount of time the system is unavailable depends upon the size of the dump. See "Dumping to a DASD data set" on page 78 for more information.

If I do dump to DASD, how much space do I need?

The maximum size of a single-volume DASD dump data set depends on the type of data set.

- Conventional sequential (DSNTYPE=BASIC) data sets can span 65,535 tracks per volume, and can hold approximately 3 GB per volume.
- Extended format (DSNTYPE=EXTREQ) data sets are supported by z/OS V1R6 and later releases. Extended format sequential data can hold 4,294,967,295 blocks per volume. The maximum size for extended format sequential is approximately 98,304 GB per volume. You cannot use striping or compression options for extended format sequential data sets. You must use the guaranteed free space option to require DFSMS to reserve space at the time that the data set is created.
- Large format data sets are supported by z/OS V1R7 and later releases. Large format (DSNTYPE=LARGE) data sets can span 16,777,215 tracks per volume. The maximum size for large format data sets is 768 GB per volume.

If you require more space than you want to allocate on a single volume, you can define a multi-volume DASD dump data set that can span up to 32 volumes of the same device type.

Note: Beginning in z/OS V1R12, SADMP supports placement of dump data sets in cylinder-managed space. In releases prior to z/OS V1R12, stand-alone data sets must remain in track-managed space.

Use the AMDSADDD REXX utility or the SADMP dump data set utility on the IPCS dialog to allocate and initialize a single volume DASD dump data set or a multi-volume DASD dump data set. This prepares the data set for use by the system where initialization is performed and for other systems that have access to the same data set using the same device numbers. For more information, see:

- "Using the AMDSADDD utility" on page 79
- SADMP option on the IPCS Dialog in *z/OS MVS IPCS User's Guide*

When using a multi-volume DASD dump data set, the device number of the first volume is specified. The other volumes are located by stand-alone dump using the information that is placed in the data set when it is initialized. All volumes are written concurrently by stand-alone dump. The data set is rejected if stand-alone dump is unable to access all volumes of the data set or if invalid control information is read from the data set during initialization.

If you do not allocate enough space in your dump data set, the stand-alone dump program prompts the operator to continue dumping to another DASD dump data set or tape volume. You can continue dumping to any stand-alone dump supported device, however, after a tape device is selected, it must be used to complete the dump even though multiple volumes might be required.

IBM recommends that you allocate multiple dump data sets to perform a complete stand-alone dump.

Can I dump to multiple dump data sets?

Stand-alone dump does allow you to dump to multiple dump data sets. By coding the DDSPROMPT=YES keyword on the AMDSADMP macro, you can generate a stand-alone dump program that allows run-time dump data set prompting.

When the Stand-alone dump program is initiated, message AMD001A is issued to prompt the operator for an output device. If a DASD device is specified and run-time dump data set prompting is active, message AMD002A is issued to prompt the operator for a dump data set name. Providing the dump data set is validly allocated and initialized on the output device, the stand-alone dump program uses the dump data set name specified. If message AMD099I is issued indicating that the dump data set is full, the operator can continue dumping to any stand-alone dump supported DASD dump data set or tape device by replying to message AMD001A (and possibly AMD002A) again. After the dump completes, message AMD104I is issued to indicate the entire set of devices and/or dump data sets that were used during the taking of the dump.

By coding DDSPROMPT=NO on the AMDSADMP macro, the stand-alone dump program is generated without run-time dump data set prompting. In this case, replying to message AMD001A with a DASD device causes the stand-alone dump program to assume that the output dump data set is named SYS1.SADMP.

Note:

1. Use the AMDSADDD REXX or the IPCS SADMP dump data set utilities to allocate and initialize the stand-alone dump data sets.
2. The stand-alone dump program must locate the dump data set on the device that is specified. Therefore, it is imperative that the necessary data set management steps be taken so that the stand-alone dump data sets are not placed into a migrated state or moved to a different volume. The dump data sets must also be exempt from any space management processing that releases unused space.
3. You can continue a dump to any stand-alone dump supported device, however, after a tape device is selected, it must be used to complete the dump even though multiple tape volumes might be required.

See the following topics for more information:

- For more information on dump data set processing, see the description of the DDSPROMPT keyword in the “Syntax of the AMDSADMP macro” on page 68.
- For more information on how to use multiple dump data sets with IPCS, see “Copying from multiple dump data sets” on page 104.
- For more information on performing tasks associated with creating, clearing, and reallocating SADMP data sets on DASD, see the SADMP option on the IPCS Dialog in *z/OS MVS IPCS User's Guide*.

What can I name my DASD dump data sets?

A stand-alone dump dump data set can be any valid MVS data set name, however, stand-alone dump has two requirements that are checked at both generation time and run-time:

- The data set name must be 44 characters or less
- The data set name must contain the text 'SADMP' as either part of, or as an entire data set qualifier

In addition, because the generation process does not perform any allocation on the output device or dump data set name, it is imperative that you ensure that the data set name specified on the OUTPUT= keyword matches exactly the dump data set name allocated by the AMDSADDD REXX or IPCS SADMP data set utilities. The following are some additional rules to follow when specifying a dump data set name:

- The data set name specified should be fully qualified (without quotation marks)
- The alphabetic characters in the dump data set name should be specified as capital letters

How much of the system should I dump?

The situation dictates the amount of information you need to diagnose the failure. You can use the DUMP keyword to control the amount of storage you want dumped. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 74 for more information.

When should I specify the dump tailoring options?

The most flexible way to specify the dump options for a stand-alone dump is to specify, on the DUMP keyword of the AMDSADMP macro, those areas of storage you normally always want dumped and additionally allow the operator who requests the dump to specify additional options by coding the PROMPT keyword on the AMDSADMP macro. In most cases, to simplify the dumping process, it is best to define any installation specific dump options on the AMDSADMP macro and not use the PROMPT keyword. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 74 for more information.

What type of security does the stand-alone dump program require?

After the stand-alone dump program is properly created on a DASD residence volume, it resides in the SYS1.PAGEDUMP.Vvolser data set. To ensure that the stand-alone dump program is available and processes successfully, do not delete the data set or move it to another volume or pack. To protect the stand-alone dump program in SYS1.PAGEDUMP.Vvolser, use a password or a security product, such as RACF. If the data set is not protected, unauthorized users can read the dump data in SYS1.PAGEDUMP.Vvolser. Also consider protecting the stand-alone dump output dump data sets from unauthorized reading.

See *z/OS Security Server RACF Security Administrator's Guide* for more information about protecting a data set.

Should I use IEBGENER or the COPYDUMP subcommand to copy a dump to a data set?

The recommended method is IPCS COPYDUMP. IPCS COPYDUMP can run without a dump directory being employed. Use the DEFER option when initiating the IPCS session to tell IPCS to defer accessing a dump directory until one is required. IPCS COPYDUMP has the ability to merge the records from a multi-volume SADMP and recapture the prioritized order used by SADMP to get the most important data into the dump data sets first.

If SADMP is allowed to complete normally, IEBGENER and similar transcription programs can produce a logically complete dump data set that IPCS can process. However, IPCS performance, particularly IPCS dump initialization, degrades as more volumes are added to the SADMP data set.

What is dumped when I run the stand-alone dump program?

The default dump contains all areas of central storage and some areas of virtual storage that are not backed by central storage. The output of the stand-alone dump program includes:

- The prefixed save areas (PSA)
- The nucleus and extended nucleus
- The system queue area (SQA) and the extended SQA
- The common service area (CSA) and the extended CSA
- Subpools 203-205, 213-215, 229, 230, 236, 237, 247, 248, and 249 for all address spaces
- The local system queue area (LSQA) and the extended LSQA for eligible address spaces
- The dump title provided by the operator; otherwise, the dump is untitled
- The processor STORE STATUS information for each processor
- Central storage from address 0 to the top of main storage (some blocks might be missing because of offline storage elements)

- Virtual storage areas selected by the DUMP keyword, or selected by the operator at runtime.
- A message log, normally consisting of all console messages issued by the dump program, including suppressed messages. (To format and print the stand-alone dump message log, use the VERBEXIT SADMPMSG subcommand or the SADMPMSG option of the IPCS dialog.)
- High virtual for the TRACE address space.
- High virtual for the DUMPSRV address space.
- High virtual for the GRS address space.
- Dump records summarizing the zeroed pages in the dump
- The full generalized trace facility (GTF) address space
- Subpool 127 in the GRS address space
- Data spaces whose names begin with ISG for the GRS address space
- All of DUMPSRV's data spaces
- The full cross-system coupling facility (XCF) address space
- All of XCF's dataspaces
- XES-related dataspaces for address spaces with an XES connection

Note that this list does not imply an order of the stand-alone dump process. During stand-alone dump processing, several different messages are issued to indicate the progress of the dumping:

- For real dump processing, AMD005I is issued.
- For both real and virtual dump processing, AMD095I is issued every 30 seconds, followed by message AMD056I indicating that dumping of virtual storage has completed and AMD104I to indicate what output devices and/or dump data sets were used by the stand-alone dump program.

Can I use my current version of the stand-alone dump program to dump a new version of z/OS?

Always use the stand-alone dump version that is generated from the same release of z/OS that you want to dump. IBM does not guarantee that a different level of stand-alone dump will successfully dump anything other than the level of z/OS it was designed for. The new version of z/OS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

When migrating to a new version of z/OS, IBM strongly recommends that you generate a new version of the stand-alone dump program built from the new z/OS system data sets. See "One-stage generation" on page 86 for more information.

Creating the stand-alone dump program

The first step in creating a stand-alone dump program is selecting a tape or DASDas the stand-alone dump IPL volume (residence volume). After you select the residence volume, you can create the stand-alone dump program. To create the stand-alone dump program, you:

1. Code the AMDSADMP macro. See "Coding the AMDSADMP macro" on page 67.
2. Assemble the macro, placing the stand-alone dump program onto the residence volume in ready-to-load form. IBM recommends that you use one-stage generation when building or creating a stand-alone dump program for the currently executing version of MVS. Use the two-stage generation to create multiple stand-alone dump programs and to create a new version of the stand-alone dump program when migrating to a new version of MVS. See "Generating the stand-alone dump program" on page 86.

MNOTES from the AMDSADMP macro

The output listing from the assembly can contain error messages, called MNOTES, that describe errors made while coding the AMDSADMP macro. To respond to one of these messages, check the specification of the macro and run the assembly step again. The meaning of the severity code is as follows:

8
: Assembly processing ends

4
: Warning

0
: Informational

AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: COMPACT=*compact* IS NOT ALLOWED. IT MUST BE YES OR NO. COMPACT=YES HAS BEEN USED.

Explanation: The system could not recognize the value specified on the COMPACT keyword. The stand-alone dump program will use the IDRC feature for the output tape if IDRC is installed.

Severity Code: 0.

AMDSADMP: CONSOLE ADDRESS *conad*
IS INVALID. IT MUST BE A DEVICE NUMBER. 001F IS SUBSTITUTED.

Explanation: The console address operand is not a valid device number of 3 or 4 hexadecimal digits.

Severity Code: 0.

AMDSADMP: CONSOLE PARM NOT
DETECTED. DEFAULT (001F, 3278) WILL BE USED.

Explanation: Either the console parameter was not specified or it was not specified correctly on the continuation statement. The parameter was probably not continued correctly on the next defined statement. Continue the interrupted parameter or field beginning in any column from 4 through 16.

See Continuing JCL Statements in *z/OS MVS JCL Reference*.

Severity Code: 0.

AMDSADMP: CONSOLE TYPE *contp*
IS INVALID. IT MUST BE A 4 DIGIT NUMBER. 3278 HAS BEEN USED.

Explanation: An incorrect console type was specified. Only 3277, 3278, 3279, or 3290 are acceptable.

Severity Code: 0.

AMDSADMP: DEFAULT OUTPUT DEVICE T0282 WILL BE USED.

Explanation: A device number was incorrectly specified, or was not specified, on the OUTPUT= parameter.

Severity Code: 0.

AMDSADMP: IPL=*ipl* IS INVALID.
FIRST CHARACTER MUST BE D OR T,
AND HAS BEEN REPLACED WITH A D.

Explanation: The IPL operand is incorrect. It is not prefixed with a 'D' or a 'T'.

Severity Code: 4.

AMDSADMP: IPL=*ipl* IS TOO LONG.
THE UNIT NAME WILL BE TRUNCATED.

Explanation: The unit name can be at most 8 characters long.

Severity Code: 4.

AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: IPLUNIT WAS NOT SPECIFIED OR IPL= TYPE (D OR T) WAS SPECIFIED INCORRECTLY. UNIT WILL BE DEFAULTED TO SYSDA.

Explanation: The IPL parameter should be specified as IPL=duuu, where 'd' is D for direct access or T for tape, and 'uuu' is a valid unit type or device number for the SADMP IPL volume as described by the UNIT=uuu JCL parameter.

System Programmer Response: A device number consists of 3 or 4 hexadecimal digits.

Severity Code: 0.

AMDSADMP: MSG=*msg* IS INVALID. IT MUST BE ALL, ACTION, OR ALLASIDS. MSG=ALL HAS BEEN USED.

Explanation: The MSG operand is not ALL, ACTION, or ALLASIDS.

Severity Code: 0.

AMDSADMP: DDSPROMPT=*ddsprompt* IS NOT ALLOWED. IT MUST BE YES OR NO. DDSPROMPT=YES HAS BEEN USED.

Explanation: The DDSPROMPT operand is incorrect. It must be either 'YES' or 'NO'. DDSPROMPT=YES is assumed.

System Action: The SADMP program will be generated with run-time dump data set prompting active.

Severity Code: 0.

AMDSADMP: OUTPUT=*output* IS INCORRECT. IT MUST BE EITHER {T|D}UNIT OR (DUNIT,DATA SET NAME).

Explanation: The OUTPUT operand is incorrect. It must be specified in one of the following formats:

- A 'T' or a 'D' followed by a device number
- A 'D' followed by a device number and a data set name pair specified within parentheses.

System Action: Generation continues, using the default for the OUTPUT operand, T0282, regardless of the format used.

System Programmer Response: The output device must be specified as a 3-digit or 4-digit device number. You can change the OUTPUT parameter at run time, if the default is not what you want.

Severity Code: 4.

AMDSADMP: OUTPUT DUMP DATA SET NAME IS INCORRECT. THE DATA SET NAME IS GREATER THAN 44 CHARACTERS.

Explanation: OUPTUT=(*Dunit,ddsname*) was specified, however, the data set name (ddsname) had a length greater than 44 characters.

System Action: Generation continues, however, no default dump data set name will be generated.

System Programmer Response: If a default dump data set name is desired, correct the OUTPUT= specification and regenerate the SADMP program.

Severity Code: 4.

AMDSADMP: OUTPUT DUMP DATA SET NAME IS INCORRECT. IT MUST CONTAIN THE TEXT 'SADMP'.

Explanation: OUPTUT=(*Dunit,ddsname*) was specified, however, the data set name (ddsname) did not contain the text 'SADMP' as either part of, or as an entire data set qualifier.

System Action: Generation continues, however, no default dump data set name will be generated.

System Programmer Response: If a default dump data set name is desired, correct the OUTPUT= specification and regenerate the SADMP program.

Severity Code: 4.

AMDSADMP Messages, Explanations and Severity Codes

**AMDSADMP: REUSEDS=*reuseds* IS NOT ALLOWED.
VALID SPECIFICATIONS ARE NEVER, CHOICE, OR ALWAYS.
REUSEDS=CHOICE HAS BEEN USED.**

Explanation: The REUSEDS operand is not NEVER, CHOICE, or ALWAYS.

System Action: Generation continues, using the default for the REUSEDS operand, CHOICE.

Severity Code: 0.

**AMDSADMP: ULABEL=NOPURGE IS NOT
POSSIBLE FOR A TAPE RESIDENCE VOLUME.**

Explanation: The ULABEL cannot be NOPURGE when the IPL device is tape. SADMP ignores your ULABEL specification.

Severity Code: 8.

**AMDSADMP: *keyword* IS AN OBSOLETE KEYWORD. IT IS IGNORED. SADMP
GENERATION CONTINUES.**

Explanation: An obsolete keyword is specified on the AMDSADMP macro. SADMP no longer requires the LOADPT or TYPE keywords to create a stand-alone dump program.

System Action: The system ignores the keyword and continues processing.

System Programmer Response: To eliminate this MNOTE, remove the indicated keyword and its associated parameter from the generation JCL.

Severity Code: 0.

**AMDSADMP: ALIB=*alib* IS NOT VALID. THE
REQUIRED SYNTAX IS ALIB=(VOLSER,UNIT).**

Explanation: The system could not recognize the parameters specified on the ALIB keyword. The correct syntax is ALIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

**AMDSADMP: NUCLIB=*nuclib* IS NOT VALID. THE REQUIRED
SYNTAX IS NUCLIB=(VOLSER,UNIT).**

Explanation: The system could not recognize the parameters specified on the NUCLIB keyword. The correct syntax is NUCLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

**AMDSADMP: MODLIB=*modlib* IS NOT VALID. THE
REQUIRED SYNTAX IS MODLIB=(VOLSER,UNIT).**

Explanation: The system could not recognize the parameters specified on the MODLIB keyword. The correct syntax is MODLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

AMDSADMP Messages, Explanations and Severity Codes

AMDSADMP: LNKLIB=*lnklib* IS NOT VALID. THE REQUIRED SYNTAX IS MODLIB=(VOLSER,UNIT).

Explanation: The system could not recognize the parameters specified on the LNKLIB keyword. The correct syntax is LNKLIB=(volser,unit), where volser is the volume serial number and unit is the UNIT=value of the device.

System Action: The system ignores this keyword and continues. The second step JCL might be incorrect.

System Programmer Response: Correct the syntax specified on the AMDSADMP macro and resubmit the JCL.

Severity Code: 8.

AMDSADMP: CONSOLE TYPE *contp* IS INVALID. NO VALUE MAY BE SPECIFIED FOR SYSC. IT WILL BE IGNORED.

Explanation: A console type was specified following the console name of SYSC. No console type is allowed for this console.

System Action: The system ignores the specification.

System Programmer Response: None.

Severity Code: 0.

AMDSADMP: CONSOLE ADDRESS SYSC MAY ONLY BE SPECIFIED FOR THE FIRST CONSOLE. IT WILL BE IGNORED.

Explanation: The console name SYSC was not specified as the first console in the console list. SYSC can only be specified as the first console.

System Action: The system ignores the specification.

System Programmer Response: None.

Severity Code: 0.

AMDSADMP: ONLY SYSTEM CONSOLE DEFINED. DEFAULT (001F,3278) WILL ALSO BE USED.

Explanation: The console named SYSC was the only console that was defined. At least one 3270 console must also be defined.

System Action: The system defined a default console of (001F,3278).

System Programmer Response: None.

Severity Code: 0.

AMDSADMP: REAL=ALL OR REAL=USED NOT SPECIFIED, THE DEFAULT REAL=ALL WILL BE USED.

Explanation: Either the REAL keyword was not specified or it was not specified correctly. REAL=ALL is the default.

Severity Code: 0.

AMDSADMP: POSITIONAL *value* IGNORED.

Explanation: A positional value other than PROMPT appeared as the first positional argument to the AMDSADMP macro. It was ignored.

System Programmer Response: None.

Severity Code: 0.

Coding the AMDSADMP macro

This section describes the coding of the AMDSADMP macro, including the following topics:

- "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 74

- “Dumping to a DASD data set” on page 78

Syntax of the AMDSADMP macro

Figure 24 on page 68 shows the syntax of the AMDSADMP macro and its parameters.

```
[symbol] AMDSADMP
[,IPL={Tunit|Dunit|DSYSDA}]
[,VOLSER={volser|SADUMP}]
[,ULABEL={PURGE|NOPURGE}]
[,CONSOLE=({cnum|(cnum,ctype) [,(cnum,ctype)]...|01F,3278})]
[,SYSUT={unit|SYSDA}]
[,OUTPUT={Tunit|Dunit|(Dunit,ddsname)|T0282}]
[,DUMP=('options')][,PROMPT]
[,MSG={ACTION|ALLASIDS|ALL}]
[,MINASID={ALL|PHYSIN}]
[,COMPACT={YES|NO}]
[,REUSEDS={CHOICE|ALWAYS|NEVER}]
[,ALIB=(volser,unit)]
[,NUCLIB=(volser,unit)]
[,MODLIB=(volser,unit)]
[,LNKLIB=(volser,unit)]
[,DDSPROMPT={YES|NO}]
[,AMD029={YES|NO}]
[,IPLEXIST={YES|NO}]
[,ADDSUMM=('options')]
[,REAL={ALL|USED}]
[,LDIPL={YES|NO}]
```

Figure 24. Format of AMDSADMP macro instruction

symbol
An arbitrary name you can assign to the AMDSADMP macro. stand-alone dump uses this symbol to create a job name for use in the initialization step.

AMDSADMP
The name of the macro.

IPL={Tunit|Dunit|DSYSDA}
Indicates the device number, device type, or esoteric name of the stand-alone dump residence volume. The first character indicates the volume type; T for tape, D for DASD.stand-alone dump uses the unit character string as the UNIT=value to allocate the residence volume for initialization.

A device number consists of 1 to 4 hexadecimal digits. To distinguish a device number from a unit type, the device number must be preceded by a slash (/); for example, you could specify `IPL=D/410F`. Otherwise, the dynamic allocation of the IPL device (IPLDEV DD-statement) may fail with reason code X'021C' (unavailable system resource).

The default is IPL=DSYSDA. When you specify IPL=T, stand-alone dump assumes T3400. When you specify IPL=D, stand-alone dump assumes DSYSDA.

Note:

1. This device also contains a work file used during stand-alone dump processing.

2. It is not recommended to place the IPL text of stand-alone dump on a volume that contains page data sets. A restart of stand-alone dump (see "Running the stand-alone dump program" on page 95) hangs during the real dump phase in this case.

VOLSER={volser|SADUMP}
Indicates the volume serial number the system is to use to allocate the residence volume for initialization. When you specify a tape volume, it must be NL (no labels). VOLSER=SADUMP is the default.

ULABEL={PURGE|NOPURGE}
Indicates whether stand-alone dump deletes (PURGE) or retains (NOPURGE) existing user labels on a DASD residence volume.When you specify NOPURGE, the stand-alone dump program is written on cylinder 0 track 0 of the residence volume, immediately following all user labels. If the user labels occupy so much space that the stand-alone dump program does not fit on track 0, the initialization program issues an error message and ends.

ULABEL=NOPURGE is the default.

CONSOLE=({cnum|(cnum,ctype)[,(cnum,ctype)]...|01F,3278})
Indicates the device numbers and device types of the stand-alone dump consoles that stand-alone dump is to use while taking the dump. When you specify CONSOLE=cnum, stand-alone dump assumes (cnum,3278). You can specify from two to 21 consoles by coding:

```
CONSOLE=((SYSC)|(cnum,ctype),(cnum,ctype),[,(cnum,ctype)]...)
```

A device number consists of 3 or 4 hexadecimal digits, optionally preceded by a slash (/). Use a slash preceding a 4-digit device number to distinguish it from a device type.

The 3277, 3278, 3279, and 3290 device types are valid, and are interchangeable.

CONSOLE=(01F,3278) is the default.

You can specify CONSOLE=SYSC for the first console only. SYSC is a constant representing the hardware system console.

Note: The specification of CONSOLE does not affect the availability of the system console.

SYSUT={unit|SYSDA}
Specifies the UNIT=value of the device that stand-alone dump uses for work files during stand-alone dump initialization. You can specify the device as a group name (for example, SYSDA), a device type (for example, 3330), or a unit address (for example, 131). SYSUT=SYSDA is the default.

OUTPUT={Tunit|Dunit|(Dunit,ddsname)|T0282}
Indicates the device type, number, and data set name that stand-alone dump uses as a default value if the operator uses the EXTERNAL INTERRUPT key to bypass console communication, or if the operator provides a null response to message AMD001A during stand-alone dump initialization. OUTPUT=T0282 is the default.

The device type can be specified as either a 'T' for tape or 'D' for DASD.

The device number consists of 3 or 4 hexadecimal digits, optionally preceded by a slash (/). Use a slash preceding a 4-digit device number to distinguish it from a device type.

If the default device is a DASD, you can also set up a default dump data set name to use by specifying both the device and the dump data set name on the OUTPUT= parameter. You can specify the first volume of a multi-volume DASD data set. If you specify a default dump data set name it must:

- Have a length that is 44 characters or less.
- Contain the text 'SADMP' as either part of, or as an entire data set qualifier.

Note that AMDSADMP processing does not allocate the data set or check to see that a valid MVS data set name has been provided. Therefore, you should insure that:

- The AMDSADDD REXX is used to allocate and initialize the same data set name specified on the OUTPUT= keyword.
- The data set name specified should be fully qualified (without quotation marks).
- The necessary data set management steps are taken so that the stand-alone dump data sets are not placed into a migrated state or moved to a different volume.
- Alphabetic characters appearing in the dump data set name should be specified as capital letters.

If the default DASD device is to be used and no dump data set name is provided, the stand-alone dump program assumes that the default dump data set name is SYS1.SADMP if the DDSPROMPT=NO parameter was also specified. Otherwise, if DDSPROMPT=YES was specified, the stand-alone dump program prompts the operator at runtime for a dump data set name to use.

Note:

1. At run-time, only a null response to message AMD001A causes the stand-alone dump program to use the default device and/or dump data set name.
2. Do not place a data set that is intended to contain a stand-alone dump on a volume that also contains a page data set that the stand-alone dump program might need to dump. When stand-alone dump initializes a page volume for virtual dump processing, it checks to see if the output dump data set also exists on this volume. If it does, the stand-alone dump program issues message AMD100I and does not retrieve any data from page data sets on this volume. Thus, the dump might not contain all of the data that you requested. This lack of data can impair subsequent diagnosis.
3. You cannot direct output to the stand-alone dump residence volume.

DUMP='*options*'
Indicates additional virtual storage that you want dumped. This storage is described as address ranges, dataspaces, and subpools in address spaces. When you do not specify DUMP, stand-alone dump does not dump any additional storage unless you specify PROMPT. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 74 for more information.

PROMPT
Causes stand-alone dump, at run time, to prompt the operator for additional virtual storage to be dumped. The operator can respond with the same information that can be specified for the DUMP keyword. When you do not specify PROMPT, stand-alone dump does not prompt the operator to specify additional storage. See "Using the DUMP or ADDSUMM keyword to request additional storage or address spaces" on page 74 for more information.

MSG={ACTION|ALLASIDS|ALL}
Indicates the type of stand-alone dump messages that appear on the console. When you specify ACTION, stand-alone dump writes only messages that require operator action. When you specify ALL, stand-alone dump writes most messages to the console. However, messages AMD010I, AMD057I, AMD076I, AMD081I, and AMD102I appear only in the stand-alone dump message log. When you specify ALLASIDS, the stand-alone dump program behaves as if MSG=ALL was specified, except that message AMD010I also appears on the console. ALL is the default.

This keyword has no effect on the stand-alone dump message log; even if you specify MSG=ACTION, the stand-alone dump virtual dump program writes all messages to the message log in the dump.

MINASID={ALL|PHYSIN}
Indicates the status of the address spaces that are to be included in the minimal dump. Specify PHYSIN to dump the minimum virtual storage (LSQA and selected system subpools) for the physically swapped-in address spaces only. Specify ALL to dump the minimum virtual storage (LSQA and selected system subpools) for all of the address spaces. ALL is the default. At run time, if PHYSIN was specified, stand-alone dump writes message AMD082I to the operator's console to warn the operator that some virtual storage might be excluded from the dump.

COMPACT={YES|NO}
COMPACT(YES) compacts the data stored on a tape cartridge if theIDRC hardware feature is available on your tape drive. If the IDRC feature is available and you do not specify the COMPACT keyword, the default is YES, so that IDRC compacts the dump data. Otherwise, the data is handled as usual.

REUSEDS={CHOICE|ALWAYS|NEVER}
Indicates whether stand-alone dump should reuse the dump data set on the specified output device when it determines that the data set is valid, however, it can contain data from a previous dump. Stand-alone dump determines this by checking to see if the first record in the data set matches the record that is written by the AMDSADDD rexx utility. When you specify ALWAYS, stand-alone dump issues message AMD094I and reuses the specified dump data set. When you specify NEVER, stand-alone dump issues message AMD093I and prompts the operator, through message AMD001A, for an output device. When you specify CHOICE, stand-alone dump informs the operator, with message AMD096A, that the data set is not reinitialized and requests permission to reuse the data set. See for more information about defining, clearing, and reallocating the dump data set.

CHOICE is the default.

ALIB=(volser,unit)
Specifies the volume serial number and UNIT=value of the volume that contains all of the following system data sets:

- SYS1.MODGEN
- SYS1.LINKLIB
- SYS1.NUCLEUS

This parameter is valid only when you are generating the stand-alone dump program using two-stage generation.

Note: The specification of the NUCLIB, LNKLIB, or MODLIB parameters overrides the corresponding value specified on the ALIB parameter.

See "Using two-stage generation of stand-alone dump when migrating" on page 93 for information on the use of this parameter.

NUCLIB=(volser,unit)
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.NUCLEUS. If you specify NUCLIB, there is no need to specify IPLTEXT, IPITEXT, DVITEXT, DPLTEXT and PGETEXT DD statements. Beginning with z/OS V1R12, this parameter is valid for one-stage generation JCL. Prior to z/OS V1R12, this parameter is valid only when you generate the stand-alone dump program using two-stage generation. See "One-stage generation" on page 86 for information on the use of this parameter.

MODLIB=(volser,unit)
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.MODGEN. This parameter is valid only when you generate the stand-alone dump program using two-stage generation. See "Using two-stage generation of stand-alone dump when migrating" on page 93 for information on the use of this parameter.

LNKLIB=(volser,unit)
Specifies the volume serial number and UNIT=value of the volume that contains the system data set SYS1.LINKLIB. This parameter is valid only when you generate the stand-alone dump program using two-stage generation. See "Using two-stage generation of stand-alone dump when migrating" on page 93 for information on the use of this parameter.

DDSPROMPT={YES|NO}
DDSPROMPT=YES allows the stand-alone dump program to prompt the operator for an output dump data set when dumping to a DASD device. When DDSPROMPT=YES is specified, after replying to message AMD001A with a DASD device number, message AMD002A is also issued to prompt the operator for a dump data set name.

DDSPROMPT=NO indicates that the stand-alone dump program should not prompt for a dump data set name when dumping to a DASD device. When DDSPROMPT=NO is specified, after replying to message AMD001A with a DASD device number, the stand-alone dump program uses data set SYS1.SADMP. DDSPROMPT=NO is the default.

Note that regardless of the DDSPROMPT= keyword value, you can always use a default device and dump data set name by specifying the OUTPUT=(Dunit,ddsname) keyword. The stand-alone dump program uses the default values specified on the OUTPUT= keyword when the operator uses the EXTERNAL INTERRUPT key to bypass console communication, or if the operator provides a null response to message AMD001A.

AMD029={YES|NO}

If AMD029=NO is specified, SADMP does not issue AMD029D when a 3270 console screen becomes full. SADMP behaves as if the operator had replied NO to AMD029D. This parameter is meaningless when the system console is used, because AMD029D is never issued for the system console. AMD029=YES is the default.

IPLEXIST={YES|NO}

If IPLEXIST=YES is specified, SADMP includes IPLEXIST with the ICKDSF parameters, so that ICKDSF does not prompt the operator with message ICK21836D if there is already IPL text on the volume. IPLEXIST=NO is the default.

ADDSUMM=(*'options'*)
Indicates additional address spaces that you want dumped during a summary phase. Default summary address spaces are always dumped during a summary phase. If you do not specify ADDSUMM, stand-alone dump dumps only the default summary address spaces unless you specify PROMPT, in which case you have the opportunity to dump additional address spaces at run time. See “Using the DUMP or ADDSUMM keyword to request additional storage or address spaces” on page 74 for more information.

REAL={ALL|USED}
Indicates which real storage to dump. Specify USED to dump real storage that is being used at the time of the dump. Specify ALL to dump all real storage (used and unused). ALL is the default.

LDIPL={YES|NO}}
When IPL=D is specified, indicates whether ICKDSF prepares (YES) the volume to be IPLed by either List-directed IPL or traditional CCW IPL, or prepares (NO) the volume to be IPLed only by traditional CCW IPL. When IPL=T is specified and LDIPL=YES is specified, AMDSADMP issues an MNOTE and forces LDIPL=NO, since tape devices do not support List-directed IPL. In order to use LDIPL=YES, ICKDSF APAR PH45198 must be installed, and RACF (or an equivalent security product) must be configured to allow ICKDSF to perform signing. AutoIPL of SADMP by a system that was IPLed via List-directed load will be a List-directed load, so SADMP must be generated with LDIPL=YES in that case.

Examples of Coding the AMDSADMP Macro

The following examples show how to code the AMDSADMP macro to create various kinds of stand-alone dump programs.

Figure 25 on page 73 shows the AMDSADMP macro coded without explicitly specified parameters to generate a direct access resident dump program. The defaults are:

```
IPL=DSYSDA
VOLSER=SADUMP
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
OUTPUT=T282
MSG=ALL
MINASID=ALL
COMPACT=YES
REUSEDS=CHOICE
DDSPROMPT=NO
REAL=ALL
```

```
DUMP1 AMDSADMP
```

Figure 25. Example: Accepting All Defaults

In Figure 26 on page 73, the IPL parameter specifies tape as the residence volume, and the VOLSER parameter identifies that tape. All other parameters are allowed to default. The defaults are:

```
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
OUTPUT=T282
MSG=ALL
MINASID=ALL
COMPACT=YES
REUSEDS=CHOICE
DDSPROMPT=NO
REAL=ALL
```

```
AMDSADMP IPL=T3400,VOLSER=SATAPE
```

Figure 26. Example: Generating an unformatted, tape resident dump program

In Figure 27 on page 73, the OUTPUT parameter directs the stand-alone dump output to dump data set SYS1.SADMP on device 450, and the REUSEDS parameter specifies that the operator be prompted about whether to reuse the dump data set. The defaults are:

```
IPL=DSYSDA
VOLSER=SADUMP
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
MSG=ALL
MINASID=ALL
COMPACT=YES
DDSPROMPT=NO
REAL=ALL
```

```
AMDSADMP OUTPUT=D450,REUSEDS=CHOICE
```

Figure 27. Example: Generating a dump program with output to DASD

In Figure 28 on page 73, the OUTPUT parameter directs the stand-alone dump output to dump data set SADMP.DDS1 on device 450. Furthermore, the DDSPROMPT=YES keyword allows for run-time dump data set prompting. The defaults are:

```
IPL=DSYSDA
VOLSER=SADUMP
ULABEL=NOPURGE
CONSOLE=(01F,3278)
SYSUT=SYSDA
MSG=ALL
MINASID=ALL
COMPACT=YES
REUSEDS=CHOICE
REAL=ALL
```

```
AMDSADMP OUTPUT=(D450,SADMP.DDS1),DDSPROMPT=YES
```

Figure 28. Example: Generating a dump program with output to DASD

Recommended specification during the build process is as follows:

```
SP(ALL) IN ASID(1,'JESXCF')
ALSO DATASPACES OF ASID(1,'JESXCF', 'APPC', 'SMSVSAM', 'CONSOLE', 'SYSBMAS')
ALSO PAGETABLES OF DATASPACES
```

If you run JES2, add:

```
ALSO SP(ALL) IN ASID('JES2')
```

Additional subpools and dataspaces might be needed, depending on your installed IBM, vendor, and locally-written products and applications.

Using the DUMP or ADDSUMM keyword to request additional storage or address spaces

You can request that stand-alone dump dump additional storage or additional address spaces in two ways:

- **Specifying DUMP options or ADDSUMM options on the AMDSADMP macro**

  As the following example show, specify the dump tailoring options described in "Dump tailoring options" on page 75 within parentheses and single quotation marks as the value of the DUMP keyword on the AMDSADMP macro.

  ```
  DUMP=('SP(5,37,18) IN ASID('JES3')')
  DUMP=('RANGE(0:1000000) IN ASID(1)')
  DUMP=('DATASPACES OF ASID('DUMPSRV')')
  DUMP=('SADMPNO IN ASID('IOSAS')')
  ```

  Note: Do not double the quotation marks within the DUMP options. The DUMP options cannot exceed 255 characters in length.

  You can also specify additional summary address space as the value of the ADDSUMM keyword on the AMDSADMP macro.

  ```
  ADDSUMM=(‘ASID(’BPX*’)’)
  ADDSUMM=('ASID(28),ASID('JOHNDOE')’)
  ADDSUMM=(’ASID(’MYDB’,40,46:48)’)
  ```

  Note: Do not double the quotation marks within the ADDSUMM options. The ADDSUMM options cannot exceed 255 characters in length.

- **Specifying additional dump options at run time**

  By coding the PROMPT keyword on the AMDSADMP macro, you can have Stand-alone dump prompt the operator to dump additional storage or specify additional address spaces to be dumped as part of summary phase. When you code PROMPT, and the virtual storage dump program gets control, stand-alone dump issues the AMD059D message:

  ```
  AMD059D ENTER 'DUMP' OR 'SET' OR ‘ADDSUMM’ WITH OPTIONS, 'LIST' OR 'END'.
  ```

  The operator can respond with one of the following:

  – DUMP followed by dump options. In this case, the ‘=’ after DUMP is optional. See "Dump tailoring options" on page 75 for the possible dump options.

  – SET followed by the MINASID or REAL options.

  – ADDSUMM followed by address space list to additionally be dumped as part of summary phase. In this case, the ‘=’ after ADDSUMM is optional. See "Dump tailoring options" on page 75 for the possible options.

  – LIST. On the console, stand-alone dump displays the current virtual storage areas to be dumped and address space list to be dumped during the summary phase.

  – END. Stand-alone dump stops prompting the operator for options and begins processing.

Figure 29 on page 75 shows a sample exchange between stand-alone dump and the operator. The operator's replies are in lowercase. Note the operator's reply to message AMD059D using the DUMP keyword and ADDSUMM keyword.

```
 AMD082I WARNING: THE MINASID SPECIFICATION HAS BEEN SET TO 'PHYSIN'.
 AMD059D ENTER 'DUMP' OR 'SET' OR ‘ADDSUMM’ WITH OPTIONS, 'LIST' OR 'END'.
     dump sp(0::9) inasid('jes2')
 AMD060I ERROR IN INPUT TEXT INDICATED BY '*':
 DUMP SP(0::9) INASID('JES2')
           *
AMD065A ENTER TEXT TO BE SUBSTITUTED FOR THE TEXT IN ERROR.
 >
 AMD060I ERROR IN INPUT TEXT INDICATED BY '*':
 DUMP SP(0:9) INASID('JES2')
              ******
 AMD065A ENTER TEXT TO BE SUBSTITUTED FOR THE TEXT IN ERROR.
 > in asid
 AMD082I WARNING: THE MINASID SPECIFICATION HAS BEEN SET TO 'PHYSIN'.
 AMD059D ENTER 'DUMP' OR 'SET' OR ‘ADDSUMM’ WITH OPTIONS, 'LIST' OR 'END'.
 > list
 AMD067I CURRENT DUMP OPTIONS:
   CSA ALSO LSQA, SP(203:205,213:215,229:230,236:237,247:249) IN ASID(PHYSIN)
   ALSO SP(0:9) IN ASID('JES2')
   ADDSUMM  ASID('ALLOCAS','ANTAS000','ANTMAIN','CATALOG','CONSOLE','DEVMAN',
   'DUMPSRV','GRS','IEFSCHAS','IOSAS','IXGLOGR','JESXCF','JES2','JES3','OMVS',
   'SMSPDSE','SMSPDSE1','SMSVSAM','WLM','SMF','SMFXC','XCFAS')
 AMD082I WARNING: THE MINASID SPECIFICATION HAS BEEN SET TO 'PHYSIN'.
 AMD059D ENTER 'DUMP' OR 'SET' OR ‘ADDSUMM’ WITH OPTIONS, 'LIST' OR 'END'.
 > end
```

Figure 29. Sample Console output from the stand-alone Dump Program

When stand-alone dump detects an error in the reply to message AMD059D, it repeats the incorrect line at the console, underscores the incorrect part with asterisks, and prompts the operator for replacement text. If the dump options exceed 255 characters, stand-alone dump marks the whole line in error.

If a system restart occurs during the virtual storage dump program, stand-alone dump re-prompts the operator for dump options. stand-alone dump does not use any of the dump options that the operator specified before the system restart.

Dump tailoring options

You can specify the dump tailoring options in one or all of the following ways:

- On the DUMP keyword of the AMDSADMP macro.
- On the ADDSUMM keyword of the AMDSADMP macro.
- By the operator in reply to message AMD059D at run time.

Following is a list of the dump tailoring options you can specify. For a complete explanation of the options, see “Explanation of dump tailoring options” on page 76.

```
{dump-spec-list|SET MINASID{ALL|PHYSIN}|SET REAL(ALL|USED)|ADDSUMM addsumm-spec-list|LIST|END}
```

dump-spec-list is one or more of the following:

- *range-spec-list* IN ASID(*address-space-list*) [ALSO...]
- DATASPACES OF *domain-spec-list*[...]
- DSP OF *domain-spec-list*[...]
- PAGETABLES OF DATASPACES

  range-spec-list is one or more of the following:

  – SP(*subpool-list*)
  – RANGE(*address-range-list*)
  – LSQA

  - HIGH VIRTUAL
  - SADMPNO

    subpool-list is one of the following:

    - *subpool-number* TO *subpool-number* [,...]
    - ALL

    address-range-list is one of the following:

    - *address* TO *address* [,...]
    - ALL

  address-space-list is one of the following:

  - *asid* TO {*asid*|*jobname*|SYSKEY|PHYSIN}[,...]
  - ALL

addsumm-spec-list is the following:

- ASID (*address-space-list*) [, ASID(*address-space-list*)][,...]

  address-space-list is the following:

  - *asid*[*dlm asid*] | *jobname*[,...]

    dlm is ‘TO’ or ‘:’.

Use the following guidelines when specifying values in your dump tailoring options:

- HIGH VIRTUAL cannot be used by itself; specify additional keywords.
- SADMPNO cannot be used by itself; specify additional keywords.
- *address* is a hexadecimal number from 0 to X'7FFFFFFF'.
- *subpool-number* is a decimal number from 0 to 255.
- *asid* is a hexadecimal number from 0 to X'FFFF'.
- *jobname* is a valid jobname enclosed in single quotation marks. Including wildcard characters is valid for jobnames. For a description of wildcard characters, see the ASID('jjjj') option in the topic “Explanation of dump tailoring options” on page 76.
- *range-spec-list* is a list of subpools, a list of storage ranges, or both.
- *domain-spec-list* is a list of address spaces.
- ‘TO’ and ‘:’ are synonyms.
- ‘DATASPACES’ and ‘DSP’ are synonyms.

Keywords, such as DATASPACES, can be truncated on the right, provided the truncated form is not ambiguous. You can enter letters in either lower-case or uppercase. Blanks can be inserted between numbers, keywords, and separators; blanks cannot be inserted within numbers or keywords.

Explanation of dump tailoring options

This section provides an explanation for each of the dump tailoring options.

RANGE(*xxxxxxxx:yyyyyyyy,xxxxxxxx:yyyyyyyy...*)
: Specifies one or more ranges of storage that you want dumped. *xxx* and *yyy* are hexadecimal addresses from 0 to X'7FFFFFFF'

RANGE(ALL)
: Specifies dumping of all storage from 0 to X'7FFFFFFF'

SP(*ddd*)
: Causes stand-alone dump to dump subpool *ddd*. *ddd* is a decimal integer from 0 to 255.

SP(*ddd*:*eee*)
: Causes stand-alone dump to dump all subpools from *ddd* to *eee*, inclusive.

SP(*ddd*:*eee*,*ddd*:*eee*,...)
Causes stand-alone dump to dump the combination of subpools that you specify.

SP(ALL)
Causes stand-alone dump to dump all subpools, from 0 to 255 inclusive.

LSQA
Causes stand-alone dump to dump the LSQA.

HIGH VIRTUAL
Causes stand-alone dump to dump all allocated paged out storage above 2G. This applies to storage that was not being excluded because of a SADMP=NO specification. See "SADMPNO" on page 77 for more information.

SADMPNO
Causes stand-alone dump to dump memory objects created with SADMP=NO in the specified address space(s). See IARV64 in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for more information.

ASID(*xxxx*:*yyyy*)
Causes stand-alone dump to dump storage for the range of address spaces whose ASIDs begin at *xxx* and end at *yyy*, inclusive. *xxx* and *yyy* are hexadecimal numbers from X'1' to X'FFFF'.

ASID('*jjj*')
Causes stand-alone dump to dump storage for the address space that jobname *jjj* identifies. Note that you must enclose the jobname in single quotation marks.

You can use wildcard characters to identify multiple jobnames. The valid wildcard characters are:

*****
Zero or more characters, up to the maximum length of the string. An asterisk can start the string, end it, appear in the middle of the string, or appear in several places in the string. A single asterisk for the jobname indicates that all jobnames match.

?
One character. One or more question marks can start the string, end it, appear in the middle of the string, or appear in several places in the string. A single question mark indicates all jobnames consisting of one character.

ASID(SYSKEY)
Causes stand-alone dump to dump storage for all address spaces whose active TCB has an associated storage key of 0 to 7.

ASID(*combination*)
You can combine any of the above specifications. An example of a valid combination is ASID(2, 'IMSJOB',SYSKEY).

ASID(PHYSIN)
Causes stand-alone dump to dump storage for physically swapped-in address spaces.

ASID(ALL)
Causes stand-alone dump to dump storage for all address spaces. Note that you cannot specify ASID(ALL) in combination with any of the other ASID specifications.

DATASPACES OF ASID(*qualifier*)
When you specify the `DATASPACES OF ASID(qualifier)` keyword, stand-alone dump dumps all data spaces owned by the specified address space. For each requested data space, stand-alone dump:

- Dumps pages backed by central storage during the central storage dump
- Copies into central storage and dumps every page that is not a first reference page and not backed by central storage

PAGETABLES OF DATASPACES
When you specify the `PAGETABLES OF DATASPACES` keyword, stand-alone dump dumps paged-out virtual storage that contains the page tables for all data spaces.

When stand-alone dump dumps the storage that you specify, stand-alone dump dumps all listed subpools and address ranges in all specified address spaces for each specification of dump options. However, stand-alone dump does not merge your specifications across the dump options that you specify. For example, to cause stand-alone dump to dump subpools 0 and 1 in address space A, and subpools 0 and 1 in address space B, enter:

```
DUMP SP(0,1) IN ASID(A,B)
```

To cause stand-alone dump to dump subpool 0 in address space A and subpool 1 in address space B, enter:

```
DUMP SP(0) IN ASID(A) ALSO SP(1) IN ASID(B)
```

Figure 30 on page 78 shows other examples of valid specifications.

```
DUMP SP(0:7,15),RANGE(0:10000000) IN ASID(SYSKEY),ASID(8)
DU  (SP(0 TO 7 OR 15),SP(255)) IN AS('TCAM')
DUMP RANGE(ALL) IN ASID(1) ALSO SP(0) IN ASID(SYSKEY,8)
DU DAT OF AS(ALL)
DUMP ( SP(0:127) IN ASID('GENER') ALSO SP(0) IN ASID('IMS') )
DUMP LSQA IN AS('MYJOB',14)
DU SP(128),LS IN ASID(C,PHYSIN)
DUMP DATASPACES OF ASID('MYJOB??')
DUMP DATASPACES OF ASID('MY*')
DUMP HIGH VIRTUAL IN ASID(C)
ADDSUMM ASID(3F),ASID('DEBBIE')
ADDSUMM ASID('MYDB*',40,46:48)
AD AS('MYJOB?'),AS(26,'REPORT')
DUMP SADMPNO IN ASID('DUMPSRV')
DUMP SADMPNO IN ASID(5,6,'XCFAS')
```

Figure 30. Example of valid specifications

Dumping to a DASD data set

When you specify DASD on the OUTPUT parameter, you direct the output of the stand-alone dump program to a predefined dump data set on one of the following types of DASD:

- 3380
- 3390
- 9345

Note: The selection of the output device (DASD or tape) can be made at both generation time and at run time. An output device specified at run time overrides an output device specified at generation time.

When preparing to take a stand-alone dump to DASD, you must allocate and initialize the dump data set using the AMDSADDD REXX or IPCS SADMP dump data set utilities.

The following requirements exist for the allocation of the DASD dump data set:

- The dump data set must have the text 'SADMP' as either part of, or as an entire data set qualifier.
- Do not place a data set that is intended to contain a stand-alone dump on a volume that also contains a page data set that the stand-alone dump program you might need to dump. When stand-alone dump initializes a page volume for virtual dump processing, it checks to see if the output dump data set also exists on this volume. If it does, the stand-alone dump program issues message AMD100I and does not retrieve any data from page data sets on this volume. Thus, the dump might not contain all of the data that you requested. This lack of data can impair subsequent diagnosis.
- The dump data set cannot be defined on the same volume that contains the IPL text of stand-alone dump.

Note: Because the data set does not have to be cataloged, there can be more than one dump data set with the same name per system. Furthermore, because the data set can be uniquely named, there can be more than one dump data set per volume.

- IBM recommends that you define the dump data set on a volume that does not contain any other data sets, especially volumes that contain sysplex couple data sets. This ensures maximum capacity when needed and avoid the possibility of other data sets being accessed by another system.
- The dump data set must be both allocated and initialized using the AMDSADDD REXX or IPCS SADMP dump data set utilities.
- Because the stand-alone dump program must be able to locate the dump data set on the output device being used, it is imperative that the necessary data set management steps be taken so that the stand-alone dump data sets are not placed into a migrated state or moved to a different volume. The dump data sets must also be exempt from any space management processing that releases unused space.

When the dump data set is filled, the stand-alone dump program prompts the operator, with message AMD001A, to specify another output device. The stand-alone dump program can continue dumping to any stand-alone dump supported device, however, after a tape device is selected, it must be used to complete the dump even though multiple tape volumes can be required.

Note: Dumping to multiple DASD dump data sets requires that each dump data set used has been preformatted by the AMDSADDD REXX or IPCS SADMP dump data set utilities.

Using the AMDSADDD utility

The REXX utility AMDSADDD resides in SYS1.SBLSCLI0. This section describes how to use the AMDSADDD REXX utility to:

- Allocate and initialize the data set. See Figure 31 on page 83 for an example of allocating and initializing the dump data set.
- Clear (reinitialize) the data set. See Figure 34 on page 85 for an example of clearing the dump data set.
- Reallocate and initialize the data set. See Figure 35 on page 86 for an example of reallocating and initializing the dump data set

The IPCS SADMP dump data set utility performs the same functions as the AMDSADDD REXX utility. See SADMP option on the IPCS Dialog in *z/OS MVS IPCS User's Guide* for more information. See *z/OS MVS IPCS Customization* for more information on the migration tasks involving AMDSADDD.

The data set allocated by the AMDSADDD REXX utility must have these characteristics:

- The data set name (DSNAME) must:
  - be 44 characters or less in length
  - contain the text 'SADMP' as either part of, or as an entire data set qualifier.

  For example, valid dump data set names are:

  - SYS1.SADMP
  - SADMP
  - SYSTEMA.SADMPDDS

  Invalid dump data set names are:

  - SYS1.DUMP.DATASET
  - SADUMP
- The block size (BLKSIZE) must be:

  DASD

  BLKSIZE

3380, 9345
20800

3390
24960

Note: Stand-alone dump processing can use a 3390 DASD defined with a block size of 20800; however, the allocated space is not fully utilized unless a block size of 24960 is used. The AMDSADDD REXX utility allocates 3390 DASD devices using a block size of 24960.

- The logical record length (LRECL) must be 4160.
- The record format (RECFM) must be FBS.
- The data set must consist of a single extent.
- The data set organization can be PS (DSORG=PS) or non-VSAM extended format (DSORG=PS-E).
- Occupies space on 1-32 volumes.

All stand-alone dump data sets that are SMS managed must have a STORCLAS with the GUARANTEED_SPACE attribute.

All DSORG=PS-E data sets must:

- Be SMS managed
- Have a DATACLAS that specifies no compression
- Have a STORCLAS that specifies a sustained data rate of zero (suppress striping).

For SMS to honor the allocation request, your installation's automatic class selection (ACS) routines must be configured to do so. For instructions on setting up an SMS environment, see the following publications:

- *z/OS DFSMS Using Data Sets*
- *z/OS DFSMSdfp Storage Administration*

You provide the volume, dump data set name, unit, space, and catalog disposition on the invocation of the AMDSADDD REXX utility. If multiple volumes are specified, then a multi-volume data set is allocated and formatted. Up to 32 volumes can be specified, all having the same device type. The amount of space specified for the data set is allocated on each volume.

Special control information is written to multi-volume data sets to allow all of the volumes to be located when the data set is written to. This includes the device number of the volume. The data set is not usable by stand-alone dump if the control information is missing or invalid. If a volume of a multi-volume data set is moved to a new device number, the data set must be re-initialized to update the control information. The data set cannot be used by a system that has the volumes attached at a device number different than the system which writes the control information.

When using multi-volume data sets, it is highly recommended that they be cataloged. This simplifies processing, as IPCS can easily be used to format and copy the dump data in the cataloged data sets.

Syntax

Note: REXX requires that the specified parameters appear in the order listed. If you do not specify a parameter, the AMDSADDD REXX utility prompts for a specification of that parameter.

```
AMDSADDD  {DEFINE|CLEAR|REALLOC}
          volser{(data set name)}
          (type[,[STORCLAS][,[DATACLAS][,[MGMTCLAS]]]])
          [space] [YES|NO] [EXTREQ|LARGE|BASIC] [OPT|NO]
```

or

```
AMDSADDD  {DEFINE|CLEAR|REALLOC}
          (volumelist){(data set name)}
          (type[,[STORCLAS][,[DATACLAS][,[MGMTCLAS]]]])
          [space] [YES|NO] [EXTREQ|LARGE|BASIC] [OPT|NO]
```

Parameters

AMDSADDD
The name of the REXX utility.

DEFINE|CLEAR|REALLOC
Indicates the function to be performed by the AMDSADDD REXX utility:

DEFINE
Allocates and initializes a new dump data set.

CLEAR
Initializes an existing dump data set again. After you use CLEAR, the data set is ready for use.

REALLOC
Deletes an existing stand-alone dump data set, then reallocates and reinitializes a new stand-alone dump data set on the same volume(s), with the sole purpose of increasing its size. If the specified dump data set does not exist, AMDSADDD converts the function to a DEFINE request and continues using DEFINE processing. If the request to reallocate and reinitialize a new dump data set cannot be satisfied (for example, if you attempt to reallocate a new data set using more cylinders than are available), AMDSADDD might delete the existing dump data set. REALLOC can change the EATTR attributes of the dump data set and can also modify the DSNTYPE but cannot change data sets from SMS to non-SMS managed and vice versa. For example, DSNTYPE=BASIC/LARGE to EXTREQ isn't allowed.

Note: When specifying the REALLOC option for an existing multi-volume data set, the same list of volumes must be specified as when the dataset was originally allocated.

volser{(data set name)}
Indicates the VOL=SER= name of the volume on which the dump data set is to be allocated. Do not use the stand-alone dump residence volume or the volumes containing the system paging data sets. Optionally, also defines the dump data set name to be allocated on the volume. If data set name is specified, it must:

- be fully qualified (without quotation marks)
- have a length of 44 characters or less
- contain the text 'SADMP' as either part of, or as an entire data set qualifier.

Note: If no data set name is specified, the AMDSADDD utility will allocate the data set SYS1.SADMP on the specified volume.

(vollist){(data set name)}
vollist is a comma delineated list of volsers to use for the data set. A multi-volume data set will be allocated using the list of volumes. The device number of the first volume is used to specify the data set to stand-alone dump.

Tip: When you take a stand-alone dump to a multi-volume data set it will be striped and take significantly less time to capture.

***(type*[,[STORCLAS][,[DATACLAS][,[MGMTCLAS]]]])**
Type indicates the device type on which the dump data set should be allocated. Valid DASD types are 3380, 3390, and 9345.

STORCLAS
The SMS storage class.

DATACLAS
The SMS data class.

MGMTCLAS
The SMS management class.

For additional information on these classes, see *z/OS MVS JCL Reference*.

space
Indicates the number of cylinders for the dump data set to be allocated. For a multi-volume data set, this amount is allocated on each volume.

The size of your dump output depends on your storage configuration and how much of that storage you choose to dump using the options of stand-alone dump. To estimate how much space, in cylinders, to allocate for your dump data set, use the number of cylinders of DASD that a typical dump to tape consumes when it has been copied to DASD for IPCS processing. If you do not allocate enough space, the stand-alone dump program prompts the operator, through message AMD001A and message AMD002A (if DDSPROMPT=YES was specified on the AMDSADMP macro), to specify a different device and/or a different dump data set so that dumping can continue.

The *space* option is not required with the CLEAR parameter. The *space* option is, however, required with the DEFINE and REALLOC parameters.

YES|NO
Specifies whether the system is to catalog the dump data set. If you want the data set to be cataloged, specify **YES** or **Y**. If you do not want the data set to be cataloged, specify **NO** or **N**. Specifying **N** allows you to allocate multiple dump data sets with the same name.

The catalog option is not required with the CLEAR parameter. The catalog option is, however, required with the DEFINE and REALLOC parameters.

EXTREQ|LARGE|BASIC
Indicates the DSNTYPE of the dump data set to be defined.

EXTREQ requests an extended format dump data set. This data set must have the attribute DSNTYPE=EXTREQ. This attribute allows the system to place the data set in cylinder-managed space on extended access volumes.

LARGE requests a large format dump data set, one with attribute DSNTYPE=LARGE that the system allows to span more than 64K tracks per volume.

BASIC indicates that a large format dump data set is not desired. BASIC can be associated with a conventional dump data set or an extended format sequential dump data set, depending on other options. BASIC is the default.

The dsntype option is not required with CLEAR parameter. The dsntype is optional with DEFINE and REALLOC parameters. The dsntype option with REALLOC must match with the existing dsntype option.

OPT|NO
Indicates the extended attributes of a dump data set. The EATTR option is not required with the CLEAR parameter.

OPT
Requests that extended attributes are optional. The system might store the dump data set in the cylinder-managed space on extended access volumes.

NO
Requests that extended attributes are not required. The default value is NO.

Examples of running AMDSADDD interactively

Figure 31 on page 83 shows an example of using the AMDSADDD REXX utility to allocate and initialize the dump data set with a size of 350 cylinders and a VOL=SER= of SAMPLE. Because no data set name is specified, AMDSADDD allocates the dump data set SYS1.SADMP on the volume SAMPLE.

Note: Stand-alone dump does not issue error messages during the processing of AMDSADDD. Stand-alone dump does, however, pass messages to the operator from other sources, such as the TSO/E ALLOC command.

```
 ------------------------  TSO COMMAND PROCESSOR  -------------------

 ENTER TSO COMMAND, CLIST, OR REXX EXEC BELOW:

 ===> exec 'sys1.sblscli0(amdsaddd)'

 What function do you want?
 Please enter DEFINE if you want to allocate a new dump dataset
 Please enter CLEAR if you want to clear an existing dump dataset
 Please enter REALLOC if you want to reallocate and clear an existing
    dump dataset
 Please enter QUIT if you want to leave this procedure
define
 Please enter VOLSER or VOLSER(dump_dataset_name) or
    (VOLLIST) or (VOLLIST) (dump_dataset_name)
sample
 Please enter the device type for the dump dataset
 Device type choices are 3380 or 3390 or 9345
 (An SMS storage class, data class, and management
  class may also be specified with the device type)
3380
 Please enter the number of cylinders
350
 Do you want the dump dataset to be cataloged?
 Please respond Y or N
y
 Specify the DSNTYPE. Reply BASIC or LARGE or EXTREQ
BASIC
 Specify the extended attributes for the dump dataset.
 Reply OPT or NO
NO
 IKJ56650I TIME-02:09:36 PM. CPU-00:00:01 SERVICE-7077780
  SESSION-02:06:31 JANUARY 20,2017

   Initializing output dump dataset with a null record:
   Dump dataset has been successfully initialized

   Results of the DEFINE request:

     Dump Dataset Name    : SYS1.SADMP
     Volume               : SAMPLE
     Device Type          : 3380
     Allocated Amount     : 350
```

Figure 31. Using AMDSADDD to allocate and initialize a basic dump data set

Figure 32 on page 84 shows an example of using the AMDSADDD REXX utility to allocate and initialize a multi-volume, large format dump data set with extended attributes and a size of 1500 cylinders on volumes DMC880 and DMC881 in storage class SADEXTND. Because no data set name is specified, AMDSADDD allocates the dump data set SYS1.SADMP on the specified volumes.

```
--------------- TSO COMMAND PROCESSOR  ------------------

 ENTER TSO COMMAND, CLIST, OR REXX EXEC BELOW:

 ===> exec 'sys1.sblscli0(amdsaddd)'

 What function do you want?
 Please enter DEFINE if you want to allocate a new dump dataset
 Please enter CLEAR if you want to clear an existing dump dataset
 Please enter REALLOC if you want to reallocate and clear an existing
    dump dataset
 Please enter QUIT if you want to leave this procedure
DEFINE
 Please enter VOLSER or VOLSER(dump_dataset_name) or
    (VOLLIST) or (VOLLIST)(dump_dataset_name)
(dmc880,dmc881)
 Please enter the device type for the dump dataset
 Device type choices are 3380 or 3390 or 9345
 (A SMS storage class, data class, and management
 class may also be specified with the device type)
(3390,SADEXTND)
 Please enter the number of cylinders (per volume)
1500
 Do you want the dump dataset to be cataloged? Please
 respond Y or N
Y
 Specify the DSNTYPE. Reply BASIC or LARGE or EXTREQ
LARGE
 Specify the extended attributes for the dump dataset.
 Reply OPT or NO
OPT
 IKJ56650I TIME-02:09:36 PM. CPU-00:00:01 SERVICE-7077780
  SESSION-02:16:31 JANUARY 20,2017

 Note: Allocated space does not match requested amount
 Amount allocated:  1512
 Amount requested:  1500

 Initializing output dump dataset with a null record:
 Dump dataset has been successfully initialized


  Results of the DEFINE request:

    Dump Dataset Name    : SYS1.SADMP
    Volume               : DMC880
                           DMC881
    Device Type          : 3390
    Allocated Amount     : 1512 (per volume)
```

Figure 32. Using AMDSADDD to allocate and initialize a multi-volume, large format dump data set

Figure 33 on page 85 shows an example of using the AMDSADDD utility to allocate and initialize an extended format dump data set ‘SADMP.SAMPLE’ with a size of 400 cylinders in the cylinder-managed space. This SMS managed dump data set spans multiple volumes SADPK1 and SADPK2. In an extended address volume environment, some systems might round up the cylinders causing the requested amount and allocated amount to be different. In this case, a message is displayed that indicates the requested amount of cylinders and the allocated amount of cylinders.

```
 ------------------------  TSO COMMAND PROCESSOR  -------------------
 ENTER TSO COMMAND, CLIST, OR REXX EXEC BELOW:
 ===> exec 'sys1.sblscli0(amdsaddd)'
 What function do you want?
 Please enter DEFINE if you want to allocate a new dump dataset
 Please enter CLEAR if you want to clear an existing dump dataset
 Please enter REALLOC if you want to reallocate and clear an existing
        dump dataset
 Please enter QUIT if you want to leave this procedure
define
 Please enter VOLSER or VOLSER(dump_dataset_name) or (VOLLIST)
 or (VOLLIST)(dump_dataset_name)
(SADPK1,SADPK2)(SADMP.SAMPLE)
 Please enter the device type for the dump dataset
 Device type choices are 3380 or 3390 or 9345
(An SMS STORAGE CLASS, DATA CLASS, AND MANAGEMENT CLASS
MAY ALSO BE SPECIFIED WITH THE DEVICE TYPE)
(3390,STORCLAS,DATACLAS,MGMTCLAS)
 Please enter the number of cylinders (per volume)
400
 Do you want the dump dataset to be cataloged?
 Please respond Y or N
y
 Specify the DSNTYPE. Reply BASIC or LARGE or EXTREQ
EXTREQ
 Specify the extended attributes for the dump dataset. Reply OPT or NO
OPT
TIME-11:54:59 PM. CPU-00:00:00 SERVICE-58954 SESSION-00:07:25 AUGUST 1,2009

Note: Allocated space does not match requested amount
Amount allocated:  420
Amount requested:  400
Initializing output dump dataset with a null record:
Dump dataset has been successfully initialized

Results of the DEFINE request:

  Dump Dataset Name    : SADMP.SAMPLE
  Volume               : SADPK1
                         SADPK2
  Device Type          : 3390
  Allocated Amount     : 420 (per volume)
```

Figure 33. Using AMDSADDD to Allocate and Initialize an Extended Dump Data Set

Figure 34 on page 85 shows an example of using the AMDSADDD REXX utility to clear (reinitialize) an existing dump data set called SADMP.DDS1 on VOL=SER=SAMPLE. In this example, the parameters are part of the invocation of the utility; therefore, AMDSADDD does not prompt for values.

```
 ------------------------  TSO COMMAND PROCESSOR  ---------------------
 ENTER TSO COMMAND, CLIST, OR REXX EXEC BELOW:
 ===> exec exec 'sys1.sblscli0(amdsaddd)' 'clear sample(sadmp.dds1) 3380'
IKJ56650I TIME-11:00:00 PM. CPU-00:00:00 SERVICE-20191 SESSION-00:09:55 JUNE
14,1994
Initializing output dump dataset with a null record:
Dump dataset has been successfully initialized
Results of the CLEAR request:

  Dump Dataset Name    : SADMP.DDS1
  Volume               : SAMPLE
  Device Type          : 3380
  Allocated Amount     : 350

***
```

Figure 34. Using AMDSADDD to Clear an Existing Dump Data Set

Figure 35 on page 86 shows an example of using the AMDSADDD REXX utility to allocate a new dump data set called SYSTEM1.SADMPDDS on VOL=SER=SMS001. In this example, the parameters are part of the invocation of the utility; therefore, AMDSADDD does not prompt for values.

Note: In an SMS environment, it is possible to have the dump data set cataloged on a different volume than the one specified. If the dump data set is allocated on a different volume, AMDSADDD issues an error message and exits. In Figure 35 on page 86, the dump data set was not allocated on the specified volume causing AMDSADDD to delete the dump data set, issue an error message and quit.

```
-----------------------TSO COMMAND PROCESSOR------------------------
===>exec exec 'sys1.sblscli0(amdsaddd)' ‘Define SMS001(SYSTEM1.SADMPDDS) 3390 100 Y LARGE’
IKJ56650I TIME-11:00:00 PM. CPU-00:00:00 SERVICE-20191 SESSION-00:09:55
JUNE 14,1994

    Error: output dump dataset not allocated on specified volume  SMS001
    Try using a Storage Class with Guaranteed Space

***
--------------------------------------------------------------------
```

Figure 35. Using AMDSADDD to Reallocate the Dump Data Set

Examples of running AMDSADDD in batch mode

The following examples show how to use JCL to allocate and initialize dump data sets.

Note: Because users cannot be prompted to enter values when invoking the AMDSADDD REXX utility in batch mode, you must specify all parameters in the order listed.

Figure 36 on page 86 shows how to use JCL to allocate and initialize the dump data set SYS1.SADMP.A1 on VOL=SER=ZOSSVA with a size of 2653 cylinders. The BASIC type of data set is allocated because the dsntype parameter is not specified.

```
//STEP1    EXEC  PGM=IKJEFT01,REGION=64M
//SYSTSPRT   DD  SYSOUT=*
//SYSTSIN    DD  *
EXEC 'SYS1.SBLSCLI0(AMDSADDD)' +
     'DEFINE ZOSSVA(SYS1.SADMP.A1) 3390 2653 N'
/*
```

Figure 36. Example: Using JCL to allocate and initialize a dump data set

Figure 37 on page 86 shows how to use JCL to allocate and initialize an extended format dump data set named SADMP.DS on VOL=SER=USRDS1 with a size of 2953 cylinders in the cylinder-managed space.

```
//SAMPLE  JOB ‘S3031,B707000,S=C’, ‘BATCH EXAMPLE’, RD=R,
//          MSGLEVEL=(1,1),CLASS=E,NOTIFY=&SYSUID,MSGCLASS=H
//STEP1    EXEC  PGM=IKJEFT01,REGION=64M
//SYSTSPRT   DD  SYSOUT=*
//SYSTSIN    DD  *
EXEC ‘SYS1.SBLSCLI0.EXEC’
‘DEFINE USRDS1(SADMP.DS)(3390,storclas) 2953 Y EXTREQ OPT’
/*
```

Figure 37. Example: Using JCL to allocate and initialize an extended format dump data set

Generating the stand-alone dump program

After coding the AMDSADMP macro, you can generate the stand-alone dump program. There are two ways to generate the stand-alone dump program:

- “One-stage generation” on page 86
- “Two-stage generation” on page 92

IBM recommends that you use one-stage generation to create the stand-alone dump program because multiple tasks are performed in one-stage.

Note: You can use either two-stage or one-stage JCL when migrating to a new version of MVS.

One-stage generation

In one-stage generation, run the AMDSAOSG program as a single job, using the AMDSADMP macro you have coded as input data on the GENPARMS control statement. The stand-alone dump utility program, AMDSAOSG, initializes a stand-alone dump residence volume in one job by dynamically allocating data sets and invoking the appropriate programs. To run the one-stage generation program, indicate one AMDSADMP macro as a control statement for DDNAME GENPARMS.

```
//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG      EXEC PGM=AMDSAOSG,REGION=5M
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=SYS1.MODGEN,DISP=SHR
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
//            SPACE=(4096,(2,1)),UNIT=SYSDA
//DSFSYSIN DD DSN=&DSFSYSIN,DISP=(,PASS),
//            SPACE=(80,(4,1)),UNIT=SYSDA
//GENPRINT DD DSN=SADMP.LIST,DISP=OLD
//GENPARMS DD *
                    AMDSADMP IPL=DSYSDA,VOLSER=SPOOL2,                            X
              CONSOLE=(1A0,3277)
        END
/*
//PUTIPL   EXEC PGM=ICKDSF,REGION=5M
//IPLDEV   DD DISP=OLD,UNIT=SYSDA,
//            VOL=(PRIVATE,RETAIN,SER=SPOOL2)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN    DD DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
```

Figure 38. Example: One-stage generation

Table 15 on page 87 contains the DDNAMES AMDSAOSG uses, and the defaults for the DDNAMES. Additional related notes follow this table.

Table 15. DDNAMES and defaults used by AMDSAOSG

ddname	Default value	Use
DPLTEXT	DSN=SYS1.NUCLEUS(AMDSADPL),DISP=SHR	Input for AMDSABLD.
DVITEXT	DSN=SYS1.NUCLEUS(AMDSADVI),DISP=SHR	Input for AMDSABLD.
GENPARMS	Must be preallocated.	Input for AMDSAOSG, passed to assembler.
GENPRINT	SYSOUT=A	Output listing from AMDSAOSG.
IPITEXT	DSN=SYS1.NUCLEUS(AMDSAIPI),DISP=SHR	Input for AMDSABLD.
IPLDEV	DSN=SYS1.PAGEDUMP.Vvolser,UNIT=iplunit, VOL=(PRIVATE,SER=iplser),	Stand-alone dump program, output from AMDSABLD. ICKDSF uses VOL keywords to describe the residence volume.
	DISP=OLD,DCB=(BLKSIZE=12288,RECFM=U, DSORG=PS), LABEL=(,NL)	Tape IPL volume.
	DISP=(NEW,KEEP),DCB=(LRECL=4096, BLKSIZE=4096,RECFM=F,DSORG=PS),SPACE=(4096,(1095),,CONTIG), LABEL=EXPDT=99366	DASD IPL volume.
IPLTEXT	DSN=SYS1.NUCLEUS(AMDSAIPD),DISP=SHR for DASD	Input for AMDSABLD.
	DSN=SYS1.NUCLEUS(AMDSAIPT),DISP=SHR for tape	
PGETEXT	DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR	Input for AMDSABLD.
SYSPRINT	Must not be pre-allocated	Temporary listings from called programs.
SYSPUNCH	DSN=&OBJ,UNIT=SYSDA,SPACE=(80,(250,50))	Object module passed from assembler to AMDSABLD.
SYSTERM	None	Assembly messages.
SYSUT1	UNIT=SYSDA,SPACE=(1700,(400,50))	Work file for assembler.

Table 15. DDNAMES and defaults used by AMDSAOSG (continued)		
ddname	**Default value**	**Use**
TRK0TEXT	Must be preallocated.	Cylinder 0, Track 0 IPL text from AMDSABLD to ICKDSF (DASD only).
DSFSYSIN	DSN=&DSFSYSIN, DISP=(,PASS), SPACE=(80,(4,1)), UNIT=SYSDA	SYSIN input for ICKDSF

Note:

1. You **must** specify the GENPARMS DDNAME on the job step.
2. You **cannot** specify the SYSPRINT and SYSIN DD statements in the job step.
3. In GENPARMS, you specify values for UNIT= and VOLSER= on the AMDSADMP macro statement.
4. You must specify SYSLIB TRK0TEXT and DFSSYSIN statements.
5.

The JCL shown in Figure 39 on page 88 generates a stand-alone dump from DASD 222 using a volume serial of SADMPM. The output is directed to the data set SYS1.SADMP on a DASD 450. Stand-alone dump determines at run-time if that device is usable. If the dump data set on device 450 is not usable, the operator will be prompted for another data set. The operator can press enter on any of the consoles at address 041, 042, 0A0, 3E0, or 3E1. The dump will include the default storage ranges in those address spaces that are physically-swapped in at the time of the dump. In addition, all storage in ASID 1 and the JES2 address spaces will be dumped. Stand-alone dump will also dump the data spaces created by the DUMPSRV address space.

```
//OSG      EXEC PGM=AMDSAOSG,REGION=5M
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=SYS1.MODGEN,DISP=SHR
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
//            SPACE=(4096,(2,1)),UNIT=SYSDA
//DSFSYSIN DD DSN=&DSFSYSIN,DISP=(,PASS),
//            SPACE=(80,(4,1)),UNIT=SYSDA
//GENPARMS DD   *
  AMDSADMP CONSOLE=((041,3277),(042,3277),(0A0,3277),                 X
                   (3E0,3277),(3E1,3277)),                            X
           DUMP='SP(ALL) IN ASID(1,'JES2') ALSO DATASPACES            X
           OF ASID('DUMPSRV')',                                       X
           IPL=D222,                                                  X
           MINASID=PHYSIN,                                            X
           OUTPUT=D450,                                               X
           REUSEDS=NEVER,                                             X
           PROMPT,                                                    X
           VOLSER=SADMPM
  END
/*
//PUTIPL   EXEC PGM=ICKDSF,REGION=4M
//IPLDEV   DD DISP=OLD,UNIT=SYSDA,
//            VOL=(PRIVATE,RETAIN,SER=SADMPM)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN    DD DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
```

Figure 39. Example: One-stage generation of stand-alone dump to a DASD

The JCL shown in Figure 40 on page 89generates a stand-alone dump from tape 333 using a volume serial of TSADMP. The output is directed to the unlabeled volume on tape 550. Stand-alone dump determines at runtime if that device is usable. If the dump data set on device 550 is not usable, the operator is prompted for another data set. The operator can press enter on any of the consoles at address 051,052, 0A0, 3E0, or 3E1. The dump includes the default storage ranges in all address spaces at the time of the dump. In addition, the data spaces of master, XCF and OMVS address spaces are also included in the stand-alone dump.

```
//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG      EXEC PGM=AMDSAOSG,REGION=5M
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=SYS1.MODGEN,DISP=SHR
//GENPARMS DD *
  AMDSADMP IPL=T333,VOLSER=TSADMP,                                    X
           CONSOLE=((051,3277),(052,3277),                             X
              (0A0,3277),(3E0,3277),(3E1,3277)),                       X
           DUMP='SP(ALL) IN ASID(ALL)                                  X
           ALSO DSP OF ASID(1,'XCFAS','OMVS')',                        X
           MINASID=ALL,                                                X
           OUTPUT=(500),                                               X
           REUSEDS=NEVER,                                              X
           PROMPT                                                      X
  END
/*
```

Figure 40. Example: One-stage generation of stand-alone dump to tape

The output from AMDSAOSG contains a listing for the stand-alone dump common communication table (CCT) and device and dump options (DDO) control blocks that contain information specified at generation time. The remainder of the output consists of messages, including message AMD064I, from both stand-alone dump and, when the residence volume is direct access, the device utility ICKDSF. Table 16 on page 89 lists the codes AMDSAOSG returns in message AMD064I.

Table 16. AMDSAOSG return codes

Return Code	Explanation
0	Residence volume initialized
4	Residence volume not initialized due to an error, or a warning was issued during AMDSADMP assembly
8	Residence volume not initialized; GENPRINT could not be opened

See *z/OS MVS System Messages, Vol 1 (ABA-AOM)* for more information about AMD064I.

Considerations when using one-stage generation

When generating the stand-alone dump program using one-stage generation, do the following:

- Ensure that the SYSLIB DDNAME concatenates SYS1.MODGEN to SYS1.MACLIB. Your installation should catalog the SYS1.MODGEN data set before generating the stand-alone dump program. Otherwise, the JCL that stand-alone dump produces will fail to create the stand-alone dump program.
- If you are generating stand-alone dump for residence on a direct access volume, AMDSAOSG creates and loads a SYS1.PAGEDUMP.Vvolser data set containing the stand-alone dump program and places an IPL text on the volume. If the volume already contains a SYS1.PAGEDUMP.Vvolser data set, AMDSAOSG will fail. While AMDSAOSG is running, the mount attribute of the volume must be PRIVATE.
- When generating the stand-alone dump program from a Magnetic Tape Subsystem, be aware of which tape format you use or you might not be able to IPL the program. Specifically, IPL processing will end abnormally if you:
  - Generate stand-alone dump on a 3490E Magnetic Tape Subsystem and use a tape subsystem other than a 3490E for IPL.
  - Generate stand-alone dump on a tape subsystem other than a 3490E and use a 3490E Magnetic Tape Subsystem for the IPL.

Using one-stage generation of stand-alone dump when migrating

When migrating to a new version of MVS, generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program. Always use a stand-alone dump version that is generated from the same release of MVS that you want to dump. IBM does not guarantee that a different level of stand-alone dump can successfully dump anything other than the level of MVS it was designed for. The new version of MVS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal one-stage generation, then add the following steps:

- Specifying the correct SYSLIB data set to ensure the new version of the AMDSADMP macro is in use.
- Beginning with z/OS V1R12, use the NUCLIB parameters on the AMDSADMP macro invocation to create the correct one-stage JCL.

Use the following JCL examples for DASD when migrating to a new level of MVS:

- One-stage generation JCL (beginning with z/OS V1R12) for a DASD, see Figure 41 on page 90.
- One-stage generation JCL (any release prior to z/OS V1R12) for a DASD, see Figure 42 on page 91.

The JCL shown in Figure 41 on page 90 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the NUCLIB is specified, one-stage JCL uses the SYS1.NUCLEUS system data sets found on the 3390 DASD with volser=NEWSYS. The stand-alone dump is saved on DASD device 560 in the SYS1.SADMP data set.

```
//ASSEMSAD JOB MSGLEVEL=(1,1)
//OSG      EXEC PGM=AMDSAOSG,REGION=5M
//STEPLIB  DD DSN=SYS1.LINKLIB,DISP=SHR,UNIT=3390,
//            VOL=SER=NEWSYS
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR,
//            UNIT=3390,VOL=SER=NEWSYS
//         DD DSN=SYS1.MODGEN,DISP=SHR,UNIT=3390,
//            VOL=SER=NEWSYS
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
//            SPACE=(4096,(2,1)),UNIT=3390
//DSFSYSIN DD DSN=&DSFSYSIN,DISP=(,PASS),
//            SPACE=(80,(4,1)),UNIT=3390
//GENPRINT DD SYSOUT=*
//GENPARMS DD *
  AMDSADMP IPL=DSYSDA,VOLSER=SADASD,                           X
         DUMP=('DATASPACES OF ASID('XCFAS',                    X
         'CTTX','APPC')'),                                     X
         MINASID=ALL,PROMPT,MSG=ALL,                           X
         CONSOLE=((020,3277),(030,3277),                       X
         (040,3277),(050,3277)),                               X
         OUTPUT=D560,                                          X
         NUCLIB=(NEWSYS,3390)
   END
/*
//PUTIPL   EXEC PGM=ICKDSF,REGION=4M
//IPLDEV   DD DISP=OLD,UNIT=SYSDA,
//            VOL=(PRIVATE,RETAIN,SER=SADASD)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN    DD DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
```

Figure 41. One-stage generation JCL for a DASD (beginning with z/OS V1R12)

Figure 42 on page 91 shows JCL that assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. It uses SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS as suggested by IPLTEXT, IPITEXT, DVITEXT, DPLTEXT and PGETEXT. The stand-alone dump is saved on DASD device 560 in the SYS1.SADMP data set.

```
//ASSEMSAD JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAOSG,REGION=5M
//STEPLIB DD DSN=SYS1.LINKLIB,DISP=SHR,UNIT=3390,
//         VOL=SER=NEWSYS
//SYSLIB  DD DSN=SYS1.MACLIB,DISP=SHR,UNIT=3390,
//         VOL=SER=NEWSYS
//         DD DSN=SYS1.MODGEN,DISP=SHR,UNIT=3390,
//         VOL=SER=NEWSYS
//IPLTEXT  DD DSN=SYS1.NUCLEUS(AMDSAIPD),DISP=SHR,
//         VOL=SER=NEWSYS,UNIT=3390
//IPITEXT  DD DSN=SYS1.NUCLEUS(AMDSAIPI),DISP=SHR,
//         VOL=SER=NEWSYS,UNIT=3390
//DVITEXT  DD DSN=SYS1.NUCLEUS(AMDSADVI),DISP=SHR,
//         VOL=SER=NEWSYS,UNIT=3390
//DPLTEXT  DD DSN=SYS1.NUCLEUS(AMDSADPL),DISP=SHR,
//         VOL=SER=NEWSYS,UNIT=3390
//PGETEXT  DD DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR,
//         VOL=SER=NEWSYS,UNIT=3390
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(,PASS),
//         SPACE=(4096,(2,1)),UNIT=3390
//DSFSYSIN DD DSN=&DSFSYSIN,DISP=(,PASS),
//         SPACE=(80,(4,1)),UNIT=3390
//GENPRINT DD SYSOUT=*
//GENPARMS DD *
  AMDSADMP IPL=DSYSDA,VOLSER=SADASD,                            X
           DUMP=('DATASPACES OF ASID('XCFAS',                   X
           'CTTX','APPC')'),                                    X
           MINASID=ALL,PROMPT,MSG=ALL                           X
           CONSOLE=((020,3277),(030,3277),                      X
                   (040,3277),(050,3277)),                      X
           OUTPUT=D560
  END
/*
//PUTIPL   EXEC PGM=ICKDSF,REGION=4M
//IPLDEV   DD DISP=OLD,UNIT=SYSDA,
//            VOL=(PRIVATE,RETAIN,SER=SADASD)
//TRK0TEXT DD DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN    DD DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
```

Figure 42. Example: One-stage generation JCL (any release) for a DASD (any release)

Use the following JCL examples for tape when migrating to a new level of MVS:

- One-stage JCL (beginning with z/OS V1R12) for tape (Figure 43 on page 91)
- One-stage JCL (any release prior to z/OS V1R12) for tape (Figure 44 on page 92)

The JCL shown in Figure 43 on page 91 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because NUCLIB is specified, it uses the SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS. Only one job step is necessary because the stand-alone dump program is saved on a tape 5B0 with a volume serial of SADMPT. The output gets directed to the data set SYS1.SADMP.SAMPLE on DASD 450.

```
//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAOSG,REGION=5M
//STEPLIB DD DSN=SYS1.LINKLIB,DISP=SHR,UNIT=3390,
//         VOL=SER=NEWSYS
//SYSLIB  DD DSN=SYS1.MACLIB,DISP=SHR,VOL=SER=NEWSYS
//         DD DSN=SYS1.MODGEN,DISP=SHR,VOL=SER=NEWSYS
//GENPARMS DD *
  AMDSADMP IPL=T5B0,VOLSER=SADMPT,                            X
           OUTPUT=(D450,SYS1.SADMP.SAMPLE),                   X
           DDSPROMPT=YES,                                     X
           NUCLIB=(NEWSYS,3390),                              X
           CONSOLE=((3E0,3278),(3E1,3278))
  END
/*
```

Figure 43. Example: One-stage JCL (beginning with z/OS V1R12) for tape

Figure 44 on page 92 shows JCL that assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. It uses the SYS1.NUCLEUS system data set found on 3390 DASD with volser=NEWSYS. Because the stand alone dump program is being saved on a tape, only one job step is necessary. The stand alone dump program is saved on TAPE 5B0 with volume serial of SADMPT. The output goes to the data set SYS1.SADMP.SAMPLE on DASD 450.

```
//SADMPGEN JOB MSGLEVEL=(1,1)
//OSG EXEC PGM=AMDSAOSG,REGION=5M
//STEPLIB DD DSN=SYS1.LINKLIB,DISP=SHR,UNIT=3390,
//        VOL=SER=NEWSYS
//SYSLIB  DD DSN=SYS1.MACLIB,DISP=SHR,VOL=SER=NEWSYS
//        DD DSN=SYS1.MODGEN,DISP=SHR,VOL=SER=NEWSYS
//IPLTEXT  DD DSN=SYS1.NUCLEUS(AMDSAIPT),DISP=SHR,
//       UNIT=3390,VOL=SER=NEWSYS
//IPITEXT  DD DSN=SYS1.NUCLEUS(AMDSAIPI),DISP=SHR,
//       UNIT=3390,VOL=SER=NEWSYS
//DVITEXT  DD DSN=SYS1.NUCLEUS(AMDSADVI),DISP=SHR,
//       UNIT=3390,VOL=SER=NEWSYS
//DPLTEXT  DD DSN=SYS1.NUCLEUS(AMDSADPL),DISP=SHR,
//       UNIT=3390,VOL=SER=NEWSYS
//PGETEXT  DD DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR,
//       UNIT=3390,VOL=SER=NEWSYS
//GENPARMS DD *
  AMDSADMP IPL=T5B0,VOLSER=SADMPT,                              X
           OUTPUT=(D450,SYS1.SADMP.SAMPLE),                     X
           DDSPROMPT=YES,                                       X
           CONSOLE=((3E0,3278),(3E1,3278))
  END
/*
```

Figure 44. Example: One-stage JCL (any release) for tape

Two-stage generation

In two-stage generation of the stand-alone dump program, you must perform two tasks:

1. Assemble the AMDSADMP macro
2. Initialize the residence volume

After you code the AMDSADMP macro, you can assemble the macro. Use the JCL shown in Figure 45 on page 92 to assemble the AMDSADMP macro. The SYSLIB data set must contain the AMDSADMP macro.

```
//ASSEMSAD     JOB        MSGLEVEL=(1,1)
//ASM          EXEC       PGM=ASMA90,REGION=4096K,PARM='DECK'
//SYSLIB       DD         DSN=SYS1.MACLIB,DISP=SHR
//             DD         DSN=SYS1.MODGEN,DISP=SHR
//SYSUT1       DD         UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT     DD         SYSOUT=(*,,STD),HOLD=YES
//SYSPUNCH     DD         DSN=D10.SYS420.STAGE3.JCL(SADMPST2),DISP=SHR
//SYSLIN       DD         SYSOUT=H
//SYSIN        DD         *
               AMDSADMP   MINASID=ALL,IPL=DSYSDA,                        X
                 DUMP=('DATASPACES OF ASID('XCFAS','CTTX','APPC')'),     X
                 VOLSER=XXXXXX,                                          X
                 CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)),  X
                 PROMPT,MSG=ALL,                                         X
                 OUTPUT=T560
               END
/*
```

Figure 45. Example: Stage-two JCL to assemble the AMDSADMP macro

The output of the assembly is a job stream that can be used to initialize the residence volume. The output of the assembly can be directed to a DASD or tape device by coding the SYSPUNCH DD card, as shown in Table 17 on page 92.

Table 17. Directing the output of assembly

Direct assembly output	**SYSPUNCH DD statement**
Tape	`//SYSPUNCH DD UNIT=tape,LABEL=(,NL),DISP=(NEW,KEEP),` `// VOL=SER=volser`

Table 17. Directing the output of assembly (continued)	
Direct assembly output	**SYSPUNCH DD statement**
New direct access data set	`//SYSPUNCH DD UNIT=dasd,SPACE=(80,(30,10)),DSN=dsname,` `// DISP=(NEW,KEEP),VOL=SER=volser`

Assembling multiple versions of AMDSADMP

You can assemble multiple versions of AMDSADMP at the same time, provided that each version specifies a different residence volume. Differentiate between versions by coding a unique symbol at thebeginning of each macro. AMDSADMP uses the symbol you indicate to create unique stage-two job names. The output from a multiple assembly is a single listing and a single object deck, which can be broken into separate jobs if desired. Use the JCL shown in Figure 46 on page 93 for coding multiple versions of AMDSADMP.

```
//MULTISAD JOB  MSGLEVEL=(1,1)
//ASM      EXEC PGM=ASMA90,PARM='DECK,NOOBJ'
//SYSLIB   DD   DSN=SYS1.MACLIB,DISP=SHR
//         DD   DSN=SYS1.MODGEN,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSIN    DD   *
TAPE       AMDSADMP IPL=T3400,VOLSER=SADMP1
DASD1      AMDSADMP VOLSER=SADMP2,MINASID=PHYSIN
DASD2      AMDSADMP VOLSER=SADMP3
           END
/*
```

Figure 46. Example: Assembling multiple versions of AMDSADMP Macro

Initializing the residence volume

When you are generating stand-alone dump for residence on a direct access volume using the stage-two JCL, a SYS1.PAGEDUMP.Vvolser data set containing the stand-alone dump program is created, loaded, and IPL text is placed on the volume. If the volume already contains a SYS1.PAGEDUMP.Vvolser data set, the stage-two job fails. While the stage-two job is running, the mount attribute of the volume must be PRIVATE.

Physical output from the assembly part of the initialization step is a listing for the stand-alone dump common communication table (CCT) and devices and dump options (DDO) control blocks that contain information specified at generation time. The remainder of the output consists of informational, error, and action messages from both stand-alone dump and, when the residence volume is direct access, the device utility ICKDSF.

When generating the stand-alone dump program from a Magnetic Tape Subsystem, be aware of which tape format you use or you might not be able to IPL the program. Specifically, IPL processing will end abnormally if you:

- Generate stand-alone dump on a 3490E Magnetic Tape Subsystem and use a tape subsystem other than a 3490E for the IPL.
- Generate stand-alone dump on a tape subsystem other than a 3490E and use a 3490E Magnetic Tape Subsystem for the IPL.

Using two-stage generation of stand-alone dump when migrating

When migrating to a new version of MVS, generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program.

Always use a stand-alone dump version that is generated from the same release of MVS that you want to dump. IBM does not guarantee that a different level of stand-alone dump will successfully dump anything other than the level of MVS it was designed for. The new version of MVS might have changed making the stand-alone dump program unable to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal two-stage generation, then add the following steps:

- Ensure that the new version of the AMDSADMP macro is being used by specifying the correct SYSLIB data set.
- Use the NUCLIB, MODLIB, LNKLIB and/or ALIB parameters on the AMDSADMP macro invocation to create the correct stage-two JCL.

The output shown in Figure 47 on page 94 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS, SYS1.MODGEN, and SYS1.LINKLIB system data sets, also found on the 3390 DASD with volser=NEWSYS.

```
//ASSEMSAD JOB  MSGLEVEL=(1,1)
//ASM      EXEC PGM=ASMA90,REGION=4096K,PARM='DECK'
//SYSLIB   DD   DSN=SYS1.MACLIB,DISP=SHR,
//             UNIT=3390,VOL=SER=NEWSYS
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=(*,,STD),HOLD=YES
//SYSPUNCH DD   DSN=D10.SYS430.STAGE3.JCL(SADMPST2),DISP=SHR
//SYSLIN   DD   SYSOUT=H
//SYSIN    DD   *
           AMDSADMP   MINASID=ALL,IPL=DSYSDA,                          X
               DUMP=('DATASPACES OF ASID('XCFAS','CTTX','APPC')'),     X
               VOLSER=SADUMP,                                          X
               CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)),  X
               PROMPT,MSG=ALL,                                         X
               OUTPUT=T560,                                            X
               ALIB=(NEWSYS,3390)
           END
/*
```

Figure 47. Example: Stage-two JCL to assemble the AMDSADMP macro

Note: Using the ALIB parameter is convenient if all of the system data sets used by the stand-alone dump program reside on the same volume. Also, note that the same results could have been achieved by coding the NUCLIB, MODLIB, and LNKLIB keywords separately with each specifying NEWSYS and 3390 for volser and unit.

Figure 48 on page 94 shows the output that assembles the version of the AMDSADMP macro contained in the SYSLIB data set, SYS1.MACLIB, found on a 3390 DASD with volser=NEWSYS. Because the MODLIB parameter is specified, the stage-two JCL will use the SYS1.MODGEN system data set found on a 3380 DASD with volser=SYS51A. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS and SYS1.LINKLIB system data sets found on a 3390 DASD with volser=SYS51B.

```
//ASSEMSAD JOB  MSGLEVEL=(1,1)
//ASM      EXEC PGM=ASMA90,REGION=4096K,PARM='DECK'
//SYSLIB   DD   DSN=SYS1.MACLIB,DISP=SHR,
//             UNIT=3390,VOL=SER=NEWSYS
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=(*,,STD),HOLD=YES
//SYSPUNCH DD   DSN=D10.SYS430.STAGE3.JCL(SADMPST2),DISP=SHR
//SYSLIN   DD   SYSOUT=H
//SYSIN    DD   *
           AMDSADMP   MINASID=ALL,IPL=DSYSDA,                          X
               DUMP=('DATASPACES OF ASID('XCFAS','CTTX','APPC')'),     X
               VOLSER=SADUMP,                                          X
               CONSOLE=((020,3277),(030,3277),(040,3277),(050,3277)),  X
               PROMPT,MSG=ALL,                                         X
               OUTPUT=T560,                                            X
               MODLIB=(SYS51A,3380),                                   X
               ALIB=(SYS51B,3390)
           END
/*
```

Figure 48. Example: Stage-two JCL to assemble the AMDSADMP macro

Note that the ALIB parameter has no effect on the SYS1.MODGEN system data set because the MODLIB parameter was specified separately. The stand-alone dump program will be generated using the cataloged system data sets if the NUCLIB, MODLIB, LNKLIB, or ALIB parameters are not specified.

Using two-stage generation for overriding

When overriding to use a different systems database, IBM recommends that you generate a new version of the stand-alone dump program. Use the new MVS system data sets to build the new version of the stand-alone dump program.

Although the current version of the stand-alone dump program might be able to dump a new version of MVS successfully, it is not guaranteed. MVS might have changed such that the stand-alone dump program would not be able to locate vital information it needs to operate.

To generate a new version of the stand-alone dump program, follow the same steps you followed for a normal two-stage generation, then add the following steps:

- Ensure that the new version of the AMDSADMP macro is being used by specifying the correct SYSLIB data set.
- Use the NUCLIB, MODLIB, LNKLIB and/or ALIB parameters on the AMDSADMP macro invocation to create the correct stage-two JCL.

The output shown in Figure 49 on page 95 assembles the version of the AMDSADMP macro contained in the SYSLIB data set SYS1.MACLIB, found on a 3390 DASD with volser=OVRIDE. Because the ALIB parameter is specified, the stage-two JCL will use the SYS1.NUCLEUS, SYS1.MODGEN, and SYS1.LINKLIB system data sets, also found on the 3390 DASD with volser=OVRIDE.

```
// EXEC ASMAC,PARM.C='DECK,NOOBJECT'
//C.SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR,UNIT=3390,VOL=SER=OVRIDE
//C.SYSPUNCH DD DSN=SMITH.TEST.CNTL(SADMP#2),DISP=OLD
//C.SYSIN DD *
        AMDSADMP IPL=D3390,                  IPL FROM DASD                  X
               VOLSER=SADIPL,                VOL=SER=SADIPL                 X
               OUTPUT=(D330,SYS1,SADMP),     DEFAULT OUTPUT DEVICE          X
               MSG=ALL,                      ALL MESSAGES TO CONSOLE        X
               DUMP=('SP(ALL) IN ASID(1) ALSO DATASPACES OF ASID(1,'JESXCF',
               'APPC','SMSVSAM',  'CONSOLES','SYSBMAS') ALSO PAGETABLES OF
               DATASPACES                                                   X
               ALSO SP(ALL) IN ASID('JES2')'),                              X
               MINASID=ALL,                  INCLUDE SWAPPED OUT SPACES     X
               REUSEDS=CHOICE,               PROMPT FOR DATASET REUSE       X
               DDSPROMPT=NO,                 OVERRIDE BUILD DATASETS        X
               ALIB=(OVRIDE,3390)
           END
/*
```

Figure 49. Example: Stage-Two JCL to assemble the AMDSADMP macro with overrides

Note: Using the ALIB parameter is convenient if all of the system data sets used by the stand-alone dump program reside on the same volume. Also, note that the same results could have been achieved by coding the NUCLIB, MODLIB, and LNKLIB keywords separately with each specifying NEWSYS and 3390 for volser and unit.

Running the stand-alone dump program

The operator usually takes a stand-alone dump for one of the following types of problems:

- Disabled wait
- Enabled wait
- Loop
- Partial system hang

When one of these problems occurs, the stand-alone dump program, residing in the SYS1.PAGEDUMP.Vvolser data set, can be run to produce a stand-alone dump. There are several procedures that can be used to run the stand-alone dump program:

- "Procedure A: Initialize and run stand-alone dump" on page 96
- "Procedure B: Restart stand-alone dump" on page 99
- "Procedure C: ReIPL stand-alone dump" on page 100
- "Procedure D: Dump the stand-alone dump program" on page 100

When to use which procedure:

- Use Procedure A to initialize the stand-alone dump program and dump storage.
- If you want to run stand-alone dump again, for instance when stand-alone dump fails, use Procedure B, Procedure C, or procedure D.
- When you want to restart stand-alone dump, try procedure B before you try Procedure C or D.
- Procedures C and D can result in the loss of some central storage from the output, whereas Procedure B usually does not.

Although the stand-alone dump program was created under the operating system, it runs as a stand-alone operation.

Procedure A: Initialize and run stand-alone dump

Use the following procedure to initialize and run a stand-alone dump.

1. Ready the residence device. If it is a tape, mount the volume on a device attached to the selected processor and ensure that the tape cartridge is write-enabled. If it is a DASD volume, ensure that it is write-enabled. A CCW IPL when not write-enabled will fail. A List-directed IPL when not write-enabled will succeed, but message AMD127I will be issued, and some pages in the dump may contain SADMP code or work areas instead of the original contents of the storage at the time that the dump was initiated.

   If you mirror the SADMP IPL volume to a device in an alternate subchannel set and have swapped so that the devices in the alternate subchannel set are now in use, prefix the SADMP IPL device number with the subchannel set id when you specify the SADMP IPL device. In this case, DASD SADMP output devices should be defined in the alternate subchannel set as well.
2. If dumping a failed stand-alone dump program, in order to diagnose the stand-alone dump failure, select the Store Status option during the IPL, or perform a manual Store Status. Otherwise, for all other cases, do `not perform a Store Status` because the machine automatically performs a Store Status when it is necessary.
3. IPL stand-alone dump, using Load Normal. Do not use Load Clear. If the Load function offers a choice between an OS or a dump program, choose a dump program. List-directed can be used for a DASD volume if LDIPL=YES was specified when SADMP was generated. AutoIPL of SADMP by a system that was IPLed via List-directed load will be a List-directed load, so SADMP must be generated with LDIPL=YES in that case.

   Stand-alone dump does not communicate with the operator console. Instead, stand-alone dump loads an enabled wait PSW with wait reason code X'3E0000'. The IPLing of the stand-alone dump program causes absolute storage (X'0'-X'18' and storage beginning at X'FC0') to be overlaid with CCWs. You should be aware of this and not consider it as a low storage overlay.

   Note: Stand-alone dump uses the PSW to communicate with the operator or system programmer.

   Stand-alone dump waits for a console I/O interrupt or an external interrupt.
4. When stand-alone dump is IPLed, you can specify a load parameter (load parm) that alters the operation of stand-alone dump. The format of the load parm is S*addddo*.

   S
      The constant S must be specified as the first character or the load parm will be ignored.

a

The *a* specification allows stand-alone dump to start using a console without the operator performing any action on it. It also allows stand-alone dump to bypass the prompts for which output device and default dump title to use. You can specify the following values for *a*:

N

No console communication requested. Use default dump device and title. Execution begins with no console messages. No prompting to the operator is allowed. If a prompt occurs, a wait state will be loaded.

O

Use the default console with the default dump device and title. No prompting to the operator is allowed. If a prompt occurs, a wait state will be loaded.

M

Use the default console with the default dump device and title. Additional prompts can be made to the operator if they are needed.

C

Use the default console. The operator must respond to all prompts.

P

Wait for an interrupt from the console device that is to be used. If you do not supply the load parm, this is the default.

dddd

The *dddd* specification is the default console device. It must be one of the devices specified as a console device on the AMDSADMP macro when the stand–alone dump was generated, or the constant SYSC for the hardware system console. If you do not specify a default console device, then the stand–alone dump will use the first console defined on the AMDSADMP macro when the stand–alone dump was generated.

The AMDSADMP macro allows you to specify SYSC as the first console in the console list. If you do this without specifying a console device in the load parm, the hardware system console will be the default console device.

o

The *o* field contains flags, and the second bit (bit 1) indicates that SADMP must start an IPL of MVS at the conclusion of its processing. If bit 1 is on, and SADMP locates an AutoIPL policy within MVS storage that specifies a re-IPL of MVS, SADMP uses the information to initiate an IPL of MVS. For details about AutoIPL, see topic about Using the automatic IPL function in *z/OS MVS Planning: Operations*.

The valid values for the *o* field are '0', '4' or blank. '0' or blank leaves all bits off. '4' sets bit 1 on. Bit 1 is intended to automate the re-IPL of MVS when SADMP is initiated manually. IBM recommends that it be left off otherwise.

If you do not use the load parm, select the system console or an operator console with a device address that is in the console list that you specified at stand-alone dump generation time (in the CONSOLE keyword of AMDSADMP). At stand-alone dump run time, the operator can choose either a console specified with the CONSOLE= keyword or the system console to control stand-alone dump operation. If an operator console is chosen, press ATTENTION or ENTER on that console. (On some consoles, you might have to press RESET first.) This causes an interruption that informs stand-alone dump of the console's address. Message AMD001A appears on the console.

a. Ready an output device. When you dump to devices that have both real and virtual addresses (for example, dumping a VM system), specify the virtual address to the stand-alone dump program. If you are dumping to tape, ensure that the tape cartridge is write-enabled and unlabeled. If you are dumping to DASD, ensure that the DASD data set has been initialized using the IPCS SADMP or AMDSADDD REXX dump data set utilities.
b. Reply with the device number for the output device. Note: A DASD output device must be in the same subchannel set as the device from which the stand-alone dump is IPLed, and only the specification of a 4-digit device number is allowed. So, for example, if in reply to the device number prompt, you enter 4180 and the stand-alone dump has been IPLed from device

15660, device 4180 is assumed to be in subchannel set 1, or in this example, 14180. If you are dumping to a DASD device and DDSPROMPT=YES was specified on the AMDSADMP macro, message AMD002A is issued to prompt the operator for a dump data set. IF DDSPROMPT=NO was specified, message AMD002A is not issued and the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.

Note:

i) Pressing ENTER in response to message AMD001A will cause the stand-alone dump program to use the default device specified on the OUTPUT= keyword of the AMDSADMP macro. If the default device is a DASD device, then pressing the ENTER key in response to message AMD001A will cause the stand-alone dump program to use both the default device and the dump data set specified on the OUTPUT= keyword of the AMDSADMP macro. If no dump data set name was provided on the OUTPUT= keyword and the DDSPROMPT=YES keyword was specified, message AMD002A is issued to prompt the operator for a dump data set. If DDSPROMPT=NO was specified, then the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.

ii) If you reply with the device number of an attached device that is not of the required device type, or if the device causes certain types of I/O errors, stand-alone dump might load a disabled wait PSW. When this occurs, use procedure B to restart stand-alone dump.

c. Stand-alone dump prompts you, with message AMD011A, for a dump title.

In Figure 50 on page 98, the dump is initialized using a load parm with no console prompts.

```
AMD083I   AMDSADMP:  STAND-ALONE DUMP INITIALIZED
AMD101I   OUTPUT DEVICE: 0330 SADMP1 SYS1.SADMP
          SENSE ID DATA: FF 3990 E9 3390 0A  BLOCKSIZE:  24,960
AMD005I   DUMPING OF REAL STORAGE NOW IN PROGRESS.
AMD005I   DUMPING OF PAGE FRAME TABLE COMPLETED.
AMD005I   DUMPING OF REAL STORAGE FOR MINIMAL ASIDS COMPLETED.
AMD005I   DUMPING OF REAL STORAGE FOR SUMMARY ASIDS COMPLETED.
AMD005I   DUMPING OF REAL STORAGE FOR SWAPPED-IN ASIDS COMPLETED.
AMD005I   DUMPING OF REAL STORAGE IN-USE REAL STORAGE COMPLETED.
AMD005I   DUMPING OF REAL STORAGE SUSPENDED.
AMD108I   DUMPING OF AUXILIARY STORAGE FOR MINIMAL ASIDS COMPLETED
AMD108I   DUMPING OF AUXILIARY STORAGE FOR SUMMARY ASIDS COMPLETED
AMD108I   DUMPING OF AUXILIARY STORAGE FOR SWAPPED-IN ASIDS COMPLETED
AMD108I   DUMPING OF AUXILIARY STORAGE FOR SWAPPED-OUT ASIDS COMPLETED
AMD056I   DUMPING OF AUXILLIARY STORAGE COMPLETED.
AMD005I   DUMPING OF REAL STORAGE RESUMED.
AMD005I   DUMPING OF AVAILABLE REAL STORAGE COMPLETED
AMD005I   DUMPING OF REAL STORAGE COMPLETED.
AMD104I   STAND-ALONE DUMP PROCESSING COMPLETED.
          DEVICE   VOLUME      USED      DATA SET NAME
    1     0330        SADMP1      43%       SYS1.SADMP
```

Figure 50. Example: Using a load parm to perform a stand-alone dump

5. When no console is available, run stand-alone dump without a console.

   a. Ready the default output device that was specified on the OUTPUT parameter on the AMDSADMP macro. For tapes, ensure that the tape cartridge is write-enabled. For DASD, ensure that the dump data set has been initialized using the AMDSADDD REXX or IPCS SADMP dump data set utilities.

   b. Enter an external interruption on the processor that stand-alone dump was IPLed from. Stand-alone dump proceeds using the default output device and/or the default dump data set.
   No messages appear on any consoles; stand-alone dump uses PSW wait reason codes to communicate to the operator.

6. Stand-Alone dump first processes the real storage in ASID order. The message AMD005I is issued after each phase to display the status of the dump.

   a. Phase 1 dumps the Page Frame Table and its related structures in virtual order.

   b. The next three phases dump real storage associated with the minimal, summary and swapped-in ASIDs in virtual order.

   c. Phase 5 dumps the In-Use real storage in real order.

7. Stand-Alone dump processes the paged-out storage in virtual order based on customer specifications. Message AMD108I is issued to display the status of the virtual phase of the dump
   a. Phases 6 to 8 dumps the paged-out storage of minimal, summary and swapped-in ASIDs. At end of phase VIII, all storage associated with the swapped-in ASIDs has been dumped.
   b. Phase 9 dumps the storage of swapped-out ASIDs.
8. Stand-Alone dump proceeds to dump the available real storage in Phase 10. The storage dumped during this phase includes the real frames that were not dumped earlier. At the completion of this phase, message AMD104I is issued to signal the end of the dump.
9. When stand-alone dump begins dumping real storage (Phase 1 to Phase 5 and Phase 10) it issues message AMD005I. Message AMD095I is issued every 30 seconds to illustrate the process of the dump. Message AMD005I will be issued as specific portions of real storage have been dumped, as well as upon completion of the real dump. Stand-alone dump can end at this step.
10. When stand-alone dump is dumping virtual storage, it issues message AMD108I as specific portions of virtual storage is dumped. Message AMD056I is issued to signal the end of virtual phase dump.
11. If you specified PROMPT on the AMDSADMP macro, stand-alone dump prompts you for additional storage that you want dumped by issuing message AMD059D.
12. Stand-alone dump dumps paged-out virtual storage, the stand-alone dump message log, and issues message AMD095I every 30 seconds to illustrate the progress of the dump.
13. When stand-alone dump completes processing, stand-alone dump unloads the tape, if there is one, and enters a wait reason code X'410000'.

See *z/OS MVS System Codes* for more information about the wait state reason codes loaded into the PSW.

Note: Some processor models do not allow selection of a specific processor to IPL from. Normally, the processor previously IPL'ed is selected again for this IPL.

Procedure B: Restart stand-alone dump

A system restart does not always work, either because it occurs at a point when stand-alone dump internal resources are not serialized, or because stand-alone dump has been too heavily damaged to function. If the restart does not work, try procedure C (reIPL).

If a dump to a DASD data set is truncated because there is not enough space on the data set to hold the dump, use a system restart to dump the original data to tape. By causing a system restart, you can reinitialize and restart a failing stand-alone dump program without losing the original data you wanted to dump.

If a permanent error occurs on the output device, the stand-alone dump program will prompt the operator to determine if a restart of the stand-alone dump program should be performed. If the operator indicates that a restart of the stand-alone dump program should be performed, then the stand-alone dump program restarts the dump using the same console and prompts the operator to specify a different output device. Continue procedure A at step 6A; see "Procedure A: Initialize and run stand-alone dump" on page 96.

For other types of stand-alone dump errors and wait states, it might be necessary for the operator to perform a manual restart of the stand-alone dump program. In this case, the operator should perform the following steps:

1. Perform a system restart on the processor that you IPLed stand-alone dump from.
2. If the restart is successful, stand-alone dump dumps central storage. If stand-alone dump abnormally ends while dumping central storage, try to restart stand-alone dump. If the restart succeeds, stand-alone dump reruns the entire dump. It will first enter wait state X'3E0000' to allow you to specify a new console and output device. You can do this to recover from an I/O error on the output device. Stand-alone dump recognizes any console in the console list and starts with the same output device defaults that are used at the IPL of stand-alone dump.
3. Continue procedure A at step 3, see "Procedure A: Initialize and run stand-alone dump" on page 96.

Procedure C: ReIPL stand-alone dump

When you reIPL stand-alone dump, the previous running of stand-alone dump has already overlaid some parts of central storage and modified the page frame table. If the virtual storage dump program was in control, a reIPL might not dump paged-out virtual storage. The number of times that you can IPL stand-alone dump to dump paged-out virtual storage is equal to the number of processors present.

To run procedure C, repeat procedure A. If the previous IPL of stand-alone dump did not load a wait state and reason code of X'250000' or higher and the reIPL succeeds, stand-alone dump usually completes processing as in procedure A. Some storage locations might not reflect the original contents of central storage because, during a previous IPL, stand-alone dump overlaid the contents. These locations include the absolute PSA and possibly other PSAs.

Procedure D: Dump the stand-alone dump program

Use a new IPL of stand-alone dump to debug stand-alone dump if stand-alone dump fails. When you use stand-alone dump to dump itself, the dump program dumps central storage only, because a dump of central storage provides enough information to diagnose a stand-alone dump error. Follow procedure A at step “2” on page 96 by performing a STORE STATUS instruction. Stand-alone dump follows procedure A steps 2 through 6, then issues message AMD088D. This message allows the operator to stop the dump after central storage has been dumped or to continue dumping virtual storage.

Stand-alone self-dump

When running a virtual storage dump and stand-alone dump error recovery detects errors in stand-alone dump, stand-alone dump can take a self-dump before proceeding. At most, stand-alone dump takes twelve self-dumps; after the twelfth request for a self-dump, stand-alone dump stops taking self-dumps, but continues to count the number of self-dump requests and continues to issue the AMD066I message. After a large number of self-dump requests, stand-alone dump terminates. Stand-alone dump places both the self-dump and the operating system dump onto the output tape or DASD.

You can use the LIST subcommand of IPCS to print stand-alone dump self-dumps. The format of the subcommand is as follows, where x = 001 - 012. See *z/OS MVS IPCS Commands* for more information.

```
LIST address COMPDATA(AMDSAxxx)
```

Running the stand-alone dump program in a sysplex

The operator usually takes a stand-alone dump in a sysplex when an MVS system is not responding. Situations that indicate that stand-alone dump should be run include:

- Consoles do not respond
- MVS is in a WAIT state
- An MVS system is in a “status update missing” condition and has been or is waiting to be removed from the sysplex
- A stand-alone dump has been requested by Level 2.

There are two high-level methods for taking a stand-alone dump of an MVS system that resides in a sysplex. Both methods emphasize the expeditious removal of the failing MVS system from the sysplex. If the failed MVS system is not partitioned out of the sysplex promptly, some processing on the surviving MVS systems might be delayed.

Method A

Use this method to take a stand-alone dump of an MVS system that resides in a sysplex. Assume that the MVS system to be dumped is “SYSA”.

1. IPL the stand-alone dump program on SYSA (see “Running the stand-alone dump Program”).

2. Issue VARY XCF,SYSA,OFFLINE from another active MVS system in the sysplex if message IXC402D or IXC102A is not already present.

   You do not have to wait for the stand-alone dump to complete before issuing the VARY XCF,SYSA,OFFLINE command.
3. Reply DOWN to message IXC402D or IXC102A.

Performing steps 2 and 3 immediately after IPLing stand-alone dump will expedite sysplex recovery actions for SYSA and allow resources held by SYSA to be cleaned up quickly, thus enabling other systems in the sysplex to continue processing.

After you IPL the stand-alone dump, MVS cannot automatically ISOLATE system SYSA through SFM. Message IXC402D or IXC102A issues after the VARY XCF,SYSA,OFFLINE command or after the XCF failure detection interval expires. You must reply DOWN to IXC402D or IXC102A before sysplex partitioning can complete.

Note: DO NOT perform a SYSTEM RESET in response to IXC402D, IXC102A after the IPL of stand-alone dump. The SYSTEM RESET is not needed in this case because the IPL of stand-alone dump causes a SYSTEM RESET. After the IPL of stand-alone dump is complete, it is safe to reply DOWN to IXC402D or IXC102A.

Method B

Use this method if there is a time delay between stopping processors, as part of the SYSTEM RESET-NORMAL function in step one, and IPLing the stand-alone dump program.

1. Perform the SYSTEM RESET-NORMAL function on SYSA.
2. Issue VARY XCF,SYSA,OFFLINE from another active MVS system in the sysplex if message IXC402D or IXC102A is not present.
3. Reply DOWN to message IXC402D or IXC102A. Performing steps two and three immediately after doing the SYSTEM RESET will expedite sysplex recovery actions for SYSA. It allows resources that are held by SYSA to be cleaned up quickly, and enables other systems in the sysplex to continue processing.
4. IPL the stand-alone dump program (see “Running the stand-alone dump program” on page 95). While this step can be done earlier, the aim of performing steps one through three is to minimize disruption to other systems in the sysplex.

After a SYSTEM RESET is performed, MVS cannot automatically ISOLATE system SYSA through SFM. Message IXC402D or IXC102A is issued after the VARY XCF,SYSA,OFFLINE command or after the XCF failure detection interval expires. You must reply DOWN to IXC402D or IXC102A before sysplex partitioning can complete.

Capturing a stand-alone dump quickly

There are times when you need to process stand-alone dump information quickly to diagnose a problem. It is important to perform the stand-alone dump process quickly, to minimize the time the system is unavailable. Sometimes a stand-alone dump is not captured because of the time that the dumping process takes. Skipping the stand-alone dump, however, can prevent the diagnosis of the system failure. Instead of skipping the stand-alone dump, it is better to spend a short time to get as much of the stand-alone dump as is possible, as quickly as possible. The following are two methods to save time when performing a stand-alone dump.

- Minimize the operator actions
- Get a partial stand-alone dump

Minimize the operator actions

Time spent waiting for the operator to reply to a message or mount a tape is idle time. Minimizing the operator actions turns the idle time into data capture time. It also simplifies the process, so that the stand-alone dump process becomes easier to do. The following are ways to minimize the operator's actions when performing a dump.

- Use the stand-alone dump LOAD parameter *SO* or *SM* to skip the prompt for the console to use to avoid other responses to messages.
- Use the default device specified on the OUTPUT= keyword of the AMDSADMP macro. If the default device is a DASD device, then pressing the ENTER key in response to message AMD001A will cause the stand-alone dump program to use both the default device and the dump data set specified on the OUTPUT= keyword of the AMDSADMP macro.
- Use REUSEDS=ALWAYS on the AMDSADMP macro to indicate that stand-alone dump should reuse the dump data set on the specified output device when it determines that the data set is valid, however, it can contain data from a previous dump. Or, you can always clear the dataset.

  Note: Be sure you do not overwrite another dump.
- Specify DDSPROMPT=NO, then the stand-alone dump program assumes that the dump data set name is SYS1.SADMP.
- Do not specify PROMPT on the AMDSADMP macro, unless requested by IBM.
- Use "fast" device for output

Get a partial stand-alone dump

While it is always best to get a complete stand-along dump, sometimes time constraints will not allow this. There is no guarantee that it will be possible to diagnose a failure from a partial stand-alone dump; however, if the choice is between no dump at all or a partial dump, then the partial dump is the best choice.

When taking a partial stand-alone dump:

- Let the stand-alone dump run for as long as you can. If you run out of time, you can stop the dump cleanly.
- Stand-alone dump tries to write out the most important information first.
  - Status information (PSW, registers, and so forth) for all CPUs
  - Critical real storage, including common storage and trace information
  - Real storage for address spaces executing at the time of the dump
  - Any remaining real storage
  - Paged out storage for swapped in address spaces
  - Paged out storage for swapped out address spaces
- Use the EXTERNAL INTERRUPT key to terminate the dumping process. This causes a clean stop, closing the output dataset properly.

In Figure 51 on page 102, the dump was ended early using the EXTERNAL INTERRUPT key.

```
AMD083I   AMDSADMP:  STAND-ALONE DUMP RESTARTED
AMD094I   0330 SADMP1 SYS.SADMP
          IS VALID, HOWEVER, IT MAY ALREADY CONTAIN DATA FROM A PREVIOUS DUMP.
          THE INSTALLATION CHOSE TO ALWAYS REUSE THE DUMP DATA SET.
AMD101I   OUTPUT DEVICE: 0330 SADMP1 SYS1.SADMP
          SENSE ID DATA: FF 3990 E9 3390 0A  BLOCKSIZE: 24,960
AMD005I   DUMPING OF REAL STORAGE NOW IN PROGRESS.
AMD005I   DUMPING OF PAGE FRAME TABLE COMPLETED.
AMD005I   DUMPING OF REAL STORAGE FOR MINIMAL ASIDS COMPLETED.
AMD005I   DUMPING OF REAL STORAGE FOR SUMMARY ASIDS COMPLETED.
AMD089I   DUMP TERMINATED DUE TO EXTERNAL KEY
AMD066I   AMDSADMP ERROR, CODE=0012, PSW=040810008101235E, COMPDATA9AMDSA002)
```

Figure 51. Example: Terminating a stand-alone dump

Copying, viewing, and printing stand-alone dump output

When stand-alone dump processing completes the dump, the output resides on a tape volume, a DASD, or a combination of devices. The easiest way to view the dump is to copy the dump to a DASD data set. When a stand-alone dump resides on multiple devices and/or dump data sets, you can concatenate the dump into one data set. After the dump is available on DASD, it can be viewed online using IPCS.

Note: If the dump resides in a DASD dump data set, IBM recommends that you copy the dump to another data set for IPCS processing and clear (reinitialize) the dump data set using the AMDSADDD or IPCS SADMP dump data set utilities. For more information, see "Using the AMDSADDD utility" on page 79 and SADMP option on the IPCS Dialog in *z/OS MVS IPCS User's Guide*.

Copying the dump to a data set

If you want to view the dump online, copy the dump to a data set. There are two tools you can use to copy the dump:

- Use the IPCS COPYDUMP subcommand when the IPCS environment has been set up on your system. This is the only option recommended if the dump was written to a multi-volume DASD data set.
- Use the IEBGENER utility when the IPCS environment has not been set up on your system. Many operators take a stand-alone dump so that the system programmer can view the dump. The operator does not require IPCS on the system because the operator will not be viewing the dump. Therefore, the operator should use the IEBGENER utility to copy the dump to a data set accessible by the system programmer's system.

For more information, see the following references:

- See *z/OS MVS IPCS Commands* for information about COPYDUMP.
- See *z/OS DFSMSdfp Utilities* for information about IEBGENER.

Copying from tape

The example below shows how to use IEBGENER to copy tape output to DASD. Two advantages of copying stand-alone dump tape output to DASD are:

- When stand-alone dump ends prematurely and does not give the stand-alone dump output (SYSUT1) an end-of-file, the SYSUT2 data set does contain an end-of file.(SYSUT2 is the data set to which stand-alone dump output is copied.) This occurs even when SYSUT2 is another tape. IEBGENER might end with an I/O error on SYSUT1; this is normal if SYSUT1 does not contain an end-of-file.
- Making SYSUT2 a direct access data set to use as input to IPCS saves IPCS processing time.

Use the JCL shown in Figure 52 on page 103 to invoke the IEBGENER utility, which will copy the stand-alone dump output from tape to a DASD data set.

```
//SADCOPY  JOB  MSGLEVEL=(1,1)
//COPY     EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT1   DD   DSN=SADUMP.TAPE,UNIT=tape,
//         VOL=SER=SADOUT,LABEL=(,NL),DISP=SHR,
//         DCB=(RECFM=FBS,LRECL=4160,BLKSIZE=29120)
//SYSUT2   DD   DSN=SADUMP.COPY,UNIT=dasd,
//         VOL=SER=SADCPY,DISP=(NEW,CATLG),
//         DCB=(RECFM=FBS,LRECL=4160,DSORG=PS),AVGREC=K,
//         SPACE=(4160,(8,4),RLSE)
```

Figure 52. Example: Copying stand-alone dump output from tape to DASD

Note: Specifying AVGREC= requires SMS be running, but the data set does not have to be SMS managed.

Copying from DASD

The example below shows how to use IEBGENER to copy DASD output to a DASD data set. After the dump is successfully copied, use the AMDSADDD REXX utility to clear (reinitialize) the dump data set and ready it for another stand-alone dump. For more information, see:

- SADMP option on the IPCS Dialog in *z/OS MVS IPCS User's Guide*.
- "Using the AMDSADDD utility" on page 79

Use the JCL shown in Figure 53 on page 104 to invoke IEBGENER, which will copy the stand-alone dump output from a DASD data set to another DASD data set.

```
//SADCOPY  JOB  MSGLEVEL=(1,1)
//COPY     EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT1   DD   DSN=SYS1.SADMP,UNIT=DASD,
//         VOL=SER=SADMP1,DISP=SHR
//SYSUT2   DD   DSN=SYS2.SADMP,UNIT=DASD,
//         DISP=(NEW,CATLG),
//         VOL=SER=SADMP2,
//         DCB=(LRECL=4160,RECFM=FBS,DSORG=PS),
//         SPACE=(CYL,(90,0),RLSE)
```

Figure 53. Example: Copying stand-alone dump output from DASD to DASD

Copying from multiple dump data sets

The stand-alone dump program allows a dump to be contained in multiple dump data sets. Therefore, when you want to view a stand-alone dump using IPCS, it is necessary to concatenate all of the dump data sets onto one DASD data set.

Use the JCL in Figure 54 on page 104 to invoke the IPCS COPYDUMP subcommand to copy stand-alone dump output from three DASD dump data sets to another data set. Note that two of the dump data sets reside on the volume SADMP1, while the third resides on the volume SADMP2.

```
//SADCOPY  JOB MSGLEVEL=(1,1)
//COPY     EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=A
//C1       DD DSN=SADMP1.DDS1,DISP=SHR,UNIT=DASD,VOL=SER=SADMP1
//C2       DD DSN=SADMP1.DDS2,DISP=SHR,UNIT=DASD,VOL=SER=SADMP1
//C3       DD DSN=SYS1.SADMP,DISP=SHR,UNIT=DASD,VOL=SER=SADMP2
//COPYTO   DD DSN=SADUMP.COPY,UNIT=DASD,
//            VOL=SER=SADCPY,DISP=(NEW,CATLG),
//            DCB=(RECFM=FBS,LRECL=4160,DSORG=PS),
//            SPACE=(4160,(8000,4000),RLSE)
//SYSTSIN DD  *
  IPCS NOPARM DEFER
  COPYDUMP OUTFILE(COPYTO) NOCONFIRM INFILE(C1, C2, C3)
  END
/*
```

Figure 54. Example: Copying a stand-alone dump from multiple DASD data sets

Use the JCL shown in Figure 55 on page 105 to invoke the IPCS COPYDUMP subcommand to copy stand-alone dump output from two DASD dump data sets and two tape volumes to a DASD data set.

```
//SADCOPY  JOB MSGLEVEL=(1,1)
//COPY     EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=A
//C1       DD  DSN=SYS1.SADMP.MAIN.DDS1,DISP=SHR,UNIT=DASD,
//             VOL=SER=SADMP1
//C2       DD  DSN=SYS1.SADMP.ALTERNAT.DDS1,DISP=SHR,UNIT=DASD,
//             VOL=SER=SADMP2
//C3       DD  DSN=SYS1.SADMP.MAIN.DDS1,DISP=SHR,UNIT=TAPE,
//             VOL=SER=SADMP3
//C4       DD  DSN=SYS1.SADMP.ALTERNAT.DDS1,DISP=SHR,UNIT=TAPE,
//             VOL=SER=SADMP4
//COPYTO   DD  DSN=SADUMP.COPY,UNIT=DASD,
//             VOL=SER=SADCPY,DISP=(NEW,CATLG),
//             DCB=(RECFM=FBS,LRECL=4160,DSORG=PS),
//             SPACE=(2080,(1600,800),RLSE)
//SYSTSIN DD  *
  IPCS NOPARM
  COPYDUMP OUTFILE(COPYTO) NOCONFIRM INFILE((C1,C2,C3,C4)
  END
/*
```

Figure 55. Example: Copying stand-alone dump output from DASD and tape

Viewing stand-alone dump output

You can view the stand-alone dump output at a terminal using IPCS. Do the following:

1. Start an IPCS session.
2. On the IPCS Primary Option Menu panel, select the SUBMIT option to copy the dump and do initial dump analysis.
3. Return to the IPCS Primary Option Menu panel. Select the DEFAULTS option.
4. IPCS displays the IPCS Default Values panel. Enter the name of the data set containing the dump on the Source line.
5. Return to the IPCS Primary Option Menu panel. Select the BROWSE, ANALYZE, or COMMAND option to view the dump.

See *z/OS MVS IPCS Commands* for information about the IPCS subcommands.

Printing stand-alone dump output

You can print an analysis of the stand-alone dump or the entire dump using IPCS.

To print an analysis of the dump in batch mode:

1. Start an IPCS session.
2. On the IPCS Primary Option Menu panel, select the SUBMIT option to copy the dump and do initial dump analysis.
3. On the IPCS Dump Batch Job Option Menu panel, enter the requested information.
4. On the next panel, enter the sysout output class. IPCS writes the dump analysis to the specified output class.
5. The system prints the dump in the printout of the output class.

To print the full dump in batch mode:

1. Use IPCS CLIST BLSCBSAP.

See *z/OS MVS IPCS User's Guide* for IPCS panels and the CLIST BLSCBSAP.

The example in Figure 56 on page 106 runs an IPCS CLIST that:

- Copies the stand-alone dump from the tape data set defined in an IEFRDER DD statement to a cataloged, direct access data set named SA1DASD.
- Analyzes and formats the dump.

- Writes the formatted dump output to a data set named IPCSPRNT. A TSO/E CLIST used for IPCS should allocate this print output data set to a sysout print class, as follows:

```
ALLOCATE DDNAME(IPCSPRNT) SYSOUT(A)
```

After the CLIST runs, the dump remains available in the SA1DASD data set for supplementary formatting jobs.

```
//PRINTJOB  JOB  MSGLEVEL=1,REGION=800M
//IPCS      EXEC IPCS,CLIST=BLSCBSAP,DUMP=SA1DASD
//IEFPROC.IEFRDER  DD DSN=SA1,DISP=OLD,UNIT=3490
//            VOL=SER=12345,LABEL=(1,NL)
/*
```

Figure 56. Example: Printing an unformatted stand-alone dump

Message output

There are three types of message output from a stand-alone dump program, as follows:

- MNOTES from the AMDSADMP macro
- Messages on the 3480, 3490, or 3590 display
- Messages on the system console or the operator console

For more information about messages on the system console or the operator console, see *MVS System Messages*.

Stand-alone dump messages on the 3480, 3490, or 3590 display

When stand-alone dump output is sent to a 3480, 3490, or 3590 magnetic tape subsystem, stand-alone dump uses the subsystem's eight-character message display to inform and prompt the operator. The leftmost position on the message display indicates a requested operator action. The eighth position (rightmost) gives additional information.

In the messages listed below, alternating indicates that there are two messages which are flashing on the display, one after the other. A blinking message is one message that is repeated on the display.

The stand-alone dump messages that can appear on the display are:

Dvolser (alternating)

MSADMP#U
: Informs the operator that a labeled tape has been rejected and a new tape must be mounted.

MSADMP#U (blinking)
: Requests that the operator mount a new tape.

RSADMP#U (blinking)
: Indicates that the stand-alone dump program has finished writing to the tape.

RSADMP# (alternating)

MSADMP#U
: Informs the operator that an end-of-reel condition has occurred and a new tape must be mounted.

SADMP# (blinking)
: Indicates that the tape is in use by stand-alone dump.

SADMP# (alternating)

NTRDY
: Informs the operator that some type of intervention is required.

The symbols used in the messages are:

#
A variable indicating the actual number of cartridges mounted for stand-alone dump. It is a decimal digit starting at 1 and increasing by 1 after each end-of-cartridge condition. When the # value exceeds 9, it is reset to 0.

D
Demount the tape and retain it for further system use, for example as a scratch tape. Stand-alone dump does not write on the tape.

M
Mount a new tape.

R
Demount the tape and retain it for future stand-alone dump use.

U
The new tape should not be file-protected.

volser
A variable indicating the volume serial number on the existing tape label.

Analyzing stand-alone dump output

The following sections describe how to analyze the output from a stand-alone dump. A stand-alone dump can indicate the following types of problems:

- Enabled wait state
- Disabled wait state
- Enabled loop
- Disabled loop

Use the information in this section to determine the type of problem the system has encountered. After the problem type is determined, see *z/OS Problem Management* for further information about diagnosing the problem type.

Collecting initial data

When an operator takes a stand-alone dump, it is important to determine the conditions of the system at the time the dump was taken. Because a stand-alone dump can be requested for a various number of problem types, the collection of problem data is imperative to determining the cause of the error.

The objectives for analyzing the output of a stand-alone dump are as follows:

- Gather symptom data
- Determine the state of the system
- Analyze the preceding system activity
- Find the failing module and component

Gathering external symptoms

When a stand-alone dump is taken, you must first question the operator or the person who requested the dump. It is important to understand the external symptoms leading up to the system problem. What was noticed before stopping the system? The answer might give you an idea of where the problem lies.

Here are a few questions you should find an answer to before continuing:

- Was the system put into a wait state?
- Were the consoles hung or locked up?
- Were commands being accepted on the operator console without a reply?
- Was a critical job or address space hung?

Gathering IPCS symptoms

After getting a list of symptoms, use IPCS to collect further symptom data. A primary symptom string is usually not available in a stand-alone dump; however, IPCS can add a secondary symptom string. In Figure 57 on page 108, the explanation of the secondary symptom string indicates an enabled wait state condition.

```
                         * * * *  S Y M P T O M  * * * *
ASR10001I The dump does not contain a primary symptom string.
  Secondary Symptom String:

    WS/E000 FLDS/ASMIORQR VALU/CPAGBACKUP FLDS/IOSCOD VALU/CLCLC0D45
    FLDS/IOSTSA VALU/CLCLDEV02

    Symptom            Symptom data     Explanation
    ---------------    -------------    -----------
    WS/E000            000              Enabled wait state code
    FLDS/ASMIORQR      ASMIORQR         Data field name
    VALU/CPAGBACKUP    PAGBACKUP        Error related character value
```

Figure 57. Example: VERBEXIT SYMPTOMS output

Determining the system state

There are several control blocks that you can view that describe the state of the system when the stand-alone dump was requested.

CSD
Describes the number of active central processors and whether the alternate CPU recovery (ACR) is active.

PSA
Describes the current environment of a central processor, its work unit, FRR stack, an indication of any locks held.

LCCA
Contains save areas and flags of interrupt handlers.

CVT
Contains pointers to other system control blocks.

Use the IPCS subcommand STATUS WORKSHEET to obtain the data that will help you determine the state of the system. For example, in Figure 58 on page 109 look for the following:

- The CPU bit mask, which indicates how many processors are online.
- The PSW at the time of the dump
- The PSATOLD. If the fields are zero, this indicates that an SRB is running and the address in SMPSW indicates the save area of the dispatcher. If the fields are nonzero, the address in PSWSV indicates the save area of the dispatcher.
- The PSAAOLD, which indicates what address space jobs are running in.

```
                             MVS Diagnostic Worksheet
Dump Title: SYSIEA01 DMPDSENQ 7/20/93
CPU Model 2064 Version 00 Serial no. 145667 Address 00
Date: 03/20/2001   Time: 05:41:26  Local
 SYSTEM RELATED DATA

 CVT  SNAME (154) ESYS       VERID (-18)
      CUCB   (64) 00FD4B68   PVTP  (164) 00FE4A10    GDA  (230) 01BE1168
      RTMCT (23C) 00F81198   ASMVT (2C0) 00FD8030    RCEP (490) 012AA3F0

 CSD  Available CPU mask: C000  Alive CPU mask: C000  No. of active CPUs: 0002

PROCESSOR RELATED DATA

 NAME             OFFSET  |  CPU 00    CPU 01
--------------------------+----------------------------------------
 PSW at time of dump      | 070E0000  070C9000
                          | 00000000  8124EE9C
 CR0 Interrupt mask       | 5EB1EE40  5EB1EE40
 CR6 I/O class mask       | FE        FE
------ LCCA --------------+----------------------------------------
 IHR1   Recursion    208  | 00        00
 SPN1/2 Spin         20C  | 0000      0000
 CPUS   CPU WSAVT    218  | 00F4BA00  00F6F550
 DSF1/2 Dispatcher   21C  | 0000      0080
 CRFL   ACR/LK flgs  2B4  | 00000000  00000000
------ PSA ---------------+----------------------------------------
 TOLD   Curr TCB     21C  | 00000000  00000000
 ANEW   ASCB         220  | 00FD3BC0  00F56180
 AOLD   Curr ASCB    224  | 00FD3280  00F56180
 SUPER  Super Bits   228  | 04000000  00000000
 CLHT   Lock Table   280  | 00FD4890  00FD4890
 LOCAL  Local lock   2EC  | 00000000  00F0D700
 CLHS   Locks held   2F8  | 00000000  00000001
 CSTK   FRR stack    380  | 00F4D4D0  00000C00
 SMPSW  SRB Disp PSW 420  | 070C0000  070C0000
                     424  | 81142B60  82039000
 PSWSV  PSW Save     468  | 070E0000  070E0000
                     46C  | 00000000  00000000
 MODE   Indicators   49F  | 08        04
```

Figure 58. Example: STATUS WORKSHEET output

You can also obtain the stored status of each central processor using the IPCS subcommand STATUS CPU REGISTERS. Watch for these bits in the first half of the PSW:

- Bits 6 and 7 indicate a disabled (04xxxxxx) or enabled (07xxxxxx) condition
- Bit 14 could indicate a wait (000A0000)
- Bits 16 and 17 indicate primary, secondary, access register (AR) or Home mode

Starting in V2R1, the worksheet displays 4-digit CPUIDs. There can be up to eight CPUs on one line, if allowed. For example:

```
PROCESSOR RELATED DATA
|
|  NAME             OFFSET  | CPU 0001  CPU 0003  CPU 0005  CPU 0006  CPU 0007  CPU 0008
CPU 0009  CPU 000A
| -------------------------
+--------------------------------------------------------------------------------
|  PSW at time of dump      | 00020000  00020000  04047000  00020000  00020000  00020000
00020000  00020000
|                           | 80000000  80000000  80000000  80000000  80000000  80000000
80000000  80000000
|                           | 00000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000
|                           | 0000001B  0000001B  10F994EC  0000001B  0000001B  0000001B
0000001B  0000001B
|  CR0 Interrupt mask       | 00800002  00800002  00800002  00800002  00800002  00800002
00800002  00800002
|  CR6 I/O class mask       | 00        00        00        00        00        00
00        00
| ------ LCCA --------------
+--------------------------------------------------------------------------------
```

In Figure 59 on page 110, the PSW indicates an enabled wait state condition. The program is running in primary mode with 24-bit addressing (bits 16 and 17 are 00 and the second word begins with 0).

```
CPU(X'0000') STATUS:
 PSW=070E0000  00000000  NO WORK WAIT
  ASCB1 at FD3280, JOB(*MASTER*), for the home ASID
  ASXB1 at FD34F8 for the home ASID. No block is dispatched
 CLTE: 01CB00E8
  +0000  BLSD..... 00000000  XDS...... 00000000  XRES..... 00000000
  +000C  XQ....... 00FD4900  ESET..... 00FD4908  ULUT..... 00FD4910
  CURRENT FRR STACK IS: SVC
  PREVIOUS FRR STACK(S): NORMAL

  GPR VALUES
      0-3  00000000  00000000  00000000  00000000
      4-7  00000000  00000000  00000000  00000000
      8-11 00000000  00000000  00000000  00000000
     12-15 00000000  00000000  00000000  00000000

  ACCESS REGISTER VALUES
      0-3  006FB01F  00000000  00000000  00000000
      4-7  00000000  00000000  00000000  00000000
      8-11 00000000  00000000  00000000  00000000
     12-15 00000000  00000000  806FA03C  00000000
```

Figure 59. Example: STATUS CPU REGISTERS output

To obtain other fields from important control blocks, use the IPCS subcommand CBFORMAT. See *z/OS MVS IPCS Commands* for information about the CBFORMAT subcommand.

You can also use the WHERE subcommand to identify particular areas in the dump. For example, if a general purpose register contains an address, use the WHERE subcommand to determine in what module that address resides. In Figure 60 on page 110, the WHERE subcommand indicates that the address is part of the READONLY nucleus.

```
NOCPU ASID(X'0001') 0124EE9C. IEANUC01.IGVSLIS1+0ADC IN READ ONLY NUCLEUS
```

Figure 60. Example: WHERE subcommand output

Analyzing an enabled wait

An enabled wait is also known as a dummy wait or a no work wait. An indication of an enabled wait is a PSW of **070E0000 00000000** or **07060000 00000000 00000000 00000000** and GPRs containing all zeroes. An enabled wait occurred when the dispatcher did not find any work to be dispatched. An enabled wait can occur because of resource contention or system non-dispatchability, among other errors.

Reviewing outstanding I/O requests

When analyzing a stand-alone dump for an enabled wait condition, check the status of the input/output requests. A display of the IOS control block and any active UCBs can help determine what was happening when the system entered the wait state.

In Figure 61 on page 111, the HOTIO field indicates that a solicited interrupt has completed with other than DCC-3 because the last time HOT-I/O detection was called. Note also that the IOQF and IOQL fields are identical, indicating that the first and last request for this device is the same.

```
                      * * * ACTVUCBS   Processing * * *
UCB AT 00F8B798:  DEVICE  001; SUBCHANNEL 0001
 UCBPRFIX: 00F8B768
    -0030  RSTEM.... 00        RSV...... 08        MIHTI.... 40
    -002D  HOTIO.... 40        IOQF..... 00F7BC00  IOQL..... 00F7BC00
    -0024  SIDA..... 0001      SCHNO.... 0001      PMCW1.... 2888
    -001E  MBI...... 0000      LPM...... 80        RSV...... 00
    -001A  LPUM..... 80        PIM...... 80        CHPID.... 21000000
    -0014            00000000  LEVEL.... 01        IOSF1.... 00
    -000E  MIHCT.... 0000      LVMSK.... 00000001  LOCK..... 00000000
    -0004  IOQ...... 00F7BC00
```

Figure 61. Example: IOSCHECK ACTVUCBS Subcommand output

Analyzing for resource contention

You can obtain information related to resource contention by using the IPCS subcommand ANALYZE. This subcommand displays contention information for I/O, ENQs, suspend locks, allocatable devices and real storage. For example, in Figure 62 on page 111, 61 units of work are waiting to be processed. The top RB is in a wait state.

```
                 CONTENTION EXCEPTION REPORT
JOBNAME=*MASTER*  ASID=0001  TCB=006E8E88
JOBNAME=*MASTER* HOLDS THE FOLLOWING RESOURCE(S):

   RESOURCE #0011:There are 0061 units of work waiting for this resource
      NAME=MAJOR=SYSIEA01 MINOR=DMPDSENQ SCOPE=SYSTEM

 STATUS FOR THIS UNIT OF WORK:
   This address space is on the SRM IN queue.
   Task non-dispatchability flags from TCBFLGS4:
    Top RB is in a wait
```

Figure 62. Example: ANALYZE subcommand output

Obtaining real storage data

Use the IPCS RSMDATA subcommand to obtain information about storage usage and any unusual condition that have occurred prior to requesting the stand-alone dump. In the RSMDATA output, if the percent usage field is 100%, there are no frames left. Also, the percent of available total fixed frames should not be a high number. If it is, there can be a program using too many resources to complete. For example, in Figure 63 on page 111, the percent of available total fixed frames is at 25%.

```
                      R S M   S U M M A R Y   R E P O R T
                           Tot real Below Prf real Dbl real      Expanded
                           -------- ----- -------- -------- -------------
In configuration . . . .     33,792 4,096   33,742        -        49,152
Available for allocation     32,672 4,089   33,742      120        49,152
Allocated  . . . . . . .     32,398 3,964   33,483      113        48,594
Percent usage  . . . . .         99    96       99       94            98
Common fixed frames  . .      3,087   317    3,087        -             -
  Percent of available .          9     7        9        -             -
Total fixed frames . . .      8,338   907        -        -             -
  Percent of available .         25    22        -        -             -
```

Figure 63. Example: RSMDATA output

You can also check the ASM control blocks to determine the statistics applicable to I/O requests. The I/O requests received and completed should be the same. In Figure 64 on page 112, note that the 509577 I/O requests received have all been completed.

```
ASMVT AT 00FD8030
509577 I/O REQUESTS RECEIVED, 509577 I/O REQUESTS COMPLETED BY ASM
240487 NON-SWAP WRITE I/O REQUESTS RECEIVED, 240487 NON-SWAP WRITE I/O
       REQUESTS COMPLETE
PART AT 01CB5310
PAGE DATA SET 0 IS ON UNIT 15B
PAGE DATA SET 1 IS ON UNIT 15B
PAGE DATA SET 3 IS ON UNIT 14A
PAGE DATA SET 4 IS ON UNIT 150
PAGE DATA SET 5 IS ON UNIT 15B
```

Figure 64. Example: ASMCHECK output

Determining dispatchability

By performing an address space analysis on the major system address space, you can determine if there is any work waiting and if the address space is dispatchable. The major address space you should analyze are:

- Master scheduler, ASID 1
- CONSOLE
- JES2/JES3
- IMS/CICS/VTAM

When you are analyzing an address space for dispatchability, keep in mind these questions:

- Are there any suspended SRBs on the queue?

  You will need to run the WEBs on ASCBSAWQ and look for WEBs that have a WEBFLAG1 field of X'000000' to check if there are any SRBs ready to be dispatched.
- Are there any ready TCBs indicated by ASCBTCBS and ASCBTCBL?

  ASCBTCBS and ASCBTCBL contain a count of the number of TCBs containing ready work to be dispatched. To find the TCBs for ASCBTCBL, look at the WEBs on the ASCBLTCS and ASCBLTCB queues that belong to the home space.
- If there is ready work, is the ASCB dispatchable (ASCBDSP1)?

  ASCBDSP1 is a non-dispatchability flag. For more information about what the values of ASCBDSP1 indicate, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
- If there is no ready work, are the TCBs in a normal wait (TCBFLGS4, TCBFLGS5, TCBNDSP)?

  A non-zero value in any of these fields indicates that the TCB is non-dispatchable.

In Figure 65 on page 113, ASCBDSP1 is X'04', indicating that this address space is not eligible for CML lock requests. The ASCBSAWQ, ASCBLTCN, and ASCBTCBS fields all contain zeroes, indicating that there is no ready work available.

```
ASCB: 00FD2B80
   +0000  ASCB..... ASCB      FWDP..... 00FC4400  BWDP..... 00000000
   +000C  LTCS..... 00000000  SVRB..... 00F4FBA8  SYNC..... 000727F4
   +0018  IOSP..... 00000000  WQID..... 0000      SAWQ..... 00000000
   +0024  ASID..... 0001      LL5...... 00        HLHI..... 01
   +002A  DPH...... 01FF      LDA...... 7F748EB0  RSMF..... C0
   +0038  CSCB..... 00000000  TSB...... 00000000
   +0040  EJST..... 0000009F  94659288
   +0048  EWST..... AEE06377  45A41803            JSTL..... 000141DE
   +0054  ECB...... 00000000  UBET..... 00000000  TLCH..... 00000000
   +0060  DUMP..... 00699D90  AFFN..... FFFF      RCTF..... 01
   +0067  FLG1..... 00        TMCH..... 00000000  ASXB..... 00FD2EA8
   +0070  SWCT..... 47BE      DSP1..... 00        FLG2..... CE
   +0076  SRBS..... 0000      LLWQ..... 00000000  RCTP..... 00000000
   +0080  LOCK..... 00000000  LSWQ..... 00000000  QECB..... 00000000
   +008C  MECB..... 00000000  OUCB..... 015178E8  OUXB..... 01517BF0
   +0098  FMCT..... 0000      LEVL..... 03        FL2A..... 00
   +009C  XMPQ..... 00000000  IQEA..... 00000000  RTMC..... 00000000
   +00A8  MCC...... 00000000  JBNI..... 00000000  JBNS..... 00FD2B18
   +00B4  SRQ1..... 00        SRQ2..... 00        SRQ3..... 00
   +00B7  SRQ4..... 00        VGTT..... 00CD7458  PCTT..... 1AB6F008
   +00C0  SSRB..... 0000      SMCT..... 00        SRBM..... 07
   +00C4  SWTL..... 00000000  SRBT..... 000015D1  E5E32000
   +00D0  LTCB..... 00000000  LTCN..... 00000000  TCBS..... 00000000
   +00DC  LSQT..... 00000000  WPRB..... 00FD2E90  NDP...... FF
   +00E5  TNDP..... FF        NTSG..... FF        IODP..... FF
   +00E8  LOCI..... 00000000  CMLW..... 00000000  CMLC..... 00000000
   +00F4  SSO1..... 000000    SSO4..... 00        ASTE..... 02900040
   +00FC  LTOV..... 7FFD8400  ATOV..... 7FFDCCA8  ETC...... 0007
   +0106  ETCN..... 0000      LXR...... 0007      AXR...... 0000
   +010C  STKH..... 00FD35C0  GQEL..... 00000000  LQEL..... 00000000
   +0118  GSYN..... 00000000  XTCB..... 006A3D90  CS1...... 00
   +0121  CS2...... 00        GXL...... 02449430
   +0128  EATT..... 0000000E  DAC0D475
   +0130  INTS..... AED8EC7B  0C7C0900            LL1...... 00
   +0139  LL2...... 00        LL3...... 00        LL4...... 00
   +013C  RCMS..... 00000000  IOSC..... 0000450A  PKML..... 0000
   +0146  XCNT..... 01F4      NSQA..... 00000000  ASM...... 00FD3520
   +0150  ASSB..... 00FD2D00  TCME..... 00000000  GQIR..... 00000000
   +0168  CREQ..... 00000000  RSME..... 02219120  AVM1..... 00
   +0171  AVM2..... 00        AGEN..... 0000      ARC...... 00000000
   +0178  RSMA..... 02219000  DCTI..... 0066E2EE
```

Figure 65. Example: SUMMARY FORMAT output (determining ready work)

For the mapping structure of WEBs under the IHAWEB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

If your address space analysis indicated that ready work was available to be dispatched, look at ASCBDSP1 to determine if the address space is dispatchable. If your address space analysis indicated that there was no ready work available to be dispatched, look at the TCBs to determine if they are in a normal wait. In Figure 66 on page 114, for example, the TCB fields indicate that the top RB is in a wait.

```
TCB: 00FD3608
   +0000  RBP...... 006FF048  PIE...... 00000000  DEB...... 00000000
   +000C  TIO...... 00000000  CMP...... 00000000  TRN...... 00000000
   +0018  MSS...... 7F7463A0  PKF...... 00        FLGS..... 00008004  00
   +0022  LMP...... FF        DSP...... FF        LLS...... 006FFD38
   +0028  JLB...... 00000000  JPQ...... 006FF200
 GENERAL PURPOSE REGISTER VALUES
     0-3  00000001  000027C4  00009FBC  00000004
     4-7  006FFF48  006FEFB8  00F6E900  0000005C
     8-11 80001E52  00C0DCE8  006F5FF0  00FCF778
    12-15 00FCF170  006FF348  80FCF1C0  806FF048
   +0070  FSA...... 00000000  TCB...... 006FF6F0  TME...... 00000000
   +007C  JSTCB.... 00FD3608  NTC...... 00000000  OTC...... 00000000
   +0088  LTC...... 006FF6F0  IQE...... 00000000  ECB...... 00000000
   +0094  TSFLG.... 00        STPCT.... 00        TSLP..... 00
   +0097  TSDP..... 00        RD....... 7F748F04  AE....... 7F746280
   +00A0  STAB..... 00F0B860  TCT...... 00000000  USER..... 00000000
   +00AC  NDSP..... 00000000  MDIDS.... 00000000  JSCB..... 00C0BE84
        .
        .
        .
   +014C  BDT...... 00000000  NDAXP.... 00000000  SENV..... 00000000
   Task non-dispatchability flags from TCBFLGS4:
    Top RB is in a wait
```

Figure 66. Example: SUMMARY FORMAT output (determining TCB in normal wait)

Analyzing a disabled wait

A disabled wait condition can be analyzed by checking the PSW at the time of the error. If bits 6 and 7 are zero and bit 14 contains a 1, there is a disabled wait. The wait state code is in byte 7, with the reason code in byte 5.

The following examples show how to determine the wait state code:

- In the following PSW, the wait state code is X'014' and the reason code is zero.

```
PSW=000E0000  00000014
```

- In another example, the wait state code is X'064' and the reason code is X'09'.

```
PSW=000A0000  00090064
```

- In z/Architecture® mode, the PSW would look like:

```
PSW=0002000 00000000 00000000 00090064
```

After you determine the wait state code from the PSW, look at the documentation for the specific wait state code for any action you can take. See *z/OS MVS System Codes* for the specific wait state code you encountered.

If you cannot find the wait state code documented, do one of the following:

- Analyze the dump to determine if it is a stand-alone dump wait state.
- Check PSASMPSW and PSAPSWSV to determine if the dispatcher loaded the wait state PSW because of an overlay. See Chapter 7, “The dump grab bag,” on page 147 for more information about storage overlays.
- Use the stored status registers to determine who loaded the wait state into the PSW.

Analyzing an enabled loop

To determine if the stand-alone dump was requested because of an enabled loop, you need to view the system trace table. Repetitive patterns in the system trace table indicate an enabled loop condition. An enabled loop, however, does not normally cause a system outage. It will cause an outage in these circumstances:

- There is a non-preemptable loop in SRB mode

- There is a loop in a high priority address space that is in TCB mode

In Figure 67 on page 115, the CLKC entries indicate an enabled loop, and because column three is all zeroes, this loop is in SRB mode. The PSW addresses on the CLKCs identify the looping program. Use the WHERE subcommand to locate the responsible program.

```
01 003E 00000000   CLKC      070C0000 8100765C  00001004 00000000
01 003E 00000000   CLKC      070C2000 81005638  00001004 00000000
01 003E 00000000   CLKC      070C0000 810056E6  00001004 00000000

01 003E 00000000   CLKC      070C0000 80FF0768  00001004 00000000

01 003E 00000000   CLKC      070C0000 80FE4E34  00001004 00000000

01 003E 00000000   CLKC      070C1000 81004BB8  00001004 00000000
```

Figure 67. Example: SYSTRACE output (recognizing an enabled loop)

Because of interrupt processing that occurs during an enabled loop, the stored status data might not point to the module causing the loop. To determine if a first level interrupt handler (FLIH) was active, view the PSASUPER field of the PSA. If the PSASUPER field is non-zero, a FLIH was active at the time of the error. Using the FLIH's save area, find the PSW and registers at the time of the error. The address in the second half of the PSW will point to the module involved in the loop. See "Problem data saved by first level interrupt handlers" on page 116 for more information.

Analyzing a disabled loop

A disabled loop is not visible in the system trace output because disabled routines do not take interrupts. Normally, a disabled loop results in a spin loop in a multiprocessor environment. When analyzing a stand-alone dump for a disabled loop, use the stored status data to determine the module involved in the loop. Also, examine the in-storage logrec buffer for entries that recovery routines have made but which were not written to the logrec data set because of a system problem. Very often it is these records that are the key to the problem solution. See "Obtaining information from the logrec recording control buffer" on page 531 for more information.

SLIP problem data in the SLIP work area

In a stand-alone dump taken after a SLIP ACTION=WAIT trap matches, problem data can be found in a work area pointed to by the PSAWTCOD field in the prefix save area (PSA). Table 18 on page 115 shows the format of this area.

Table 18. Work area pointed to by the PSAWTCOD field

Offset	Length	Content
0 (0)	1	RTM/SLIP processing environment indicator: • X'01': RTM1 • X'02': RTM2 • X'03': MEMTERM • X'04': PER
1 (1)	2	Logical processor identifier (CPUID)
3 (3)	1	System mask, if offset 0 is 2 (RTM2)
4 (4)	4	Pointer to general purpose registers 0 through 15 at the time of the event
8 (8)	4	Pointer to the 16-byte program status word (PSW) at the time of the event

Table 18. Work area pointed to by the PSAWTCOD field (continued)		
Offset	**Length**	**Content**
12 (C)	4	One of the following, as indicated by the RTM/SLIP processing environment indicator at offset 0 of the work area: • Pointer to the system diagnostic work area (SDWA), if offset 0 is 1 (RTM1) • Pointer to the recovery termination manager 2 (RTM2) work area (RTM2WA), if offset 0 is 2 (RTM2) • Pointer to the address space control block (ASCB) being ended, if offset 0 is 3 (MEMTERM) • Pointer to the PER code, if offset 0 is 4 (PER)
16 (10)	4	Pointer to cross memory information (control registers 3 and 4) at the time of the event
20 (14)	4	Pointer to access registers AR0 through AR15 at the time of the event. Pointer to the high 32 bits of the 64-bit GPRs, or 0 if not available. See Wait State 01B in the *z/OS MVS System Codes* for more information.

Problem data saved by first level interrupt handlers

If processing is stopped or an error occurs in one of the first level interrupt handlers (FLIH), you might need to determine the PSW and registers of the interrupted program. Field PSASUPER has bits to indicate if an FLIH was in control:

- PSAIO for the IO FLIH
- PSASVC for the SVC FLIH
- PSAEXT for the external FLIH
- PSAPI for the program interrupt FLIH

The following tables show where each FLIH saves the PSW and registers for interrupted tasks or SRB:

- Table 19 on page 116
- Table 20 on page 117
- Table 21 on page 117
- Table 22 on page 118

Table 19. Problem data saved by the SVC FLIH for task and SRB code			
Code giving up control	**Data saved**	**Field receiving data**	**Control block**
All SVCs, initially	General purpose registers 7-9	PSAGPREG	PSA
	General purpose registers, if a problem occurred	LCCASGPR	LCCA
All SVCs	PSW	RBOPSW	Requestor's RB
	Cross memory status	XSBXMCRS	XSB
	PCLINK stack header	XSBSTKE	XSB
	EAX	XSBEAX	XSB
	Access registers 0-15	STCBARS	STCB
	Current linkage stack entry pointer	STCBLSDP	STCB
Type 1 and 6 SVCs	General purpose registers 0-15	TCBGRS	TCB
Type 2, 3, and 4 SVCs	General purpose registers 0-15	RBGRSAVE	SVRB

Table 20. Problem data saved for a program check for task and SRB code

Code giving up control	Data saved	Field receiving data	Control block
Initially for non-recursive program interruptions	General purpose registers 0-15	LCCAPGR2	LCCA
	PSW	LCCAPPSW	LCCA
	ILC/PINT	LCCAPINT	LCCA
	TEA	LCCAPVAD	LCCA
	TEA AR number	LCCAPTR2	LCCA
	Control registers 0-15	LCCAPCR2	LCCA
	Access registers 0-15	LCCAPAR2	LCCA
Initially for recursive program interruptions	General purpose registers 0-15	LCCAPGR1	LCCA
	PSW	LCCAPPS1	LCCA
	ILC/PINT	LCCAPIC1	LCCA
	TEA	LCCAPTE1	LCCA
	TEA AR number	LCCAPTR2	LCCA
	Control registers 0-15	LCCAPCR1	LCCA
	Access registers 0-15	LCCAPAR1	LCCA
Initially for monitor call interruptions that occur during page fault or segment fault processing	General purpose registers 0-15	LCCAPGR3	LCCA
	PSW	LCCAPPS3	LCCA
	ILC/PINT	LCCAPIC3	LCCA
	TEA	LCCAPTE3	LCCA
	TEA AR number	LCCAPTR3	LCCA
	Control registers 0-15	LCCAPCR3	LCCA
	Access registers 0-15	LCCAPAR3	LCCA
Initially for all trace buffer full interruptions	General purpose registers 0-15	LCCAPGR4	LCCA
For unlocked tasks for page faults or segment faults that require I/O; problem data is moved from the LCCA	Registers		TCB and STCB
	PSW		RB
	Other status		XSB
For locked tasks for page faults or segment faults that require I/O; problem data is moved from the LCCA	Registers		IHSA
	PSW		IHSA
	Other status		XSB for IHSA
For SRBs for page faults or segment faults that require I/O; SRB is suspended, no status is saved			

Table 21. Problem data saved by the I/O FLIH for task and SRB code

Code giving up control	Data saved	Field receiving data	Control block
Initially	General purpose registers 0-15	SCFSIGR1	SCFS
	Control registers 0-15	SCFSICR1	SCFS
	Access registers 0-15	SCFSIAR1	SCFS

Table 21. Problem data saved by the I/O FLIH for task and SRB code (continued)			
Code giving up control	**Data saved**	**Field receiving data**	**Control block**
For unlocked tasks	General purpose registers 0-15	TCBGRS	TCB
	PSW	RBOPSW	RB
	Cross memory status	XSBXMCRS	XSB
	EAX	XSBEAX	XSB
	Access registers 0-15	STCBARS	STCB
	Current linkage stack entry pointer	STCBLSDP	STCB
For locally locked tasks	General purpose registers 0-15	IHSAGPRS	IHSA for locked address space
	PSW	IHSACPSW	IHSA for locked address space
	Access registers 0-15	IHSAARS	IHSA for locked address space
	Current linkage stack entry pointer	IHSALSDP	IHSA for locked address space
	Cross memory status	XSBXMCRS	XSB for locked address space
	EAX	XSBEAX	XSB for locked address space
For SRBs and non-preemptive TCBs	General purpose registers 0-15	SCFSIGR1	SCF
	PSW	FLCIOPSW	PSA

Table 22. Problem data saved by the external FLIH for task and SRB code			
Code giving up control	**Data saved**	**Field receiving data**	**Control block**
Initially	General purpose registers 7-10	PSASLSA	PSA
For locally locked tasks	General purpose registers 0-15	IHSAGPRS	IHSA
	PSW	IHSACPSW	IHSA
	Access registers 0-15	IHSAARS	IHSA
	Current linkage stack entry pointer	IHSALSDP	IHSA
	Cross memory status	XSBXMCRS	XSB
	EAX	XSBEAX	XSB
Unlocked tasks	General purpose registers 0-15	TCBGRS	TCB
	PSW	RBOPSW	RB
	Access registers 0-15	STCBARS	STCB
	Current linkage stack entry pointer	STCBLSDP	STCB
	Cross memory status	XSBXMCRS	XSB
	EAX	XSBEAX	XSB

Table 22. Problem data saved by the external FLIH for task and SRB code (continued)

Code giving up control	Data saved	Field receiving data	Control block
For SRBs and non-preemptive TCBs	General purpose registers 0-15	SCFSXGR1	SCFS
	PSW	SCFSXPS1	SCFS
	Control registers 0-15	SCFSXCR1	SCFS
	Access registers 0-15	SCFSXAR1	SCFS
If first recursion	General purpose registers 0-15	SCFSXGR1	SCFS
	PSW	SCFSXPS2	SCFS
	Control registers 0-15	SCFSXCR2	SCFS
	Access registers 0-15	SCFSXAR2	SCFS
If second recursion	General purpose registers 0-15	SCFSXGR3	SCFS
	PSW	FLCEOPSW	PSA
	Control registers 0-15	SCFSXCR3	SCFS
	Access registers 0-15	SCFSXAR3	SCFS

Chapter 5. ABEND dump

An ABEND dump shows the virtual storage predominately for an unauthorized program. Typically, a dump is requested when the program cannot continue processing and abnormally ends. An operator can also request an ABEND dump while ending a program or an address space.

The system can produce three types of ABEND dumps, one unformatted dump (SYSMDUMP) and two formatted dumps (SYSABEND and SYSUDUMP). These dumps are produced when a program cannot continue processing and a DD statement for an ABEND dump was included in the JCL for the job step that has ended. The data included is dependent on:

- Parameters supplied in the IEAABD00, IEADMR00, and IEADMP00 parmlib members for SYSABENDs, SYSMDUMPs, and SYSUDUMPs, respectively.
- A determination by the system
- ABEND, CALLRTM, or SETRP macro dump options
- IEAVTABX, IEAVADFM, or IEAVADUS installation exit processing

IBM recommends the use of SYSMDUMP, the unformatted dump. Unformatted dumping is more efficient because only the storage requested is written to the data set, which allows the application to capture diagnostic data and be brought back online faster. Also, pre-formatted dumps force the system to select a single set of reports, too many for the diagnosis of many problems, and too few for others. Unformatted dumps allow the analyst to determine, from a wide variety of reports, what information to use and how it is presented.

Use SYSUDUMP for diagnosis of program problems that need simple problem data. A SYSABEND dump, through the IBM supplied defaults, supplies more of the system information related to the application program's processing than a SYSUDUMP. The additional information may be better suited for complex problem diagnosis.

Note: The last of dump DDs SYSABEND,SYSUDUMP or SYSMDUMP will be used.

This section covers the following topics, which describe how to use ABEND dumps:

Synopsis of ABEND dumps

Use Table 23 on page 122 as a quick reference for the three types of ABEND dumps. If you need further information about ABEND dumps, refer to the sections following this table.

Table 23. Types of ABEND dumps

Obtaining the dump	Receiving the dump	Dump contents
SYSABEND: Assembler macro in any program: • ABEND with DUMP • SETRP with DUMP=YES Assembler macro in an authorized program: • ABEND with DUMP • CALLRTM with DUMP=YES • SETRP with DUMP=YES Operator command on a console with master authority: • CANCEL with DUMP For full information, see "Obtaining ABEND dumps" on page 123.	Formatted dump in a data set with the ddname of SYSABEND: • In SYSOUT; print in the output class or browse at a terminal • On tape or direct access: print in a separate job or browse at a terminal • On a printer (Not recommended; the printer cannot be used for anything else for the duration of the job step.) For full information, see "Obtaining ABEND dumps" on page 123.	Default contents: summary dump for the failing task and other task data. See "Contents of ABEND dumps" on page 127. Customized by all of the following: • IEAADB00 parmlib member • Parameter list on the requesting ABEND, CALLRTM, or SETRP macro • Recovery routines invoked by the recovery termination manager (RTM) • Cumulative from all CHNGDUMP operator commands with SYSABEND • Installation-written routines at the IEAVTABX, IEAVADFM, and IEAVADUS exits For full information about customization, see "Customizing ABEND dump contents" on page 132.
SYSMDUMP: Assembler macro in any program: • ABEND with DUMP • SETRP with DUMP=YES Assembler macro in an authorized program: • ABEND with DUMP • CALLRTM with DUMP=YES • SETRP with DUMP=YES Operator command on a console with master authority: • CANCEL with DUMP For full information, see "Obtaining ABEND dumps" on page 123.	Unformatted dump in a data set with the ddname of SYSMDUMP: • On tape or direct access; use IPCS to format and print/view the dump For full information, see "Obtaining ABEND dumps" on page 123.	Default contents: summary dump and system data for the failing task. See "Contents of ABEND dumps" on page 127. Customized by all of the following: • IEADMR00 parmlib member • Parameter list on the requesting ABEND, CALLRTM, or SETRP macro • Recovery routines invoked by the recovery termination manager (RTM) • Cumulative from all CHNGDUMP operator commands with SYSMDUMP • Installation-written routines at the IEAVTABX exit For full information about customization, see "Customizing ABEND dump contents" on page 132.

Table 23. Types of ABEND dumps (continued)

Obtaining the dump	Receiving the dump	Dump contents
SYSUDUMP: Assembler macro in any program: • ABEND with DUMP • SETRP with DUMP=YES Assembler macro in an authorized program: • ABEND with DUMP • CALLRTM with DUMP=YES • SETRP with DUMP=YES Operator command on a console with master authority: • CANCEL with DUMP For full information, see "Obtaining ABEND dumps" on page 123.	Formatted dump in a data set with the ddname of SYSUDUMP: • In SYSOUT; print in the output class or browse at a terminal • On tape or direct access; print in a separate job or browse at a terminal • On a printer (Not recommended; the printer cannot be used for anything else for the duration of the job step.) For full information, see "Obtaining ABEND dumps" on page 123.	Default contents: summary dump for the failing task. See "Contents of ABEND dumps" on page 127. Customized by all of the following: • IEADMP00 parmlib member • Parameter list on the requesting ABEND, CALLRTM, or SETRP macro • Recovery routines invoked by the recovery termination manager (RTM) • Cumulative from all CHNGDUMP operator commands with SYSUDUMP • Installation-written routines at the IEAVTABX, IEAVADFM, and IEAVADUS exits For full information about customization, see "Customizing ABEND dump contents" on page 132.

Obtaining ABEND dumps

You can obtain SYSABEND, SYSUDUMP, and SYSMDUMP dumps using one process. To obtain a specific type of ABEND dump, specify the correct DD statement in your JCL as shown in Table 24 on page 123; for more information about these statements, see *z/OS MVS JCL Reference*:

Table 24. Summary: DD statements to specify for specific ABEND dumps

Dump Type	DD statement
SYSABEND	`//SYSABEND DD ...`
SYSUDUMP	`//SYSUDUMP DD ...`
SYSMDUMP	`//SYSMDUMP DD ...`

Provide a data set to receive the dump, then arrange to view the dump. If a data set is not provided, the system ignores a request for an ABEND dump. When setting up the data set, determine if it will contain privileged data. If so, protect it with passwords or other security measures to limit access to it.

Because ABEND dumps provide information to debug application programs, the data they have access to is limited. Authorized programs require special processing to allow the information they can access into a dump. ABEND dump processing issues an IEA848I message when violations occur. The primary facility for dumping authorized data is through the SDUMPX macro, however, two security FACILITY classes are provided that allow installations to permit ABEND dumps to contain authorized data:

IEAABD.DMPAUTH
For access to programs that are protected by the PROGRAM facility, or UNIX set-user-ID and/or set-group-ID programs that gain privilege.

IEAABD.DMPAKEY
For programs that execute in authorized keys. Plus, for ABEND dumps requiring access to GTF trace data.

See *z/OS Security Server RACF Security Administrator's Guide* for additional details. For details on the SDUMPX macro, see

- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *z/OS MVS Programming: Authorized Assembler Services Guide*.

Data set for dump

Define the data set in either:

- The JCL for the job step, for batch processing
- The logon procedure for a TSO/E userid, for foreground processing

Define the data set in a DD statement with a ddname of SYSABEND, SYSMDUMP, or SYSUDUMP. The ddname for the data set determines how the dump can be printed or viewed, what the dump contains, and how the dump contents can be customized. The first two effects are discussed in the following topics.

The system writes the dump in a sequential data set using the basic sequential access method (BSAM). The dump data set can be on any device supported by BSAM. Note that the system provides a data control block (DCB) for the dump data set and opens and closes the DCB.

You can also use extended format sequential data sets as dump data sets for ABEND dumps. Extended format sequential data sets have the following features:

- Have a greater capacity than sequential data sets
- Support striping
- Support compression

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for ABEND dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. Abend dump fully supports placement of dump data sets in cylinder-managed space.

VIO for ADDRSPC=REAL: A SYSMDUMP DD statement must specify a virtual input/output (VIO) data set if the job or step to be dumped is running in nonpageable virtual storage, that is, the JCL JOB or EXEC statement specifies ADDRSPC=REAL.

Preallocate data sets for SYSMDUMP dumps

You may use any dataset name you wish for the SYSMDUMP dataset. However, the dataset name SYS1.SYSMDPxx will be treated specially. If you use the data set naming convention of SYS1.SYSMDPxx for a DISP=SHR data set, the system writes only the first dump, with all subsequent dump requests receiving system message IEA849I. The data set can be either a magnetic tape unit or a direct access storage device (DASD) data set.

When using this naming convention, you must manage the dump data set to use the same data set repeatedly for SYSMDUMP dumps. For subsequent dumps, you must initialize the SYS1.SYSMDPxx data set with an end-of-file (EOF) mark as the first record.

Naming Convention: You must use SYS1.SYSMDPxx, where xx is 00 through FF and identifies the exact data set to be used.

Data Set Disposition: If you specify DISP=SHR with the SYS1.SYSMDPxx naming convention, the facility that enables the system to write only the first dump becomes active.

If you specify DISP=SHR without the SYS1.SYSMDPxx naming convention, the system writes a new dump over the old dump when the same data set is the target for multiple dumps. This also happens for multiple dumps within the same job if each step does not specify FREE=CLOSE on the SYSMDUMP DD statement.

For dispositions other than DISP=SHR, the system uses the data set as if it were any other MVS data set. If you specify DISP=MOD, the system writes the dump following the previous dump, so that the data set contains more than one dump. If you specify DISP=OLD, the system writes a new dump over the old dump when the same data set is the target for multiple dumps.

Data Set Management: To minimize the loss of subsequent dumps, your installation exit should follow these steps for the management of SYS1.SYSMDPxx data sets:

1. Intercept system message IEA993I. The system issues this message when it writes the dump to the SYS1.SYSMDPxx data set.
2. Copy the dump onto another data set.
3. Clear the SYS1.SYSMDPxx data set by writing an EOF mark as the first record, making it available for the next SYSMDUMP dump to be written on the data set.

The installation exit routine can be one of the following:

- IEAVMXIT
- The exit routine specified on the USEREXIT parameter in the MPFLSTxx parmlib member

See *z/OS MVS System Messages, Vol 6 (GOS-IEA)* for a description of system messages IEA849I and IEA993I. See *z/OS MVS Installation Exits* for information about the installation exit routine.

Process for obtaining ABEND dumps

Obtain an ABEND dump by taking the following steps for each job step where you want to code a dump:

1. Code a DD statement in the JCL for every job step where a dump would be needed. The statement can specify one of the following:

   - Direct access
   - SYSOUT
   - Tape
   - Printer (Not recommended; printer cannot be used for anything else for duration of job step.)

   The presence of the DCB attributes enables the system-determined block size process to select an efficient block size for the DASD selected. For more information, see *z/OS DFSMS Using Data Sets*. Your installation can make specification of these attributes unnecessary through local SMS class selection routines.

   For example, the following SYSOUT SYSABEND DD statement places the dump in sysout output class A. In the example, output class A is a print class. The system prints a dump written to this class when printing the class.

   ```
   //SYSABEND  DD  SYSOUT=A
   ```

   The following example places a SYSUDUMP dump on a scratch tape. In the example, TAPE is an installation group name. DEFER specifies that the operator is to mount the tape only when the data set is opened; thus, the operator will not mount the tape unless it is needed for a dump. The system deletes the data set if the job step ends normally; in this case, the data set is not needed because no dump was written. The system keeps the data set if the step ends abnormally; the data set contains a dump. A future job step or job can print the dump.

   ```
   //SYSUDUMP  DD  DSN=DUMPDS,UNIT=(TAPE,,DEFER),DISP=(,DELETE,KEEP)
   ```

2. Place the DD statement in the JCL for the job step that runs the program to be dumped or in the logon procedure for a TSO/E user ID.

   The following example shows a SYSABEND DD statement in the logon procedure for a TSO/E user ID. A dump statement must appear in the logon procedure in order to process a dump in the foreground. The system keeps the data set if the job step ends abnormally.

   ```
   //SYSABEND  DD  DSN=MYID3.DUMPS,DISP=(OLD,,KEEP)
   ```

3. If you need to diagnose a program that does not contain code for an ABEND dump, code one of the following:

   - ABEND assembler macro with a DUMP parameter in a problem program or an authorized program

The following example shows an ABEND macro that ends a program with a user completion code of 1024 and requests a dump:

```
ABEND 1024,DUMP
```

- SETRP assembler macro with a DUMP=YES parameter in the recovery routine for a problem program or an authorized program

  The following example shows a SETRP macro in an ESTAE recovery routine for a problem program. The address of the system diagnostic work area (SDWA) is in register 1, which is the default location.

  ```
  SETRP DUMP=YES
  ```

- CALLRTM assembler macro with a DUMP=YES parameter in an authorized program

  The following example shows a macro in an authorized program. The CALLRTM macro ends a program and requests a dump. Register 5 contains the address of the task control block (TCB) for the program.

  ```
  CALLRTM TYPE=ABTERM,TCB=(5),DUMP=YES
  ```

4. If you need to diagnose a program that already contains code for an ABEND dump, and that program is already abending, skip step 5.
5. If you need to diagnose a program that already contains code for an ABEND dump, but the program is not currently abending, ask the operator to enter a CANCEL command with a DUMP parameter on the console with master authority.

   For example, to cancel a job and request a dump, ask the operator to use either of the following:

   ```
   CANCEL  BADJOB,DUMP
   CANCEL  STARTING,A=1234,DUMP
   ```

   To cancel a user ID and request a dump, ask the operator to use either of the following commands:

   ```
   CANCEL  U=MYID3,DUMP
   CANCEL  U=*LOGON*,A=5678,DUMP
   ```

6. The system writes a formatted dump to the data set defined in step 1.

Printing and viewing dumps

You can print or view the different types of ABEND dumps as follows:

SYSABEND and SYSUDUMP dumps: These two dumps are formatted as they are created. They can be:

- In a SYSOUT data set. The system can print the dump when printing the output class. To view at a terminal, use a facility that allows the viewing of JES SPOOL data sets.
- On a tape or direct access data set. Print the dump in a separate job or job step or view the dump at a terminal by browsing the data set containing the dump.

  A convenient way to print the dump is in a later job step that runs only if an earlier job step abnormally ends and, thus, requests a dump. For this, use the JCL EXEC statement COND parameter.
- Sent directly to a printer. Note this is not recommended; the printer cannot be used for anything else while the job step is running, whether a dump is written or not.

Figure 68 on page 127 shows JCL that uses the IEBPTPCH facility to print a formatted dump data set. In this example, a SYSABEND dump is printed. The same JCL can be used for a SYSUDUMP. Because the system formats the dump when creating it, the IEBPTPCH utility program can print the dump. The dump is in a data set named DUMPDS on tape.

```
              ⋮
//PRINT      EXEC   PGM=IEBPTPCH
//SYSPRINT   DD     SYSOUT=A
//SYSUT1     DD     DSN=DUMPDS,UNIT=TAPE,DISP=(OLD,DELETE)
//SYSUT2     DD     SYSOUT=A
//SYSIN      DD     *
   PRINT     PREFORM=A,TYPORG=PS
/*
```

Figure 68. Example: Using IEBPTPCH to print a dump

SYSMDUMP dumps: This dump is unformatted when created. The system can write the dump to tape or direct access. Use IPCS to format the dump and then view it at a terminal or print it. SYSMDUMP dumps are especially useful for diagnosing errors because IPCS can produce specific information for specific requests. See *z/OS MVS IPCS User's Guide* for more information.

Contents of ABEND dumps

You can specify the contents of an ABEND dump by specifying parameters on the ddname in the JCL for the program. This topic discusses the IBM-supplied default contents and contents available through customization.

All three ABEND dumps contain a summary dump, although the SYSMDUMP summary dump contains less information than the SYSABEND and SYSUDUMP summary dumps. The SYSUDUMP consists of only the summary dump. The SYSABEND dump also contains task data, while the SYSMDUMP also contains system data. The SYSMDUMP dump is a synchronous SVC dump and contains data similar to the data in an SVC dump.

Note:

1. ABEND dumps do not include hiperspaces. To include hiperspace in an ABEND dump, read the data from the hiperspace into address space storage that is being dumped.
2. If some needed areas are not included by default, see "Customizing ABEND dump contents" on page 132 for ways to add the areas.

Determining current ABEND dump options

Use a DISPLAY DUMP operator command to get the dump mode and options in effect for SVC dumps and ABEND SYSABEND, SYSMDUMP, and SYSUDUMP dumps. The system displays the mode and options in message IEE857I.

For example, to determine the mode and options, enter the following command:

```
DISPLAY DUMP,OPTIONS
```

If the options listed are not the ones desired, use a CHNGDUMP operator command to change them.

See *z/OS MVS System Commands* for the DISPLAY and CHNGDUMP operator commands. For a description of these messages, see *MVS System Messages*.

Default contents of ABEND dumps

The contents of the three ABEND dumps are detailed in the following two tables. Table 25 on page 128 shows dump contents alphabetically by the parameters that specify the areas in the dumps. To select a dump, decide what areas will be used to diagnose potential errors. Find the areas in the tables. The symbols in columns under the dump indicate how the area can be obtained in that dump. The symbols are:

D
: IBM-supplied default contents

M
: Available on the macro that requests the dump

P
: Available in the parmlib member that controls the dump options

X
: Available on the CHNGDUMP operator command that changes the options for the dump type

Table 25. Summary: dump contents by parameter

Parameter	Dump Contents	ABEND Dump to SYSUDUMP	ABEND Dump to SYSABEND	ABEND Dump to SYSMDUMP
ALL	All the dump options available in a SYSMDUMP dump, except the NOSYM and ALLNUC options			X
ALLNUC	The DAT-on and DAT-off nucleuses			P X
ALLPA	All link pack areas, as follows: • Job pack area (JPA) • Link pack area (LPA) active for the task being dumped • Related Supervisor Call (SVC) modules	M P X	D M P X	M
ALLPDATA	All the program data areas	P X	P X	
ALLSDATA	All the system data areas	P X	P X	P
ALLVNUC	The entire virtual control program nucleus, including: • Prefixed save area (PSA) • System queue area (SQA) • Local system queue area (LSQA)	M P X	M P X	M
CB	Control blocks for the task being dumped	M P X	D M P X	M
CSA	Common service area (CSA) (that is, subpools 227, 228, 231, 241)			P X
DM	Data management control blocks for the task being dumped: • Data control block (DCB) • Data extent block (DEB) • Input/output block (IOB)	M P X	D M P X	M
ENQ	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks	P X	D P X	

Table 25. Summary: dump contents by parameter (continued)

Parameter	Dump Contents	ABEND Dump to SYSUDUMP	ABEND Dump to SYSABEND	ABEND Dump to SYSMDUMP
ERR	Recovery termination manager (RTM) control blocks for the task being dumped: • Extended error descriptor (EED) for RTM • Registers from the system diagnostic work area (SDWA) • RTM2 work area (RTM2WA) • Set task asynchronous exit (STAE) control block (SCB)	M P X	D M P X	M
GRSQ	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks			P X
IO	Input/output supervisor (IOS) control blocks for the task being dumped: • Execute channel program debug area (EXCPD) • Unit control block (UCB)	M P X	D M P X	M
JPA	Job pack area (JPA): module names and contents	M P X	M P X	M
LPA	Active link pack area (LPA): module names and contents	M P X	M P X	M P X
LSQA	Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255)	M P X	D M P X	D M P X
NOSYM	No symptom dump (message IEA995I)	P X	P X	P X
NUC	Read/write portion of the control program nucleus (that is, only non-page-protected areas of the DAT-on nucleus), including: • Communication vector table (CVT) • Local system queue area (LSQA) • Prefixed save area (PSA) • System queue area (SQA)	M P X	M P X	D M P X
PCDATA	Program call information for the task	M P X	M P X	M
PSW	Program status word (PSW) when the dump was requested	M P X	D M P X	M P X
Q	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks	M	M	M

Table 25. Summary: dump contents by parameter (continued)

Parameter	Dump Contents	ABEND Dump to SYSUDUMP	ABEND Dump to SYSABEND	ABEND Dump to SYSMDUMP
REGS	Registers at entry to ABEND, that is, when the dump was requested: • Access registers • Floating-point registers • General registers • Vector registers, vector status register, and vector mask register for a task that uses the Vector Facility	M P X	D M P X	M
RGN	Allocated pages in the private area of each address space being dumped, including subpools 0 - 127, 129 - 132, 203 - 205, 213 - 215, 223 - 225, 229, 230, 236, 237, 244, 249, 251 - 255			D P X
SA or SAH	Save area linkage information, program call linkage information, and backward trace of save areas	M P X	D M P X	M
SPLS	Storage allocated in user subpools 0 - 127, 129 - 132, 244, 251, and 252 for the task being dumped Note that SUBPLST in the macro parameter list for a SYSABEND or SYSUDUMP dump overrides SPLS in the dump options list, but only for the dump being requested.	M P X	D M P X	M
SQA	System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248) The control blocks for the failing task in the SQA include: • Address space control block (ASCB) • Job scheduler address space control block (JSAB)	M P X	M P X	D M P X
SUBTASKS	Storage for the task being dumped and program data for all of its subtasks	M P X	M P X	D M
SUM	Summary dump, see "Default contents of summary dumps in ABEND dumps" on page 131	D M P X	D M P X	D M P X
SWA	Scheduler work area (SWA); that is, subpools 236 and 237	M P X	M P X	D M P X
TRT	System trace and generalized trace facility (GTF) trace, as available	M P X	D M P X	
	System trace, as available			D M P X

Default contents of summary dumps in ABEND dumps

If only a summary dump is requested, as in a SYSUDUMP dump that is not customized, the summary information is together, because it forms the entire dump. When a summary dump is combined with other dump options, the summary dump information is scattered throughout the dump. In Table 26 on page 131, an S indicates that a summary dump is available with the dump type.

Table 26. Default contents of summary dumps in ABEND dumps

Summary Dump Contents	ABEND Dump to SYSUDUMP	ABEND Dump to SYSABEND	ABEND Dump to SYSMDUMP
Completion code: The system or user completion code if an ABEND macro requested the dump and, if it exists, the accompanying reason code	S	S	S
Control blocks for the failing task, including: • **ASCB** (address space control block) • **CDE** (contents directory entry) • **LLE** (load list element) • **RB** (request block) • **TCB** (task control block) • **TIOT** (task input/output table) • **XL** (extent list)	S	S	
Control blocks for the recovery termination manager (RTM): • **EED** (extended error descriptor) for RTM • **Registers** from the system diagnostic work area (SDWA) • **RTM2WA** (RTM2 work area) • **SCB** (set task asynchronous exit (STAE) control block)	S	S	S
Dump header, mapped by the AMDDATA macro			S
Dump index	S	S	
Dump title: The job and step being dumped, the time and date of the dump, the dump identifier, and the processor	S	S	S
Load module, if the PSW points to an active load module: • **Name** • **Module Contents** • **Offset** into the load module of the failing instruction • **Module pointed to in the last PRB** (program request block)	 S S S S	 S S S S	 S
PSW (program status word) at entry to ABEND, that is, when the dump was requested. The PSW includes the instruction length code and the interrupt code for the failing instruction.	S	S	S
Registers at entry to ABEND, that is, when the dump was requested	S	S	S
Save areas of register contents	S	S	
Storage: 4 kilobytes before and 4 kilobytes after the addresses in the PSW and the registers. The dump shows, by ascending address, only the storage that the user is authorized to access. Duplicate addresses are removed.	S	S	S
System trace table entries for the dumped address space	S	S	

Table 26. Default contents of summary dumps in ABEND dumps (continued)

Summary Dump Contents	ABEND Dump to SYSUDUMP	ABEND Dump to SYSABEND	ABEND Dump to SYSMDUMP
TCB summary: Information from the task control blocks (TCB) in the address space being dumped	S	S	
Virtual storage map: The subpools in the address space being dumped: • Subpool number • Subpool key • The owning or sharing task control block (TCB) • The beginning address and length of each allocated area • The beginning address and length of each free area	S	S	

Customizing ABEND dump contents

The ddname of the data set for the ABEND dump determines how the contents can be customized. The system determines the contents of a particular ABEND dump from the options list the system maintains for the type of dump. The dump options list can be customized, cumulatively, by all the ways shown in the following tables; thus, for example, a SYSMDUMP ABEND dump written for an ABEND macro can be completely different from the default SYSMDUMP ABEND dump described in this document.

- Table 27 on page 133
- Table 28 on page 135
- Table 29 on page 136

For more information about the topics described in this section, see the following references:

- See *z/OS MVS Initialization and Tuning Reference* for parmlib members.
- See *z/OS MVS System Commands* for the CHNGDUMP operator command.
- See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* and *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for the ABEND, SETRP, SNAP, SNAPX, ESTAE, ESTAEX, and ATTACH or ATTACHX with ESTAI macros.
- See *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* and *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for the SETRP and CALLRTM macros.
- See *z/OS MVS Installation Exits* for the IEAVTABX, IEAVADFM, and IEAVADUS installation exits.

Recommendations for customizing ABEND dumps: How an installation customizes dumps should depend on the usual use of each type of dump. The IBM-supplied dump options for ABEND dumps are designed for the following uses:

- SYSABEND dumps: For diagnosis of complex errors in any program running under the operating system
- SYSMDUMP dumps: For diagnosis of system problems when the dump is requested in a program
- SYSUDUMP dumps: For diagnosis of program problems needing simple problem data

For SYSUDUMP dumps, the IBM-supplied IEADMP00 member specifies the default contents as only a summary dump. An installation should consider using the IEADMP00 member as supplied, because it offers a small dump for simple problems.

Program areas in dumps: To request a meaningful dump for a particular program, code an ABEND macro that points to a macro parameter list. Specify in the list the data areas that are needed to diagnose the abnormally ending program but that are not specified in the parmlib member for the dump. Two examples are:

- If the task that is ending has subtasks and they might cause an error, specify PDATA=SUBTASKS in the macro parameter list to dump the subtasks.

- To see only the subpools used by the program, specify the subpool numbers in a SUBPLST option for a SYSABEND dump. The SPLS option, which is a default for SYSABEND dumps, writes all user subpools. Leaving SPLS in the dump options may make the dump bigger than needed. Note that SUBPLST in the macro parameter list overrides SPLS in the current dump options.

Nucleus areas in dumps: Dump options control the parts of the nucleus that appear in a dump. A diagnostician seldom needs to analyze all of the nucleus. An installation can eliminate nucleus areas from dumps. If the IBM-supplied defaults are used, an SYSMDUMP ABEND dump contains the read/write DAT-on nucleus.

An installation can obtain one copy of the DAT-off nucleus to use in any problem by entering a DUMP operator command.

The ABEND dump options that control dumping of the nucleus areas are:

Option
: **Nucleus aArea**

SDATA=NUC
: Read/write DAT-on nucleus

SDATA=ALLNUC
: All of the DAT-on nucleus: read/write and read-only

Customizing SYSABEND dump contents

Table 27 on page 133 summarizes how to customize the contents of SYSABEND dumps.

Table 27. Customizing SYSABEND dump contents

SYSABEND Customization	Effect	Example

Replacing IEAABD00 parmlib member (by using the IEBUPDTE utility).	**Change occurs**: At system initialization
	
	What changes: IEAABD00 contains the IBM-supplied default dump options. Replacing IEAABD00 changes the dump options for SYSABEND.
		To add program call data and the link pack area to all SYSABEND dumps, while retaining the IBM-supplied options, use IEBUPDTE to change the IEAABD00 member to contain:
		
		`SDATA=(LSQA,CB,ENQ,TRT,ERR,`
		`DM,IO,SUM,PCDATA)`
		`PDATA=(PSW,REGS,SPLS,ALLPA,`
		`SA,LPA)`

Using a macro parameter list.
 	
The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: At dump request
	
	What changes: The macro parameter list options are added to the dump options list, but only for the dump being requested.
	
	Note that SUBPLST in the macro parameter list overrides SPLS in the dump options list, but only for the dump being requested.
		To add program call data and the link pack area to this SYSABEND dump, code in the program:
		
		`ABEND 76,DUMP,`
		
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`SDATA=PCDATA,`
		
		`PDATA=LPA,MF=L`

Table 27. Customizing SYSABEND dump contents (continued)		
SYSABEND Customization	**Effect**	**Example**

Recovery routines invoked by the recovery termination manager:
• FRRs (function recovery routines) for a system component
• ESTAE/ESTAI recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro
• ARRs (associated recovery routines)
 	
These routines issue SETRP macros. To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: Just before dumping
	What changes: The SETRP macro parameter list options are added to the dump options list, but only for the dump being requested.
		To add program call data and the link pack area to this SYSABEND dump, code in the recovery routine:
		`SETRP ,DUMP=YES,`
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`SDATA=PCDATA,`
		`PDATA=LPA,MF=L`

Entering CHNGDUMP operator command with SYSABEND parameter on a console with master authority.	**Change occurs**: Immediately when entered
	What changes:
	• **For ADD:** CHNGDUMP options are added to the IEAABD00 options, previous CHNGDUMP options, and all macro parameter list options. The options remain added until a CHNGDUMP DEL,SYSABEND operator command is entered.
	• **For OVER:** CHNGDUMP options override all other dump options.
	• **For DEL:** All CHNGDUMP options are deleted and the dump options in IEAABD00 are used again.
	When more than one CHNGDUMP operator command with SYSABEND is entered, the effect is cumulative.
		To add program call data and the link pack area to all SYSABEND dumps until changed by CHNGDUMP DEL,SYSABEND, enter:
		`CHNGDUMP SET,ADD,SYSABEND,`
		`SDATA=PCDATA,PDATA=LPA`
		To return to the IEAABD00 options, enter:
		`CHNGDUMP DEL,SYSABEND`

Through IEAVTABX installation exit name list.	**Change occurs**: Just before dumping **What changes**: The routine can add or delete options from the dump options, but only for the current dump.	See *z/OS MVS Installation Exits*.
Through IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run. IEAVADUS is one installation routine.	**Change occurs**: During dumping. The routine runs during control block formatting of a dump with the CB option. **What changes**: The routine can add control blocks to the dump.	See *z/OS MVS Installation Exits*.

Customizing SYSMDUMP dump contents

Table 28 on page 135 contains a summary of how to customize the contents of SYSMDUMP dumps.

Table 28. Customizing SYSMDUMP dump contents

SYSMDUMP Customization	Effect	Example

Replacing IEADMR00 parmlib member (by using the IEBUPDTE utility).	**Change occurs**: At system initialization
	What changes: IEADMR00 contains the IBM-supplied default dump options. Replacing IEADMR00 changes the dump options for SYSMDUMP.
		To add the link pack area to all SYSMDUMP dumps, while retaining all the IBM-supplied defaults, use IEBUPDTE to change the IEADMR00 member to contain:
		`SDATA=(NUC,SQA,LSQA,SWA,TRT`
		`,RGN,`
		`SUM,LPA)`

Using a macro parameter list.
The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: At dump request
	What changes: The macro parameter list options are added to the dump options list, but only for the dump being requested.
		To add the link pack area to this SYSMDUMP dump, code in the program:
		`ABEND 76,DUMP,`
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`PDATA=LPA,MF=L`

Recovery routines invoked by the recovery termination manager:
• FRRs (function recovery routines) for a system component
• ESTAE/ESTAI recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro
• ARRs (associated recovery routines)
These routines issue SETRP macros. To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: Just before dumping
	What changes: The SETRP macro parameter list options are added to the dump options list, but only for the dump being requested.
		To add the link pack area to this SYSMDUMP dump, code in the recovery routine:
		`SETRP ,DUMP=YES,`
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`PDATA=LPA,MF=L`

Entering CHNGDUMP operator command with SYSMDUMP parameter on a console with master authority.	**Change occurs**: Immediately when entered
	What changes:
	• **For ADD:** CHNGDUMP options are added to the IEADMR00 options, previous CHNGDUMP options, and macro parameter list options. The options remain added until a CHNGDUMP DEL,SYSMDUMP operator command is entered.
	• **For OVER:** CHNGDUMP options override all other dump options.
	• **For DEL:** All CHNGDUMP options are deleted and the dump options in IEADMR00 are used again.
	When more than one CHNGDUMP operator command with SYSMDUMP is entered, the effect is cumulative.
		To add the link pack area to all SYSMDUMP dumps until changed by CHNGDUMP DEL,SYSMDUMP, enter:
		`CHNGDUMP`
		`SET,ADD,SYSMDUMP=(LPA)`
		To return to the IEADMR00 options, enter:
		`CHNGDUMP DEL,SYSMDUMP`

Table 28. Customizing SYSMDUMP dump contents (continued)

SYSMDUMP Customization	Effect	Example
Through IEAVTABX installation exit name list.	**Change occurs**: Just before dumping **What changes**: The routine can add or delete options from the dump options, but only for the current dump.	See *z/OS MVS Installation Exits*.

Customizing SYSUDUMP dump contents

Table 29 on page 136 contains a summary of how to customize the contents of SYSUDUMP dumps.

Table 29. Customizing SYSUDUMP dump contents

SYSUDUMP Customization	Effect	Example

Replacing IEADMP00 parmlib member (by using the IEBUPDTE utility).	**Change occurs**: At system initialization
	
	What changes: IEADMP00 contains the IBM-supplied default dump options. Replacing IEADMP00 changes the dump options for SYSUDUMP.
		To add program call data and user subpool storage to all SYSUDUMP dumps, while retaining the summary dump, use IEBUPDTE to change the IEADMP00 member to contain:
		
		`SDATA=(SUM,PCDATA)`
		`PDATA=SPLS`

Using a macro parameter list.
 	
The DUMPOPT or DUMPOPX parameter on the ABEND or CALLRTM macro points to the parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: At dump request
	
	What changes: The macro parameter list options are added to the dump options list, but only for the dump being requested.
	
	Note that SUBPLST in the macro parameter list overrides SPLS in the dump options list, but only for the dump being requested.
		To add program call data and user subpool storage to this SYSUDUMP dump, code in the program:
		
		`ABEND 76,DUMP,`
		
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`SDATA=PCDATA,`
		
		`PDATA=SPLS,MF=L`

Recovery routines invoked by the recovery termination manager:
• FRRs (function recovery routines) for a system component
• ESTAE/ESTAI recovery routines established by an ESTAE or ESTAEX macro or the ESTAI parameter of an ATTACH or ATTACHX macro
• ARRs (associated recovery routines)
 	
Theseroutines issue SETRP macros. To customize the dump contents, the DUMPOPT or DUMPOPX parameter on the SETRP macro points to a parameter list. The list is usually created by a list-form SNAP or SNAPX macro.
	Change occurs: Just before dumping
	
	What changes: The SETRP macro parameter list options are added to the dump options list, but only for the dump being requested.
		To add program call data and user subpool storage to this SYSUDUMP dump, code in the recovery routine:
		
		`SETRP ,DUMP=YES,`
		
		`DUMPOPT=PARMS`
		`PARMS    SNAP`
		`SDATA=PCDATA,`
		
		`PDATA=SPLS,MF=L`

Table 29. Customizing SYSUDUMP dump contents (continued)

SYSUDUMP Customization	Effect	Example

Entering CHNGDUMP operator command with SYSUDUMP parameter on a console with master authority.	**Change occurs**: Immediately when entered
	
	What changes:
	• **For ADD:** CHNGDUMP options are added to the IEADMP00 options, previous CHNGDUMP options, and all macro parameter list options. The options remain added until a CHNGDUMP DEL,SYSUDUMP operator command is entered.
	• **For OVER:** CHNGDUMP options override all other dump options.
	• **For DEL:** All CHNGDUMP options are deleted and the dump options in IEADMP00 are used again.
	
	When more than one CHNGDUMP operator command with SYSUDUMP is entered, the effect is cumulative.
		To add program call data and user subpool storage to all SYSUDUMP dumps until changed by CHNGDUMP DEL,SYSUDUMP, enter:
		
		`CHNGDUMP SET,ADD,SYSUDUMP,`
		` SDATA=PCDATA,PDATA=SPLS`
		
		To return to the IEADMP00 options, enter:
		
		`CHNGDUMP DEL,SYSUDUMP`

Through IEAVTABX installation exit name list.	**Change occurs**: Just before dumping **What changes**: The routine can add or delete options from the dump options, but only for the current dump.	See *z/OS MVS Installation Exits*.
Through IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run and IEAVADUS is one installation routine.	**Change occurs**: During dumping. The routine runs during control block formatting of a dump with the CB option. **What changes**: The routine can add control blocks to the dump.	See *z/OS MVS Installation Exits*.

Analyzing an ABEND dump

Note: A SYSMDUMP ABEND dump is always a synchronous SVC dump. To analyze a SYSMDUMP, see "Analyzing an SVC dump" on page 36.

ABEND dumps written to SYSABEND and SYSUDUMP data sets are useful for analyzing problems in a program running under the operating system. This program can be called any of the following:

- Installation-provided program
- An application program
- A non-authorized program
- A problem program
- A program in the private area

ABEND dumps are written for problems detected in two ways:

- **Software-detected problem**, such as:
  - A nonzero return code from a called module
  - A program check, abend code X'0Cx', that a recovery routine changes to another abend code
  - An erroneous control block queue
  - Not valid input to a system service

- **Hardware-detected problem**, which is a program check, abend code X'0Cx', that a recovery routine does not change to another abend code

Analysis Procedure

To analyze a SYSABEND or SYSUDUMP, take the following steps:

1. **Collect and analyze logrec error records.**

   Check all logrec error records related to the abended task. Determine if any records show an earlier system problem; if so, continue diagnosis with that problem. Because of recovery and percolation, a SYSABEND or SYSUDUMP dump can be the end result of an earlier system problem.

2. **Collect and analyze messages about the problem.** Use time stamps to select messages related to the problem:

   - The job log
   - The system log (SYSLOG) or operations log (OPERLOG)

   Check the messages for earlier dumps written while the abended task was running. Determine if these earlier dumps indicate an earlier system problem; if so, continue diagnosis with that problem.

3. **Analyze the dump**, as described in the following steps.

   Note: After the problem and before the dump, recovery tried to reconstruct erroneous control block chains before ending the task. If the problem proves to be in a system component, a SYSABEND or SYSUDUMP dump cannot be used to isolate it because of the recovery actions; these dumps are useful only for problems in the private area.

4. **Obtain the abend code, reason code, job name, step name, and program status word (PSW) from the dump title** at the beginning of the dump.

   If the completion code is USER=dddd, an application program issued an ABEND macro to request the dump and to specify the completion code.

   If the completion code is SYSTEM=hhh, a system component ended the application program and a recovery routine in the program requested the dump. The application program probably caused the abend.

   Reference See *z/OS MVS System Codes* for an explanation of the abend code.

5. **Analyze the RTM2WA**, as follows:

   - In the TCB summary, find the task control block (TCB) for the failing task. This TCB has the abend code as its completion code in the CMP field. In the TCB summary, obtain the address of the recovery termination manager 2 (RTM2) work area (RTM2WA) for the TCB.
   - In the RTM2WA summary, obtain the registers at the time of the error and the name and address of the abending program.
   - If the RTM2WA summary does not give the abending program name and address, probably an SVC instruction abnormally ended.
   - If the RTM2WA summary gives a previous RTM2WA for recursion, the abend for this dump occurred while an ESTAE or other recovery routine was processing another, original abend. In recursive abends, more than one RTM2WA may be created. Use the previous RTM2WA to diagnose the original problem.

   For information about the RTM2WA, SDWA, and TCB data areas, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

6. **Analyze the dump for the program name**. Obtain the program name from the RTM2WA summary. If the name field is zero, do the following:

   - Find the control blocks for the task being dumped.
   - The last request blocks are SVRBs. In the WLIC field in an SVRB, find the following SVC interruption codes:

  - X'33' for a SNAP SVC interruption
  - X'0C' for a SYNCH SVC interruption
- The program request block (PRB) for the abending program immediately precedes these SVRBs.
- When the dump contains more than one CDE, determine the first and last address for each CDE. The entry point address is the first address. Add the length to the entry point address to obtain the last address. Compare these addresses to the address in the right half of the PSW in the dump header; the PSW address falls between the first and last addresses of the correct CDE.

  Note that the leftmost digit in the PSW address denotes addressing mode and is not part of the address.
- In that CDE, the NAME field gives the program name.

7. **Locate the failing program module** in the hexadecimal dump.
8. **Find the instruction that caused the abend.**

   The PSW in the dump header is from the time of the error. Obtain the address in the right half of the PSW. The leftmost digit denotes addressing mode and is not part of the address.

   For most problems, subtract the instruction length in the ILC field of the dump header from the PSW address to obtain the address of the failing instruction. Do not subtract the instruction length in the following cases; the failing instruction is at the PSW address.

   - Page translation exception.
   - Segment translation exception.
   - Vector operation interruption.
   - Other interruptions for which the processing of the instruction identified by the old PSW is nullified. See *z/Architecture Principles of Operation* for the interruption action.
   - If access registers were being used at the time of the error, so that the access list entry token (ALET) may be incorrect.

   Subtract the failing instruction address from the failing module address. Use this offset to find the matching instruction in the abending program's assembler listing.
9. **For an abend from an SVC or system I/O routine, find the last program instruction.**

   If the abend occurred in a system component running on behalf of the dumped program, find the last instruction that ran in the program, as follows:

   - For an abend from an SVC routine, look in the last PRB in the control blocks for the task being dumped. The right half of the PSW in the RTPSW1 field contains the address of the instruction following the SVC instruction.
   - For an abend from a system I/O routine, look in the save area trace. This trace gives the address of the I/O routine branched to. The return address in that save area is the last instruction that ran in the failing program.
10. **For an abend from an SVC or system I/O routine, determine the cause of the abend**, using the following:

    - For an abend from an SVC, look in the system trace table for SVC entries matching the SVRBs in the control blocks for the task being dumped.
    - For an abend from an I/O routine, look in the system trace table for I/O entries issued from addresses in the failing program. The addresses are in the PSW ADDRESS column.

    If SVC entries match the dumped blocks or the I/O entries were issued from the failing program, the system trace table was not overlaid between the problem and the dump.

    In this case, start with the most recent entries at the end of the trace. Back up to the last SVC entry with the TCB address of the abending task. Go toward the end of the trace, looking for indications of the problem. See Chapter 8, “System trace,” on page 153 for more information.
11. **For a program interrupt, determine the cause of the abend**, using the registers at the time of the error in the RTM2WA and in the SVRB following the PRB for the abending program.

Also, look at the formatted save area trace for input to the failing module.

12. **For an abend in a cross memory environment**, do the following to analyze the dump.

    Many services are requested by use of the Program Call (PC) instruction, rather than by SVCs or SRBs. When an abend is issued by the PC routine, the OPSW field in the RB contains the instruction address of the PC routine that issued the abend. The SVRB contains the registers of the PC routine.

    Do the following to look for the registers and PSW at the time the PC instruction was issued:

    - For a stacking PC, find the registers in the linkage stack. Any entries on the linkage stack are before the RBs in the dump.
    - For a basic PC, find the registers in the PCLINK stack. Any entries on the PCLINK stack are after the RBs in the dump.

    For a stacking PC, find the linkage stack entry that corresponds to the RB/XSB for the program. The LSED field of the linkage stack entry and the XSBLSCP field in the corresponding XSB have the same value. From the linkage stack entry, obtain the registers and the PSW at the time the stacking PC was issued. The address in the PSW points to the instruction following the PC instruction in the abending program.

    For a basic PC, determine the caller from the PCLINK stack. To locate the PCLINK stack element (STKE):

    - The STKEs appear in the dump following all of the RBs. If the dump contains more than one STKE, the pointer to the STKE for the PC involved in the problem is in the XSBSTKE field of the XSB associated with the RB for the abending program.
    - The RBXSB field in the RB points to the XSB.
    - The XSBSEL field in the XSB points to the current STKE.

    In the STKE, the STKERET field contains the return address of the caller of the PCLINK service.

    For information about the STKE and XSB data areas, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Chapter 6. SNAP dump

This topic (SNAP Dump) contains programming interface information.

A SNAP dump shows virtual storage areas that a program, while running, requests the system to dump. A SNAP dump, therefore, is written while a program runs, rather than during abnormal end. The program can ask for a dump of as little as a one byte field to as much as all of the storage assigned to the current job step. The program can also ask for some system data in the dump.

A SNAP dump is especially useful when testing a program. A program can dump one or more fields repeatedly to let the programmer check intermediate steps in calculations. For example, if a program being developed produces incorrect results, requests for SNAP dumps can be added to the program to dump individual variables. The first time that incorrect storage is encountered should narrow down the section of code causing the error.

Obtaining SNAP dumps

Provide a data set to receive the dump, then arrange to print the dump. The SNAP or SNAPX macros in a program can place their dumps in the same or different data sets; the DCB parameter in each SNAP or SNAPX macro indicates the data set.

When setting up a dump data set, determine if the data set will contain privileged data. If so, protect it with passwords or other security measures to limit access to it.

You can use extended format sequential data sets as dump data sets for SNAP dumps. Extended format sequential data sets have the following features:

- Have a greater capacity than sequential data sets
- Support striping
- Support compression

Using DSNTYPE=LARGE: In z/OS V1R7 and later releases, sequential data sets that use DSNTYPE=LARGE are allowable for ABEND dumps when the systems that are involved in processing a DSNTYPE=LARGE data set are migrated to V1R7 prior to their use. If analysis using an earlier release is required, use z/OS V1R7 to transcribe the dump into a data set supported by the earlier release.

Placing dump data sets in cylinder-managed space: In z/OS V1R11 and later releases, extended format sequential data sets can be placed in either track-managed space or cylinder-managed space. SNAP dump fully supports placement of dump data sets in cylinder-managed space.

Obtain a SNAP dump by taking the following steps:

1. Code a DD statement in the JCL for the job step that runs the problem program to be dumped with a ddname other than SYSUDUMP, SYSABEND, SYSMDUMP, or another restricted ddname. The statement can specify that the output of the SNAP dump should be written to one of the following:

   - Direct access.
   - Printer. Note that a printer is not recommended, except when running under z/VM®, because the printer cannot be used for anything else while the job step is running, whether a dump is written or not. Under z/VM you can use a virtual printer. This allows you to see or print the partial output on a real printer while the program is running while only using a small amount of system resources.
   - SYSOUT. SNAP dumps usually use SYSOUT.
   - Tape.

Example: SYSOUT DD Statement for SNAP Dump: The following example places a SNAP dump in sysout output class A. In the example, output class A is a print class. When the system prints the output class, the system will print any dumps written to the class.

```
//SNAP1     DD  SYSOUT=A
```

Example: Tape DD Statement for SNAP Dump: The following example places a SNAP dump on a tape. In the example, TAPE is a group name established by the installation.

```
//SNAP2     DD  DSN=DUMPDS,UNIT=TAPE,DISP=(,KEEP,KEEP)
```

The system keeps the data set when the job step ends, whether normally or abnormally. In either case, SNAP dumps are taken throughout processing, regardless of the way the step ends.

Example: Direct Access DD Statement for SNAP Dump: The following example places a SNAP dump on direct access, for example, the 3350 direct access storage.

```
//SNAP3     DD  DSN=SNAPSHOT,UNIT=3350,DISP=(,KEEP,KEEP),
//               VOLUME=SER=12345,SPACE=(1680,(160,80))
```

The system writes the dump in a sequential data set using the basic sequential access method (BSAM). The dump data set can be on any device supported by BSAM.

2. In the problem program:

   a. Specify a data control block (DCB) for the data set to receive the dump. For a standard dump, which has 120 characters per line, the DCB must specify:

      - BLKSIZE=882 or 1632
      - DSORG=PS
      - LRECL=125
      - MACRF=(W)
      - RECFM=VBA

      For a high-density dump, which has 204 characters per line and will be printed on an APA 3800 printer, the DCB must specify:

      - BLKSIZE=1470 or 2724
      - DSORG=PS
      - LRECL=209
      - MACRF=(W)
      - RECFM=VBA

   b. Code an OPEN macro to open the DCB.

      Before you issue the SNAP or SNAPX macro, you must open the DCB that you designate on the DCB parameter, and ensure that the DCB is not closed until the macro returns control. To open the DCB, issue the DCB macro with the following parameters, and issue an OPEN macro for the data set:

      ```
      DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=nnn,LRECL=xxx,
      and DDNAME=any name but SYSABEND, SYSMDUMP or SYSUDUMP
      ```

      If the system loader processes the program, the program must close the DCB after the last SNAP or SNAPX macro is issued.

   c. Code a SNAP or SNAPX assembler macro to request the dump.

      Example: Coding the SNAP Macro: In the following example, the SNAP macro requests a dump of a storage area, with the DCB address in register 3, a dump identifier of 245, the storage area's starting address in register 4, and the ending address in register 5:

      ```
           SNAP  DCB=(3),ID=245,STORAGE=((4),(5))
      ```

Repeat this macro in the program as many times as wanted, changing the dump identifier for a unique dump. The system writes all the dumps that specify the same DCB to the same data set.

Example: Two SNAP Dump Requests in a Program: The following example shows a problem program that requests two SNAP dumps. Both SNAP macros in the example specify the same data control block (DCB) to place both dumps in the same data set. Each dump has a different identifier: PIC3 for the first dump, PIC4 for the second. Both dumps show the same areas: the control blocks. Thus, the programmer can see these areas at two points in the program's processing.

```
SNAPDCB   DCB    BLKSIZE=882,DSORG=PS,LRECL=125,MACRF=(W),RECFM=VBA
           .
           .
           .
          OPEN   SNAPDCB,OUTPUT
          LA     3,SNAPDCB
          SNAP   DCB=(3),ID=PIC3,SDATA=CB
           .
           .
           .
          SNAP   DCB=(3),ID=PIC4,SDATA=CB
          CLOSE  SNAPDCB
```

d. Close the DCB with a CLOSE assembler macro.

For more information, see the following references:

- See *z/OS DFSMS Macro Instructions for Data Sets* for coding the DCB, OPEN, and CLOSE macros.
- See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* and *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for required parameters on the DCB macro and for coding the SNAP or SNAPX macro.

3. Print or view the data set. The output of the SNAP or SNAPX macro is a standard EBCDIC data set with ANSII characters in column one. This data set can be edited.

The dumps are formatted as they are created. Printing depends on the location of the dump when it is created:

Location
: **Printing**

SYSOUT
: The system prints the dump(s) when printing the output class.

On a tape or direct access data set
: Print the dump(s) in a separate job or job step.

Printer
: The system prints the dump(s) as they are created.

To view SNAP dumps at a terminal, browse the data set containing the dump.

Example: Printing a SNAP Dump The following JCL prints a SNAP dump. Because the system formats the dump when creating it, the IEBPTPCH utility program can print the dump. The dump is in the SNAPSHOT data set.

```
              .
              .
              .
//PRINT     EXEC  PGM=IEBPTPCH
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSN=SNAPSHOT,UNIT=3350,DISP=(OLD,DELETE),
//           VOLUME=SER=12345
//SYSUT2    DD    SYSOUT=A
//SYSIN     DD    *
   PRINT    TYPORG=PS
/*
```

Customizing SNAP dump contents

You can customize the contents of SNAP dumps in one of the following ways:

- Through installation exits.
- Through parameters on the SDUMP or SDUMPX macro.

Customizing through installation exits

An installation can customize the contents of SNAP dumps through the IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run and IEAVADUS is one installation routine. The installation exit routine runs during control block formatting of a dump when the CB option is specified on the SNAP or SNAPX macro. The routine can format control blocks and send them to the data set for the dump. See *z/OS MVS Installation Exits* for more information.

Customizing through the SNAP or SNAPX macro

The parameters on the SNAP or SNAPX macro determine the dump contents. The macro can specify any or all of the areas listed in Table 30 on page 144.

Note that the parameters cannot request that a Hiperspace be included in the dump. To include Hiperspace data in a SNAP dump, read the data from the Hiperspace into address space storage that is being dumped. See *z/OS MVS Programming: Extended Addressability Guide* for more information about manipulating data in Hiperspace storage.

Table 30. Customizing dumps using through the SNAP or SNAPX macro

Parameter	Dump Contents
ALLPA	All link pack areas, as follows: • Job pack area (JPA) • Link pack area (LPA) active for the task being dumped • Related supervisor call (SVC) modules
ALLVNUC	The entire virtual control program nucleus
CB	Control blocks for the task being dumped
DM	Data management control blocks for the task being dumped: • Data control block (DCB) • Data extent block (DEB) • Input/output block (IOB)
ERR	Recovery termination manager (RTM) control blocks for the task being dumped: • Extended error descriptor (EED) for RTM • Registers from the system diagnostic work area (SDWA) • RTM2 work area (RTM2WA) • Set task asynchronous exit (STAE) control block (SCB)
IO	Input/output supervisor (IOS) control blocks for the task being dumped: • Execute channel program debug area (EXCPD) • Unit control block (UCB)
JPA	Job pack area (JPA): module names and contents
LPA	Link pack area (LPA) active for the task being dumped: module names and contents
LSQA	Local system queue area (LSQA) allocated for the address space (that is, subpools 203 - 205, 213 - 215, 223 - 225, 229, 230, 233 - 235, 249, 253 - 255)

Table 30. Customizing dumps using through the SNAP or SNAPX macro (continued)	
Parameter	**Dump Contents**
NUC	Read/write portion of the control program nucleus (that is, only non-page-protected areas of the DAT-on nucleus), including: • Communication vector table (CVT) • Local system queue area (LSQA) • Prefixed save area (PSA) • System queue area (SQA)
PCDATA	Program call information for the task
PSW	Program status word (PSW) when the dump is requested
Q	Global resource serialization control blocks for the task being dumped: • Global queue control blocks • Local queue control blocks
REGS	Registers when the dump is requested: • Access registers • Floating-point registers • General registers • Vector registers, vector status register, and vector mask register for a task that uses the Vector Facility
SA or SAH	Save area linkage information, program call linkage information, and backward trace of save areas
SPLS	Storage allocated in user subpools 0 - 127, 129 -132, 244, 251, and 252 for the task being dumped
SQA	System queue area (SQA) allocated (that is, subpools 226, 239, 245, 247, 248)
SUBTASKS	Storage for the task being dumped and program data for all of its subtasks
SWA	Scheduler work area (SWA) (that is, subpools 236 and 237)
TRT	System trace and generalized trace facility (GTF) trace, as available
—	One or more data spaces identified on the SNAPX macro
—	One or more storage areas, identified by beginning and ending addresses on the SNAP or SNAPX macro
—	One or more subpools, identified by subpool number on the SNAP or SNAPX macro

Chapter 7. The dump grab bag

A dump contains information about an error that can help you identify a problem type. Using interactive problem control system (IPCS), the information about the error is formatted to provide a quick and effective method of retrieval.

The hints that follow apply to processing all kinds of dumps: SVC dumps, stand-alone dumps, and SYSMDUMP dumps.

This section covers the following topics:

Problem data for storage overlays

Always be aware of the possibility of a storage overlay when analyzing a dump. System problems in MVS are often caused by storage overlays that destroy data, control blocks, or executable code. The results of such an overlay vary. For example:

- The system detects an error and issues an abend code, yet the error can be isolated to an address space. Isolating the error is important in discovering whether the overlay is in global or local storage.
- Referencing the data or instructions can cause an immediate error such as a specification exception (abend X'0C4') or operation code exception (abend X'0C1').
- The bad data is used to reference a second location, which then causes another error.

When you recognize that the contents of a storage location are not valid and subsequently recognize the bit pattern as a certain control block or piece of data, you generally can identify the erroneous process/component and start a detailed analysis.

Analyzing the damaged area

Once you determine that storage is bad or overlaid, try to identify the culprit. First, determine the extent of the bad data. Look for EBCDIC data or module addresses in storage to identify the owner. Any type of pattern in storage can indicate an error and identify the program that is using the damaged storage. Look at the data on both sides of the obviously bad areas. See if the length of the bad area is familiar; that is, can you relate the length to a known control block length, data size, MVC length? If so, check various offsets to determine their contents and, if you recognize some, try to determine the exact control block. In Figure 69 on page 148, for example, storage from CSA shows a pattern of allocated blocks.


```
00CFD000   00000000   00000000   E5C7E3E3   080000F1   | ........VGTT...
00CFD010   00000020   00000000   0000E3D8   00000000   | ..........TQ...
00CFD020   00BE3D30   00000000   E5C7E3E3   080000F1   | ........VGTT...
00CFD030   00000020   00CFD008   0000E260   00000000   | ......}...S-...
00CFD040   00CA6A60   00000000   17000080   E5E2C90F   | ...-........VSI
00CFD050   00CFE018   00CFD0B8   C8E2D4D3   4BD4C3C4   | ..\...}.HSML.MC
00CFD060   E24BC4C1   E3C14040   40404040   40404040   | S.DATA
00CFD070 TO 00CFD07F (X'00000010' bytes)--All bytes contain X'40', C' '
00CFD080   40404040   000E0042   00000000   00000000   |    ...........
00CFD090   00000000   00000000   00000000   17CA0000   | ...............
00CFD0A0   1FFE0000   C2C3E3F3   D3C90001   00000000   | ....BCT3LI.....
00CFD0B0   00000000   40080000   E2E8E2F1   4BE4C3C1   | .... ...SYS1.UC
00CFD0C0   E34BC5D5   E5F2F600   00000168   00CFD0C8   | T.ENV26.......}
00CFD0D0   F1000000   00CFD230   00CFD22C   13C9C4C1   | 1.....K...K..ID
00CFD0E0   C8C5C240   00100150   00CFD244   00000160   | HEB ...&;.K....
```

Figure 69. Example: Recognizing a pattern

Even if you do not recognize the pattern, take one more step. Can you determine the offset from some base that would have to be used in order to create the bit pattern? If so, the fact that there is a certain bit pattern at a certain offset can be helpful.

For example, a BALR register value (X'40D21C58') at an offset X'C' can indicate that a program is using this storage for a register save area (perhaps caused by a bad register 13). Another field in the same overlaid area might trigger recognition.

Repetition of a pattern can indicate a bad process. If you can recognize the bad data you might be able to relate that data to the component or module that is causing the error. This provides a starting point for further analysis.

Common bad addresses

The following are commonly known as bad addresses. If you recognize these in the code you are diagnosing, focus your problem source identification on these areas:

- X'000C0000', X'040C0000', or X'070C0000', and one of these addresses plus some offset. These are generally the result of some code using 0 as the base register for a control block and subsequently loading a pointer from 0 plus an offset, thereby picking up the first half of a PSW in the PSA.

  Look for storage overlays in code pointed to by an old PSW. These overlays result when 0 plus an offset cause the second half of a PSW to be used as a pointer.
- X'C00', X'D00', X'D20', X'D28', X'D40', and other pointers to fields in the normal functional recovery routine (FRR) stack. Routines often lose the contents of a register during a SETFRR macro expansion and incorrectly use the address of the 24-byte work area returned from the expansion.
- Register save areas. Storage might be overlaid by code doing a store multiple (STM) instruction with a bad register save area address. In this case, the registers saved are often useful in determining the component or module at fault.

Problem data from the linkage stack

The linkage stack is used to identify a program that requested a system service, if the service was entered by a branch instruction.

For example, to see the linkage stack entry that is associated with address space identifier (ASID) X'1A', use the IPCS subcommand:

```
SUMMARY FORMAT ASID(X'1a')
```

The resulting dump for the linkage stack associated with the address space (see Figure 70 on page 149) shows one entry, as follows:

BAKR STATE ENTRY
: A Branch and Stack (BAKR) instruction caused this entry.

SASN..1A and PASN..1A
At the time of the BAKR, the program was not in cross memory mode. When the branching program is not in cross memory mode, secondary address space number (SASN) and primary address space (PASN) are identical. If the program had been in cross memory mode, SASN and PASN would not have been identical.

PSW..070C0000 80FD7618
The return address of the branch caused by the BAKR is FD7618. This address is in the right half of the program status word (PSW).

```
LINKAGE STACK ENTRY  01  LSED: 7F7490B0
  LSE: 7F749010
     GENERAL PURPOSE REGISTER VALUES
     00-03.... 7FFEB410  04504DF4  04532000  04541FFF
     04-07.... 04504CE4  81150380  00000028  04504B50
     08-11.... 04503A75  04502A76  04501A77  04504630
     12-15.... 84500A78  00000000  80FD7618  8450FAF8
     ACCESS REGISTER VALUES
     00-03.... 00000000  00000000  00000000  00000000
     04-07.... 00000000  00000000  00000000  00000000
     08-11.... 00000000  00000000  00000000  00000000
     12-15.... 00000000  00000000  00000000  00000000
1S/A ON AQFT PER SYSTEM HUNG 22:30 08/30/88       24  09:42:48 10/14/88

PKM..8000 SASN..001A EAX..0000 PASN..001A PSW..070C0000  80FD7618
    TARG... 8450FB12  MSTA... 0451E300  00000000
    TYPE... 84
     BAKR STATE ENTRY
    RFS... 0F38      NES... 0000
```

Figure 70. Example: Viewing a linkage stack entry

Many system services are called through branches. For branch entry services, use register 14 to identify the calling program. Look for the problem in the calling program. See *z/OS MVS Programming: Extended Addressability Guide* for more information about the linkage stack.

Problem data for modules

For a module, the system saves and restores status from different locations, depending on the processing mode of the module when it lost control. Use the IPCS STATUS CPU subcommand to find out the mode of the module that had been currently running for each processor. Use the saved status as problem data for diagnosis.

Processing modes

The processing modes follow. Code always runs in one or more of these modes. For example, code running in task or service request block (SRB) mode can also be either locally locked or physically disabled.

- **Task mode** is the most common processing mode. All programs given control by ATTACH, ATTACHX, LINK, LINKX, XCTL, and XCTLX macros run in task mode.
- **SRB mode** is code that runs from one of the service request block (SRB) queues.
- **Physically disabled mode** is reserved for high-priority system code that manipulates critical system queues and data areas. This mode is usually combined with supervisor state and key 0 in the PSW. The combination ensures that the routine can complete its function before losing control. The mode is restricted to just a few modules in the system, for example, interrupt handlers, the dispatcher, and programs that are holding a global spin lock.
- **Locked mode** is for code that runs in the system while holding a lock.
- **Cross memory mode**. Cross memory mode is defined by:
  - **Primary address space**: Address space identifier (ASID) in control register 3
  - **Secondary address space**: ASID in control register 4

  - **Home address space**: Address of the address space control block (ASCB) in the PSAAOLD field
  - **PSW S-bit** (bit 16 of the PSW): Indicator of current addressability:
    - S-bit=0 - To the primary address space
    - S-bit=1 - To the secondary address space

    When primary addressability and secondary addressability are to the home address space and the S-bit=0, the work is not in cross memory mode.
- **Access register (AR) mode**, where a program can use the full set of assembler instructions (except MVCP and MVCS) to manipulate data in another address space or in a data space. Unlike cross memory, access registers allow full access to data in many address spaces or data spaces.

Problem data from recovery work areas

You can use the recovery work area (RWA) to find the failing module. In most cases, you would use the TCB and RB structure to find the failing module instead of the RWA. Use the RWA in the following situations:

- When an SVC dump is requested in a SLIP trap. In this dump, the current status at the time of the problem is in the recovery save areas or in the SDUMP SQA 4K buffer. See "Reading the SDUMPX 4K SQA buffer" on page 47 for more information.
- When the problem is in the recovery process itself.
- When a stand-alone dump is written because of a suspected loop.

The recovery work areas are:

- Logrec records
- Logrec buffer in the system: obtained by a VERBEXIT LOGDATA subcommand
- System diagnostic work area (SDWA), including the variable recording area (VRA) formatted in logrec records and in the logrec buffer.
- Functional recovery routine (FRR) stacks: described in the next topic.
- Recovery termination manager (RTM) data areas, including the RTM2 work area (RTM2WA): formatted by a SUMMARY FORMAT subcommand or obtained in a formatted ABEND or SNAP dump by the ERR option.

The RTM2WA and SDWA blocks contain registers, PSW, and other time of problem information. Use these blocks in diagnosis when they are associated with a task control block (TCB).

See Chapter 15, "Recording logrec error records," on page 517 for more information. For information about the control blocks, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For details about the IPCS commands, see *z/OS MVS IPCS Commands*.

Problem data for ACR

When alternate CPU recovery (ACR) is active at the time of the dump, the search argument in IPCS STATUS WORKSHEET output contains the symptom:

```
FLDS/CSDACR
```

Pre-Processing phase data

If ACR is active, problem data for the pre-processing phase are:

- The CSDCPUAL field of the common system data (CSD) indicates which processor failed and which is still running
- A system trace table entry with ACR in the Ident column indicates that ACR began and identifies the failing processor

- Use the CSD online mask to determine which CPU's LCCA to examine. Use the IPCS subcommand CBFORMAT to examine the failing CPU's LCCA.
- The WSACACR in the CPU work save area vector table (WSAVTC) for both processors' logical configuration communication areas (LCCA) points to a copy of the PSAs and FRR stacks for both processors.
- The LCCADCPU in both processors points to the LCCA of the failing processor and the LCCARCPU points to the LCCA of the running processor

Note that a dump shows the PSA of the failed processor when the running processor initiated ACR. The normal FRR stack, pointers to other FRR stacks, locks, PSA super bits, and other data reflect the processor at the time of the failure.

Post-Processing phase data

ACR issues message IEA858E when it completes and resets the CSDACR flag to X'00'.

Data obtained by IPCS

Use the following IPCS subcommand to see all the LCCAs and the CSD:

```
STATUS CPU DATA WORKSHEET
```

For more information about control blocks, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For information about the IPCS commands, see *z/OS MVS IPCS Commands*.

Problem data for machine checks

The hardware uses a machine check interruption to tell the control program that it has detected a hardware malfunction. Machine checks vary considerably in their impact on software processing:

- **Soft errors**: Some machine checks notify software that the processor detected and corrected a hardware problem that required no software recovery action.
- **Hard errors**: Other hardware problems detected by a processor require software-initiated action for damage repair. Hard errors also require software recovery to verify the integrity of the process that experienced the failure.

The machine check interrupt code (MCIC) in the PSA FLCMCIC field describes the error causing the interrupt. An MCIC can have more than one bit on to indicate more than one failing condition.

For a machine check, the system writes a logrec error record. The error record contains the MCIC, except when:

- The LRBMTCKS bit in field LRBMTERM of the logrec buffer (LRB) is ON to indicate that the machine check old PSW and the MCIC are both zero.
- The LRBMTINV bit in field LRBMTERM is ON to indicate that the machine check old PSW is nonzero but the MCIC is zero.

Hard errors cause FRR and ESTAE processing.

See *z/Architecture Principles of Operation* for a complete description of the MCIC.

Chapter 8. System trace

System trace provides an ongoing record of hardware events and software events that occurs during system initialization and operation. The system activates system tracing at initialization, which runs continuously, unless your installation has changed the IBM-supplied system tracing. After system initialization, you can use the TRACE operator command on a console with master authority to customize system tracing.

System trace writes trace data in system trace tables in the trace address space. System trace maintains a trace table for each processor.

Because system trace usually runs all the time, it is useful for problem determination. While system trace and the generalized trace facility (GTF) lists many of the same system events, system trace also lists events occurring during system initialization, before GTF tracing can be started. System trace also traces branches and cross-memory instructions, which GTF cannot do.

The following topics explain system trace in detail:

- "Customizing system tracing" on page 153
- "Receiving system trace data in a dump" on page 154
- "Formatting system trace data in a dump" on page 155
- "Reading system trace output" on page 155.

Customizing system tracing

The system starts system tracing during system initialization and the trace runs continually. There are, however, a few things you can do to alter system tracing:

- "Increasing the size of the system trace table" on page 153.
- "Tracing branch instructions" on page 154.

Increasing the size of the system trace table

System trace tables reside in fixed storage on each processor. The default trace table size is 1 MB per processor, but you can change it using the TRACE ST command. You might, however, want to increase the size of the system trace table from the default 1 MB when:

- You find that the system trace does not contain tracing from a long enough time period.
- You want to trace branch instructions (using the BR=ON option on the TRACE ST command when you start tracing).

Do the following to increase the size of the trace table:

- Enter the TRACE ST command to change the size of the system trace table. For example, to increase the size of the trace table from the default 1 megabyte per processor to 2 megabytes per processor:

```
TRACE ST,2M
```

- Enter the TRACE ST command with the BUFSIZE (or BUFSIZ) parameter to change the total size of the system trace table allocated. For example, in a system that has 10 processors, to restart system tracing and increase the size of the trace table from the default of 10 megabytes (1 megabytes per processor) to 20 megabytes (2 megabytes per processor):

```
TRACE ST,BUFSIZ=20M
```

Remember: Choose a reasonable value for the system trace storage after considering the available central storage and the actual storage required for system trace. Increasing the system trace storage to a large value amount may cause shortage of pageable storage in the system.

Tracing branch instructions

System tracing allows you the option of tracing branch instructions, such as BALR, BASR, BASSM and BAKR, along with other system events. The mode tracing option is separate from branch tracing.

Attention: Branch tracing ON can affect your system performance and use very large amounts of storage. Do not use branch tracing as the default for system tracing on your system. Only use it for short periods of time to solve a specific problem. The default system tracing does not include branch instructions.

IBM provides two health checks, SYSTRACE_BRANCH and SYSTRACE_MODE, to help identify when branch tracing may be affecting your system performance. For example, if branch tracing has been set on for a long time, you will receive message IEAH801E. For more information, see System trace checks (IBMSYSTRACE) in *IBM Health Checker for z/OS User's Guide*. For information about related messages, see *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

When you want to trace branch instructions such as BALR, BASR, BASSM and BAKR, do the following:

- Turn on system tracing with branch tracing using the TRACE command from a console with master authority:

  ```
  TRACE ST,BR=ON
  ```

  Because tracing branch instructions tends to significantly increase the number of trace entries being generated, you can increase the size of the trace tables when you turn tracing on.

  – To increase the size of the trace tables for each processor from the default 1 MB to 2 MB, issue:

    ```
    TRACE ST,2M,BR=ON
    ```

  – To increase the size of total storage for trace buffers (that is, the sum of the storage set aside for trace table entries on all the installed processors) to 2 megabytes:

    ```
    TRACE ST,BUFSIZ=2M,BR=ON
    ```

For more information, see the following documentation:

- *z/Architecture Principles of Operation* describes the branch instruction trace entries and the mode trace entries that MVS combines with them (and are generated by the hardware). MVS enables or disables the production of these entries by manipulating control register bits. The trace table entries that are not 'branch (or mode)' entries that are generated by MVS software through the TRACE or TRACG instructions. See the Tracing topic for information.
- For a description of the TTE from mapping macro IHATTE, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Receiving system trace data in a dump

System trace writes trace data in system trace tables in the trace address space. System trace maintains a trace table for each processor. Obtain the trace data in a dump that included option SDATA=TRT. Table 31 on page 154 shows the dumps that have TRT in their default options and how to request trace data for dumps that do not include the data by default.

System trace data that are requested by RTM and ABDUMPs will only receive the most current 64K of data for each CPU when the number of concurrent snapshots could affect system availability. This is referred to as a mini-trace table snapshot. For more information about working with system trace tables, see the SYSTRACE section in *z/OS MVS IPCS Commands*.

Table 31. Dumps that have TRT in their default options

Dump	How to obtain trace data
ABEND dump to SYSABEND	Default

Table 31. Dumps that have TRT in their default options (continued)

Dump	How to obtain trace data
ABEND dump to SYSMDUMP	Default
ABEND dump to SYSUDUMP	Default
SNAP dump	Request SDATA=TRT
Stand-alone dump	Default
SVC dump for SDUMP or SDUMPX macro	Default
SVC dump for DUMP operator command	Default
SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP	Default
Any dump customized to exclude trace data	Request SDATA=TRT

Formatting system trace data in a dump

- For formatted dumps, system trace formats the system trace data and the system prints it directly.
- For unformatted dumps, use the IPCS SYSTRACE subcommand to format and print or view the trace data in the dump.

Reading system trace output

The following topics describe system trace table entries (TTE) as they appear in a dump formatted with the IPCS SYSTRACE subcommand.

- "Example of a system trace in a dump" on page 155
- "Summary of system trace entry identifiers" on page 156 shows a table of the system trace identifiers for each system trace entry in a dump and shows where you can find the format of the entry in this section. If you are looking for a particular entry start with this table, because many of the entries are similar and are grouped together.
- "ACR trace entries" on page 158 through "USRn trace entries" on page 203 shows the format for each type of trace entry. For the detailed format of TTEs, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Example of a system trace in a dump

Figure 71 on page 156 shows system trace entries. IPCS formatted the entries from an example SVC dump. Note that system trace data in an ABEND dump has the same format. The subcommand issued from the IPCS Subcommand Entry panel was:

```
SYSTRACE TIME(LOCAL)
```

The oldest trace entries appear first in the trace; the newest entries are at the end. An asterisk (*) before an identifier indicates an unusual condition; see the format of the entry for an explanation.

```
--------------------------------------------------- System Trace Table ---------------------------------------------------------
--                                                  ------------------                                                        --
--                                                  ------------------                                                        --
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Local-------- CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE                    Date-02/19/2016
001C-000A 006D9E88 SSCH  03B92 00  03   237DC020  02380448 53C2A041 6D83A8E8                              15:54:49.794644583 0048
                                                      3B4D
001C-000A 006D9E88 SSCH  03B4D 00  03   02475020  0237DD90 53C2A001 6D83A868                              15:54:49.986753500 0046
001C-000A 006D9E88 SSCH  03A3D 00  03   03A239B0  02374EE0 53C2A001 6D83AD68                              15:54:50.010469553 0030
001C-000A 20D8B900 SRB         00000000_016206A8  0000000A 02069F40 02069F6C       00           000A 000A 15:54:50.012521911 0030
                               07040000 80000000  006D9E88 20
001C-000A 20D8B900 PC      ...   0      07464E4A              00318           Resume   SRB
001C-000A 20D8B900 SSRV    119          8745F3C8  207DD318 8005216B 20792828  Resume                      15:54:50.012538814 0030
                                                  00000000
001C-000A 20D8B900 PR      ...   0      07464E4A 01408F44                                       000A
001C-000A 006D9E88 SSCH  03A3D 00  03   02411568  02374EE0 53C2A001 6D83AC68                              15:54:50.013370673 0030
001C-000A 006D9E88 SSCH  03AAE 00  03   031A38B0  02378AA8 53C2A001 6D83A7E8                              15:54:50.014232347 0030
                                                      3A3D
001C-000A 006D9E88 SSCH  03AAE 00  03   237DC020  02378AA8 53C2A001 6D83ACE8                              15:54:50.015341827 002B
                                                      3A3D
001C-000A 21061E00 SRB         00000000_016206A8  0000000A 20DFCF78 20DFCFA4       00           000A 000A 15:54:50.058874719 0032
                               07040000 80000000  006D9E88 20
001C-000A 031AC800 SRB         00000000_016206A8  0000000A 20DFCF78 20DFCFA4       00           000A 000A 15:54:50.059031815 0032
                               07040000 80000000  006D9E88 20
0023-000A 006D9E88 SSCH  03A3D 00  03   03A23020  02374EE0 53C2A001 6B28FCE8                              16:25:35.116331493 004C
0022-000A 006D9E88 SSCH  03B6D 00  03   03A23020  0237F310 53C2A001 6FA1DBE8                              16:25:35.135359104 0015
0022-000A 20700980 SRB         00000000_016206A8  0000000A 20DFCF78 20DFCFA4       00           000A 000A 16:25:35.149944230 0015
                               47040000 80000000  006D9E88 20
0022-000A 20700980 PC      ...   0      07464E4A              00318           Resume   SRB
0022-000A 20700980 SSRV    119          8745F3C8  03A87AA8 80076F4A 2293FEE8  Resume                      16:25:35.149963318 0015
                                                  00000000
0022-000A 20700980 PR      ...   0      07464E4A 01408F44                                       000A
001D-000A 21451D80 SRB         00000000_016206A8  0000000A 20DFCF78 20DFCFA4       00           000A 000A 16:25:35.200529334 0027
                               47040000 80000000  006D9E88 20
001D-000A 21451D80 PC      ...   0      07464E4A              00318           Resume   SRB
001D-000A 21451D80 SSRV    119          8745F3C8  206ECAD0 800E713F 21A941F8  Resume                      16:25:35.200544172 0027
                                                  00000000
001D-000A 21451D80 PR      ...   0      07464E4A 01408F44                                       000A
001D-000A 03A35100 SRB         00000000_01223632  0000000A 02D1614C 82D16120       FF           000A 000A 16:25:35.247026387 0015
                               47040000 80000000  006D47E8 00
001D-000A 03A35100 SSRV    129          81223776  02D1613C 00000000 00000000  Post                        16:25:35.247027125 0015
                                                  00000000
001D-000A 006D47E8 DSP         00000000_01407008  00000000 00000080 02D1613C  00000000 00000000 000A 000A 16:25:35.247030291 0015
                               47040000 80000000
001D-000A 006D47E8 SSRV     78          81407106  0000F503 00000080 02D16120  Freemain                    16:25:35.247036644 0015
                                                  000A0000
001D-000A 006D47E8 SVCR     2F 00000000_290F8C3A  00000000 91000000 290F9598                              16:25:35.247037242 0015
                               47540000 80000000                                                                             0015
001D-000A 006D47E8 SVC      2F 00000000_290F8C3A  00000000 91000000 290F9598  STimer   Set                16:25:35.247039829 0015
                               47540000 80000000                                                                             0015
```

Figure 71. Example: system trace in an SVC dump

Summary of system trace entry identifiers

This topic summarizes all the system trace entries by identifier. Because many trace entries are similar, they are described together. Use Table 32 on page 156 to locate the format for a particular entry.

For example, in the trace entry shown in Figure 72 on page 156, the system trace identifier is SVC. Look up SVC in Table 32 on page 156 to find where the SVC trace entry format is described. In this case, the SVC trace entry is described in "SVC, SVCE, and SVCR trace entries" on page 198.

```
01 000C 00AFF090   SVC    1 070C2000 00EB19CC  00000000 00000001 00C13340
```

Figure 72. Example: Finding the format for an SVC entry

Table 32. References for system trace entry format description

Identifier (Ident)	Description	For format, see:
ACR	Alternate CPU recovery	"ACR trace entries" on page 158
AINT	Adapter interruption	"AINT trace entries" on page 159
ALTR	Alteration of trace option	"ALTR trace entries" on page 160
BR	Branch through a BAKR, BALR, BASR, or BASSM instruction	"BR trace entries" on page 161
BSG	Branch on subspace group	"BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162
CSCH	Clear subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
DSP	Task dispatch	"DSP, SRB, SSRB, and WAIT trace entries" on page 170

Table 32. References for system trace entry format description (continued)

Identifier (Ident)	Description	For format, see:
EXT	General external interruptions: • CALL - external call interruption • CLKC - clock comparator interruption • EMS - emergency signal interruption • SS - service signal interruption • TIMR - timer interruption • WTI - warning track interruption	"CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries" on page 164
HSCH	Halt subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
I/O	Input/output interruption	"CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries" on page 164
MCH	Machine check interruption	"CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries" on page 164
MOBR	Change of addressing mode along with a change of instruction address	"MODE and MOBR trace entries" on page 172
MODE	Change of addressing mode	"MODE and MOBR trace entries" on page 172
MSCH	Modify subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
PC	Program Call control instruction	"BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162
PCIL	PCI load instruction	"PCIL trace entries" on page 173
PCIS	PCI store instruction	"PCIS trace entries" on page 174
PDMX	PCIE adapter interruption de-multiplexing event	"PDMX trace entries" on page 175
PGM	Program interruption	"PGM, SPER and SPR2 trace entries" on page 176
PR	Program Return control instruction	"BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162
PT	Program Transfer control instruction	"BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162
RCVY	Recovery event	"RCVY trace entries" on page 177
RSCH	Resume subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
RST	Restart interruption	"CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries" on page 164
SIGA	Signal adapter operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
SPER	SLIP program event recording	"PGM, SPER and SPR2 trace entries" on page 176
SPIN	Starting, middle, or stopping of a system spin.	"SPIN trace entries" on page 181
SPR2	SLIP program event recording, when STDATA is specified	"PGM, SPER and SPR2 trace entries" on page 176
SRB	Initial service request block dispatch	"DSP, SRB, SSRB, and WAIT trace entries" on page 170
SSAR	Set Secondary Address Space Number control instruction	"BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries" on page 162

Table 32. References for system trace entry format description (continued)		
Identifier (Ident)	Description	For format, see:
SSCH	Start subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
SSRB	Suspended service request block dispatch	"DSP, SRB, SSRB, and WAIT trace entries" on page 170
SSRV	System service entered by a Program Call (PC) instruction or a branch	"SSRV trace entries" on page 187
SUSP	Lock suspension	"SUSP trace entries" on page 197
SVC	Supervisor call interruption	"SVC, SVCE, and SVCR trace entries" on page 198
SVCE	SVC error	"SVC, SVCE, and SVCR trace entries" on page 198
SVCR	SVC return	"SVC, SVCE, and SVCR trace entries" on page 198
SYNS	Synchronous (zHyperLink) I/O start	"SYNS and SYNE trace entries" on page 200
SYNE	Synchronous (zHyperLink) I/O end	"SYNS and SYNE trace entries" on page 200
TIME	Timer services	"TIME trace entries" on page 202
USRn	User event	"USRn trace entries" on page 203
WAIT	Wait task dispatch	"DSP, SRB, SSRB, and WAIT trace entries" on page 170
XSCH	Cancel subchannel operation	"CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries" on page 167
?EXPL	The SYSTRACE subcommand cannot identify the system trace entry	N/A

ACR trace entries

An ACR trace entry represents failure of a processor and subsequent entry into the alternate CPU recovery component.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   fail wu-addr- *ACR    cpu                    psaeepsw flg-crex psacstk-  psaclhs- psalocal           timestamp---------
cp--
                                                  psasuper psamodew           psaclhse
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `fail`: Home address space identifier (ASID) of the failing processor

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

  ACR
  : Alternate CPU recovery.

CD/D
: `cpu`: The failing processor address from the PSACPUPA field of the PSA

PSW----- Address-
Blank

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `flg-crex`: LCCACREX field of the logical configuration communication area (LCCA) for the failing processor.
- `psacstk-`: PSACSTK field from the prefix save area (PSA) from the failing processor.
- `psaeepsw`: PSAEEPSW field in the PSA. Bytes 1 and 2 contain the failing processor's address. Bytes 3 and 4 contain the external interruption code.
- `psamodew`: PSAMODEW field in the PSA
- `psasuper`: PSASUPER field in the PSA

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA from the failing processor.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA from the failing processor.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA from the failing processor.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

AINT trace entries

An AINT trace entry represents an adapter interruption.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- AINT   aism i/o-old-pswaddr    isc                         psaclhs- psalocal           timestamp---------
cp--
                               i/o-old-pswcntl    desc     desc     desc      psaclhse
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

AINT
Adapter interruption.

CD/D
`aism`: The adapter interruption source mask.

PSW----- Address-
`i/o-old-pswaddr / i/o-old-pswcntl`: I/O old PSW.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `isc`: Interruption subclass.
- `desc`: Description of the adapter types represented by the adapter interruption source mask (IQP, PCIE, Crypto, CF).

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

ALTR trace entries

An ALTR trace entry represents alteration of the system trace options. Alter the options with a TRACE ST operator command.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- *ALTR                          tobtropt gpr0---- gpr1----                   pasd sasd timestamp---------
cp--
                                                  pol-buf-
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

ALTR
Alteration of the trace option. An asterisk (*) always appears before ALTR to indicate an unusual condition.

CD/D
Blank

PSW----- Address-
Blank

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `tobtropt`: Trace options in control register 12 format, from the TOBTROPT field of the system trace option block (TOB)
- `gpr0----`: General register 0
- `gpr1----`: General register 1
- `pol-`: The number of processor with tracing active or suspended, from the TOBTRPOL field of the TOB
- `buf-`: The number of trace buffers per processor, from the TOBTRBUF field of the TOB

PSACLHS-
Blank

PSACLHSE
Blank

PSALOCAL
Blank

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

BR trace entries

A BR trace entry represents processing of a Branch and Link (BALR), Branch and Save (BASR), Branch and Save and Set Mode (BASSM), or Branch and Stack (BAKR) instruction, when the R_2 field in the instruction is not zero. These branches are traced only when a TRACE operator command requests branch tracing by BR=ON.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   last wu-addr- BR          address- address-  address- address- address-  address- etc.
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`last`: Last home address space identifier (ASID) in the trace buffer.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

BR
Branch instruction

CD/D
Blank

PSW----- Address-
Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
PSACLHS-
PSACLHSE
PSALOCAL
PASD
SASD
`address-`: Successful branch address, repeated for consecutive branches on the BR entry. Addresses appear in the following formats:

Addressing mode and location	Appearance
24-bit address	xxxxxx
31-bit address	xxxxxxxx
64-bit address with zeros in high order bits	00_xxxxxxxx
64-bit address with non-zero high order bits	xxxxxxxx_xxxxxxxx

Time *Format*
Blank

CP--
Blank

BSG, PC, PR, PT, PTI, SSAR and SSIR trace entries

These trace entries represent processing of a cross memory instruction:

- A BSG trace entry represents a Branch on Subspace group (BSG) control instruction
- A PC trace entry represents a Program Call (PC) control instruction
- A PR trace entry represents a Program Return (PR) control instruction
- A PT trace entry represents a Program Transfer (PT) control instruction
- A PTI trace entry represents a Program Transfer with Instance (PTI) control instruction
- An SSAR trace entry represents a Set Second Address Space Number (SSAR) control instruction
- An SSIR trace entry represents a Set Secondary Address Space Number with Instance (SSAIR) control instruction

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   last wu-addr- PC          pkey-flg pc-addr-           pc#-----

pr   last wu-addr- PR          psw-key- pr-addr-           pr-faddr                             pasd

pr   last wu-addr- PT          psw-key- pt-addr-           pt-asid-

pr   last wu-addr- PTI         psw-key- pt-addr-           pt-asid-

pr   last wu-addr- SSAR        newsasid                                                              sasd

pr   last wu-addr- SSSR        newsasid                                                              sasd

pr   last wu-addr- BSG         alet     bsg-addr
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `last`: Last home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

  PC
  : Program Call control instruction

  PR
  : Program Return control instruction

  PT
  : Program Transfer control instruction

  PTI
  : Program Transfer with Instance (PTI) control instruction

  SSAR
  : Set Secondary Address Space Number control instruction

  SSIR
  : Set Secondary Address Space Number with Instance (SSAIR) control instruction

  BSG
  : Branch on Subspace Group control instruction

CD/D
: Blank

PSW----- Address-

- `alet`: ALET word during BSG execution
- `newsasid`: New SASID from the SSAR instruction
- `return--`: Caller's return address
- `psw-key-`: Program status word (PSW) key
- `pkey-flag`: Program status word (PSW) key and flags. The flag value is either blank or a hexadecimal value of 1-3:
  - 0 - PSW bit 31 was replaced by a zero and PSW bit 31 was a zero before being replaced.
  - 1 - PSW bit 31 was replaced by a one and PSW bit 31 was a zero before being replaced.
  - 2 - PSW bit 31 was replaced by a zero and PSW bit 31 was a one before being replaced.
  - 3 - PSW bit 31 was replaced by a one and PSW bit 31 was a one before being replaced.

- `pc-addr-`: Return address from the PC instruction. The low bit of the address is 0 if the PC was issued in supervisor state and 1 if the PC was issued in problem state
- `pr-addr-`: New instruction address as updated by the PR instruction. The low bit of the address is 0 if the resulting PSW will be supervisor state and 1 if the resulting PSW will be problem state
- `pt-addr-`: New instruction address as updated by the PT instruction
- `bsg-addr`: New instruction address as updated by the BSG instruction

Addresses appear in the following formats:

Addressing mode and location	Appearance
24-bit address	xxxxxx
31-bit address	xxxxxxxx
64-bit address with zeros in high order bits	00_xxxxxxxx
64-bit address with non-zero high order bits	xxxxxxxx_xxxxxxxx

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `pc#-----`: PC number from the PC instruction
- `pr-faddr`: Address of the location following the PR instruction
- `pt-asid-`: New ASID specified on the PT instruction

PSACLHS-
This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
Blank

PSALOCAL
This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
`pasd`: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD
`sasd`: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

Time *Format*
Blank

CP--
Blank

CALL, CLKC, EMS, EXT, I/O, MCH, RST, SS, TIMR, and WTI trace entries

The following trace entries represent an interruption:

- These entries represent external interruptions:
  - A CALL trace entry is for an external call
  - A CLKC trace entry is for a clock comparator
  - An EMS trace entry is for an emergency signal
  - An EXT trace entry is for a general external interruption
  - An SS trace entry is for a service signal
  - A TIMR trace entry is for a timer interruption

  - A WTI trace entry is for a warning track interruption
  - A Key entry for an interrupt key interruption
  - A TimA entry for a timing alert interruption
- An I/O trace entry is for an I/O interruption
- An MCH trace entry is for a machine check
- An RST trace entry is for a restart

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- EXT    CALL ext-old- psw-----  psaeepsw pccarph-           psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-                              psaclhse

pr   home wu-addr- EXT    CLKC ext-old- pswaddr-  psaeepsw tqe-addr tqe-asid  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-                              psaclhse

pr   home wu-addr- EXT     EMS ext-old- pswaddr-  psaeepsw pccaemsi pccaemsp  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-  pccaemse                    psaclhse

pr   home wu-addr- EXT    code ext-old- pswaddr-  psaeepsw                    psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-                              psaclhse

pr   home wu-addr- I/O     dev i/o-old- pswaddr-  flg-ctl- ccw-addr dvch-cnt  psaclhs- psalocal pasd sasd timestamp---------
cp--
                           dev i/o-old- pswctrl-  ucb-addr ext-stat           psaclhse

pr   home wu-addr- *MCH        mch-old- pswaddr-  machine- chk-code psasuper  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               mch-old- pswctrl-                              psaclhse

pr   home wu-addr- *RST        rst-old- pswaddr-  gpr15--- gpr0---- gpr1----  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               rst-old- pswctrl-  psasuper psamodew

pr   home wu-addr- EXT      SS ext-old- pswaddr-  psaeepsw psaeparm msf-bcmd  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-  flg-brsp mssfasid mssfatcb  psaclhse

pr   home wu-addr- EXT    TIMR ext-old- psw-----  psaeepsw pccarph-           psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-                              psaclhse

pr   home wu-addr- EXT     WTI ext-old- psw-----  psaeepsw pccarph-           psaclhs- psalocal pasd sasd timestamp---------
cp--
                               ext-old- pswctrl-                              psaclhse
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

EXT
: General external interruption. An asterisk before EXT indicates that the interrupt is a malfunction alert (MFA) or is the result of pressing the External Interrupt key.

I/O
: I/O interruption. An asterisk before I/O indicates that one of the following bits in IRBFLAGS field of the interrupt request block (IRB) is ON. The IRBFLAGS field is in the Unique-1 column of the I/O entry.

- IRBN for path not operational
- IRBSALRT for alert status

MCH
Machine check interruption.

RST
Restart interruption.

CD/D

CALL
External call external interruption

CLCK
Clock comparator external interruption

EMS
Emergency signal external interruption

SS
Service signal external interruption

TIMR
Timer interruption

WTI
Warning track interruption

code
External interruption code

dev
Device number associated with the I/O or, for a co-processor device, the I/O co-processor identifier, for example, ADM

PSW----- Address-
The z/Architecture 128-bit PSW address appears on two lines:

- `pswaddr-`: Two words, containing the 64-bit address portion of the PSW
- `pswctrl-`: Two words, containing the 64-bit "control" portion of the PSW

The PSW represents different data, depending on the type of interrupt that was traced:

- `ext-old- pswaddr- / ext-old- pswctrl-`: External old program status word (PSW)
- `i/o-old- pswaddr- / i/o-old- pswctrl-`: I/O old PSW
- `mch-old- pswaddr- / mch-old- pswctrl-`: Machine check old PSW
- `rst-old- pswaddr- / rst-old- pswctrl-`: Restart old PSW

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `ccw-addr`: Address of the channel command word (CCW) for the I/O
- `-cnt`: Residual count
- `dvch`: Device status and subchannel status
- `ext-stat`: Extended status word
- `flg-brsp`: Maintenance and service support facility (MSSF) hardware flags and MSSF response code
- `flg-ctl-`: IRBFLAGS field in the IRB and the subchannel control bytes
- `gpr15--- gpr0---- gpr1----`: General registers 15, 0, and 1
- `machine- chk-code`: Machine check interruption code from the FLCMCIC filed in the prefix save area (PSA)
- `msf-bcmd`: Service processor command word
- `mssfasid`: Service processor address space ID
- `mssfatcb`: Service processor TCB address

- `pccaemse`: PCCAEMSE field from the physical configuration communication area (PCCA)
- `pccaemsi`: PCCAEMSI field from the PCCA
- `pccaemsp`: PCCAEMSP field from the PCCA
- `pccarph-`: PCCARPB field from the PCCA
- `psaeepsw`: PSAEEPSW field in the PSA. For CALL and EMS, bytes 1 and 2 contain the issuing processor's address. For all entries, bytes 3 and 4 contain the external interruption code.
- `psaeparm`: PSAEPARM field in the PSA, containing the MSSF buffer address
- `psamodew`: PSAMODEW field in the PSA
- `psasuper`: PSASUPER field in the PSA
- `tqe-asid`: ASID of the associated timer queue element (TQE)
- `tqe-addr`: TQE address
- `ucb-addr`: Unit control block (UCB) address

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

CSCH, HSCH, MSCH, RSCH, SSCH, SIGA and XSCH trace entries

These trace entries represent an input/output operation:

- A CSCH trace entry represents a clear subchannel operation
- An HSCH trace entry represents a halt subchannel operation
- An MSCH trace entry represents a modify subchannel operation
- An RSCH trace entry represents a resume subchannel operation
- An SSCH trace entry represents a start subchannel operation
- An SIGA trace entry represents a signal adapter operation
- An XSCH trace entry represents a cancel subchannel operation

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   asid wu-addr- CSCH    dev cc  di   iosbaddr  ucb-addr ioq-addr asc-iosb                               timestamp---------
cp--

pr   asid wu-addr- HSCH    dev cc  di   iosbaddr  ucb-addr ioq-addr asc-iosb                               timestamp---------
cp--

pr   asid wu-addr- MSCH    dev cc       iosbaddr  ucb-addr f1f2pmom mbi-t2lb                               timestamp---------
cp--

pr   asid wu-addr- RSCH    dev cc  di   iosbaddr  ucb-addr                                                 timestamp---------
cp--

pr   asid wu-addr- SSCH    dev cc  di   iosbaddr  ucb-addr orb-wrd2 orb-wrd3                               timestamp---------
cp--
                                                  orb-wrd4 cap-addr bdev

pr   asid wu-addr- XSCH    dev cc  di   iosbaddr  ucb-addr ioq-addr bdev                                   timestamp---------
cp--

pr   asid wu-addr- SIGA    dev cc  fc   qib-addr  subsysid q-mask-1 q-mask-2                               timestamp---------
cp--
                                                  ucb-addr
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`asid`: Address space identifier (ASID) related to the I/O.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

CSCH
Clear subchannel operation

HSCH
Halt subchannel operation

MSCH
Modify subchannel operation

RSCH
Resume subchannel operation

SSCH
Start subchannel operation

XSCH
Cancel subchannel operation

SIGA
Signal adapter operation

An asterisk before RSCH, SSCH, or SIGA indicates that the condition code associated with the I/O was not 0.

CD/D
`dev`: One of the following:

- The device number associated with the I/O, which will include the subchannel set identifier when appropriate.
- ADMF, if the IOSADMF macro was transferring data

PSW----- Address-

- `cc`: Condition code in bits 2 and 3 associated with the I/O

- `di`: Driver identifier associated with the I/O
- `fc`: Function code associated with the I/O
- `iosbaddr`: I/O supervisor block (IOSB) address associated with the I/O
- `qib-addr`: Queue identification block (QIB) address associated with the I/O

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `asc-iosb`: IOSB address for the associated SSCH request for the I/O
- `bdev`: The base device number if the I/O is associated with an alias device.
- `cap-addr`: Captured unit control block (UCB) address associated with the SSCH I/O. This field is blank if a below 16 megabyte UCB or actual above 16 megabyte UCB address was used for the start subchannel (SSCH) operation. The address of the actual above 16 megabyte UCB is in the `ucb-addr` field.
- `f1f2pmom`: From the subchannel information block (SCHIB) associated with the I/O, as follows:

  f1
  SCHFLG1 flag field

  f2
  SCHFLG2 flag field

  pm
  SCHLPM field

  om
  SCHPOM field
- `ioq-addr`: I/O queue (IOQ) address associated with the I/O
- `mbi-t2lb`:

  mbi-
  SCHMBI field from the SCHIB

  t2
  IOSOPT2 field from the IOSB

  lb
  IOSFLB field from the IOSB
- `orb-wrd2`: Word 2 of the operation request block (ORB) associated with the I/O
- `orb-wrd3`: Word 3 of the operation request block (ORB) associated with the I/O
- `orb-wrd4`: Word 4 of the operation request block (ORB) associated with the I/O
- `q-mask-1`: Read or write queue mask associated with the I/O
- `q-mask-2`: Read queue mask associated with the I/O
- `subsysid`: Subsystem ID associated with the I/O
- `ucb-addr`: Unit control block (UCB) address associated with the I/O

PSACLHS-
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

PSACLHSE
Blank

PSALOCAL
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

PASD
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

SASD
This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

DSP, SRB, SSRB, and WAIT trace entries

These trace entries represent the dispatch of a unit of work:

- A DSP trace entry represents dispatch of a task
- An SRB trace entry represents the initial dispatch of a service request
- An SSRB trace entry represents dispatch of a suspended service request
- A WAIT trace entry represents dispatch of the wait task

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- DSP         dsp-new- pswaddr-  psamodew gpr0---- gpr1----  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               dsp-new- pswctrl-

pr   home wu-addr- SRB         srb-new- pswaddr-  safnasid gpr0---- gpr1----  srbhlhi-          pasd sasd timestamp---------
cp--
                               srb-new- pswctrl-  purgetcb flg-srb-

pr   home wu-addr- SSRB        ssrb-new- pswaddr- safnasid          gpr1----  psaclhs4 psalocal pasd sasd timestamp---------
cp--
                               ssrb-new- pswctrl- purgetcb

pr   home wu-addr- WAIT                                                                                  timestamp---------
cp--
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

DSP
Task dispatch

SRB
Initial service request dispatch

SSRB
Suspended service request dispatch

WAIT
Wait task dispatch

CD/D
Blank

PSW----- Address-

The z/Architecture 128-bit PSW address appears on two lines:

- `pswaddr`: Two words, containing the 64-bit address portion of the PSW
- `pswctrl`: Two words, containing the 64-bit "control" portion of the PSW

The PSW represents different data, depending on the type of work that was dispatched:

- `dsp-new- pswaddr / dsp-new- pswctrl`: Program status word (PSW) to be dispatched
- `srb-new- pswaddr / srb-new- pswctrl`: PSW to receive control on the SRB dispatch
- `ssrb-new pswaddr / ssrb-new pswctrl`: PSW to receive control on the SSRB redispatch

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `gpr0----`: General register 0
- `gpr1----`: General register 1
- `psamodew`: PSAMODEW field in the PSA
- `safnasid`: LCCASAFN field in the logical configuration communication area (LCCA) and the related ASID
- `flg-srb`: SRBFLGS field from the SRB
- `purgetcb`: TCB (located in address space of the scheduler of the SRB or SSRB) that gets control if the SRB or SSRB abends and percolates

PSACLHS-

- `psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.
- `psaclhs4`: PSACLHS4 field of the PSA
- `srbhlhi-`: SRBHLHI field in the SRB

This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE

Blank

PSALOCAL

`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD

`pasd`: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD

`sasd`: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

Time *Format*

`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--

`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

MODE and MOBR trace entries

These trace entries represent a change of addressing mode:

- A MODE trace entry represents a change into or out of 64-bit addressing mode
- A MOBR trace entry represents a change into or out of 64-bit addressing mode along with a change of instruction address

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE
pr   last wu-addr- MODE        target   address-  address- address- address-  address- etc.
pr   last wu-addr- MOBR        target   address-  address- address- address-  address- etc.
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`last`: Last home address space identifier (ASID) in the trace buffer.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

MODE
Addressing mode change instruction

MOBR
Addressing mode change combined with a branch instruction

CD/D
Blank

PSW-----
`target`: Target addressing mode.

24 OR 31
Target addressing mode is either 24-bit or 31-bit.

64
Target addressing mode is either 64-bit.

Address-
Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
PSACLHS-
PSACLHSE
PSALOCAL
PASD
SASD
`address-`: Target address. Addresses appear in the following formats:

Addressing mode and location	Appearance
24-bit address	xxxxxx
31-bit address	xxxxxxxx
64-bit address with zeros in high-order bits	00_xxxxxxxx
64-bit address with nonzero high-order bits	xxxxxxxx_xxxxxxxx

Time *Format*
Blank

CP--
Blank

PCIL trace entries

A PCIL trace represents a PCI load instruction.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- PCIL   pfid cc       reqaddr-  traceid  operand1_operand1  psaclhs- psalocal           timestamp---------
cp--
                                                           operand2_operand2  psaclhse
                                                           operand3_operand3
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

PCIL
PCI load instruction.

CD/D
`pfid`: The PCIE function identifier for the PCIE device.

PSW----- Address-

- `cc`: Condition code from the PCI load instruction.
- `reqaddr`: Address of the program that requested the PCI load instruction to be issued.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `traceid`: Program defined trace identifier that can be used to determine why the PCI load instruction is being issued.
- `operand1_operand1`: First operand on the PCI load instruction.
- `operand2_operand2`: Second operand on the PCI load instruction.
- `operand3_operand3`: Third operand on the PCI load instruction.

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
: `timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
: `cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

PCIS trace entries

A PCIS trace represents a PCI store instruction.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- PCIS   pfid cc       reqaddr-  traceid  operand1_operand1  psaclhs- psalocal           timestamp---------
cp--
                                                           operand2_operand2  psaclhse
                                                           operand3_operand3
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

  PCIS
  : PCI store instruction.

CD/D
: `pfid`: The PCIE function identifier for the PCIE device.

PSW----- Address-

- `cc`: Condition code from the PCI store instruction.
- `reqaddr`: Address of the program that requested the PCI store instruction to be issued.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `traceid`: Program defined trace identifier that can be used to determine why the PCI store instruction is being issued.
- `operand1_operand1`: First operand on the PCI store instruction.
- `operand2_operand2`: Second operand on the PCI store instruction.
- `operand3_operand3`: Third operand on the PCI store instruction.

PSACLHS-
: `psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
: `psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
: `psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

PDMX trace entries

A PDMX trace represents a PCIE adapter interruption de-multiplexing event.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- PDMX   pfid pdmxaddr           devtype  cback@   cbparm1   psaclhs- psalocal           timestamp---------
cp--
                                                  cbparm2                     psaclhse
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

PDMX
PCIE adapter interruption de-multiplexing event.

CD/D
`pfid`: PCIE function identifier for the PCIE device.

PSW----- Address-
`pdmxaddr`: Address of the program that is performing the de-multiplexing.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `devtype`: PCIE device type.
- `cback@`: Callback routine address.
- `cbparm1`: First word of callback routine parameters.
- `cbparm2`: Second word of callback routine parameters.

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

PGM, SPER and SPR2 trace entries

These trace entries represent a program event:

- A PGM trace entry is for a program interrupt
- An SPER trace entry is for a PER event requested in a SLIP trap
- An SPR2 trace entry is for a PER event requested in a SLIP trap, when the STDATA keyword is specified on the trap

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- PGM    code pgm-old- pswaddr-  ilc-code tea-----           psaclhs- psalocal pasd sasd timestamp---------
cp--
                               pgm-old- pswctrl-           tea-----           psaclhse

pr   home wu-addr- SPER   code pgm-old- pswaddr-  ilc-code          trap----  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               pgm-old- pswctrl-  per-addH per-addL           psaclhse

pr   home wu-addr- SPR2   code pgm-old- pswaddr-  var1     var2     var3      psaclhs- psalocal pasd sasd timestamp---------
cp--
                               pgm-old- pswctrl-  var4     var5     spc-exc
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

PGM
Program interruption. An asterisk (*) before PGM indicates an unusual condition. PGM trace entries for program interrupts that may be resolved are not flagged. If the program interrupt is not resolved, then a subsequent RCVY trace entry is created and flagged with an asterisk.

SPER
SLIP program event recording

CD/D

- `code` for PGM entry: Program interruption code
- `code` for SPER entry: PER number

PSW----- Address-

The z/Architecture 128-bit old PSW appears on two lines:

- `pswaddr`: Two words, containing the 64-bit address portion of the PSW
- `pswctrl`: Two words, containing the 64-bit "control" portion of the PSW

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `ilc-code`: Instruction length code and interruption code.
- `per-addH`: high-order bits of the SLIP/PER status address.
- `per-addL`: low-order bits of the SLIP/PER status address.
- `tea-----`: Translation exception address. In the high-order bit, 0 indicates primary and 1 indicates secondary.
- `trap----`: SLIP/PER trap identifier in the form ID=xxxx.
- `var1`, `var2`, `var3`, `var4`, `var5`: Each contains one word of variable data as specified by the STDATA keyword.
- `spc-exc`: The message `SpaceExc` if more than five words of variable data are requested in the STDATA keyword.

PSACLHS-

`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE

`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL

`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD

`pasd`: Primary ASID (PASID) at trace entry.

SASD

`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*

`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--

`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

RCVY trace entries

A RCVY trace entry represents entry into a recovery routine following an error or interruption. Several types of recovery events require reentry in a new environment or address space. Table 33 on page 177 summarizes when an RCVY trace event requires reentry. RCVY also writes records to the system trace table for ESTAE type recovery exits and when the ESTAE type recovery exit requests retry.

Table 33. RCVY trace events that require reentry

Trace Entry for Recovery Event	Reentry	Trace Entry for Reentry
RCVY ABT	Required only if the task to be ended resides in an address space other than the current home address space	
RCVY ITRM	Always required	RCVY ITRR, if the unit of work ending is locally locked or has an EUT FRR established

Table 33. RCVY trace events that require reentry (continued)		
Trace Entry for Recovery Event	**Reentry**	**Trace Entry for Reentry**
RCVY MEM	Always required	RCVY MEMR
RCVY ABTR		
RCVY RCML	Always required	RCVY RCMR
RCVY STRM	Always required	RCVY STRR, if the unit of work ending is in SRB mode, is locally locked, or has an EUT FRR established

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format------- CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- *RCVY  ABRT                    trk-----                    psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  ABT           return--  comp---- reas---- rc------  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  asid---- tcb-----           psaclhse

pr   home wu-addr- *RCVY  ABTR                    comp---- reas---- rc------  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  asid---- tcb-----           psaclhse

pr   home wu-addr- *RCVY  DAT                     comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  FRR  frr-new- psw-----  comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                    fpw-----  psaclhse

pr   home wu-addr- *RCVY  ITRM          return--  comp---- reas----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  ITRR                    comp---- reas----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  MCH                     comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  MEM           return--  comp---- reas---- rc------  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  asid----                    psaclhse

pr   home wu-addr- *RCVY  MEMR                    comp---- reas----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  asid----                    psaclhse

pr   home wu-addr- *RCVY  PERC                    comp---- reas----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                    fpw-----  psaclhse

pr   home wu-addr- *RCVY  PROG                    comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  RCML          return--  comp---- reas---- asid----  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  RCMR                    comp---- reas----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  RESM retry--- pswaddr-  comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                               retry--- pswctrl-  cpu-----          fpw-----  psaclhse

pr   home wu-addr- *RCVY  RSRT                    comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  RTRY retry--- pswaddr-  comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                               retry--- pswctrl-                    fpw-----  psaclhse

pr   home wu-addr- *RCVY  SABN                    comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  SPRC                    comp---- reas---- psasuper  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  asid---- tcb----- fpw-----  psaclhse

pr   home wu-addr- *RCVY  SRBT          return--  comp---- reas---- rc------  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                         srbidtoken-------------------------  psaclhse

pr   home wu-addr- *RCVY  STRM          return--  comp---- reas---- tcb-----  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  STRR                    comp---- reas---- tcb----   psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  pppp---- pppp----           psaclhse

pr   home wu-addr- *RCVY  ESTA          exit----  sdwa---- parm64-- parm----  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                  alet---- scb-----           psaclhse

pr   home wu-addr- *RCVY  ESTR retry--- retry---  exit---- scb-----           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse

pr   home wu-addr- *RCVY  SKFE          exit----  scb-----                    psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                              psaclhse
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

RCVY
Recovery event. An asterisk (*) always appears before RCVY to indicate an unusual condition.

CD/D
Type of recovery event, as follows:

- ABRT: Abort processing for an unrecoverable error during any recovery termination management (RTM) processing
- ABT: Request for abnormal end of a task by a CALLRTM TYPE=ABTERM macro, with a system or user completion code
- ABTR: Rescheduling of a CALLRTM TYPE=ABTERM request for end of a task, when the task is not in the home address space
- DAT: RTM1 entered for a dynamic address translation (DAT) error
- ESTA: RTM has set up this ESTAE type recovery exit to receive control. Note that when a RCVY ESTA record follows another RCVY ESTA record without an intervening RCVY ESTR record, this implies that the first recovery exit either abended or percolated
- ESTR: Retry was requested by this ESTAE type recovery exit. The exit and scb fields in this record can be used to help match it to its corresponding RCVY ESTA record.
- FRR: RTM1 processing to invoke a function recovery routine (FRR)
- ITRM: The system requested RTM1 to end an interrupted task
- ITRR: ITRM reentry, to process a request to end an interrupted task
- MCH: RTM1 entered for a machine check interruption
- MEM: Request for abnormal memory end by a CALLRTM TYPE=MEMTERM macro, with a completion code
- MEMR: Processing for an abnormal memory end following a MEM event
- PERC: Percolation from RTM1 to RTM2 to continue recovery processing
- PROG: RTM1 was entered for a program check interruption
- RCML: RTM1 was entered to perform special end processing for a task in a failing address space. The failing address space held the local lock of another address space.
- RCMR: RCML reentry, to process an abnormal end by a resource manager
- RESM: Resume from an FRR after a RESTART request following an RSRT entry
- RSRT: RTM entered for a RESTART request from the operator
- RTRY: Retry from an FRR
- SABN: The system requested RTM1 to end abnormally the current unit of work
- SKFE: RTM has bypassed giving control to a FESTAE recovery exit because its address is zero. This situation can happen because of a timing window in FESTAE processing, in which case it is not a concern
- SPRC: Final percolation from service request block (SRB) recovery
- SRBT: Request for abnormal termination of a pre-emptable SRB by a CALLRTM TYPE=SRBTERM macro, with a system or user completion code
- STRM: The system requested RTM1 to end abnormally a suspended task
- STRR: STRM reentry, to process the abnormal end of a suspended task

PSW----- Address-
For the RESM and RTRY entries, this field contains the z/Architecture 128-bit old PSW, which appears on two lines:

- `pswaddr-`: Two words, containing the 64-bit address portion of the PSW
- `pswctrl-`: Two words, containing the 64-bit "control" portion of the PSW

- `exit----`: Address of an ESTAE type recovery exit. This is always zero for RCVY SKFE entries.
- `retry--- retry---` : Retry address requested by an ESTAE type recovery exit. When retry is to CVTBSM0F, contains the contents of 64-bit GPR15 at the time of the retry request instead of the address of CVTBSM0F.

The `frr-new- psw-----` field contains the new program status word (PSW) to give control to the FRR. For all other types of recovery events, the `return--` field contains the caller's return address.

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `alet----`: Contains the ALET of the parameter area for ESTAE type recovery exits with an ALET qualified parameter area pointer. Otherwise, it contains zero.
- `asid----`: Target ASID for end processing.
  - In a SPRC entry, the ASID is for the task that will be abnormally ended by SRB-to-task percolation. If this field and the

  `tcb-----` field are zero, then no SRB-to-task percolation is performed.
- `exit----`: Contains the ESTAE type recovery exit address
- `comp----`: System or user completion code
- `cpu-----`: Target processor for a restart error indicated on a request for an FRR resume, after an operator RESTART request
- `fpw-----`: FRR processing word, in the following format:

```
rstxxxxxp xxxxxxxx ssssssss eeeeeeee
```

r
Bit 0 = 1 means a resource manager entry to the FRR

s
Bit 1 = 1 means the FRR was skipped

t
Bit 2 = 1 means the FRR is EUT=YES

p
Bit 7 = 0 means a not serialized SRB-to-task percolation

Bit 7 = 1 means a serialized SRB-to-task percolation

ssssssss
The stack index, which is an index of the FRR stack. The index means the following:

- 0 Normal stack
- 1 SVC I/O dispatcher super stack
- 2 Machine check super stack
- 3 PC FLIH super stack
- 4 External FLIH super stack 1
- 5 External FLIH super stack 2
- 6 External FLIH super stack 3
- 7 Restart super stack
- 8 ACR super stack
- 9 RTM super stack

eeeeeeee
The entry index, which is an index of the FRR entry on the stack. The index ranges from 0 through 16. If the current stack is a super stack, an index of 0 indicates a super FRR.

- `parm----`: Contains the 31-bit parameter address, or the lower 32 bits of the 64-bit parameter address, which is provided to the ESTAE type recovery exit when it gains control.
- `parm64--`: Contains the upper 32 bits of the 64-bit parameter address when the ESTAE type recovery exit was established in AMODE 64
- `pppp---- pppp----`: PSW of the interrupted unit of work.
  - The instruction in the PSW may not be the cause of the failure. For example, an interruption can occur because a time limit expired, so that the interrupted instruction is not at fault.
- `psasuper`: PSASUPER field in the prefix save area (PSA)
- `rc------`: Return code from CALLRTM
- `reas----`: Reason code accompanying the completion code appearing in the entry. If not provided, `NONE`.
- `srbidtoken`: Uniquely identifies the preemptable SRB, provided via the IEAMSCHD macro and consisting of four unique words on the second line for the RCVY SRBT trace entry only.
- `tasn----`: Target ASID for RCML reentry
- `tcb-----`: Target task control block (TCB) for end processing
  - In a SPRC entry, the TCB is for the task that will be abnormally ended by SRB-to-task percolation. If this field and the `asid----` field are zero, then no SRB-to-task percolation is performed.
  - In a STRM or STRR entry, a TCB address of zero indicates that the request was for ending of a suspended SRB.
- `trk-----`: RTM1 error tracking area
- `scb-----`: Contains the address of the STAE control block that represents this ESTAE type recovery exit. Note that for ARR and IEAARR recovery routines, RTM creates a 'pseudo-SCB'. Thus the same SCB address can be seen for multiple ARR or IEAARR recovery exits
- `sdwa----`: Contains the address of the system diagnostic work area that is provided to the ESTAE type recovery exit. If no SDWA is available to the exit, the field contains 0000000C

PSACLHS-
: `psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
: `psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
: `psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
: `pasd`: Primary ASID (PASID) at trace entry.

SASD
: `sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
: `timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
: `cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

SPIN trace entries

A SPIN trace entry represents the starting (at least one second in), the middle (when special processing is done), or the stopping of a system spin attempting to obtain a resource. The spinning module will identify the resource within the trace entry.

```
PR   ASID WU-Addr- Ident  CD/D      PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format------- CP--
                                                       Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- SPIN   SRC/{S|P}          return--  spin-dur holder-- rstrscid  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                                   psaclhse

pr   home wu-addr- SPIN   ACO/{S|P}          return--  spin-dur holder-- infncode  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       ascb-add inp-parm           psaclhse

pr   home wu-addr- SPIN   BBR/{S|P}          return--  spin-dur cpu-spin infncode  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       reg1----                    psaclhse

pr   home wu-addr- SPIN   INT/{S|P}          return--  spin-dur cpu-spin           psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                                   psaclhse

pr   home wu-addr- SPIN   LKX/{S|P}          return--  spin-dur lockword lock-add  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       pllhsom- plclhsp- lock-ent  psaclhse

pr   home wu-addr- SPIN   RI/{S|M|P}         return--  spin-dur cpu-spin req-code  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       rcv-addr pccaaddr           psaclhse

pr   home wu-addr- SPIN   SGP/{S|P}          return--  spin-dur cpu-spin para-reg  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       sigpcode sigpstat           psaclhse

pr   home wu-addr- SPIN   SPN/{S|P}          return--  spin-dur cpu-spin reg0---   psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                       reg1---- reg3----           psaclhse

pr   home wu-addr- SPIN   MTC/{S|P}          return--  spin-dur phycpu-- asid----  psaclhs- psalocal pasd sasd timestamp--------- cp--
                                                                                   psaclhse
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

  SPIN
  : System spin.

CD/D
: The CD/D field indicates the spinning module, using the last two or three characters of its module name, followed by a forward slash (/) and S (start), M (middle), or P (stop). For more information about each spinning module, see “Spinning modules” on page 183.

PSW----- Address-
: `return--`: Caller's return address

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
: For more information about each spinning module, see “Spinning modules” on page 183.

PSACLHS-
: `psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
: `psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
: `psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
: `pasd`: Primary ASID (PASID) at trace entry.

SASD
: `sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
: `timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--

`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

Spinning modules

The following section summarize information about the spinning modules.

BLWRESRC

CD/D

SRC/S
The start spin entry of the BLWRESRC module.

SRC/P
The stop spin entry of the BLWRESRC module.

Unique-1

spin-dur
Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2

holder--
CPU that is a current holder.

Unique-3

rstrscid
The restart resource ID of the caller (input parameter 1).

IEAVEAC0

CD/D

AC0/S
The start spin entry of the IEAVEAC0 module.

AC0/P
The stop spin entry of the IEAVEAC0 module.

Unique-1

spin-dur
The spin duration so far (bytes 3,4,5, and 6 of time since the start of spin).

Unique-2

holder--
CPU that is a current holder.

Unique-3

infncode
The input function code.

Unique-4

ascb-add
Current_ASCB

Unique-5

inp-parm
input_parm

IEAVEBBR

CD/D

BBR/S
The start spin entry of the IEAVEBBR module.

BBR/P
The stop spin entry of the IEAVEBBR module.

Unique-1

spin-dur
The spin duration so far (bytes 3,4,5, and 6 of time since the start of spin).

Unique-2

cpu-spin
The CPU address being spun for.

Unique-3

infncode
The input function code.

Unique-4

reg1----
input reg 1: when applicable, the ASID in bits 16-31.

IEAVEINT

CD/D

INT/S
The start spin entry of the IEAVEINT module.

INT/P
The stop spin entry of the IEAVEINT module.

Unique-1

spin-dur
The spin duration so far (bytes 3,4,5, and 6 of time since the start of spin).

Unique-2

cpu-spin
The CPU address being spun for.

IEAVELKX

CD/D

LKX/S
The start spin entry of the IEAVELKX module.

LKX/P
The stop spin entry of the IEAVELKX module.

Unique-1

spin-dur
Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2

lockword
The lock word that contains the CPU address being spun for.

Unique-3

lock-add
Input reg 11 (lockword address).

Unique-4

pllhsom-
Lock held obtained mask (PLLHSOM, via input reg 12).

Unique-5

plclhsp-
Lock held string pointer (PLCLHSP, via input reg 12).

Unique-6

lock-ent
Input reg 13. The lock routine entry point address.

IEAVERI

CD/D

RI/S
The start spin entry of the IEAVERI module.

RI/M
The middle spin entry of the IEAVERI module.

RI/P
The stop spin entry of the IEAVERI module.

Unique-1

spin-dur
The spin duration so far (bytes 3,4,5, and 6 of time since the start of spin).

Unique-2

cpu-spin
The CPU address being spun for.

Unique-3

req-code
input register 0. Request code and, when appropriate, ASID.

Unique-4

rcv-addr
Input reg 12. Receiving routine's entry point address.

Unique-5

pccaaddr
Input reg 1. PCCA address of the receiving CPU. If this identifies the same CPU as Unique 2, Unique 2 value can be used.

IEAVESGP

CD/D

SGP/S
The start spin entry of the IEAVESGP module.

SGP/P
The stop spin entry of the IEAVESGP module.

Unique-1

spin-dur
Spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2

cpu-spin
The CPU address being spun for.

Unique-3

para-reg
Input reg 1. Parameter register for status and prefix order codes.

Unique-4

sigpcode
Input reg 2. The SIGP order code.

Unique-5

sigpstat
The status returned from the last SIGP

IEAVESPN

CD/D

SPN/S
The start spin entry of the IEAVESPN module.

SPN/P
The stop spin entry of the IEAVESPN module.

Unique-1

spin-dur
The spin duration so far (bytes 3, 4, 5, and 6 of time since the start of spin).

Unique-2

cpu-spin
The CPU address being spun for.

Unique-3

reg0----
Input reg 0.

Unique-4

reg1----
Input reg 1.

Unique-5

reg3----
Input reg 3.

IEAVTMTC

CD/D

MTC/S
The start spin entry of the IEAVTMTC module.

MTC/P
The stop spin entry of the IEAVTMTC module.

Unique-1

spin-dur
The spin duration so far (bytes 3,4,5, and 6 of time since the start of spin).

Unique-2

phycpu--
The physical CPU number of some CPU that is still running with the terminating ASCB for "S" and 0 for "P". If an ACR condition is encountered, value can also be 0 for "S".

Unique-3

asid----
The ASID that is the target of CALLRTM TYPE=MEMTERM.

SSRV trace entries

An SSRV trace entry represents entry to a system service. The service can be entered by a PC instruction or a branch.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- SSRV   ssid          return--  data---- data---- data----  desc---- desc---- pasd sasd timestamp---------
cp--
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

SSRV
Request for a system service

CD/D
`ssid`: One of the following SSRV entry identifiers:

ssid (hexadecimal)	Macro for SSRV request	Component
0001	WAIT	Task management
0002	POST	Task management
0004	GETMAIN	Virtual storage management
0005	FREEMAIN	Virtual storage management
000A	GETMAIN, FREEMAIN	Virtual storage management
005F	SYSEVENT	System resource manager
0078	GETMAIN, FREEMAIN	Virtual storage management
007A	SPI, SPIINT	Service processor interface
0100	ETCON	PC/AUTH
0101	ETCRE	PC/AUTH
0102	ATSET	PC/AUTH
0103	AXSET	PC/AUTH
0104	AXEXT	PC/AUTH

ssid (hexadecimal)	Macro for SSRV request	Component
0105	AXFRE	PC/AUTH
0106	AXRES	PC/AUTH
0107	ETDES	PC/AUTH
0108	ETDIS	PC/AUTH
0109	LXFRE	PC/AUTH
010A	LXRES	PC/AUTH
010E	SUSPEND	Supervisor control
010F	RESUME	Supervisor control
0110	SCHEDULE	Supervisor control
0111	SCHEDULE	Supervisor control
0112	SCHEDULE	Supervisor control
0113	DSGNL	Supervisor control
0114	RISGNL	Supervisor control
0115	RPSGNL	Supervisor control
0116	SCHEDULE	Supervisor Control
0117	SCHEDULE	Supervisor Control
0118	SUSPEND	Supervisor Control
0119	RESUME	Supervisor Control
011A	RESUME	Supervisor Control
011B	RESUME	Supervisor Control
011C	SCHEDULE	Supervisor Control
011D	IEAMSCHD	Supervisor Control
011E	Pause (IEAVPSE / IEAVPSE2 / IEA4PSE / IEA4PSE2)	Supervisor Control
011F	Release (IEAVRLS / IEAVRLS2 / IEA4RLS / IEA4RLS2)	Supervisor Control
0120	Timer DIE	Supervisor Control
0128	WAIT	Task management
0129	POST	Task management
012A	POST	Task management
012B	POST	Task management
012C	ASCBCHAP	Task management
012D	STATUS	Task management
012E	STATUS	Task management
0132	STORAGE OBTAIN	Virtual storage management
0133	STORAGE RELEASE	Virtual storage management
0146	SPI, SPIINT	Service processor interface
014B	IARV64	Real storage management

ssid (hexadecimal)	Macro for SSRV request	Component
014C	ISGENQ	Global resource serialization
014D	ENQ/RESERVE	Global resource serialization
014E	DEQ	Global resource serialization
014F	SYSCALL	UNIX System Services
0150	ICYDIE	DFSMS Media Manager
0151	CHANGKEY	Real storage management
1000	IARST64	Real storage management
1001	IARCP64	Real storage management
1050	CF CPU command or internal request	Reconfiguration

PSW----- Address-
`return--`:

- For PC/AUTH, supervisor control, and task management: Caller's return address if the service was entered by a branch; 0 if the service was entered by a PC instruction
- For virtual storage management: For SSRV 132 (Storage Obtain) and SSRV 133 (Storage Release), it is the ALET. For other VSM SSRVs (004, 005, 00A, 078), it is the caller's return address.
- For z/OS UNIX System Services: the syscall code.
- For real storage management (IARV64): Bytes as follows:

Byte	Description		
0	Request type identifier:		
		01	GETSTOR
		02	GETSHARED
		03	DETACH
		04	PAGEFIX
		05	PAGEUNFIX
		06	PAGEOUT
		07	DISCARDDATA
		08	PAGEIN
		09	PROTECT
		0A	SHAREMEMOBJ
		0B	CHANGEACCESS
		0C	UNPROTECT
		0D	CHANGEGUARD
		0F	GETCOMMON
		11	PCIEFIX
		12	PCIEUNFIX
		13	CHANGEATTRIBUTE
1	GETSTOR GETSHARED Request flags:		
	1...	....	COND=YES request
	.1..	....	FPROT=NO request
	..1.	....	CONTROL=AUTH request (only applies to GETSTOR)

Byte	Description		
	...1	....	SVCDUMPRGN=NO request (only applies to GETSTOR)
	....	1...	CHANGEACCESS=GLOBAL request (only applies to GETSHARED)
	....	.1..	GUARDLOC=HIGH request (only applies to GETSTOR)
	....	..1.	INORIGIN request (only applies to GETSTOR)
1	DETACH Request flags:		
	1...	....	COND=YES request
	.1..	....	MATCH=USERTOKEN request
	..1.	....	AFFINITY=SYSTEM request
	...1	....	OWNER=NO request
1	SHAREMEMOBJ Request flags:		
	1...	....	COND=YES request
	.1..	....	SVCDUMPRGN=NO request
1	CHANGEGUARD Request flags:		
	1...	....	COND=YES request
	.1..	....	TOGUARD request
	..1.	....	FROMGUARD request
1	PAGEFIX Request flags:		
	1...	....	LONG=NO request
1	DISCARDDATA Request flags:		
	1...	....	CLEAR=NO request
	.1..	....	KEEPREAL=NO request
1	CHANGEACCESS Request flags:		
	1...	....	READONLY request
	.1..	....	SHAREDWRITE request
	..1.	....	HIDDEN request
1	GETCOMMON Request flags:		
	1...	....	COND=YES request
	.1..	....	FPROT=NO request
1	PCIEFIX Request flags:		
	1...	....	LONG=NO request
1	CHANGEATTRIBUTE Request flags:		
	1...	....	SENSITIVE specified
	.1..	....	SENSITIVE=YES
	..1.	....	SENSITIVE=NO
2	Keys Used flag:		
	1...	....	KEY specified
	.1..	....	USERTOKEN specified
	..1.	....	TTOKEN specified
	...1	....	CONVERTSTART specified
	....	1...	GUARDSIZE64 request

Byte	Description		
	....	.1..	CONVERTSIZE64 request
	....	..1.	MOTKN specified
3	Miscellaneous byte: – Storage Key for GETSTOR, GETSHARED, and GETCOMMON requests – Number of ranges in range list for range list requests – 0 for all other requests		

- For CONFIGURE CPU: Bytes as follows:

Byte	Description
0 - 1	Target CPU ID
2	Internal flags
3	Bits 0 - 3: Reserved Bits 4 - 7: Direction and source **Value** **Meaning** **0** Online or Offline at MSI time **1** Online Operator request **2** Offline Operator request **3** Online WLM request **4** Offline WLM request

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

`data----`: Data. The unique trace data for each event is obtained from data areas. The areas for PC/AUTH, supervisor control, and task management are in the *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

- For an SSRV request to the PC/AUTH component: the PCTRC data area
- For an SSRV request to supervisor control: the SPTRC data area
- For an SSRV request to task management: the TMTRC data area
- For an SSRV request to virtual storage management, the data is:
  - Under Unique-1: Information input to the VSM STORAGE (OBTAIN and RELEASE) service: Bytes as follows:

Byte	Description		
0	Flags:		
	X...	....	Reserved.
	.1..	....	CALLRKY=YES was specified.
	..1.	....	AR 15 is in use.
	..0.	....	AR 15 is not in use.
	...1	....	LOC=(*nnn*,64) was specified. Storage can be backed above the bar.
	....	1...	CHECKZERO=YES was specified.
	....	0...	CHECKZERO=NO was specified explicitly, or by default.
	....	.1..	TCBADDR was specified on STORAGE OBTAIN or RELEASE.

Byte	Description		
	....	..00	OWNER=HOME was specified explicitly, or by default.
	....	..01	OWNER=PRIMARY was specified.
	....	..10	OWNER=SECONDARY was specified.
	....	..11	OWNER=SYSTEM was specified.
1	Storage key (bits 0 through 3). Ignore when CALLRKY=YES is flagged in byte 0.		
2	Subpool number.		
3	Request flags:		
	1...	....	ALET operand specified.
	.1..	....	Storage can be backed anywhere.
	..00	....	Storage must have callers residency.
	..01	....	Storage must have a 24-bit address.
	..10	....	The request is for an explicit address.
	..11	....	Storage can have a 24- or 31-bit address.
	....	1...	Maximum and minimum request.
	....	.1..	Storage must be on a page boundary.
	....	..1.	Unconditional request.
	....	...0	OBTAIN request.
	....	...1	RELEASE request.

- Under Unique-2:
  - In an SSRV trace entry for a VSM STORAGE OBTAIN or GETMAIN, one of the following:
    - The length of the storage successfully obtained
    - The maximum storage requested, if the storage was not obtained
  - In an SSRV trace entry for a VSM STORAGE RELEASE or FREEMAIN:
    - the length of the storage to be released, or zero if a subpool release was requested.
- Under Unique-3:
  - In an SSRV trace entry for a VSM STORAGE OBTAIN or GETMAIN, one of the following:
    - The address of the storage successfully obtained, if you specified address; otherwise, zero.
    - The minimum storage requested, if the storage was not obtained
  - In an SSRV trace entry for a VSM STORAGE RELEASE or FREEMAIN:
    - The address of the storage to be released.
- Under Unique-4:
  - Left 2 bytes under Unique-4: ASID of the target address space
  - Next byte under Unique-4: Reserved
  - Right byte under Unique-4:

    If the GETMAIN/FREEMAIN/STORAGE OBTAIN/STORAGE RELEASE is unconditional, an abend will be issued and the SSRV trace entry 3rd byte of Unique-4 will contain X'FF'. If the GETMAIN/FREEMAIN/STORAGE OBTAIN/STORAGE RELEASE is conditional, no abend will be issued and the SSRV trace entry 3rd byte of Unique-4 will contain the actual return code from the storage service.

- For an SSRV request to real storage management (SSID 14B), the IARV64 data is:
  - Under Unique-1

      - Return Code/Abend Code (4 bytes)
  - Under Unique-2
      - Reason Code (4 bytes)
  - Under Unique-3
      - ALET specified on the IARV64 request (4 bytes)
  - Additional Unique fields depending on the IARV64 service that follows:
      - GETSTOR/GETSHARED/GETCOMMON
          - Origin address of the memory object - 8 bytes
          - Size of the memory object - 8 bytes
          - User token - 8 bytes
      - DETACH
          - Memory object start address (for MATCH=SINGLE requests) zeroes (for MATCH=USERTOKEN requests) - 8 bytes
          - User token - 8 bytes
      - PAGEFIX, PAGEUNFIX, PAGEOUT, PAGEIN, DISCARDDATA, CHANGEACCESS, PROTECT, UNPROTECT, PCIEFIX, PCIEUNFIX
          - Address of rangelist - 8 bytes
          - VSA from 1st range list entry - 8 bytes
          - Number of blocks from 1st range list entry - 8 bytes
      - CHANGEGUARD
          - Memory object start (if ConvertStart was not specified), or convert start address (if ConvertStart was specified) - 8 bytes
          - Number of segments to be converted - 8 bytes
      - SHAREMEMOBJ
          - Range list address - 8 bytes
          - VSA from 1st range list entry - 8 bytes
          - User token - 8 bytes
- In an SSRV trace entry for global resource serialization with SSID (14C), the ISGENQ data is:
  - Under Unique-1:
      - Return address (4 bytes)
  - Under Unique-2:
      - Two bytes of flags as follows:

Byte	Description		
1	Flags:		
	01..	....	REQUEST=OBTAIN
	10..	....	REQUEST=CHANGE
	11..	....	REQUEST=RELEASE
	..1.	....	COND=YES
	...0	0...	SCOPE=STEP
	...1	0...	SCOPE=SYSTEM
	...1	1...	SCOPE=SYSTEMS
	....	.1..	CONTROL=SHARED

Byte	Description		
	....	.0..	CONTROL=EXCLUSIVE
	....	..1.	RESERVEVOLUME=YES
	....	...1	SYNCHRES=YES
2	Flags:		
	1...	....	SYNCHRES=NO
	.1..	....	An exit changed the request.
	..1.	....	WAITTYPE=ECB
	...1	....	CONTENTIONACT=Fail
	....	1...	RESLIST=YES
	....	.1..	RNLs changed scope.
	....	..1.	TEST=YES
	....	...1	RNL=NO

Note: If the last bit of byte one and the first bit of byte two are both off, the system default for SYNCHRES is used.

ISGENQ reason code (2 bytes): If a list request was provided, this field will provide the reason code for the particular list entry in error. If more than one entry is in error, it will provide the highest reason code.

– Under Unique-3:

  - Primary ASID (2 bytes)
  - The last 2 bytes may represent
    - X'FFFF' if an incomplete trace entry. An incomplete entry may be the result of a program check or an error was detected. The entry will be populated only with data we know we can trust. Therefore, some flags may only be partially filled in. To avoid confusion, having a X'FFFF' as a device number and having the reserve request bit off will inform the user the entry is incomplete. (2 bytes)
    - X'0000' if not a reserve request (2 bytes)
    - Device number if a reserve request (2 bytes)

– Under Unique-4

  - First 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.

– Under Unique-5

  - Last 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.

- In an SSRV trace entry for global resource serialization with SSID (14D) the ENQ and SSID (14E) the DEQ, the information is:

  – Under Unique-1:

    - Return address (4 bytes)

  – Under Unique-2:

    - Refer to the PEL mapping for explanation of PELLAST and PELXFLG1. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
    - 3 bytes of flags.
    - Byte 1 is:

Byte	Description		
1	Flags:		
	0...	....	Exclusive request
	.0..	....	STEP
	.1..	....	SYSTEM
	.0..	....	SYSTEM w/UCB
	.0..	1...	SYSTEMS w/UCB
	.1..	0...	SYSTEMS
	..1.	0...	An exit changed the request
	...1	1...	RNLs changed scope
	....	.000	RET=NONE
	....	.001	RET=HAVE
	....	..010	RET=CHNG
	....	.011	RET=USE
	....	.100	RET=ECB
	....	.101	RESERVED
	....	.110	RESERVED
	....	.111	RET=TEST

    - Byte 2 represents PELLAST
      - Bit 4 is ignored.
    - Byte 3 represents PELXFLG1
      - Bit 8 is ignored.
  - ENQ return code (SSID14D) (1 byte) or DEQ return code (SSID 14E) (1 byte)
    - If a list request was provided, this field will provide the Return Code for the particular list entry in error. If more than one entry is in error, it will provide the highest Return Code.
    - If ABEND, this field is in the form X'Fn' where n signifies the first hex digit of the ABEND code. For example, a X'F7' signifies a X'738' ABEND and X'F4' signifies a X'438' ABEND.
- Under Unique-3:
  - Primary ASID (2 bytes)
  - The last 2 bytes may represent
    - X'FFFF' if an incomplete trace entry. An incomplete entry may be the result of a program check or an error was detected. The entry will be populated only with data we know we can trust. Therefore, some flags may only be partially filled in. To avoid confusion, having a X'FFFF' as a device number and having the reserve request bit off will inform the user the entry is incomplete. (2 bytes)
    - X'0000' if not a reserve request (2 bytes)
    - Device number if a reserve request (2 bytes)
- Under Unique-4
  - First 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.
- Under Unique-5

  - Last 4 bytes of the QNAME (4 bytes). For a list request, this represents the first QNAME in the request.

- For an SSRV request to UNIX system services, the data is:
  - Under Unique-1

    The address of the PPRT control block
  - Under Unique-2:

    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the first parameter, otherwise the first four bytes of the first parameter, if available. Zero, if parameter not available.
  - Under Unique-3

    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the second parameter, otherwise the first four bytes of the second parameter, if available. Zero, if parameter not available.
  - Under Unique-4

    For an 8 byte parameter of an AMODE 64 caller, the low four bytes of the third parameter, otherwise the first four bytes of the third parameter, if available. Zero, if parameter not available.
- For an SSRV request to DFSMS Media Manager, the data is:
  - Under Unique-1

    The address of the IOSB control block
  - Under Unique-2

    High order word of MMRE address (Media Manager Request Element) or zero
  - Under Unique-3

    Low order word of MMRE address
  - Under Unique-4

    High order word of MMIB address (Media Manager Information Block) or zero
  - Under Unique-5

    Low order word of MMIB address
- For an SSRV request to real storage management (SSID 151), the CHANGKEY data is:
  - Under Unique-1

    The address of the first byte of the first page of the virtual storage area whose key is to be changed
  - Under Unique-2

    The address of the first byte of the last page of the virtual storage area whose key is to be changed
  - Under Unique-3
    - Byte 0: If at least one page in the range was GETMAIN assigned, bits 0 - 4 contain the original storage key and fetch protection status, and bits 5 - 7 are undefined; otherwise, byte 0 contains zero.
    - Byte 1: Bits 0 - 4 contain the new storage key and fetch protection status, and bits 5 - 7 are undefined.
    - Bytes 2 - 3: Diagnostic flags.
  - Under Unique-4

    Diagnostic flags
- In an SSRV trace entry for CONFIGURE CPU with SSID (1050), the information is:

– Under Unique-1

  Contents of an internal return code field.

– Under Unique-2

  Shows which 8-byte block of CSD_CPU_ALIVE.

– Under Unique-4 through Unique-5

  The 8-byte contents of CSD_CPU_ALIVE mask at the 8-byte block offset in Unique-2 as updated by the CF CPU command.

PSACLHS-
`desc----`: String for the current lock held, from the PSACLHS field of the PSA. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
Blank.

PSALOCAL
`desc---`: Locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
`pasd`: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD
`sasd`: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SSRV trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

SUSP trace entries

An SUSP trace entry represents a request for a suspend type lock when the requestor had to be suspended because the lock was not available.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- SUSP                 return--  rb-addr- suspndid rel-addr  psaclhs- psalocal           timestamp---------
cp--
                                                  ssrbaddr                    psaclhse
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

SUSP
Lock suspension

CD/D
Blank

PSW----- Address-
`return--`: Caller's return address

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `rb-addr-`: Address of the suspended request block (RB)
- `rel-addr`: Address associated with the type of lock suspension:
  - 0: For LOCL lock
  - ASCB address: For CML lock
  - Lockword address: For CEDQ, CLAT, CMS, and CSMF locks
- `ssrbaddr`: Address of the suspended service request block (SSRB)
- `suspndid`: Identifier of the lock suspension type: CEDQ, CLAT, CML, CMS, CSMF, or LOCL

PSACLHS-
`psaclhs-`: String for the current lock held, from the PSACLHS field of the PSA.

PSACLHSE
`psaclhse`: Extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: Locally locked address space indicator, from the PSALOCAL field of the PSA.

PASD
Blank

SASD
Blank

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

SVC, SVCE, and SVCR trace entries

These trace entries represent a supervisor event:

- An SVC trace entry is for processing of a Supervisor Call (SVC) instruction
- An SVCE trace entry is for an error during processing of an SVC instruction
- An SVCR trace entry is for return from SVC instruction processing

```
PR   ASID Wu-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- SVC    code svc-old- pswaddr-  gpr15--- gpr0---- gpr1----                               timestamp---------
cp--
                               svc-old- pswctrl-

pr   home wu-addr- SVCE   code svc-old- pswaddr-  gpr15--- gpr0---- gpr1----  psaclhs- psalocal pasd sasd timestamp---------
cp--
                               svc-old- pswctrl-  env-data                    psaclhse

pr   home wu-addr- SVCR   code ret-new- pswaddr-  gpr15--- gpr0---- gpr1----                               timestamp---------
cp--
                               ret-new- pswctrl-
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows. An asterisk before SVC, SVCE, or SVCR indicates that the SVC is for an abend (SVC D) and the abend is not for a normal end of task, that is, bit X'08' in the leftmost byte of register 1 (in the Unique-3 column) is not on.

SVC
: Supervisor call (SVC) interruption

SVCE
: SVC error

SVCR
: SVC return

CD/D
: `code`:

- For SVC and SVCE, and for SVCR when not X'FFxx': SVC number.
- For SVCR when X'FF00': completion of the system-initiated processing involved with ATTACH, LINK, SYNCH, or XCTL processing before the target routine getting control.
- For SVCR when X'FF01': initial system-initiated processing involved with XCTL processing prior to the target routine getting control.

PSW----- Address-
: The z/Architecture 128-bit old PSW, which appears on two lines:

- `pswaddr`: Two words, containing the 64-bit address portion of the PSW
- `pswctrl`: Two words, containing the 64-bit "control" portion of the PSW

The contents of this field varies, depending on the type of supervisor event and the value of `code`:

- `ret-new- pswaddr / ret-new- pswctrl`:
  - For SVC and SVCE, and for SVCR when `code` is not X'FFxx': Program status word (PSW) to receive control when the SVC is dispatched again.
  - For SVCR when `code` is X'FF00': PSW of the target routine that will get control as a result of ATTACH, LINK, SYNCH, or XCTL processing.
  - For SVCR when X'FF01': PSW of a system routine that will get control as a result of initial system processing involved with XCTL.
- `svc-old- pswaddr / svc-old- pswctrl`: SVC old PSW

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

`gpr15--- gpr0---- gpr1----`: General registers 15, 0, and 1, except for SVCE, other than SVCE D (ABEND macro), for which gpr15 contains one of the following values:

- **00000004** - issuer of SVC was in SRB mode
- **00000008** - issuer of SVC was locked
- **0000000C** - issuer of SVC was disabled
- **00000010** - issuer of SVC was in cross memory mode
- **00000014** - issuer of SVC was in EUT FRR mode
- **00000018** - issuer of SVC was in AR mode

`env-data`: Characteristics of failing environment, from the PSAMODEW field of the PSA.

PSACLHS-
`psaclhs-`: For SVCE, string for the current lock held, from the PSACLHS field of the PSA. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PSACLHSE
`psaclhse`: For SVCE, extended string for the current lock held, from the PSACLHSE field of the PSA.

PSALOCAL
`psalocal`: For SVCE, locally locked address space indicator, from the PSALOCAL field of the PSA. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

PASD
`pasd`: Primary ASID (PASID) at trace entry. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

SASD
`sasd`: Secondary ASID (SASID) at trace entry. This field will contain descriptive text for some SVC trace entries. The descriptive text will not appear in SNAP, SYSUDUMP, or SYSABEND output.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

SYNS and SYNE trace entries

These trace entries represent a synchronous I/O (zHyperLink) event:

- A SYNS trace entry represents the start of a synchronous I/O operation.
- A SYNE trace entry represents the end of a synchronous I/O operation.

```
PR   ASID Wu-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE
pr   asid wu-addr- SYNS    dev t dd op  pfid      iosbaddr len1len2 rr
                                        funchand
pr   asid wu-addr- SYNE    dev t dd op  pfid      iosbaddr len1len2 rr
                                        funchand  rc  rcq
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) related to the I/O.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

SYNS
Start of synchronous (zHyperLink) I/O operation.

SYNE
End of synchronous (zHyperLink) I/O operation. An asterisk before SYNE indicates that the operation did not complete successfully.

CD/D
`dev`: The device number associated with the request.

PSW----- Address-
The z/Architecture 128-bit old PSW, which appears on two lines:

- `t`: The type of synchronous I/O request:

  I
  An initiate type request. An initiate request is used when multiple synchronous I/O requests need to be performed in parallel. There is one initiate request issued for each synchronous I/O to be started in parallel. Normally, the system traces only the start of an initiate request (SYNS event). However, if the initiate request fails, the end of the initiate request is also traced (SYNE event).

  C
  A complete type request. A complete request is used to wait for previously initiated requests to complete. There is one complete request issued for each synchronous I/O that was initiated successfully. The system traces only the end of the complete request (SYNE event).

  O
  A standalone (only) type request. The system traces the start (SYNS event) and end (SYNE event) of a standalone request.

- `dd`: Driver identifier associated with the I/O
- `op`: Operation to be performed
- `pfid`: PCIE function identifier
- `funchand`: PCIE function handle

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `iosbaddr`: IOSB address for the I/O request.
- `len1`: First data length field
- `len2`: Second data length field
- `rr`: Record count
- `rc`: Response code
- `rcq`: Response code qualifier

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the same format as the time stamp on logrec data set records.

CP--
`cp--`: Two hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

TIME trace entries

A TIME trace entry represents a dynamic time-of-day (TOD) clock adjustment by the timer services component.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- TIME   code                    word1--- word2--- data----                   pasd sasd timestamp---------
cp--
                                                  data---- data---- data----
```

PR
`pr`: Identifier of the processor that produced the TTE.

ASID
`home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
`wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
The TTE identifier, as follows:

TIME
Timer service

CD/D
`code`: Contains a value of 1, indicating that word1 and word2 contain the amount of time that the system advances the time-of-day (TOD) clock when the TOD clock and the External Time Reference (ETR) get out of synchronization.

PSW----- Address-
`return`: Return address of the program that issued the PTRACE macro

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6
`word1, word2`: For a code value of 1, the amount of time that the system advances the TOD clock when the TOD clock and the ETR get out of synch.

PSACLHS-
Blank

PSACLHSE
Blank

PSALOCAL
Blank

PASD
`pasd`: Primary ASID (PASID) at trace entry.

SASD
`sasd`: Secondary ASID (SASID) at trace entry.

Time *Format*
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

USRn trace entries

A USRn trace entry represents processing of a PTRACE macro in an authorized program. The trace entry contains data from the macro.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Format-------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

pr   home wu-addr- USRn                return--   data---- data---- data----                   pasd sasd timestamp---------
cp--
                                                  data---- data---- data----

pr   home wu-addr- USRn                return--   idc- rbc data---- data----                   pasd sasd timestamp---------
cp--
                                                  data---- data---- data----
```

PR
: `pr`: Identifier of the processor that produced the TTE.

ASID
: `home`: Home address space identifier (ASID) associated with the TTE.

WU-Addr-
: `wu-addr-`: Address of the task control block (TCB) for the current task or the work element block (WEB).

Ident
: The TTE identifier, as follows:

  USRn
  : User event. n is a number from X'0' to X'F'.

CD/D
: Blank

PSW----- Address-
: `return`: Return address of the program that issued the PTRACE macro

Unique-1/Unique-2/Unique-3
Unique-4/Unique-5/Unique-6

- `data----`: User-defined data from the PTRACE macro
- `idc-`: PTRACE identification count
- `rbc`: Relative byte count

PSACLHS-
: This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

PSACLHSE
: Blank.

PSALOCAL
: This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

PASD
: `pasd`: Primary ASID (PASID) at trace entry. This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

SASD
: `sasd`: Secondary ASID (SASID) at trace entry. This field contains descriptive text for some SVC, SSRV, and PC trace entries. The descriptive text will not appear in SNAP/SYSUDUMP/SYSABEND output.

Time ***Format***
`timestamp---------`: Time-of-day (TOD) clock value when system trace created the trace entry. The value is in the format that was specified for TIME. If TIME was not specified, the default format is TIME(HEX).

CP--
`cp--`: Four hex digits of the processor model dependent information, which is intended to identify the physical CP that made the trace entry. CP is only provided when formatting SYSTRACE under IPCS. CP is not provided for SYSUDUMP, SYSABEND, or SNAP.

Multiple trace entries for a user event

A single user event appears in more than one trace entry if the PTRACE macro requests recording of more than 5 fullwords of trace data. For example, the following PTRACE macro requests recording of 11 fullwords of trace data:

```
PTRACE TYPE=USER3,REGS=(2,12),SAVEAREA=STANDARD
```

For this macro, system trace places three entries in the trace table. The entries contain the following:

- The first entry contains the 5 fullwords of trace data in registers 2 through 6.
- The second entry contains the 5 fullwords of trace data in registers 7 through 11.
- The third entry contains the fullword of trace data in register 12.

If the program issuing the PTRACE macro is interrupted, the three trace entries may not be consecutive in the trace table. The multiple entries contain continuation information as the data for Unique-1. The format of the continuation information is as follows:

```
nnnn hhh
```

nnnn
The PTRACE identification count. This is a hexadecimal number assigned by PTRACE to all entries for one macro processing.

hhh
The byte offset, in hexadecimal, of the next byte of trace data, which is under Unique-2. For the first entry, the offset is X'000'. For the second entry, the offset is X'014'. For the third entry, the offset is X'028'.

For example, Figure 73 on page 204 shows three trace entries from the preceding PTRACE macro.

```
PR   ASID WU-Addr- Ident  CD/D PSW----- Address-  Unique-1 Unique-2 Unique-3  PSACLHS- PSALOCAL PASD SASD Time Hex----------
CP--
                                                  Unique-4 Unique-5 Unique-6  PSACLHSE

01   000C 00AFF090 USR3                 81A007B6  003D 000 00000001 00000002                    000C 000C 9DAA507461CB3E02
cp--
                                                  00000003 00000004 00000005
01   000C 00AFF090 USR3                 81A007B6  003D 014 00000006 00000007                    000C 000C 9DAA507461CB3E02
cp--
                                                  00000008 00000009 0000000A
01   000C 00AFF090 USR3                 81A007B6  003D 028 0000000B                             000C 000C 9DAA507461CB3E02
cp--
```

Figure 73. Example: Three trace entries from the PTRACE macro

Chapter 9. Master trace

Master trace maintains a table of all recently issued system messages. This creates a log of external system activity; the other traces log internal system activity. Master trace is activated automatically at system initialization, but you can turn it on or off using the TRACE command.

Master trace can help you diagnose a problem by providing a log of the most recently issued system messages. For example, master trace output in a dump contains system messages that may be more pertinent to your problem than the usual component messages issued with a dump.

The following sections contain more information about how to request, customize, and use the master trace.

Master trace and the hardcopy log

Master trace lists the same messages that the system saves automatically and permanently in the hardcopy log, but the entries are maintained in a wraparound table, which means that master trace overwrites old entries when the table is full. You can use master trace data in a dump as a substitute for the hardcopy log when the dump contains the required messages. If the master trace table wraps and overwrites the messages related to your problem before you can request a dump, the dump will not contain useful messages.

Consider the following conditions:

- The master trace table wraps at 9 p.m.
- The system issues messages related to a problem between 9:10 and 9:20 p.m.
- The system issues an SVC dump at 9:30 p.m.

In this example, the messages pertinent to the problem will be in the master trace data in the dump, since the problem occurred between the time the trace table wrapped and the time the dump was issued.

To print the system-managed data set containing the hardcopy log, use the JESDS parameter of the OUTPUT JCL statement.

Customizing master trace

At initialization, the master scheduler sets up a master trace table of 24 kilobytes. A 24-kilobyte table holds about 336 messages, assuming an average length of 40 characters. You can change the size of the master trace table or specify that no trace table be used by changing the parameters in the SCHEDxx parmlib member.

You can also change the size of the table using the TRACE command. For example, to change the trace table size to 36 kilobytes, enter:

```
TRACE MT,36K
```

See *z/OS MVS Initialization and Tuning Reference* for more information about the SCHEDxx member.

Requesting master trace

Start, change, or stop master tracing by entering a TRACE operator command from a console with master authority. For example, to start the master tracing:

```
TRACE MT
```

To stop master tracing:

```
TRACE MT,OFF
```


You can also use the TRACE command to obtain the current status of the master trace. The system displays the status in message IEE839I. For example, to ask for the status of the trace, enter:

```
TRACE STATUS
```

In the output shown in Figure 74 on page 206, master tracing is active with a trace table of 140 kilobytes, as indicated by **MT=(ON,140K)**.

```
TRACE STATUS
IEE839I ST=(ON,0500K,01000K) AS=ON  BR=OFF EX=ON  MT=(ON,140K)
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
```

Figure 74. Example: TRACE STATUS output

If you want to check the current status of system, master, and component tracing, use the DISPLAY TRACE command. The system displays the status in message IEE843I. For example, to ask for the status of the three traces, enter:

```
DISPLAY TRACE
```

In Figure 75 on page 206, master tracing is active with a master trace table of 140 kilobytes, as indicated by MT=(ON,140K).

```
DISPLAY TRACE
IEE843I 15.17.14  TRACE DISPLAY 564
        SYSTEM STATUS INFORMATION
 ST=(ON,0500K,01500K) AS=ON  BR=OFF EX=ON  MT=(ON,140K)
 COMPONENT MODE  COMPONENT MODE  COMPONENT MODE  COMPONENT MODE
 --------------------------------------------------------------
 SYSGRS    ON    SYSSPI    OFF   SYSSMS    OFF   SYSDLF    MIN
 SYSOPS    ON    SYSXCF    ON    SYSLLA    MIN   SYSXES    ON
 SYSAPPC   ON    SYSRSM    ON    SYSAOM    OFF   SYSVLF    MIN
 CTTX      MIN
```

Figure 75. Example: DISPLAY TRACE output

See *z/OS MVS System Commands* for details about the TRACE and DISPLAY operator commands. See *z/OS MVS System Messages, Vol 7 (IEB-IEE)* for information about IEE839I and IEE843I messages.

Receiving master trace

Master trace writes trace data in the master trace table, which resides in the master scheduler address space (ASID 1). You can obtain master trace data in a stand-alone, SVC, or unformatted dump, if the dump options list includes TRT to request trace data. Format the master trace data by specifying the IPCS VERBEXIT MTRACE subcommand or using the IPCS Trace Processing selection panel. Table 34 on page 206 shows the dumps that contain master trace data.

Table 34. Summary of dumps that contain master trace data

Dump	Master trace data in the dump?
Stand-alone dump	Default
SVC dump for SDUMP or SDUMPX macro	Default
SVC dump for DUMP operator command	Default
SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP	Default
Any unformatted dump customized to exclude trace data	Yes, Request SDATA=TRT
ABEND dump to SYSABEND	Not available

Table 34. Summary of dumps that contain master trace data (continued)	
Dump	**Master trace data in the dump?**
ABEND dump to SYSMDUMP	Not available
ABEND dump to SYSUDUMP	Not available
SNAP dump	Not available

See *z/OS MVS IPCS Commands* for information about the VERBEXIT MTRACE subcommand. See *z/OS MVS IPCS User's Guide* for information about the panel.

Reading master trace data

The following topics in this section show the format of master trace entries:

- "Master trace output formatted in a dump" on page 207
- "Master trace table in storage" on page 208

Master trace output formatted in a dump

The entries in the master trace table are listed in first-in, first-out (FIFO) order, which resembles a hardcopy log. The messages might not be in chronological order because presumably the messages were not put in the master trace table in the order the messages were issued.

Figure 76 on page 207 shows an example of master trace data in a dump that is formatted by IPCS. The subcommand that is issued on the **IPCS Subcommand Entry** panel:

```
VERBEXIT MTRACE
```

```
  *** MASTER TRACE TABLE ***

  TAG  IMM DATA  |----------------------- MESSAGE DATA ----------------->|

0001 00000013  N C040000 SCOTT01  03147 21:24:22.76          00000000  $HASP468 JES2 INIT DECK PROCESSED
0001 00000013  NC0000000 SCOTT01  03147 21:24:22.77 INTERNAL 00000290  REPLY 0002,N1  AUTH=(NET=YES),NAME=SCOTT01
0001 00000013  W C040000 SCOTT01  03147 21:24:22.76          00000000 *0002 $HASP469 REPLY PARAMETER STATEMENT, CANCEL, OR END
0001 00000009  NRC040000 SCOTT01  03147 21:24:22.77 INTERNAL 00000090  IEE600I REPLY TO 0002 IS;N1  AUTH=(NET=YES),NAME=SCOTT01
0001 00000013  N C040000 SCOTT01  03147 21:24:22.77          00000290  $HASP466 CONSOLE   STMT   126  N1  AUTH=(NET=YES),
               NAME=SCOTT01
0001 00000013  N C040000 SCOTT01  03147 21:24:22.77          00000090  $HASP826 NODE(1)      NAME=SCOTT01,AUTH=(DEVICE=YES,
               JOB=YES,
0001 00000013  N C040000 SCOTT01  03147 21:24:22.77          00000090  $HASP826              NET=YES,SYSTEM=YES),TRANSMIT=BOTH,
0001 00000013  N C040000 SCOTT01  03147 21:24:22.77          00000090  $HASP826              RECEIVE=BOTH,HOLD=NONE,PENCRYPT=NO,
0001 00000013  N C040000 SCOTT01  03147 21:24:22.78          00000090  $HASP826              ENDNODE=NO,REST=0,SENTREST=ACCEPT,
0001 00000013  N C040000 SCOTT01  03147 21:24:22.78          00000090  $HASP826              COMPACT=0,LINE=0,LOGMODE=,LOGON=0,
0001 00000013  N C040000 SCOTT01  03147 21:24:22.78          00000090  $HASP826              PASSWORD=(VERIFY=(NOTSET),
               SEND=(NOTSET)),
0001 00000013  N C040000 SCOTT01  03147 21:24:22.78          00000090  $HASP826              PATHMGR=YES,PRIVATE=NO,SUBNET=,
               TRACE=NO
0001 00000013  NC0000000 SCOTT01  03147 21:24:22.78 INTERNAL 00000290  REPLY 0003,END
0001 00000013  W C040000 SCOTT01  03147 21:24:22.78          00000000 *0003 $HASP469 REPLY PARAMETER STATEMENT, CANCEL, OR END
0001 00000009  NRC040000 SCOTT01  03147 21:24:22.78 INTERNAL 00000090  IEE600I REPLY TO 0003 IS;END
0001 00000013  N C040000 SCOTT01  03147 21:24:22.78          00000290  $HASP466 CONSOLE    STMT   127   END
0001 00000013  N 0000000 SCOTT01  03147 21:24:22.79          00000290  IEF196I IEF285I   CONSOLE.OSV142.PARMLIB
                    KEPT
0001 00000013  N 0000000 SCOTT01  03147 21:24:22.79          00000290  IEF196I IEF285I   VOL SER NOS= D72666.
0001 00000013  N 0000000 SCOTT01  03147 21:24:22.79          00000290  IEF196I IEF285I   SYS1.PARMLIB
                    KEPT
0001 00000013  N 0000000 SCOTT01  03147 21:24:22.79          00000290  IEF196I IEF285I   VOL SER NOS= D72666.
```

Figure 76. Example of master trace data in a dump formatted by IPCS

The meaning of the highlighted text in the preceding example is as follows:

TAG
A halfword that contains the identity of the caller. **TAG** can be:

Tag
Caller

000
Reserved

001
WTO SVC

002
Master scheduler

003
Trace command

Current identifiers are defined in the macro, IEZMTPRM, which maps the parameter list.

IMM DATA
A fullword of immediate data, consisting of the 32 bits defined by the caller. The significance of the immediate data is defined by the caller.

MESSAGE DATA
The message. If a problem occurs during processing, the line that follows the message indicates the problem.

Master trace table in storage

This topic describes master trace data as it is recorded in the master trace table in the master scheduler address space. You can use this information to write your own formatting or analysis routines for master trace information. Master trace places entries in FIFO order. Thus, a current entry is in front of the older entries. When the table is full, master trace wraps, and resumes recording entries at the end of the table.

Note that the messages may not be in chronological order because the messages may not be put in the master trace table in the order in which they are issued.

You can locate the master trace table from the communication vector table (CVT), as shown in Table 35 on page 208.

Table 35. How to locate master trace table from CVT

At the location:	In field:	Find the following address:
CVT+X'94'	CVTMSER	IEEBASEA (master scheduler resident data area)
IEEBASEA+ X'8C'	BAMTTBL	Start of the master trace table

The unformatted master trace table in the master scheduler address space contains a header and, for each message logged in the table, an entry. The following two topics show the fields in the header and an entry. The master trace table header and entries are mapped by the MTT mappings in the IEEZB806 macro, which can be found in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Header in the master trace table

<table>
<tr><th colspan="2">TABLE ID</th><th>CURRENT</th><th>START</th><th>END</th></tr>
<tr><td>SUBPOOL</td><td>LENGTH</td><td colspan="3">WRAP TIME</td></tr>
<tr><td colspan="2">WRAP POINT</td><td>RESERVED1</td><td>DATA LENGTH</td><td>RESERVED2...</td></tr>
</table>

TABLE ID
A fullword field containing *MTT*. MTT is an eye-catcher to mark the beginning of the master trace table.

CURRENT
A fullword field containing the address of the current (most recently stored) entry.

START
A fullword field containing the address of the first byte of the trace area.

END
A fullword field containing the address of the first byte beyond the end of the trace area.

SUBPOOL
A one-byte field containing the number of the subpool in which this table resides.

LENGTH
A three-byte field containing the length, in bytes, of the table header and the area containing the entries. This length is the default table size or the size specified on the TRACE command.

WRAP TIME
A double word field containing a time, either when the table was initialized or when the last table wrap occurred. The time is in the **XXHH:MM:SS.T** form:

XX
Possible values can be IT or WT.

IT
Indicates the time that the table was initialized.

WT
Indicates the time the table last wrapped.

HH
hours

MM
minutes

SS
seconds

T
tenths of a second

WRAP POINT
A fullword field containing the address of the first byte of the last entry stored before the most recent table wrap.

Note: This address is initialized to zero and remains zero until the first table wrap.

RESERVED1
A fullword reserved field.

DATA LENGTH
A fullword field containing the length, in bytes, of the data area part of the table.

RESERVED2
A 21-word field.

Entry in the master trace table

FLAGS	TAG	IMM DATA	LEN	CALLER-PASSED DATA

Entry header
10-byte header for the entry.

FLAGS
A halfword that contains the flags that are set by the caller in the parameter list that is passed to master trace.

TAG
A halfword that contains the identity of the caller. **TAG** can be:

Tag
: **Caller**

0000
: Reserved

0001
: WTO SVC

0002
: Master scheduler

0003
: Trace command

Current identifiers are defined in the macro, IEZMTPRM, which maps the parameter list.

IMM DATA
: A fullword that contains 32 bits defined by the caller. Master trace stores these bits in the table without checking them for validity. The significance of IMMEDIATE DATA is defined by the caller; likely values are a counter, a control block address, or flags describing the passed data.

LEN
: A halfword that contains the length of the caller-passed data.

CALLER-PASSED DATA
: A variable-length field that contains the data that is provided by the caller.

The master trace table entries vary in length. If the caller specifies the length of the caller-passed data as zero, the entry in the master trace table consists of only the 10-byte header.

Chapter 10. The Generalized Trace Facility (GTF)

The generalized trace facility (GTF) is a service aid you can use to record and diagnose system and program problems. GTF is part of the MVS system product, and you must explicitly activate it by entering a START GTF command.

Use GTF to record a variety of system events and program events on all of the processors in your installation. If you use the IBM-supplied defaults, GTF lists many of the events that system trace lists, showing minimal data about them. However, because GTF uses more resources and processor time than system trace, IBM recommends that you use GTF when you experience a problem, selecting one or two events that you think might point to the source of your problem. This will give you detailed information that can help you diagnose the problem. You can trace combinations of events, specific incidences of one type of event, or user-defined program events that the GTRACE macro generates. For example, you can trace:

- Channel programs and associated data for start and resume subchannel operations, in combination with I/O interruptions
- I/O interruptions on one particular device
- System recovery routine operations

The events that GTF traces are specified as options in a parmlib member. You can use the IBM supplied parmlib member or provide your own. Details of GTF operation, which include storage that is needed, where output goes, and recovery for GTF are defined in a cataloged procedure in SYS1.PROCLIB.

GTF can trace system and program events both above and below 16 megabytes. For each event it traces, GTF produces trace records as its output. You can have GTF direct this output to one of the following places:

- A trace table in virtual storage.
- A data set on a tape or direct access storage device (DASD).

Choose a trace table for your GTF output when maintaining good system performance is very important to your installation. The trace table cannot contain as much GTF trace data as a data set, but will not impact performance as much as a data set because there is no I/O overhead.

Choose a data set or sets when you want to collect more data than will fit in a trace table. Writing trace data to a data set does involve I/O overhead, so choosing this option will impact system performance more than a trace table.

GTF can use only one table in virtual storage, but can use up to 16 data sets. If you specify more than one data set, all of them must reside on devices of the same class, tape, or DASD.

Other components, such as OPEN/CLOSE/EOV and VSAM have special GTF support. See *z/OS DFSMSdfp Diagnosis* for complete details.

GTF and IPCS

You can use IPCS to merge, format, display, and print GTF output. See *z/OS MVS IPCS User's Guide* and *z/OS MVS IPCS Commands* for information about the COPYTRC, GTFTRACE, and MERGE subcommands, and the trace processing option of the IPCS dialog.

GTF and the GTRACE macro

You can use GTF in combination with the GTRACE macro, provided you activate GTF with TRACE=USR. Then, your programs can issue GTRACE macros to generate trace records, which GTF can store in the trace table. See *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about coding the GTRACE macro.

GTF and system trace

You can use GTF in combination with system trace. System trace records predetermined system events, and provides minimal details about each event. Supplement system trace information by selecting specific GTF options to provide more detailed information about system and user events. For further information about system trace, see Chapter 8, “System trace,” on page 153.

Using IBM defaults for GTF

IBM supplies both a SYS1.PARMLIB (also called parmlib) member that contains predefined GTF trace options and a cataloged procedure for GTF, should you want to use IBM's defaults for GTF operation. You can override some of the default options by specifying certain parameters on the START command that activates GTF.

The IBM-supplied parmlib member of GTF trace options

IBM supplies the GTFPARM parmlib member, which contains the GTF trace options shown in Figure 77 on page 212

```
TRACE=SYSM,USR,TRC,DSP,PCI,SRM
```

Figure 77. IBM-Supplied GTFPARM parmlib member

Briefly, the following options request GTF traces. For more details about these trace options, see “GTF trace options” on page 227.

SYSM
: Selected system events

USR
: User data that the GTRACE macro passes to GTF

TRC
: Trace events associated with GTF

DSP
: Dispatchable units of work

PCI
: Program-controlled I/O interruptions

SRM
: Trace data associated with the system resource manager (SRM)

The IBM-supplied cataloged procedure

IBM supplies the GTF cataloged procedure, which resides in SYS1.PROCLIB. The cataloged procedure defines GTF operation, output location, recovery facilities, trace output data sets, and the parmlib member that contains GTF options and defaults. Figure 78 on page 212 illustrates the content of the IBM supplied cataloged procedure.

```
//GTFNEW PROC MEMBER=GTFPARM
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',
// TIME=1440,REGION=4M
//IEFRDER DD DSNAME=SYS1.TRACE,UNIT=SYSDA,SPACE=(TRK,20),
// DISP=(NEW,KEEP)
//SYSLIB DD DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
```

Figure 78. IBM-Supplied GTF Cataloged Procedure

The statements in this cataloged procedure are:

PROC
Defines the GTF cataloged procedure.

EXEC

PGM=AHLGTF
Calls for the system to run program AHLGTF.

PARM='MODE=EXT,DEBUG=NO,TIME=YES',
The parameters selected specify that GTF direct trace data to a data set on tape or DASD, attempt recovery if it encounters an error, and give every trace record a timestamp. See the explanation for the EXEC statement under "Setting up a cataloged procedure" on page 214 for detailed information.

TIME=1440
The amount of time, in seconds, that GTF remains active.

REGION=4M
Specifies the maximum size of the storage that GTF can use.

IEFRDER DD
Defines the trace output data set:

- The common data set name is SYS1.TRACE.
- The data set can reside on a DASD if:
  - The primary space amount is large enough for the data set to contain at least 20 physical blocks. After completely filling the primary space, GTF will overlay previously written records with new trace records, starting at the beginning of the output data set.
  - The set is in basic format, large format sequential, or does not have a DSORG value.
  - The set is a VSAM linear data set with a control interval size (CISIZE) of 32 KB.
- The data set can reside on a tape if the tape has IBM standard labels or no labels. If GTF fills the volume, the system requests another volume to continue writing. GTF does not overlay the trace records of this GTF written instance.

Restrictions to interactions with installation SMS routines:

- A VSAM linear data set with either an extended format or conventional format with a control interval size (CISIZE) of 32 K can be substituted.
- Neither extended sequential nor VSAM data sets, other than linear data sets with the required CISIZE, should be used.

Note: Wrapping is not supported for Extended Format Sequential data sets, which are treated as NOWRAP even if NOWRAP is not specified.

SYSLIB DD
Defines the IBM-supplied GTFPARM parmlib member that contains GTF trace options and their default values. Multiple instances of GTF can be active at the same time. Each instance of GTF requires a unique trace data set. The default trace data set in the cataloged procedure can be overridden by specifying a different data set on the START command, or by setting up a cataloged procedure for each instance of GTF to be activated.

Customizing GTF

You can customize GTF to the needs of your installation by either overriding IBM's defaults through the START GTF command, or providing your own parmlib member and cataloged procedure for GTF. Customize GTF in one of the following ways:

- Predefine the GTF trace options in a parmlib or data set member. See "Defining GTF trace options" on page 214.
- Set up a cataloged procedure. See "Setting up a cataloged procedure" on page 214.

- Override the defaults in the IBM supplied GTF cataloged procedure using the START command. See "Using the START command to invoke GTF" on page 220.
- Determine how much storage GTF needs for the trace options you choose. See "Determining GTF's storage requirements" on page 218.
- Specify trace options directly through the console after entering the START command. See "Specifying or changing GTF trace options through system prompting" on page 221.

Defining GTF trace options

If you supply your own parmlib member or data set containing GTF trace options, you can select any of the options listed in "GTF trace options" on page 227. Each instance of GTF can be activated with the same or different set of options.

The member containing predefined trace options does not have to reside in the parmlib member. GTF will accept any data set specified in the SYSLIB DD statement of the cataloged procedure, or in the START command, as long as that data set's attributes are compatible with those of SYS1.PARMLIB.

Setting up a cataloged procedure

Set up your own GTF cataloged procedure when you want to control details of GTF operation such as:

- Amount of storage needed for tracing
- Recovery for GTF
- Number and type of trace output data sets.

If you choose to supply your own cataloged procedure, include the following statements:

PROC
Defines your cataloged procedure.

EXEC

PGM=AHLGTF
Calls for the system to run program AHLGTF.

PARM='*parm, parm...*'
Options specified on the PARM parameter specify where GTF writes trace data and the amount of storage needed for GTF to collect and save trace data in various memory dump types. *parm* can be any of the following:

```
MODE={INT|EXT|DEFER}
SADMP={nnnnnnK|nnnnnnM|40K}
SDUMP={nnnnnnK|nnnnnnM|40K}
NOPROMPT
NOWRAP
ABDUMP={nnnnnnK|nnnnnnM|0K}
BLOK={nnnnn|nnnnnK|nnnnnM|40K}
SIZE = {nnnnnnK|nnnnnnM|1024K}
TIME=YES
DEBUG={YES|NO}
```

MODE={INT|EXT|DEFER}
Defines where GTF writes the trace data. MODE=INT directs data to a trace table in virtual storage, and MODE=EXT directs data to a data set on tape or DASD.

If MODE=INT, each instance of GTF directs the trace data to a separate trace table in virtual storage, and ignores any DD statements that define GTF output data sets. Choose this option when it is important to you to maintain good system performance while GTF runs. The trace table cannot contain as much GTF trace data as a data set, but does not impact performance as much as a data set because there is no I/O overhead.

If MODE=EXT or MODE=DEFER, each instance of GTF directs the output to a separate trace data set defined by GTFOUTxx or IEFRDER DD statements. MODE=EXT is the default value.

Choose MODE=EXT or MODE=DEFER when you want to collect more data than fits in a trace table. Writing trace data to a data set does involve I/O overhead, so choosing one of these options impact system performance more than MODE=INT.

MODE=DEFER places the trace data in the GTF address space until you enter the STOP GTF command. Every instance of GTF runs in its own address space. During GTF end processing, each instance of GTF transfers the data from its address space to the output data set.

The amount of data transferred for MODE=EXT or MODE=DEFER is one of the following:

- The default amount
- The amount specified on the SADMP|SA keyword

When the trace output medium is full, GTF continues as follows:

- **Int:** GTF resumes recording at the beginning of the buffer. Thus, GTF writes over earlier trace data.
- **Direct access:** GTF resumes recording at the beginning of the data set when the primary allocation is full. Thus, GTF writes over earlier trace data.
- **Tape**: GTF writes an end-of-file record. The tape is rewound and unloaded, then a new volume is mounted on the tape unit. If GTF has only one tape data set and only one unit for the data set, GTF does not write trace records while the tape is unavailable, thus losing trace data.

  GTF can write to multiple tape units in two ways:

  – Multiple GTFOUTxx DD statements can specify tape data sets. When GTF fills one data set, it changes to the next data set.
  – The IEFRDER DD statement can specify two tape units. In this case, GTF resumes writing the current trace data on the other unit, while rewinding and unloading the full volume.

SADMP|SA=nnnnnnK|nnnnM|<u>40K</u>}

Specifies the amount of storage needed to save GTF trace data for stand-alone memory dumps. Specify the amount of storage in terms of either kilobytes (K) or megabytes (M). The minimum amount is 40 K, and the maximum is 2048 M minus 400 K, or 2096752 K. GTF rounds up the amount to the block size boundaries for DASD data sets, or 32 K boundaries for tape data sets or internal mode. The default value for this parameter is 40 K (rounded up to the correct boundary).

SDUMP|SD={nnnnnnK|nnnnM|<u>40K</u>}

Specifies the amount of storage needed to save GTF trace data for SVC memory dumps. Specify the amount of storage in terms of either kilobytes (K) or megabytes (M). The minimum amount is zero, and the maximum cannot exceed the maximum amount of storage defined by the SADMP parameter. GTF rounds up the amount to the block size boundaries for DASD data sets, or 32 K boundaries for tape data sets or internal mode. The default value for this parameter is 40 K (rounded up to the correct boundary).

NOPROMPT|NP

If specified, indicates that you are not prompted to specify trace options. Message AHL125A and AHL100A are not issued. Use this parameter when you have a parmlib member set up with the wanted GTF options and you want to avoid multiple replies in a sysplex environment.

NOWRAP|NW

If specified, indicates to the system to stop writing to one or more data sets when the data sets are full. With NOWRAP, the system uses the primary and secondary extents of one or more data sets. Default behavior to wrap the data is used if NW is not specified. In this case, the system uses the primary extent or extents.

ABDUMP|AB={nnnnnnK|nnnnM|<u>0K</u>}

Specifies the amount of GTF trace data to be formatted in an ABEND or SNAP memory dump. Specify the amount of trace data in terms of either kilobytes (K) or megabytes (M). The minimum amount is zero, and the maximum cannot exceed the maximum amount of storage defined by the SADMP parameter. GTF rounds up the amount to the block size boundaries for DASD data sets, or 64 K boundaries for tape data sets or internal mode. The default value for this parameter is 0 K, which means that no GTF data appears in ABEND or SNAP memory dumps.

For ABEND or SNAP memory dumps. GTF formats only those records that are directly associated with the failing address space. GTF does not format the channel program trace data associated with the failing address space.

BLOK={nnnnn|nnnnnK|nnnnnM|40K}
Specifies the amount of virtual storage (E)CSA that GTF uses to collect trace data. Specify this storage amount in 4096-byte pages (nnnnn), or in bytes (nnnnnK or nnnnnM). The maximum amount is 99999; the default is 40 K. If the amount is less than 40 K, GTF uses the default.

SIZE={nnnnnK|nnnnnM|1024K}
Specifies the size of the buffers. Specify this amount in bytes (nnnnnK or nnnnnM). The range for the size keyword is 1 M to 2046 M. The maximum amount is 2046 M; the default is 1024 K. If the amount is less than 1024 K, GTF uses the default.

TIME=YES

Specifies that every GTF trace record have a timestamp, as well as the block timestamp associated with every block of data. The timestamp is the eight-byte time of day (TOD) clock value at the local time in which GTF puts the record into the trace table. GTF does not accept TIME=NO; all output records have timestamps. Local time is calculated by using a time zone offset that GTF establishes at the time that the trace starts. If the system time zone offset is changed during tracing, for example in response to Daylight Saving Time going into effect, local times that are formatted by GTF do not correspond with system times afterward.

When you use IPCS to format and print the trace records, a timestamp record follows each trace record. You can use these timestamp records to calculate the elapsed time between trace entries. The timestamp record is described in "Time stamp records" on page 247.

DEBUG={YES|NO}
Specifies whether GTF should attempt recovery after it encounters an error. If DEBUG=YES, GTF does not attempt any recovery. Instead, GTF will issue an error message and end after encountering any error so that the contents of the trace table immediately before the error remain intact.

If DEBUG=NO, which is the default, GTF does the following:

- For errors in GTF processing, GTF continues processing after doing one or more of the following:
  - Flagging the trace record or trace record field that is associated with the error.
  - Issuing a message to the console to inform that an error occurred.
  - Suppressing the error or function in which the error occurred.
- For errors that do not occur in GTF itself, GTF ends abnormally. If GTF stops processing that does not cause any other task to also stop.

TIME=1440
Specifies unlimited processor time for GTF.

REGION=nnnnK|M
Specifies the maximum size of the storage that GTF can use. If the REGION parameter is omitted, then the GTF program defaults to using the REGION value that was specified during the system installation. For more information about determining the value for REGION, see "Determining GTF's storage requirements" on page 218. For more information about the REGION parameter, see *z/OS MVS JCL Reference* .

IEFRDER DD or GTFOUTxx DD
Defines the trace output data set or data sets. This statement is required only if you do one of the following:

- Specify MODE=EXT or MODE=DEFER
- Use the default MODE=EXT

IEFRDER DD can be used, but does not have to be used, for one trace output data set. Additional data sets must be defined on GTFOUTxx DD statements, where xx are one or two characters that are valid in DDNAMES. The trace output data set or data sets must be unique and cannot be shared across active instances of GTF. For more information, see "Guidelines for defining GTF trace output data sets in a cataloged procedure" on page 217.

SYSLIB DD

Optionally include a SYSLIB DD statement to define the IBM-supplied member, or the installation-supplied member that contains GTF trace options. If the member exists, GTF uses the options in that member. If the member does not exist, GTF issues an error message and stop.

If you code a procedure that does not contain a SYSLIB DD statement, GTF issues message AHL100A to prompt for options after the START GTF command. In response, you can supply the wanted trace options through the console. See "Specifying or changing GTF trace options through system prompting" on page 221 for examples of specifying options through the console.

Guidelines for defining GTF trace output data sets in a cataloged procedure

The trace output data sets must be specific to each instance of GTF and can be defined in the cataloged procedure. Each instance of GTF to be started must have a separate cataloged procedure, or if the same cataloged procedure is used, then a different trace data set must be supplied with the START command

Use the following guidelines for specifying trace output data sets on the IEFRDER DD or GTFOUTxx DD statements:

- You can define up to 16 output data sets for GTF to use. If you define more than 16 data sets, GTF will accept the first 16 and ignore the rest.
- If GTF cannot open all of the data sets, it issues a message that identifies those that are unopened, and continues processing with those that are open.
- Do not specify the RLSE option while using the SPACE parameter because the output data sets are opened and closed more than once while GTF runs.

  Note: If the GTF trace output resides on an SMS volume, you should ensure that the SMS management class does not allow partial release.
- Do not request secondary extents for trace data sets. GTF will only use the first extent.

  To obtain the maximum degree of control over the number of trace entries for which space is allocated, specify space allocation in blocks, using a block length that matches the BLKSIZE of your trace data set. Do not specify any secondary space. Use the CONTIG option to request contiguous blocks. For example, if your BLKSIZE is 8192, code the SPACE keyword as follows:

  ```
  SPACE=(8192,(500,0),,CONTIG)
  ```
- All data sets must be in the same device class: either DASD or tape, but not both. If you mix device classes, GTF will ignore the tapes and use only DASD. However, the data sets can have different device types; for example, you can mix 3380 and 3390 device types.
- If the data set is on a DASD:
  - The data set can be sequential, basic format, large format, or extended format. Basic format is limited to 65535 tracks on the volume. Large format is limited to 16,777,215 tracks per volume. The extended format limit is much higher. Extended format data sets can be compressed format, striped, or both. You cannot use the WRAP option in compressed format.
  - The data set must be linear if it is VSAM and must have a 32 KB control interval size. The data set might be extended format, striped, and might have the extended addressing option to allow it to be larger than 4 GB.
- When NOWRAP is not specified, the primary space request is consulted and only control intervals contained within that space are used. Unlike non-VSAM data sets used for the default WRAP processing,

the primary space does not have to be satisfied using a single extent. A data set must be empty or have been defined with the REUSE attribute. If neither of the two conditions exists, GTF rejects the use of the data set.

When NOWRAP is not specified, the primary space request is consulted and only control intervals contained within that space are used. Unlike non-VSAM data sets used for the default WRAP processing, the primary space does not have to be satisfied using a single extent. A data set must be empty or have been defined with the REUSE attribute. If neither of the two conditions exists, GTF rejects the use of the data set.

When NOWRAP processing is requested, control intervals are filled until GTF is stopped or the data set is full.

Note: GTF and CTRACE accept a single VSAM linear data set as output. VSAM's support for striping can increase data rate without the complexity that is associated with the use of distinct data sets.

- GTF and CTRACE support placement of NOWRAP traces in cylinder-managed space. WRAP traces placed in VSAM linear data sets can reside in cylinder-managed space too. WRAP traces in non-VSAM data sets cannot be placed in large format data sets, extended format data sets, or cylinder-managed space.
- To ensure the most efficient GTF processing, do not specify any particular block size for the output data set or data sets in either:
  - The cataloged procedure for GTF
  - The JCL, TSO/E commands, or interactive system productivity facility/program development facility (ISPF/PDF) panels that you might use to preallocate the data sets.

  The system computes an optimal block size when it opens each data set.

  Note: If you want GTF to use an unlabeled tape as the output data set, you must specify the logical record length and block size when you allocate that data set.
- If you define more than one data set, you should ensure that the number of paths to the data sets equals the number of data sets.
- You can specify the number of channel programs for each output data set using the NCP parameter on each DD statement. The NCP value determines the rate at which GTF transfers data to the output data sets. For example, if you want to transfer data to your data sets at a rate of 25 buffers per second and you have 5 data sets, you will need to specify an NCP value of 5. GTF then transfers data to the 5 data sets at a rate of 5 buffers per second per data set for a total rate of 25 buffers per second.

  The maximum value for NCP is 255. If you do not specify a value for NCP, or if you specify a value less than four, GTF will use the following default values:
  - For tape: four
  - For DASD: the number of output blocks per track, which is multiplied by four.
- If, when you enter the START command, you override any of the DD statements for multiple output data sets, you must use symbolic parameters in those DD statements. See "Using the START command to invoke GTF" on page 220 for more information.

Determining GTF's storage requirements

The storage that GTF requires depends on the trace options that you choose. Modern systems consume large amounts of virtual storage, and GTF is no exception. Use the REGION parameter to specify at least 20 MB of available storage for GTF. Then, if you need to impose constraints on the private storage of the GTF address space, use the REGION parameter to restrict GTF's usage. If the REGION parameter is omitted, then the GTF program defaults to using the REGION value that was specified during the system installation. For general information about the REGION parameter, see *z/OS MVS JCL Reference* .

After determining the GTF trace options, use Figure 79 on page 219 to determine several storage requirements for either your cataloged procedure or the START GTF command. There are several types of storage to calculate:

- Extended pageable link pack area (EPLPA)
- System queue area (SQA)
- Extended common service area (ECSA)
- Region storage

Use the formulas in Figure 79 on page 219 to calculate the amount of storage needed for each storage type. Then add them all together to arrive at the final figure to specify on the REGION parameter. For information about the options mentioned in the figure, see "GTF trace options" on page 227.

Extended Pageable Link Pack Area

Fix = Opt + Prmpt + 8K

Fix: Fixed storage in pageable EPLPA while GTF is active.

Opt: Sum of storage required for each GTF option specified. See the table below to calculate OPT.

Prmpt: Optional additional 1.5K if any prompting options specified.

8K: 8K required for services.

Option	Size Required
SYSM	4K
SYS with DSP and/or SRM and/or RNIO	7K
SYS, SYSP	18K
PI, DSP, PIP	2.5K
EXT	2K
IO, IOP, SIO, SIOP, SSCH, SSCHP	6K
SVC, SVCP	10K
SRM, RR, RNIO	3K
SLIP	8K
USR, USRP	1.5K
PCI, TRC	No Requirement
CCW, CCWP	9.3K

Notes:

1. When you specify more than one event from a line, the size requirement is the same as if you specified only one option. For example, DSP and PI require 2.5K.
2. For the maximum storage requirement round up the storage requirement for each option you specified to the nearest 4K boundary.
3. For the minimum storage requirement, round up the 'FIX' value to the nearest 4K boundary.

Example:

1) Options = PIP, DSP, SLIP
   Fix = 10.5 + 1.5 + 8 = 20K minimum or
   = 12 + 1.5 + 8 = 21.5 = 24K maximum
2) Options = SYSM, SRM USR, TRC
   Fix = 8.5 + 0 + 8K = 16.5 = 20K minimum or
   = 12 + 0 + 8K = 20K maximum

System Queue Area

SQA = 16500 + REG + SAVE + CBLOC

SQA: System Queue Area storage requirement.

REG: 232 bytes per processor are required for register save areas, regardless of whether or not GTF is active.

SAVE: 1352 bytes per processor are required for save/work areas when GTF is active.

CBLOC: 1700-2200 bytes are needed for control blocks when GTF is active.

Notes:

1. When you specify PCI and either CCW or CCWP, GTF requires the following additional SQA storage:

   16 + 1600 * (value of PCITAB in bytes)

2. When you specify either CCW or CCWP, GTF uses 4096 additional bytes of the SQA for each processor.
3. When you specify USRP, GTF uses 4096 additional bytes ofthe SQA for each processor.

Extended Common Service Area (CSA)

ESQA = N

N: Approximately 4500 times the number of blocks specified on the BLOCK= keyword parameter of the GTF START command.

The default is 45056 bytes.

Region Storage

SUBPOOL: GTF uses a default of 1031 kilobytes of storage for true data.

REGION: GTF requires a minimum of an 800K virtual region to run.

Figure 79. GTF storage requirements

Starting GTF

Multiple instances of GTF can be active in a system at the same time. When you activate GTF, each instance operates as a system task, in its own address space. The only way to activate GTF is to enter a START GTF command from a console with master authority. Using this command, select either IBM's procedure or your cataloged procedure for GTF. The cataloged procedure defines GTF operation; you can accept the defaults that the procedure establishes, or change the defaults by specifying certain parameters on the START GTF command.

Because GTF sends messages to a console with master authority, enter the command only on a console that is eligible to be a console with master authority. Otherwise, you cannot view the messages from GTF that verify trace options and other operating information.

Each instance of GTF can be assigned a unique identifier that is specified on the START GTF command after the GTF keyword. This will allow you to recognize and control specific instances of GTF. If a unique identifier is not specified, the operating system assigns the default, which is the device number of the device where the trace data set resides. See the example in the topic "Starting GTF with trace output to an existing data set on tape" on page 223 for an instance of GTF with the default identifier.

Using the START command to invoke GTF

The START command, without any parameters other than the IBM-supplied procedure name and an identifier, uses the defaults of the cataloged procedure. If that source JCL contains a DD statement for the data set member of predefined trace options, GTF will issue a message that lists those options, and will allow you to override them. Otherwise, GTF will prompt you to specify trace options directly through the console. See "Specifying or changing GTF trace options through system prompting" on page 221 for more information.

To invoke GTF, enter the following START command. For information about the command and parameters you can use to change the GTF cataloged procedure, see *z/OS MVS System Commands*.

```
{START|S}{GTF|membername}[.identifier]
```

Guidelines for overriding JCL statements in the GTF cataloged procedure

You can override the parameters of only one output data set using the **keyword=option** parameter on the START command. If you have defined more than one output data set, and you used IEFRDER as the DDNAME for one of the DD statements, the keywords specified on the START command will override the attributes of the data set that IEFRDER defines. If you want to alter the attributes of another data set, or more than one data set, you must:

- Use symbolic parameters in the JCL DD statements for those attributes you want to change. You cannot use DD statement keywords as symbolic parameters; for example, you cannot code UNIT=&UNIT;
- Assign values to the symbolic parameters in the EXEC or PROC statements in the cataloged procedure.
- Specify keywords in the START command to override the symbolic parameter values specified on the EXEC or PROC statements.

Examples of overriding the JCL statements in the GTF cataloged procedure

The following shows examples of setting up a cataloged procedure when you want to override JCL statements in the procedure using the **keyword=option** parameter on the START command. Note that the DD statement parameters in both of the following procedures are for example only; the needs of your installation might require you to provide DD parameters in addition to, or other than, DSNAME, UNIT, and DISP.

If you want to alter just one data set using the START command, your cataloged procedure could look like Figure 80 on page 221.

```
//GTFABC    PROC   MEMBER=GTFPARM
//IEFPROC   EXEC   PGM=AHLGTF,REGION=2880K,TIME=1440,
//                 PARM=('MODE=EXT,DEBUG=NO')
//IEFRDER   DD     DSNAME=SYS1.GTFTRC,UNIT=SYSDA,
//                 SPACE=(4096,20),DISP=(NEW,KEEP)
//SYSLIB    DD     DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//GTFOUT1   DD     DSNAME=SYS1.TRACE1,UNIT=SYSDA,DISP=(NEW,KEEP)
//GTFOUT2   DD     DSNAME=SYS1.TRACE2,UNIT=SYSDA,DISP=(NEW,KEEP)
//GTFOUT3   DD     DSNAME=SYS1.TRACE3,UNIT=SYSDA,DISP=(NEW,KEEP)
```

Figure 80. Example: altering one data set

Enter **START GTFABC,,,UNIT=TAPE**, to alter only the data set that IEFRDER defines.

If you want to alter the attributes of more than one data set with the START command, use the JCL statements in Figure 80 on page 221 in your cataloged procedure.

```
//GTFABC  PROC MEMBER=GTFPARM,NAME1='SYS1.TRACE1',
//             NAME2='SYS1.TRACE2',NAME3='SYS1.TRACE3',
//IEFPROC EXEC PGM=AHLGTF,REGION=2880K,TIME=1440,
//             DEVICE='SYSDA',DSPS='OLD'
//          PARM=('MODE=EXT,DEBUG=NO')
//SYSLIB  DD   DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//GTFOUT1 DD   DSNAME=&NAME1,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT2 DD   DSNAME=&NAME2,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT3 DD   DSNAME=&NAME3,UNIT=&DEVICE,DISP=&DSPS;
```

Figure 81. Example: Altering More Than One Data Set

Enter **START GTFABC,,,DEVICE=TAPE**, to override the default value of the UNIT parameter for each output data set in your cataloged procedure.

See *z/OS MVS JCL Reference* for more information about using symbolic parameters in JCL statements.

Specifying or changing GTF trace options through system prompting

After you enter the START command, GTF issues message AHL100A or AHL125A which allows you to specify or change trace options. If the cataloged procedure or START command did not contain a member of predefined options, GTF issues message AHL100A, which allows you to enter trace options. If the procedure or command did include a member of predefined options, GTF identifies those options by issuing the console messages AHL121I and AHL103I. Then you can accept the options, or reject and specify new options.

GTF allows overlapping of trace options when multiple instances are active. This sequence of messages appears as:

```
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER memname OF PDS dsname
AHL103I TRACE OPTIONS SELECTED -
keywd=(value),...,keywd=(value)
keywd,keywd,...,keywd
AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
```

Note: If you specify NOPROMPT or NP on the START GTF command, the system will not issue message AHL125A to request the respecification of trace options or the continuation of initialization.

If you choose to reject any options in the member, you are rejecting all of the options specified in that member. Respecifying trace options does not modify the options in the data set member.

The format of the response is:

```
TRACE=trace option[,trace option]...
```

The trace options determine the amount of storage GTF requires. See "Determining GTF's storage requirements" on page 218.

GTF will accept the trace options listed under "GTF trace options" on page 227.

Examples of starting GTF

In this topic you will find the following examples:

- "Starting GTF with a cataloged procedure and parmlib member" on page 222
- "Starting GTF with internal tracking" on page 222
- "Starting GTF with trace output to an existing data set on tape" on page 223
- "Starting GTF with trace options stored in SYS1.PARMLIB" on page 223
- "Starting GTF without trace options in a member" on page 224
- "Starting GTF to trace VTAM remote network activity" on page 224

Starting GTF with a cataloged procedure and parmlib member

Figure 82 on page 222 shows GTF started with a cataloged procedure that indicates the GTFPARM parmlib member. The trace options are specified in the parmlib member record. In this example, message AHL103I displays the options specified in the GTFPARM member: TRACE=SYSM, DSP, PCI, SRM, TRC, USR. This example shows the messages and the reply generated by the initial START command, and the GTFPARM specifications that are in effect. This instance of GTF can be recognized by the EXAMPLE 1 identifier.

```
START GTF.EXAMPLE1
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER GTFPARM OF PDS SYS1.PARMLIB
AHL103I TRACE OPTIONS SELECTED--SYSM,USR,TRC,DSP,PCI,SRM
00 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 00,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 82. Example: Starting GTF with a Cataloged Procedure

Starting GTF with internal tracking

Figure 83 on page 222 shows GTF, with identifier EXAMPLE2, started with MODE=INT. The trace data is maintained in virtual storage and is not recorded on an external device. In this example, you can override the trace options given in the supplied parmlib member:

```
START GTF.EXAMPLE2,,,(MODE=INT),DSN=NULLFILE
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER memname OF PDS dsname
AHL103I TRACE OPTIONS SELECTED - SYSM,USR,TRC,DSP,PCI,SRM
00 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 00,TRACE=IO,SSCH,SVC,DSP
AHL103I TRACE OPTIONS SELECTED -- DSP,SVC,IO,SSCH
01 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 01,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 83. Example: Starting GTF with internal tracking

Starting GTF with trace output to an existing data set on tape

Figure 84 on page 223 shows how the START command is used to direct GTF trace output to an existing data set on tape rather than to an existing data set on a DASD. The device type and volume serial number are supplied. The disposition and name of the trace data set are changed from DISP=(NEW,KEEP) and DSNAME=SYS1.TRACE to DISP=(OLD,KEEP) and DSNAME=TPOUTPUT. The specified tape has a volume serial of TRCTAP and resides on a 3400 tape drive. Note that the GTFPARM parmlib member is used to specify the trace options.

Here the GTF keyword is not followed by a unique identifer and defaults to volume serial number.

```
START GTF,3400,TRCTAP,(MODE=EXT),DISP=OLD,DSNAME=TPOUTPUT

AHL103I TRACE OPTIONS SELECTED--SYSM,DSP,PCI,SRM,TRC,USR

00 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U

REPLY 00,U

AHL031I GTF INITIALIZATION COMPLETE
```

Figure 84. Example: Start GTF, trace output to an existing data set on tape

Starting GTF with trace options stored in SYS1.PARMLIB

Figure 85 on page 223 shows how to store trace options in a member of SYS1.PARMLIB. This can save you time when starting GTF. First store one or more combinations of trace options as members in SYS1.PARMLIB, and include a SYSLIB DD statement in the cataloged procedure. When you start GTF, GTF will then retrieve the trace options from SYS1.PARMLIB, instead of prompting you to supply them through the console. GTF displays the trace options for you, and then issues message AHL125A, to which you can reply **U** to accept the parmlib options.

This example shows the job control statements and utility JCL statements needed to add trace options to SYS1.PARMLIB using IEBUPDTE.

```
//GTFPARM    JOB     MSGLEVEL=(1,)
//           EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSPRINT   DD      SYSOUT=A
//SYSUT2     DD      DSNAME=SYS1.PARMLIB,DISP=SHR
//SYSIN      DD      DATA
./       ADD    NAME=GTFA,LIST=ALL,SOURCE=0
TRACE=SYSP,USR
SVC=(1,2,3,4,10),IO=(D34,D0C),SSCH=ED8,PI=15
./       ADD    NAME=GTFB,LIST=ALL,SOURCE=0
TRACE=IO,SSCH,TRC
./       ADD    NAME=GTFC,LIST=ALL,SOURCE=0
TRACE=SYS,PCI
/*
```

Figure 85. Example: Starting GTF with trace options stored in SYS1.PARMLIB

A sample SYSLIB DD statement to be included in a GTF cataloged procedure might look like this:

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

The new member name can also be specified on the START command while using the IBM-supplied GTF procedure, as in the following example:

```
S GTF,,,(MODE=EXT,TIME=YES),MEMBER=GTFB
```

For more information see the following references:

- See *z/OS DFSMSdfp Utilities* for descriptions of the statements.
- See *z/OS MVS JCL Reference* for descriptions of the statements.

- See *z/OS MVS Initialization and Tuning Reference* for further information about SYS1.PARMLIB.

Starting GTF without trace options in a member

Figure 86 on page 224 shows an installation-written procedure where there is no predefined member with trace options specified. The procedure contains no SYSLIB DD statement. When GTF is started with a procedure containing no SYSLIB DD statement, message AHL100A is issued to prompt for GTF trace options.

In this example, an installation-written cataloged procedure, USRPROC, is invoked to start GTF in external mode to a direct access data set, ABCTRC, on device 250. The trace options selected result in trace data being gathered for:

- All SVC and IO interruptions
- All SSCH operations
- All matching SLIP traps with a tracing action specified or SLIP traps in DEBUG mode
- All dispatcher events
- All issuers of the GTRACE macro will have their user data recorded in the trace buffers.

The trace data is written into the data set ABCTRC. (Note that when the end of the primary extent is reached, writing continues at the beginning.)

```
START USRPROC,250,333005,(MODE=EXT),DSN=ABCTRC

00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00,TRACE=SVC,SSCH,IO,DSP,SLIP,USR
AHL103I TRACE OPTIONS SELECTED--USR,DSP,SVC,IO,SLIP,SSCH
01 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 01,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 86. Example: Starting GTF without trace options in a member

Starting GTF to trace VTAM remote network activity

GTF can trace VTAM activity only if VTAM is started with the GTF option. See *z/OS Communications Server: SNA Operation* for details. In Figure 87 on page 225, GTF options are not stored in parmlib; the trace options are entered at the console. Three GTF options are required to record all VTAM traces:

- RNIO must be specified so that the VTAM I/O trace can function for an NCP or a remote device attached to the NCP.
- IO or IOP must be specified so that the VTAM I/O trace can function for a local device.
- USR or USRP must be specified so that the VTAM buffer and the NCP line traces can function.

You must enter the START GTF command before a trace can be activated from VTAM.

```
START MYPROC.EXAMPLE8,,,(MODE=EXT)
00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00,TRACE=RNIO,IO,USRP
AHL103I TRACE OPTIONS SELECTED--IO,USRP,RNIO
01 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 01,USR=(FF0,FF1),END
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 87. Example: Starting GTF to trace VTAM remote network activity

Stopping GTF

You can enter the STOP command at any time during GTF processing. The amount of time you let GTF runs depends on your installation and the problem you are trying to capture, but a common time is between 15 and 30 minutes.

If you are running GTF to gather information related to a problem for which a SLIP trap has been defined, you can instruct SLIP to stop all instances of GTF when the trap conditions are satisfied. For additional information, see the SLIP command documentation in *z/OS MVS System Commands*.

To stop GTF processing, enter the STOP command. The STOP command must include either the GTF identifier specified in the START command, or the device number of the GTF trace data set if you specified MODE=EXT or MODE=DEFER to direct output to a data set. If you have not specified the GTF identifier in the START command, then those instances of GTF will have the same identifier: the volume serial number.

If you are not sure of the identifier or the device number of the trace data set, enter the following command:

```
DISPLAY A,LIST
```

Figure 88 on page 225 shows the output produced by the DISPLAY A,LIST command. In this example, the identifier for GTF is EVENT1.

```
CNZ4105I 16.47.46 DISPLAY ACTIVITY   FRAME LAST   F      E   SYS=SY1
 JOBS     M/S    TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM     OAS
00000    00005    00000      00016    00008    00000/00300       00000
 LLA      LLA      LLA       NSW  S  VLF      VLF      VLF      NSW  S
 JES2     JES2     IEFPROC   NSW  S  SDSF     SDSF     SDSF     NSW  S
 GTF      EVENT1   IEFPROC   NSW  S
```

Figure 88. Example: recognizing GTF identifier in DISPLAY A,LIST output

You must enter the STOP command at a console with master authority. The general format of the STOP command is as follows:

```
{STOP|P} identifier
```

For example, to stop GTF for the identifier EVENT1 (as shown in Figure 88 on page 225), enter the command:

```
STOP EVENT1
```

When the STOP command takes effect, the system issues message AHL006I. If the system does not issue message AHL006I, then GTF tracing continues, remaining active until a STOP command takes effect or the next initial program load (IPL). When this happens, you will not be able to restart GTF tracing. In this case, you can use the FORCE ARM command to stop GTF.

If there were several functions started with the same identifier on the START command, using the same identifier on the STOP command will stop all those functions.

If the volume serial number is used on the STOP command, all instances of GTF with trace data directed to a data set on that volume serial are stopped. This is independent of the identifier assigned to each instance of GTF.

For example, if three instances of GTF are active with the identifiers EX1, EX2, and EX3 directing trace data to different data sets to the same volume with volume serial number 1020, then the following command will stop all the 3 instances of GTF.

```
STOP 1020
```

See *z/OS MVS System Commands* for more information about the STOP and FORCE ARM commands.

You can also use an identifier to stop GTF. In this example, the following command starts GTF tracing with the identifier EXAMPLE and with trace data maintained in the GTF address space. The DSN keyword is entered to prevent allocation of an external trace data set as specified in the cataloged procedure.

```
  START GTF.EXAMPLE,,,(MODE=INT),DSN=NULLFILE
```

To stop GTF tracing, you would issue the following command:

```
  STOP EXAMPLE
```

In some instances, you may need to display the active jobs before stopping GTF. The example shown in Figure 89 on page 226 starts GTF tracing with trace data recorded on an external device, data set GTF.TEST01. Another instance of GTF with an identifier EX1 is started with trace data directed to another data set on the same volume. Note that you do not have to specify MODE=EXT, because it is the default.

```
S GTF,,,DSNAME=GTF.TEST01,VOLUME=SER=IPCS01,DISP=OLD
S GTF.EX1,DSNAME=GTF.TEST02,VOLUME=SER=IPCS01
```

Figure 89. Example: Starting instances of GTF

Because it is not apparent which is the GTF recording device, you have to display active jobs with the DISPLAY A,LIST command before stopping GTF. In Figure 90 on page 226, the device number for GTF is 0227.

```
CNZ4105I 16.54.36 DISPLAY ACTIVITY FRAME LAST F E SYS=SY1
 JOBS     M/S    TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM     OAS
00000    00006    00000      00028    00008    00002/00300       00003
 IGVDGNPP IGVDGNPP IGVDGNPP OWT  S  VLF      VLF      VLF      NSW  S
 VTAM     VTAM     VTAM     NSW  S  J273     J273     IEFPROC  NSW  S
 GTF      0227     IEFPROC  NSW  S  GTF      EX1      IEFPROC  NSW  S
```

Figure 90. Example: DISPLAY A,LIST command output

If you only want to stop only the second instance of GTF, issue the following command:

```
STOP EX1
```

If you want to stop both instances, issue the following command:

```
STOP 227
```

GTF trace options

This topic describes the GTF options you can specify through either system prompting in response to the START GTF command or in a predefined parmlib or data set member. However, GTF will not use certain combinations of options; see Table 36 on page 231 for a list of those combinations.

Some GTF trace options also require keywords. If you specify options requiring keywords in the member or data set containing the predefined options, it must also contain the associated keywords. These are explained in "Prompting keywords" on page 231.

ASIDP
: Requests that GTF tracing be limited to a subset of address spaces. ASIDP requests GTF prompting for one to five address space identifiers (ASID) in which you want GTF tracing to occur. ASIDP works only with a GTF option that generates tracing, such as SVC or IO. For information about responding to GTF prompts, see "Prompting keywords" on page 231.

CCW
: Requests tracing of channel programs and associated data for I/O events. CCW is valid only if the other trace options include SSCH, SSCHP, IO, or IOP. See Table 36 on page 231.

CCWP
: Requests tracing of channel programs and associated data for I/O events, and requests GTF prompting for the following information:

- Tracing channel command words (CCW) or device command words (DCW) for start subchannel (SSCH) operations or I/O interruptions or both
- Tracing the request and response blocks for synchronous I/O (zHyperLink) end events
- Maximum number of CCWs or device command words (DCW) for each event
- Maximum number of bytes of data for each CCW or DCW or synchronous I/O request
- Optional input/output supervisor block (IOSB), input/output block extension (IOBE, zHPF channel programs, and synchronous I/O requests only), and error recovery procedure work area (EWA) tracing
- Size of the program controlled interrupt (PCI) table

For information about responding to GTF prompts, see "Prompting keywords" on page 231.

CCWP is valid only if the other trace options include SSCH, SSCHP, IO, or IOP. See Table 36 on page 231.

CSCH
: Requests recording for all clear subchannel operations. See Table 36 on page 231 for more information on combining this option with other GTF options.

DSP
: Requests recording for all dispatchable units of work: service request block (SRB), local supervisor routine (LSR), task control block (TCB) and Supervisor Call (SVC) prolog dispatch events. If you specify both the SYSM and DSP trace options, GTF records minimal trace data for DSP. Otherwise, GTF records comprehensive trace data for DSP.

EXT
: Requests comprehensive recording for all external interruptions. See Table 36 on page 231 for more information on combining this option with other GTF options.

HSCH
: Requests recording for all halt subchannel operations. See Table 36 on page 231 for more information on combining this option with other GTF options.

IO
: Requests recording of all non-program-controlled I/O interruptions and synchronous I/O (zHyperLink) end events. Unless you also specify the PCI trace option, GTF does not record program-controlled interruptions. See Table 36 on page 231 for more information on combining this option with other GTF options.

IOX
Requests recording of all non-program-controlled I/O interruptions providing a summary of a complete channel program for the I/O interruption in an I/O summary trace record. Unless you also specify the PCI trace option, GTF does not record program-controlled interruptions.

IOP
Requests GTF prompting for specific device numbers for which you want GTF to record non-program-controlled I/O interruptions and synchronous I/O (zHyperLink) end events. Unless you specify the PCI trace option, GTF does not record program-controlled interruptions. See Table 36 on page 231 for more information on combining this option with other GTF options. For information about responding to GTF prompts, see "Prompting keywords" on page 231.

IOXP
Requests GTF prompting for specific device numbers for which you want GTF to record non-program-controlled I/O interruptions providing a summary of a complete channel program for the I/O interruption in an I/O summary trace record. Unless you specify the PCI trace option, GTF does not record program-controlled interruptions. For more information on responding to GTF prompts, see "Prompting keywords" on page 231.

If an installation chooses to specify either IO or IOP in addition to IOX or IOXP, they will receive IOX records for DASD and tape devices and IO records for all other devices.

JOBNAMEP
Requests that GTF tracing be limited to a subset of jobs. JOBNAMEP requests GTF prompting for one through five job names for which you want GTF tracing to occur.

These job names can be generic, as well as specific, job names. If you want to specify generic job names, use * or % in the job name.

The asterisk is a placeholder for one or more valid job name characters, or indicates no characters. For example, if you enter JOBNAMEP=I*MS*, GTF will process trace data for address spaces with job names IABMS01, IAMS, IMS, IMSA, IMS00012, and so on. However, if you enter JOBNAMEP=*MASTER*, that job name represents only the master address space.

The percent symbol is a placeholder for a single valid job name character. For example, if you enter JOBNAMEP=I%MS%%, GTF will process trace data for address spaces with job names IAMS01 and IXMSBC, but not job names IMS001 or I999MS. The combination %* is a placeholder for at least one character.

JOBNAMEP works only with a GTF option that generates tracing, such as SVC or IO. For information on responding to GTF prompts, see "Prompting keywords" on page 231.

MSCH
Requests recording for all modify subchannel operations. See Table 36 on page 231 for more information on combining this option with other GTF options.

PCI
Requests recording of intermediate status interruptions in the same format as other I/O trace records that GTF creates. Specifically, PCI causes GTF to record program-controlled I/O interruptions, initial status request interruptions, resume subchannel operation instruction, and suspend channel program interruptions. When you select specific devices as a result of prompting for I/O events (IOP trace option), GTF records intermediate status interruptions for only those devices. PCI is valid only when the other trace options include IO, IOP, SYS, SYSM, or SYSP.

PCIE
Requests tracing of PCI load and store instructions, adapter interrupts, and PCIE de-multiplexing requests.

PFIDP
Requests that GTF tracing of PCIE-related events be limited to a subset of the PCIE function identifiers (PFIDs). PFIDP requests GTF prompting for 1 to 256 PFIDs or PFID ranges in which you want GTF tracing to occur. PFIDP is only valid when the PCIE trace option is specified. For information on responding to GTF prompts, see "Prompting keywords" on page 231.

PI
Requests comprehensive recording for all program interruptions (0-255). See Table 36 on page 231 for more information on combining this option with other GTF options.

PIP
Requests GTF prompting for those interruption codes for which you want GTF to record program interruptions. For information about responding to GTF prompts, see "Prompting keywords" on page 231. See Table 36 on page 231 for more information on combining this option with other GTF options.

RNIO
Requests recording of all Virtual Telecommunications Access Method (VTAM) network activity. If you specify both the SYSM and RNIO trace options, GTF will record minimal trace data for RNIO. Otherwise, GTF records comprehensive trace data for RNIO.

RR
Requests comprehensive recording of data associated with all invocations of recovery routines (such as STAE and ESTAE routines). GTF creates a trace record describing the activity of the recovery routine when control passes from the recovery routine back to the recovery termination manager (RTM). See Table 36 on page 231 for more information on combining this option with other GTF options.

{SIO|SIOP}
If you specify the SIO or SIOP trace option, GTF processes that request as a request for SSCH or SSCHP. GTF issues message AHL138I to indicate this substitution. Subsequent messages refer to the original SIO or SIOP trace option.

Note: The SIO keyword is provided only for compatibility; it is recommended that you use the SSCH keyword instead. The SIOP option is provided only for compatibility; it is recommended that you use the SSCHP option instead.

SLIP
Requests that a trace entry be made each time:

- A match occurs for a SLIP trap with ACTION=TRACE
- A SLIP trap with the SLIP DEBUG option is checked

The amount of data and the type of SLIP trace record to be built is specified on the SLIP command.

SRM
Requests recording of trace data each time the system resource manager (SRM) is invoked. If you specify both the SYSM and SRM trace options, GTF records minimal trace data for SRM. Otherwise, GTF records comprehensive trace data for SRM.

SSCH
Requests recording for start subchannel and resume subchannel operations and synchronous I/O (zHyperLink) start events. See Table 36 on page 231 for more information on combining this option with other GTF options.

SSCHP
Requests GTF prompting for the specific device numbers for which you want GTF to record start subchannel and resume subchannel operations, and synchronous I/O (zHyperLink) end events. For information about responding to GTF prompts, see "Prompting keywords" on page 231. See Table 36 on page 231 for more information on combining this option with other GTF options.

SVC
Requests comprehensive recording for all SVC interruptions. See Table 36 on page 231 for more information on combining this option with other GTF options.

SVCP
Requests GTF prompting for those SVC numbers for which you want data recorded. For information about responding to GTF prompts, see "Prompting keywords" on page 231. See Table 36 on page 231 for more information on combining this option with other GTF options.

SYS
Requests recording of comprehensive trace data for all of the following:

- Clear subchannel operations

- External interruptions
- Halt subchannel operations
- I/O interruptions
- Modify subchannel operations
- Program interruptions
- Recovery routines
- Start subchannel and resume channel operations
- SVC interruptions.

Because specifying SYS automatically causes GTF to trace all of these events, GTF will ignore the following trace options if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC. See Table 36 on page 231 for more information on combining this option with other GTF options.

SYSM
Requests recording of minimal trace data for the same events as SYS.

Because specifying SYSM automatically causes GTF to trace all of these events, GTF will ignore the following trace options if if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC.

If if you specify DSP, RNIO, or SRM in addition to SYSM, GTF produces minimal, rather than comprehensive, trace data for those events.

SYSP
Requests recording for the same events as the SYS option, but causes GTF to prompt for selection of specific SVC, IO, SSCH, and PI events that you want recorded. For information about responding to GTF prompts, see "Prompting keywords" on page 231.

Because specifying SYSP automatically causes GTF to trace all of these events, GTF will ignore the following trace options if you specify them in any form: CSCH, HSCH, MSCH, SSCH, EXT, IO, PI, RR, SVC. See Table 36 on page 231 for more information on combining this option with other GTF options.

TRC
Requests recording of those trace events that are associated with GTF itself. Unless you request TRC, GTF will not trace these events. TRC works only with a GTF option that generates tracing, such as SVC or IO.

USR
Requests recording of all data that the GTRACE macro passes to GTF. You must specify USR or USRP to trace data from the GTRACE macro. Use USRP for specific events. If USR is used instead of USRP, the trace data set might be full of unwanted records. When you code the GTRACE macro but do not specify USR or USRP, GTF ignores the GTRACE macro. See Table 36 on page 231 for more information on combining this option with other GTF options.

Reference

See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about coding the GTRACE macro.

USRP
Requests GTF prompting for specific event identifiers (EID) of the data that the GTRACE macro passes to GTF. The EIDs represent user, program product, or IBM subsystem and component events. See Table 39 on page 236 for a list of EID values.

See Table 36 on page 231 for more information on combining this option with other GTF options. For information about responding to GTF prompts, see "Prompting keywords" on page 231.

XSCH
Requests recording all cancel subchannel operations.

See Table 36 on page 231 for more information on combining this option with other GTF options. For information about responding to GTF prompts, see Table 37 on page 231.

Combining GTF options

Table 36 on page 231 shows those TRACE options that GTF will *not* use in combination. If two or more options from the same row are specified, GTF uses the option that has the lower column number and ignores the other options. For example, if you specify both SYSP and PI (see row D), GTF uses SYSP (column 2) and ignores PI (column 5).

Table 36. Combining GTF options

Row	Columns				
	1	2	3	4	5
A	SYSM	SYSP	SYS	SSCHP	SSCH
B	SYSM	SYSP	SYS	IOP, IOXP	IO, IOX
C	SYSM	SYSP	SYS	SVCP	SVC
D	SYSM	SYSP	SYS	PIP	PI
E	SYSM	SYSP	SYS	EXT	
F	SYSM	SYSP	SYS	RR	
G	SYSM	SYSP	SYS	CSCH	
H	SYSM	SYSP	SYS	HSCH	
I	SYSM	SYSP	SYS	MSCH	
J	SYSM	SYSP	SYS	XSCH	
K	CCWP	CCW			
L	USRP	USR			

If an installation chooses to specify either IO or IOP in addition to IOX or IOXP, they will receive IOX records for DASD and tape devices and IO records for all other devices.

Prompting keywords

When you specify any of the trace options listed in Table 37 on page 231, GTF prompts for specific values by issuing message AHL101A:

```
AHL101A SPECIFY TRACE EVENT KEYWORDS - keyword=,...,keyword=
```

The keywords issued in the message correspond to the trace options specified. Enter only the trace event keywords appearing in the message text. The trace options and their corresponding keywords are:

Table 37. GTF trace options and corresponding prompting keywords

Trace Option	Prompting Keyword	Number of Prompting Values Allowed
ASIDP	ASID=	5
CCWP	CCW=	N/A
IOP, IOXP, SYSP	IO=SSCH=	Unlimited
IOP, IOXP, SSCHP, SYSP	IO=SSCH=	Unlimited
JOBNAMEP	JOBNAME=	5
PFIDP	PFID=	256
PIP, SYSP	PI=	50
SSCHP, SIOP, SYSP	SIO=	Unlimited
SSCHP, SIOP, SYSP	SSCH=	Unlimited

Table 37. GTF trace options and corresponding prompting keywords (continued)		
Trace Option	**Prompting Keyword**	**Number of Prompting Values Allowed**
SVCP, SYSP	SVC=	50
USRP	USR=	50

Note:

1. The SIO keyword is provided only for compatibility; it is recommended that you use the SSCH keyword instead. The SIOP option is provided only for compatibility; it is recommended that you use the SSCHP option instead.
2. Tracing a PAV base device number will cause all PAV aliases associated with that base device number to also be traced. If I/O tracing is needed to locate issues related to PAV alias device numbers when they are not associated with a PAV base device number, specify the device numbers of the PAV alias devices explicitly.

Guidelines for specifying values for prompting keywords: Use the following guidelines when replying to message AHL101A for prompting keywords:

- If you do not specify a reply for each of the keywords displayed in message AHL101A, GTF records all the events for that trace option, which increases the amount of storage that GTF requires. IBM recommends that you specify values for each keyword displayed, selecting the values that will help you debug your problem.
- You can only enter values for keywords displayed in message AHL101A.
- GTF limits the number of specific values that you can supply through prompting, see Table 37 on page 231 for the maximum number of values allowed for each keyword. If you specify more than the maximum values, GTF issues a message to which you reply by respecifying values for all appropriate keywords.
- Keep in mind that prompting increases the amount of storage that GTF requires, because storage requirements depend on the trace options you specify. See “Determining GTF's storage requirements” on page 218 for further information.
- Within a given reply, each keyword that you specify must be complete. If you need more values for a keyword than will fit into one reply, repeat the keyword in the next reply, and code the additional values for that keyword. The following are examples of correct replies:

  ```
  REPLY 01 IO=(191-193),SVC=(1,2,3,4,5)
  REPLY 01 SVC=(6,7,8,9,10)
  ```

  The maximum number of values that GTF allows for a keyword does not change, regardless of whether you enter one or more replies to specify all the values for the keyword.
- After supplying all keywords and values, you must enter the END keyword, which signifies that the event definition is complete. If the system does not find the END keyword in a reply, the system issues message AHL102A to prompt for additional event keywords and values. When the system finds the END keyword, the system issues message AHL103I to list all of the trace options that are in effect.

For sample prompting sequences, see “Examples of sample prompting sequences” on page 236.

Use the following keywords when GTF prompts for values by issuing message AHL101A:

ASID=(*asid1*[,*asidn*]...[,*asid5*])

Specifies one through five identifiers for address spaces in which you want GTF tracing to occur. The values ‘asid1’ through ‘asid5’ are hexadecimal numbers from X'0001' to the maximum number of entries in the address space vector table (ASVT). If you specify ASIDP, but do not specify ASID= before replying END, GTF traces all address space identifiers.

If the number of values for ASIDP requires more than one line, and a particular ASID value is incorrect, GTF allows you to respecify the correct value without having to reenter all ASIDs.

If you specify both ASIDP and JOBNAMEP, GTF will trace address spaces that ASIDP did not identify, if some of the jobs that JOBNAMEP identifies run in other address spaces.

CCW=([S|I|SI][,CCWN=*nnnnn*][,DATA=*nnnnn*][,IOSB][,PCITAB=n])

Specifies different options for tracing channel programs. If you specify CCW= more than once, GTF uses the last specification of CCW=.

If you specify CCWP, but do not specify a value for keyword CCW=, GTF's default CCW tracing depends on what other trace options were specified. The following table shows the defaults for CCW tracing depending on other trace options specified:

Table 38. CCW defaults for selected TRACE options

Other Trace Options Selected	CCW Subparameter Defaults
SSCH or SSCHP	S
IO or IOP or IOX or IOXP	I
SSCH or SSCHP or IO or IOP or IOX or IOXP	SI
PCI	PCITAB=1
SSCH or SSCHP or IO or IOP or IOX or IOXP	CCWN=50
SSCH or SSCHP or IO or IOP or IOX or IOXP	DATA=20

```
Examples:
TRACE=IO,CCWP            CCW defaults to: CCW=(I,CCWN=50,DATA=20)
TRACE=IOP,SSCH,PCI,CCWP  CCW defaults to: CCW=(SI,CCWN=50,DATA=20,PCITAB=1)
```

If you specify an option more than once in one line, GTF uses the last specification of that option. An exception is that GTF uses the first specification of S, I, or SI. If a line contains an error, GTF prompts you to respecify the value.

S|I|SI
Specifies the type of I/O event for which you want channel programs traced. If you specify more than one option, GTF uses the first option.

S
Specifies GTF tracing of channel programs for start subchannel and resume subchannel operations. CCW=S works only with the SSCH or SSCHP trace options.

I
Specifies GTF tracing of channel programs for I/O interruptions, including program-controlled interruptions if you specify PCI as a trace option, and synchronous I/O (zHyperLink) end events. CCW=I works only with the IO or IOP trace options.

SI
Specifies GTF tracing of channel programs for start subchannel and resume subchannel operations and I/O interruptions, and synchronous I/O (zHyperLink) end events. CCW=SI works only with either SSCH or SSCHP and either IO or IOP as trace options.

CCWN=*nnnnn*
Specifies the maximum number of CCWs or DCWs traced for each event. The value *nnnnn* is any decimal number from 1 to 32767. The default is 50.

DATA=*nnnnn*
Specifies the maximum number of bytes of data traced for each CCW or DCW, or indirect data address word. The value *nnnnn* is any decimal number from 0 to 32767. The default is 20.

For non-zHPF channel programs, GTF traces *nnnnn* bytes of data for each CCW, or each indirect data address word (IDAW), or modified indirect data address word (MIDAW). For zHPF channel programs, GTF traces *nnnnn* bytes of data for each DCW, or each transport indirect data address word (TIDAW). For synchronous I/O operations, GTF traces*nnnnn* bytes for each synchronous I/O data address word (SDAW).

For start subchannel or resume subchannel operations, GTF does not trace data for read, read backwards, or sense commands in the channel programs. If no data is being transferred, regardless of the type of I/O operation, GTF does not trace data for read, read backwards, or sense commands.

For synchronous I/O operations, GTF only traces data at the end of the synchronous I/O operation (synchronous I/O end event) if the operation is successful (response code X'0010').

When the data count in the CCW or DCW or the amount of data associated with the indirect data address wordis equal to or less than *nnnnn*, GTF traces all data in the data buffer. When the data count in the CCW or DCWor the amount of data associated with the indirect data address word is greater than *nnnnn*, GTF traces data only from the beginning and end of the data buffer. If you examine the traced data, you can tell whether the channel completely filled the buffer on a read operation.

GTF uses a different CCW or DCW tracing method for a data transfer that is in progress when an I/O interruption occurs. Instead of using the data count in the CCW or DCW, GTF tracing depends on the transmitted data count. The transmitted data count is the difference between the data count in the CCW or DCW and the residual count in the subchannel status word (SCSW), for non-zHPF channel programs and the transport status block (TSB), for zHPF channel programs.

- When the residual count is greater than the data count in the CCW or DCW, then GTF traces all of the data in the CCW or DCW.
- When the transmitted data count is less than or equal to *nnnnn*, GTF traces all of the transmitted data.
- When the transmitted data count is greater than *nnnnn*, GTF traces data only from the beginning and end of the transmitted data.

IOSB
Specifies tracing of the input/output supervisor block (IOSB), the input/output block extension (IOBE), for zHPF channel programs and synchronous I/O requests, and, if available, the error recovery procedure work area (EWA), for all events. If you do not specify IOSB, then GTF performs IOSB and EWA tracing only if GTF encounters an exceptional condition when tracing a channel program.

PCITAB=*n*
Specifies a decimal number of 100-entry increments for GTF to allocate in an internal program-controlled interruption (PCI) table. The value of *n* is an integer from 1 to 9. The default is 1 (100 entries).

The PCI table keeps track of the channel programs that use PCI. One entry in the PCI table contains information about a program-controlled interruption in one channel program. An entry in the PCI table includes a CCW address and an IOSB address.

IO=(*DEVCLASS=xxxx,DEVCLASS=xxxx,devnum1* [*,devnumn...,devnum*])
Specifies devices for which you want I/O interruptions or synchronous I/O end events traced. Devices are specified by entering a device number or a device class.

The device number must be specified in hexadecimal and is not the same as the subchannel number. If you specify any combination of IO= and SSCH=, and IO=SSCH=, the combined number of device numbers for all prompting keywords is unlimited. Specify device numbers individually, or as a range of device numbers with a dash (-) or colon (:) separating the lowest and highest number in the range. For example, to trace I/O interruptions for device numbers 193 through 198, you specify IO=(193-198).

The device class must be specified with the DEVCLASS= keyword parameter, which provides the ability to trace all devices in the specified device class. The allowable keyword parameters are:

```
TAPE  (magnetic tape devices)
COMM  (communications)
DASD  (direct access storage device)
DISP  (display)
UREC  (unit/record)
CTC   (channel to channel)
```

If you specify IOP, IOXP, or SYSP and does not specify IO= in the response to the prompting messages, GTF processing proceeds as if you specified IO, IOX or SYS event keywords respectively.

For the following examples, the I/O device numbers and associated device types listed below are used:

I/O Device Number
: **Device Type**

230
: 3390 DASD

450
: 3490 Tape Drive

575
: 3480 Tape Drive

663
: 3380 DASD

020
: 3274 Communications Controller

In this example, the resulting trace includes information for all DASD devices and one 3490 tape drive at address 450.

```
IO=(DEVCLASS=DASD,450)
```

In the following example, the resulting trace includes information for all DASD and TAPE devices and the communications controller at address 020.

```
IO=(DEVCLASS=DASD,DEVCLASS=TAPE,020)
```

In this example, the resulting trace includes information for the devices at addresses 450, 575, and 663.

```
IO=(450-663)
```

**IO=SSCH=(*DEVCLASS=xxxx,DEVCLASS=xxxx,*devnum1[,devnumm....
[,devnum])**

Specifies devices for which you want I/O interruptions and start subchannel operations, and synchronous I/O start and end events traced. See the IO= prompting keyword for a description of how to specify devices to be traced.

JOBNAME=(*jobname1*[,*jobnamen*]...[,*jobname5*])

Specifies one through five job names for which you want GTF tracing to occur. The values *job1* through *job5* must be valid job names.

These job names can be generic, as well as specific, job names. If you want to specify generic job names, you must use * or % in the job name.

The asterisk is a placeholder for one or more valid job name characters, or indicates no characters. For example, if the you enter JOBNAMEP=I*MS*, GTF will process trace data for address spaces with job names IABMS01, IAMS, IMS, IMSA, IMS00012, and so on. However, if you code JOBNAMEP=*MASTER*, that job name represents only the master address space.

The percent symbol is a placeholder for a single valid job name character. For example, if you code JOBNAMEP=I%MS%%, GTF will process trace data for address spaces with job names IAMS01 and IXMSBC, but not job names IMS001 or I999MS. The combination %* is a placeholder for at least one character.

If you specify JOBNAMEP, but do not specify JOBNAME before replying END, GTF traces all job names.

If the number of values for JOBNAMEP requires more than one line, and a particular job name value is incorrect, GTF allows you to respecify the correct value without having to reenter all job names.

If you specify both ASIDP and JOBNAMEP, GTF will trace jobs that JOBNAMEP did not identify, if some of the address spaces that ASIDP identifies contain jobs that JOBNAMEP did not identify.

PFID=(*pfid1*[,*pfidn*]...[,*pfid256*])

Specifies 1 to 256 PFIDs or PFID ranges for which you want GTF tracing to occur. The values are hexadecimal numbers from 0 to FFFFFFFF. Leading zeroes are allowed. For example: 1, 01, and 00000001 are all accepted. If a PFID range is specified, the first number in the range must be lower than the second number in the range.

Duplicate PFID values are ignored. Overlapping PFID ranges are accepted without issuing an error message.

If you specify PFIDP, but do not specify PFID= before replying END, GTF traces all PFIDs. If the number of values for PFIDP requires more than one line, and a particular PFID value is incorrect, GTF allows you to re-specify the correct value without having to reenter all PFIDs.

PI=(*code0*[,*coden*]...[,*code50*])

Specifies 1 through 50 program interruption codes, in decimal notation, that you want traced. If you specify PIP or SYSP, and do not specify PI= in response to this prompting message, GTF traces all program interruptions.

SSCH=(*DEVCLASS=xxxx*,*DEVCLASS=xxxx*,*devnum1*[,*devnumn....*, devnum])

Specifies devices for which you want start subchannel operations and synchronous I/O start events traced. See the IO= “Prompting keywords” on page 231 for a description of how to specify devices to be traced.

SVC=(*svcnum1*[,*svcnumn*]...[,*svcnum50*])

Specifies 1 through 50 SVC numbers, in decimal notation, that you want traced. If you specify SVCP or SYSP, and do not specify SVC= in response to the prompting message, GTF traces all SVC numbers. Both SVC entry and exit are be recorded.

USR=(*event1*[,*eventn*]...[,*event50*])

Specifies 1 through 50 user event identifiers (EIDs) for which you want user data traced. The values for USR are three-digit hexadecimal numbers, as follows:

Table 39. Event identifiers and the types of events they represent

Identifier (Hex)	Type of Event
000-3FF	User
400-5FF	Reserved for program products
600-FFF	Reserved for IBM subsystems and components

If you specify USRP and do not specify USR= in response to the prompting message, all instances of GTRACE issued with TEST=YES will return with an indication that tracing is not active.

Examples of sample prompting sequences

This example shows how to store prompting keywords in a SYS1.PARMLIB member.

If you start GTF with options requiring prompting keywords stored in SYS1.PARMLIB, these prompting keywords must also appear in the parmlib member. If prompting keywords are used in the parmlib member without the replies included, GTF will not obtain the replies from the console. GTF will use the options without the prompting (for example, SVCP becomes SVC). A SYSLIB DD statement in your cataloged procedure causes GTF to read the prompting keywords from the specified parmlib member. The second and subsequent logical records in the member should contain only the prompting keywords.

GTF uses either the END keyword, or end-of-file on the member as the indicator that there is no more prompting input from parmlib. If the number of events for one keyword require more than one record, respecify the keyword in a subsequent prompting record with the additional events, as follows:

```
Record #1 TRACE=IOP,SVCP,SSCH

Record #2 IO=(D34,D0C),SVC=(1,2,3)

Record #3 SVC=(4,5,6,7,8,9,10),END
```

Do not respecify the keyword through the system console at this point, because GTF will then override all of the options and keywords in the parmlib member.

When GTF finishes reading the options and prompting keywords in the parmlib member, it displays the options through message AHL103I:

```
AHL103I TRACE OPTIONS SELECTED--IOP,SVCP,SSCH
AHL103I IO=(D34,D0C),SVC=(1,2,3,4,5,6,7,8,9,10)
```

This message may be a multiple-line message, depending on the number of options you select. If the set of devices specified for IO= and SSCH= are identical, message AHL103I will show them as if specified by use of IO=SSCH.

After GTF displays all of the options specified, you then have the opportunity to accept the parmlib options, or completely change the options by respecifying them through the console by replying to the following message:

```
AHL125A RESPECIFY TRACE OPTIONS OR REPLY U.
```

In Figure 91 on page 238, you started GTF in external mode to the data set defined in the cataloged procedure. You selected two trace options in reply 00:

- SYSP requests that GTF trace specific system event types.
- USRP requests that GTF trace specific user entries that the GTRACE macro generates.

Message AHL101A instructed you to specify values for the SVC, IO, SSCH, PI, and USR keywords. In reply 01 to message AHL101A, you selected:

- Five SVCs
- Two devices for non-program-controlled I/O interruptions
- One device for SSCH operations
- Three user event identifiers.

GTF does not record any other SVC, IO, and SSCH events. Because you did not specify any program interruption codes for PI=, GTF would trace all program interruptions.

```
START MYPROC.EXAMPLE7,,,(MODE=EXT)
00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00,TRACE=SYSP,USRP
01 AHL101A SPECIFY TRACE EVENT KEYWORDS--IO=,SSCH=,SVC=,PI=,USR=
01 AHL101A SPECIFY TRACE EVENT KEYWORDS--IO=SSCH=
REPLY 01,SVC=(1,2,3,4,10),IO=(191,192),USR=(10,07A,AB)
02 AHL102A CONTINUE TRACE DEFINITION OR REPLY END
REPLY 02,SSCH=282,END
AHL103I TRACE OPTIONS SELECTED--SYSP,PI,IO=(191,192),SSCH=(282)
AHL103I SVC=(1,2,3,4,10),USR=(010,07A,0AB)
03 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 03,U
```

Figure 91. Example: Specifying prompting trace options SYSP and USRP

In Figure 92 on page 238 you started GTF in external mode, using the trace options defined in the data set specified in the cataloged procedure. You are prompted for the information as follows:

- Message AHL100A prompts for trace options.
- In reply 00, you selected six trace options: SSCHP, IOP, PCI, CCWP, SVC, and JOBNAMEP.
- Message AHL101A prompts to specify values for the IO, SSCH, CCW and JOBNAME prompting keywords.
- In reply 01, you select one device for tracing both IO and SSCH events and limit GTF tracing to one job.
- In reply 02, you specify five options for CCW tracing.

The final result of these specifications is that GTF traces CCWs for both start subchannel operations and I/O interruptions at device 580 for the job BACKWARD, and all SVCs in BACKWARD's address space. GTF would allocate 200 entries in the PCI table, and trace up to 100 CCWs or DCWs, up to 40 bytes of data for each CCW or DCW, and the IOSB, IOBE, and EWA.

```
START USRPROC,,,(MOD=EXT)
00 AHL100A SPECIFY TRACE OPTIONS
REPLY 00, TRACE=SSCHP,IOP,PCI,CCWP,SVC,JOBNAMEP
01 AHL101A SPECIFY TRACE EVENT KEYWORDS
        --IO=,SSCH=,CCW=,JOBNAME=,IO=SSCH=
REPLY 01,JOBNAME=(BACKWARD),IO=SSCH=580
02 AHL102A CONTINUE TRACE DEFINITION OR REPLY END
REPLY 02,CCW=(CCWN=100,DATA=40,PCITAB=2,IOSB,SI),END
AHL103I TRACE OPTIONS SELECTED--PCI,SVC,IO=SSCH=(580)
AHL103I CCW=(SI,IOSB,CCWN=100,DATA=40,PCITAB=2)
AHL103I JOBNAME=(BACKWARD)
03 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 03,U
```

Figure 92. Example: Specifying prompting trace options

Receiving GTF traces

GTF writes trace data in GTF trace tables in the GTF address space in storage. GTF trace data in storage are printed or viewed as part of a dump, if the dump options list includes TRT to request trace data. The following table shows the dumps that have TRT in their default options. For unformatted dumps that are printed or viewed through IPCS, format the trace data by specifying the IPCS GTFTRACE subcommand or using the IPCS Trace Processing selection panel.

To format and print GTF trace data in a GTFOUTxx or IEFRDER data set, specify the IPCS GTFTRACE subcommand or use the IPCS Trace Processing selection panel.

If the GTF data was created for VTAM diagnosis, you can use the ACF/TAP program to format the VTAM data.

Dump	How to obtain trace data
ABEND dump to SYSABEND	Default
ABEND dump to SYSMDUMP	Not available
ABEND dump to SYSUDUMP	Request SDATA=TRT
SNAP dump	Request SDATA=TRT
Stand-alone dump	Default
SVC dump for SDUMP or SDUMPX macro	Default
SVC dump for DUMP operator command	Default
SVC dump for SLIP operator command with ACTION=SVCD, ACTION=STDUMP, ACTION=SYNCSVCD, or ACTION=TRDUMP	Default
Any dump customized to exclude trace data	Request SDATA=TRT

For more information, see the following references:

- See *z/OS MVS IPCS Commands* for the GTFTRACE subcommand.
- See *z/OS MVS IPCS User's Guide* for the panel interface.

Combining, extracting, and merging GTF trace output

GTF trace data can be combined with other data or extracted from dumps and data sets using two IPCS subcommands: COPYTRC and MERGE.

Use consolidated or merged trace output to show the chronology of events around the time of an error. Specify start and stop times for the merge to see events beginning a little before the error occurred and ending a little after. On the CTRACE and GTFTRACE subcommands, specify the jobs and address space identifiers (ASID) involved, so that the merged output contains only pertinent trace records.

Merging is most useful when several components are running traces; the system can also be running a GTF trace. Each component puts its trace records into its own buffers independently. GTF is independent from all of the component traces. You can merge these separate records into one chronological sequence to make diagnosis easier.

See *z/OS MVS IPCS Commands* for more information about COPYTRC and MERGE.

Combining and extracting GTF output

Use the IPCS COPYTRC subcommand to do one or more of the following:

- Consolidate GTF trace data into one data set from:
  - Multiple GTF data sets
  - Multiple GTF data sets, dumps, or both
  - More than one system

- Extract GTF trace data from SVC dumps and stand-alone dumps
- Extract from merged data the GTF trace data for a specified list of systems

If you have GTF set up to write data for one system to multiple data sets, you can use the IPCS COPYTRC subcommand to consolidate the data into one data set. You should do this before you consolidate GTF data from multiple systems with the MERGE or COPYTRC subcommands.

Figure 93 on page 240 shows an example of a GTF cataloged procedure with 3 data sets defined for GTF data from system SYS01.

```
//GTFABC    PROC   MEMBER=GTFPARM
//IEFPROC   EXEC   PGM=AHLGTF,REGION=2880K,TIME=1440,
//                 PARM=('MODE=EXT,DEBUG=NO')
//IEFRDER   DD     DSNAME=SYS1.GTFTRC,UNIT=SYSDA,
//                 SPACE=(4096,20),DISP=(NEW,KEEP)
//SYSLIB    DD     DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//GTFOUT1   DD     DSNAME=SYS01.DSN1,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT2   DD     DSNAME=SYS01.DSN2,UNIT=&DEVICE,DISP=&DSPS;
//GTFOUT3   DD     DSNAME=SYS01.DSN3,UNIT=&DEVICE,DISP=&DSPS;
```

Figure 93. Example: Consolidating GTF output from multiple data sets

From IPCS, issue the following command to consolidate the data from the data sets defined in the cataloged procedure into one data set, GTF.SYS01:

```
COPYTRC TYPE(GTF)
        INDATASET(SYS01.DSN1,SYS01.DSN2,SYS01.DSN3)
        OUTDATASET(GTF.SYS01)
```

In Figure 94 on page 240, the COPYTRC subcommand is used to consolidate data from 3 systems, in data sets GTF.SYS01, GTF.SYS02, and GTF.SYS03, into one output data set, GTF.ALLSYS.

```
COPYTRC TYPE(GTF)
        INDATASET(GTF.SYS01,GTF.SYS02,GTF.SYS03)
        OUTDATASET(GTF.ALLSYS)
```

Figure 94. Example: Consolidating GTF output from multiple systems

Note that just one data set per system was used on the COPYTRC command. For best results, if you have more than one data set for a system, you should first consolidate those using a separate instance of the COPYTRC command.

To format the output data set for GTF data, issue the following IPCS subcommand:

```
GTFTRACE DSNAME(ALLSYS)
```

Merging trace output

Use the IPCS MERGE subcommand to merge multiple traces into one chronological sequence. The traces can be all of the following:

- Component traces from the same dump on direct access storage (DASD)
- Component traces from different dumps on DASD
- GTF trace records from a dump or data set and on tape or DASD

In Figure 95 on page 241, the MERGE subcommand is used to consolidate and format data from 3 systems, in data sets GTF.SYS04, GTF.SYS05, and GTF.SYS06, into one chronological sequence in output data set, GTF.SYSALL.

```
MERGE
GTFTRACE DSNAME(GTF.SYS04)
GTFTRACE DSNAME(GTF.SYS05)
GTFTRACE DSNAME(GTF.SYS06)
MERGEEND
```

Figure 95. Example: Merging GTF output from multiple systems

Reading GTF output

This topic shows the format of the trace records that GTF creates. When you select your tracing options carefully, GTF provides detailed information about the system and user events where your problem lies, making it easier to diagnose.

This section contains the following topics:

- "Formatted GTF trace output" on page 243 which has information about trace records formatted by the IPCS GTFTRACE subcommand.
- "Unformatted GTF trace output" on page 297 which has information about unformatted trace records.

"Trace options" on page 241 lists the GTF trace options and the trace records they generate in GTF trace output. Use this list to correlate the options you selected with their associated trace records. Some trace options in the table do not have trace records associated with them:

- ASIDP - Specifies that GTF trace only events from the select address spaces.
- JOBNAMEP - Specifies that GTF trace only events in selected jobs.
- END - Specifies the end of prompting keyword values specified.
- TRC - Specifies that GTF tracing includes the GTF address space.

Trace options

Trace Options
: **Trace Record Identifier**

ASIDP
: N/A

CCW
: CCW, TCW, Synch I/O

CCWP
: CCW, TCW

CSCH
: CSCH

DSP
: DSP, LSR, SDSP, SRB

END
: N/A

EXT
: EXT

HSCH
: HSCH

IO
: EOS, INTG, IO, IOCS, SYNE

IOP
: EOS, INTG, IO, IOCS, SYNE

IOX
IOX

IOXP
IOXP

JOBNAMEP
N/A

MSCH
MSCH

PCI
PCI

PCIE
PCILG, PCISTG, ADINT, PCIDMX

PFIDP
N/A

PI
PGM, PI

PIP
PGM, PI

RNIO
RNIO

RR
FRR, STAE

SIO
RSCH, SSCH

SIOP
RSCH, SSCH

SLIP
SLIP

SRM
SRM

SSCH
RSCH, SSCH, SYNS

SSCHP
RSCH, SSCH, SYNS

SVC
SVC

SVCP
SVC

SYS
CSCH, EOS, EXT, FRR, HSCH, INTG, IO, IOCS, MSCH, PGM, PI, SSCH, STAE, SVC, SYNE, SYNS

SYSM
CSCH, EOS, EXT, FRR, HSCH, INTG, IO, IOCS, MSCH, PGM, PI, SSCH, STAE, SVC, SYNE, SYNS

SYSP
CSCH, EOS, EXT, FRR, HSCH, INTG, IO, IOCS, MSCH, PGM, PI, SSCH, STAE, SVC, SYNE, SYNS

TRC
N/A

USR
USR

USRP
USR

XSCH
XSCH

Formatted GTF trace output

This topic describes GTF trace output records formatted by the IPCS GTFTRACE subcommand. In each formatted record, the length of each field is indicated by the number of characters. The characters indicate the type of data in the field, as follows:

c
Character

d
Decimal

h
Hexadecimal

x
Variable information

y
Variable information

The CCW trace record format uses additional letters to distinguish parts of fields.

A trace record can contain indicators to denote unusual conditions that occurred while GTF was tracing the event for the record. The indicators are:

N/A	Not applicable. The field does not apply in this record. In a 2-byte field, not applicable appears as N/.
U/A	Unavailable. GTF could not gather the information. In a 2-byte field, unavailable appears as U/.
PPPPPPPP	Unavailable because of a page fault encountered while GTF was gathering the data (SVC only).
SSSSSSSS	Unavailable because of security considerations (SVC only).
********	Unavailable because of an error that occurred while GTF was gathering the data or due to the data being paged out.
X'EEEE'	Unavailable because of a severe error that occurred while GTF was gathering the data. This value appears in the first 2 data bytes of the trace record. The contents of the trace record are unpredictable.

Trace record identifiers

Each trace record has an identifier to indicate the type of record. Table 40 on page 243 lists the identifiers alphabetically and gives the page that shows the format for the record.

Table 40. Summary of trace record identifiers

Trace Record Identifier	GTF Trace Record	Parameter in SYS1.PARMLIB Member or Operator Reply	For format, see:
****	Time stamp		"Time stamp records" on page 247
****	Lost event		"Lost event records" on page 248
ADINT	Adapter interruption	PCIE	"ADINT trace records" on page 248
CCW	Non-zHPF channel program	CCW	"CCW trace records" on page 249
CSCH	Clear subchannel operation	CSCH, SYS, SYSM, SYSP	"CSCH and HSCH trace records" on page 251

Table 40. Summary of trace record identifiers (continued)

Trace Record Identifier	GTF Trace Record	Parameter in SYS1.PARMLIB Member or Operator Reply	For format, see:
DSP	Task dispatch	DSP	"DSP and SDSP trace records" on page 252
EOS	End-of-sense interruption	IO, IOP, SYS, SYSM, SYSP	"EOS, INTG, IO, IOCS, and PCI trace records" on page 253
EXT	General external interruption	EXT, SYS, SYSM, SYSP	"EXT trace records" on page 257
FRR	Functional recovery routine return	RR, SYS, SYSM, SYSP	"FRR trace records" on page 259
HEXFORMAT	Unformatted trace event		"HEXFORMAT, SUBSYS, and SYSTEM trace records" on page 260
HSCH	Halt subchannel operation	HSCH, SYS, SYSM, SYSP	"CSCH and HSCH trace records" on page 251
INTG	Interrogate input/output interruption	IO, IOP, SYS, SYSM, SYSP	"EOS, INTG, IO, IOCS, and PCI trace records" on page 253
IO	Input/output interruption	IO, IOP, SYS, SYSM, SYSP	"EOS, INTG, IO, IOCS, and PCI trace records" on page 253
IOCS	Input/output interruption with concurrent sense	IO, IOP, SYS, SYSM, SYSP	"EOS, INTG, IO, IOCS, and PCI trace records" on page 253
IOX	Input/output interruption summary record format	IOX, IOXP, SYS, SYSM, SYSP	"IOX trace records" on page 261
LSR	Local supervisor routine dispatch	DSP	"LSR trace records" on page 264
MSCH	Modify subchannel operation	MSCH, SYS, SYSM, SYSP	"MSCH trace records" on page 265
PCI	Program-controlled input/output interruption	PCI	"EOS, INTG, IO, IOCS, and PCI trace records" on page 253
PCIDMX	PCIE de-multiplexing event	PCIE	"PCIDMX trace records" on page 265
PCILG	PCI load instruction	PCIE	"PCILG trace records" on page 266
PCISTG	PCI store instruction	PCIE	"PCISTG trace records" on page 267
PGM	Program interruption	PI, PIP, SYS, SYSM, SYSP	"PGM and PI trace records" on page 268
PI	Program interruption	PI, PIP, SYS, SYSM, SYSP	"PGM and PI trace records" on page 268
RNIO	VTAM remote network input/output event	RNIO	"RNIO trace records" on page 270
RSCH	Resume subchannel	SSCH, SSCHP	"RSCH trace records" on page 270
SDSP	Task re-dispatch	DSP	"DSP and SDSP trace records" on page 252
SLIP	SLIP program event interruption	SLIP	"SLIP trace records" on page 271
SRB	Service request block routine dispatch or re-dispatch	DSP	"SRB trace records" on page 277
SRM	System resources manager return	SRM	"SRM trace records" on page 278

Table 40. Summary of trace record identifiers (continued)

Trace Record Identifier	GTF Trace Record	Parameter in SYS1.PARMLIB Member or Operator Reply	For format, see:
SSCH	Start subchannel operation	SSCH, SSCHP, SYS, SYSM, SYSP	"SSCH trace records" on page 279
STAE	STAE or ESTAE recovery routine return	RR, SYS, SYSM, SYSP	"STAE trace records" on page 280
SUBSYS	Unformatted trace event		"HEXFORMAT, SUBSYS, and SYSTEM trace records" on page 260
SVC	Supervisor call interruption	SVC, SVCP, SYS, SYSM, SYSP	"SVC and SVCR trace records" on page 281
SVCR	Supervisor call exit	SVC, SVCP, SYS, SYSM, SYSP	"SVC and SVCR trace records" on page 281
SYNE	Synchronous I/O end	IO, IOP, SYS, SYSM, SYSP	"SYNS and SYNE trace records" on page 283
SYNS	Synchronous I/O start	SSCH, SSCHP, SYS, SYSM, SYSP	"SYNS and SYNE trace records" on page 283
SYNCH I/O	Synchronous I/O request information	CCW	"SYNCH I/O trace records" on page 285
SYSTEM	Unformatted trace event		"HEXFORMAT, SUBSYS, and SYSTEM trace records" on page 260
TCW	zHPF channel program	CCW	"TCW trace records" on page 287
USR	User event	USR, USRP	"USR trace records" on page 289
XSCH	Cancel subchannel operation	XSCH, SYS, SYSM, SYSP	"XSCH trace record" on page 294

Example of formatted GTF trace output

This section contains screen images that show GTF records. IPCS produced the screens from an example dump. These records are in comprehensive format and are time stamped.

The GTFTRACE subcommand was issued on the IPCS Subcommand Entry panel shown in Figure 96 on page 246.

```
IPCS OUTPUT STREAM ------------------------------------------ LINE 0 COLS 1 78
COMMAND ===>                                                  SCROLL ===> CSR

 ****************************** TOP OF DATA **********************************

   **** GTFTRACE DISPLAY OPTIONS IN EFFECT ****
 SSCH=ALL  IO=ALL  CCW=SI
 SVC=ALL  PI=ALL
 EXT  RNIO  SRM  RR  DSP  SLIP
 **** GTF DATA COLLECTION OPTIONS IN EFFECT: ****
 Minimum tracing for IO, SSCH, SVC, PI, EXT, and FRR events
 All GTRACE events requested
 All events associated with the execution should be traced
 All DISPATCHER events traced
 PCI events are to be traced
 System resource manager events traced

              **** GTF TRACING ENVIRONMENT ****
        Release: SP7.8.0   FMID: HBB7780   System name: FIRST
        CPU Model: 2097  Version: FF  Serial no. 170067
   SVC   CODE.... 002      ASCB.... 00FB8100 CPU..... 0000
                           PSW..... 07041000 80000002 00000000 0557C0D0
                           TCB..... 006F0BF8 R15..... 00FDCAEA R0...... 00000000
                           R1...... 806F2BE0
                  GMT-01/09/2011 00:21:13.668101  LOC-01/08/2011 19:21:13.668101
   SVCR  CODE.... 002      ASCB.... 00FB8100 CPU..... 0000
                           PSW..... 07041000 80000002 00000000 0557C0D0
                           TCB..... 006F0BF8 R15..... 813BCF7C R0...... 813BCF7C
                           R1...... 02514607
                  GMT-01/09/2011 00:21:13.668143  LOC-01/08/2011 19:21:13.668143
```

Figure 96. Example: IPCS subcommand entry panel for GTFTRACE

Figure 97 on page 246 shows records for the start subchannel operation (SSCH) event.

```
IPCS OUTPUT STREAM ------------------------------------------ LINE 0 COLS 1 78
COMMAND ===>                                                  SCROLL ===> CSR
 SRB   ASCB.... 00FC1500 CPU..... 0000
                         PSW..... 07040000 80000000 00000022 0217E07C
                         R15..... 8217E050 SRB..... I
                         TYPE.... INITIAL DISPATCH OF SRB
                GMT-01/09/2011 00:39:44.155668  LOC-01/08/2011 19:39:44.155668
 DSP   ASCB.... 00FC1500 CPU..... 0000
                         PSW..... 07040000 80000078 00000000 013AB828
                         TCB..... 006FF148 R15..... 813AB828 R0...... 006E1BC0
                         R1...... 0217E050
                GMT-01/09/2011 00:39:44.155742  LOC-01/08/2011 19:39:44.155742
 SSCH.... 00982    ASCB.... 00FB8880 CPUID... 0000     JOBN.... JES2
                   RST..... 0FC27620 VST..... 02626620 DSID.... 006DCFEC
                   CC...... 00       SEEKA... 00000000 15000D07
                   GPMSK... 00       OPT..... 00       FMSK.... 00
                   DVRID... 02       IOSLVL.. 01       UCBLVL.. 01
                   UCBWGT.. 00       BASE.... 00982
                   ORB..... 00F1D4E0 13C2D081 0F1FDC68 0000FE00 00000000
                            00000000 00000000 00000000
               GMT-01/09/2011 00:21:32.948888  LOC-01/08/2011 19:21:32.948888
```

Figure 97. Example: GTF record for SSCH events

The screen images in Figure 98 on page 247 and Figure 98 on page 247 show records for two input/output (IO) interruption events. The last two rows in the I/O statistics section will only appear for zHPF I/O events.

```
  IO...... 00982    ASCB.... 00FB8880 CPUID... 0000     JOBN.... JES2
                    PSW..... 07041000 80000000 00000000 05722F66
                    IRB..... 10C04007 0FC27360 0C000000 00800002 00000000
                    TCB..... 006FF368 SENSE... N/A      FLA..... 40
                    OPT..... 00       DVRID... 02       IOSLVL.. 01
                    UCBLVL.. 01       UCBWGT.. 00       BASE.... 00982

                    I/O Statistics:
                    Connect. 00000000 Pending. 01BE0000 Discon.. 01A80000
                    CUQ..... 00000000 DAO..... 00000000 Devbsy.. 00000000
                    ICMR.... 00000000 StartCt. 00000000 SamplCt. 0BEF0000
                    ZTotdev. D7C20000 ZDefer.. 01310000 ZCUQ.... 00000000
                    ZDevBsy. 00000000 ZDAO.... 00000000
                    IntrDly. hhhhhhhh
                 GMT-01/09/2011 00:21:32.944548  LOC-01/08/2011 19:21:32.944548

  **** GTFTRACE DISPLAY OPTIONS IN EFFECT ****
SSCH=ALL  IO=ALL  CCW=SI
SVC=ALL  PI=ALL
EXT  RNIO  SRM  RR  DSP  SLIP
 **** GTF DATA COLLECTION OPTIONS IN EFFECT: ****
IO filtering requested
CCW trace prompting
IO CCW records
SSCH CCW records
All records timestamped
SSCH prompting
*** DATE/TIME: GMT-01/09/11 21:14:10    LOC-01/09/11 21:14:10.009827
```

Figure 98. Example: GTF records for IO interruption events

```
              GMT-01/09/11 21:14:10.009803    LOC-01/09/11 21:14:10.009803
 PCI..... 0000     ASCB.... 00000000 CPUID... 0000     JOBN.... ........
                   PSW..... 00000000 00000000 00000000 00000000
                   IRB..... 00000000 00000000 00000000 00000000 00000000
                   TCB..... 00000000 SENSE... 0000     FLA..... 00
                   OPT..... 00       DVRID... 00       IOSLVL.. 00
                   UCBLVL.. 00       UCBWGT.. 00       BASE.... 00000
              GMT-01/09/11 21:14:10.009955    LOC-01/09/11 21:14:10.009955
 EOS..... 10000    ASCB.... 00000000 CPUID... 8861     JOBN.... ........
                   PSW..... 00000000 00000000 00000000 00000000
                   IRB..... 00000000 00000000 00000000 00000000 00000000
                   TCB..... 00000000 SENSE... 0000     FLA..... 00
                   OPT..... 00       DVRID... 00       IOSLVL.. 00
                   UCBLVL.. 00       UCBWGT.. 00       BASE.... 00000
              GMT-01/09/11 21:14:10.078486    LOC-01/09/11 21:14:10.078486
 CSCH.... 10000    ASCB.... 00000000 CPUID... 0000     JOBN.... ........
                   DEV..... 0000     SFLS.... 0000     SID..... 00000000
                   CC...... 00       DVRID... 00       ARDID... 00
                   IOSLVL.. 00       UCBLVL.. 00       UCBWGT.. 00
                   BASE.... 00000
              GMT-01/09/11 21:14:10.099752    LOC-01/09/11 21:14:10.099752
 HSCH.... 10000    ASCB.... 00000000 CPUID... 8861     JOBN.... ........
                   DEV..... 0000     SFLS.... 0000     SID..... 00000000
                   CC...... 00       DVRID... 00       ARDID... 00
                   IOSLVL.. 00       UCBLVL.. 00       UCBWGT.. 00
                   BASE.... 00000
              GMT-01/09/11 21:14:10.119803    LOC-01/09/11 21:14:10.119803
  ****************************** END OF DATA ***********************************
```

Figure 99. Example: More GTF records for IO interruption events

Formatted trace records for events

The following sections describe different types of formatted trace records.

Time stamp records

Time stamp records mark the time an event occurred.

Record Format After Each Trace Record

```
GMT-mm/dd/yy hh:mm:ss:dddddd    LOC-mm/dd/yy hh:mm:ss.dddddd
```

GMT-mm/dd/yy hh:mm:ss
: Month/day/year and Greenwich mean time given in hour:minute:second format.

LOC-mm/dd/yy hh:mm:ss.dddddd
Month/day/year and local time given in hour:minute:second.microsecond format. Local time is calculated using a time zone offset established when tracing starts.

Source index records

Source index records are added when GTF trace records are consolidate using the IPCS COPYTRC subcommand. The records identify the system that produced the GTF trace record.

Record Format After Each Trace Record, if the GTF trace records are consolidated with the IPCS COPYTRC subcommand:

```
   SOURCE INDEX: 01
```

The source index record indicates that the GTF trace record was produced by the system with identifier 01. Identifiers include the system name and the trace options in effect for that system. The identifiers are listed at the top of the IPCS report.

Lost event records

A lost event record indicates that GTF lost the trace records for one or more events because of an error or overflow of the trace buffer.

Record Format When GTF Trace Buffer is Lost due to Error

```
 **** ONE TRACE BUFFER LOST TIME hh.mm.ss.dddddd
```

hh.mm.ss.dddddd
The time of day (hour.minute.second.microsecond) when GTF placed the first trace record in the buffer.

The size of the GTF trace buffer is:

- Equal to the blocksize used by GTF when writing the trace data, if GTF is writing the trace records to a data set on a direct access storage device (DASD). The system displays the blocksize in message AHL906I. If the records are to be written to a data set, the system issues message AHL906I after starting GTF.
- 32,760 bytes, if GTF is writing the trace records to a data set on tape.
- 32,768 bytes, if GTF is writing the trace records only into internal trace buffers.

Record Format for Number of Trace Events Lost due to Errors or Trace Buffer Overflow

```
 ****** LOST EVENTS NUM dddddddddddd LOCAL TIME mm/dd/yyyy hh.mm.ss.nnnnnn ***
```

dddddddddd
The number of lost events

mm/dd/yyyy
The date (in month/day/year format) when GTF placed the first trace record in the current trace buffer.

hh.mm.ss.dddddd
The time of day (hour.minute.second.microsecond) when GTF placed the first trace record in the current trace buffer.

ADINT trace records

ADINT records represent adapter interruptions.

```
ADINT...         ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                 PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                 ISC..... hh       AISM.... hh
```

ASCB hhhhhhhh
Address of the ASCB for the address space that was active when the adapter interruption occurred on this CPU.

CPUID hhhh
Address of the processor on which the adapter interruption occurred.

JOBN cccccccc
Name of the job associated with the address space that was active when the adapter interruption occurred on this CPU.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word (PSW) stored when the interruption occurred.

ISC hh
Interruption subclass.

AISM hh
Adapter source identification mask.

CCW trace records

A CCW record represents the processing of a non-zHPF channel program. CCW trace records appear following EOS, IO, IOCS PCI, RSCH, or SSCH trace records; they do not appear alone. Any of the formats can appear in any combination in one CCW trace record.

```
  CCW CHAIN   FORMAT d          ccc               DEV..... hhhh
  ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
  Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd | cccccccc |
  Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd | cccccccc |
  Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
                              .
                              --Back half of split data--
                              .
                             dddddddd dddddddd | cccccccc |
                             dddddddd dddddddd | cccccccc |
  Fhhhhhhh ---CCW-- ---CCW-- dddddddd dddddddd | cccccccc |
 IDAW hhhhhhhh_hhhhhhhh hhhh hhhhhhhh hhhhhhhh | cccccccc |
                             hhhhhhhh hhhhhhhh | cccccccc |
MIDAW hhhhhhhh hhhhhhhh hhhh hhhhhhhh hhhhhhhh | cccccccc |
      hhhhhhhh_hhhhhhhh      hhhhhhhh hhhhhhhh | cccccccc |
 IOSB hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
 EWAx hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
```

FORMAT d ccc
Format (d) and type of trace event (ccc): EOS, IO, IOCS, PCI, RSCH, or SSCH. Format is either zero or 1.

DEV shhhh
DEV snnnn
Device number from the UCBCHAN field of the UCB. This number is qualified with the subchannel identifier (UCBSID).

ASCB hhhhhhhh
Same as the ASCB field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.

CPU hhhh
Same as the CPU ID field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.

JOBN cccccccc
Same as the job named (JOBN) field in the IO, IOCS, SSCH, RSCH, PCI, or EOS base record.

Fhhhhhhh
Fullword address of the CCW. If the high order bit of the address is on, this is the real address of the CCW, otherwise this is the virtual address of the address of the CCW.

---CCW--
Is the CCW command. The command is either a format 0 or format 1 CCW.

- Format 0 CCW is in the format ooaaaaaa ffuubbbb
- Format 1 CCW is in the format ooffbbbb aaaaaaaa

oo
op code.

aaaaaa
Real address of data associated with the CCW. If indirect address words (IDAWs) are present, this is the address of the IDAW list.

aaaaaaaa
Fullword real address of data associated with the CCW. If IDAWs are present, this is the address of the IDAW list.

ff
CCW flags; if this flag is .... .1.. , then this indicates that an IDAW list is present. If this flag is .... ..1., then a suspend of the channel program was requested. If this flag is ....1 , then a modified indirect addressing word list is present.

uu
Not used by hardware; could contain a nonzero character.

bbbb
Byte count.

dddddddd dddddddd | cccccccc |
Information transferred by the CCW. If there is not a series of dashes in this field, then all transferred data is displayed in four byte sections.

--Back half of split data--
Indicates there were more bytes of information transferred than were specified on the START command. The default value is 20 bytes, but you can specify the number of bytes to be shown. The specified value is halved; for an odd number, the larger section is shown first. The first section of data displayed comes from the beginning of the buffer from which the data was transferred. The last section comes from the end of the buffer.

IDAW hhhhhhhh or hhhhhhhh_hhhhhhhh hhhh
Contents of the IDAW, a fullword real address for 31–bit IDAWs or a doubleword real address for 64–bit IDAWs, followed by a halfword specifying the length of the data at that address. The data at the address follows the halfword length. The *hhhhhhhh_hhhhhhhh* version of this parameter specifies 64–bit.

MIDAW hhhhhhhh hhhhhhhh hhhh hhhhhhhh_hhhhhhhh
The modified indirect addressing word (MIDAW), which is 16 bytes, formatted in the GTF trace with the first 8 bytes on the first line containing the flags and data length and the second line containing the 64 bit data address. The length of the data is replicated after the first 8 bytes of MIDAW data to make it easier to read and maintain consistency with the IDAW format. The data at the address follows the halfword length. The data for a MIDAW is not formatted if the skip indicator is on.

IOSB hhhhhhhh
Fullword virtual address of the IOSB followed by the contents of the IOSB. The fullword at offset X'34' of the IOSB points to an error recover procedure work area (EWA), or is zero. The EWA is traced and documented directly below the IOSB and is formatted in the same manner as the IOSB.

EWAx hhhhhhhh
Fullword virtual address of the error recovery procedure work area, followed by the contents of EWA.

CSCH and HSCH trace records

CSCH and HSCH records represent a clear subchannel operation and a halt subchannel operation.

Record Formats

```
CSCH.... shhhh   ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                DEV..... hhhh     SFLS.... hhhh     SID..... hhhhhhhh
                CC...... hh       DVRID... hh       ARDID... hh
                IOSLVL.. hh       UCBLVL.. hh
                UCBWGT.. hh       BASE.... shhhh
```

```
HSCH.... shhhh   ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                DEV..... hhhh     SFLS.... hhhh     SID..... hhhhhhhh
                CC...... hh       DVRID... hh       ARDID... hh
                IOSLVL.. hh       UCBLVL.. hh
                UCBWGT.. hh       BASE.... shhhh
```

CSCH shhhh
HSCH shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that started the I/O operation.

CPUID hhhh
Address of the processor on which the I/O operation started

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the task that requested the I/O operation

N/A
No job is associated with the requested I/O

DEV hhhh
Device number from the UCBCHAN field of the UCB.

SFLS hhhh
Start flags from the UCBSFLS field of the UCB.

SID hhhhhhhh
Subchannel ID from the UCBSID field of the UCB.

CC hh
CSCH or HSCH condition code in bits 2 - 3.

DVRID hh
Driver ID value from the IOSDVRID field of the IOSB.

ARDID hh
One of the following:

hh
Associated request driver ID from the IOSDVRID field of the IOSB

U/
Unavailable because the IOQ was unavailable

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

DSP and SDSP trace records

A DSP record represents dispatching of a task. An SDSP record represents re-dispatching of a task after an SVC interruption. SDSP interruptions also build SVC exit records with label SDSP. When both DSP and SVC options are in effect, the SVCR format of trace record is produced by IPCS.

If the trace data contains an SVC exit record, the label that appears in the formatted output will depend on the options selected during IPCS.

1. If the SVC option is selected in the IPCS dialog, the SVC exit record and the SVC number will appear with the label SVCR.
2. If only the DSP option is chosen in the IPCS dialog, the formatted output record will remain unchanged; DSP and SDSP labels will appear in the formatted output and no SVC number is present.
3. If both DSP and SVC options are active in IPCS, the SVCR along with SVC number will appear.

It can be concluded, if SVC is one of the options selected during IPCS formatting, all SVC exit records will appear with label SVCR along with SVC number.

Minimal Trace Record Formats

```
DSP   ASCB.... hhhhhhhh CPU..... hhhh     PSW..... hhhhhhhh hhhhhhhh
                        PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
                        R1...... hhhhhhhh
```

```
SDSP  ASCB.... hhhhhhhh CPU..... hhhh     PSW..... hhhhhhhh hhhhhhhh
                        PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
                        R1...... hhhhhhhh
```

Comprehensive Trace Record Formats

```
 DSP          ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              DSP-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
              TCB..... hhhhhhhh MODN.... yyyyyyyy R15..... hhhhhhhh
              R0...... hhhhhhhh R1...... hhhhhhhh
```

```
 SDSP         ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              DSP-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
              TCB..... hhhhhhhh MODN.... yyyyyyyy R15..... hhhhhhhh
              R0...... hhhhhhhh R1...... hhhhhhhh
```

ASCB hhhhhhhh
Address of address space control block.

CPU hhhh
Address of processor on which the task is dispatched.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
DSP-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word under which the task is dispatched.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the task being dispatched

N/A
The record is for a system or started task

PPPPPPPP
A page fault occurred

****
An internal error occurred

TCB hhhhhhhh
Address of the task control block.

R15 hhhhhhhh
R0 hhhhhhhh
R1 hhhhhhhh
Data that will appear in general registers 15, 0, and 1 when the task is dispatched.

MODN cccccccc
cccccccc is one of the following:

mod_name
The name of a module that will receive control when the task is dispatched.

WAITTCB
Indicates that the system wait task is about to be dispatched.

SVC–T2
Indicates that a type 2 SVC routine that resides in the nucleus is about to be dispatched.

SVC–RES
Indicates that a type 3 SVC routine or the first load module of a type 4 SVC routine is about to be dispatched. The routine is located in the pageable link pack area (PLPA).

SVC–cccc
Indicates that the second or subsequent load module of a type 4 SVC routine is about to be dispatched. The module is located in the fixed or pageable link pack area (LPA). The last four characters of the module name are cccc.

****IRB*****
Indicates that an asynchronous routine with an associated interruption request block (IRB) is about to be dispatched. No module name is available.

***ccccccc**
Indicates that error fetch is in the process of loading an error recovery module. The last seven characters of the module name are ccccccc.

PPPPPPPP
A page fault occurred

****
An internal error occurred

EOS, INTG, IO, IOCS, and PCI trace records

EOS records represent an end of sense interruption.

```
EOS..... shhhh  ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                IRB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh SENSE... hhhh     FLA..... hh
                OPT..... hh       DVRID... hh       IOSLVL.. hh
                UCBLVL.. hh       UCBWGT.. hh       BASE.... shhhh

                I/O Statistics:
                Connect. hhhhhhhh Pending. hhhhhhhh Discon.. hhhhhhhh
                CUQ..... hhhhhhhh DAO..... hhhhhhhh Devbsy.. hhhhhhhh
                ICMR.... hhhhhhhh StartCt. hhhhhhhh SamplCt. hhhhhhhh
                ZTotdev. hhhhhhhh ZDefer.. hhhhhhhh ZCUQ.... hhhhhhhh
                ZDevBsy. hhhhhhhh ZDAO.... hhhhhhhh
                IntrDly. hhhhhhhh IOSQTim. hhhhhhhh
```

INTG records represent an input/output (I/O) interruption that is used to signal the completion of a zHPF interrogate request.

```
INTG.... shhhh  ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                IRB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh SENSE... hhhh     FLA..... hh
                OPT..... hh       DVRID... hh       IOSLVL.. hh
                UCBLVL.. hh       UCBWGT.. hh       BASE.... shhhh
```

IO records represent an I/O interruption.

```
IO...... shhhh  ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                IRB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh SENSE... hhhh     FLA..... hh
                OPT..... hh       DVRID... hh       IOSLVL.. hh
                UCBLVL.. hh       UCBWGT.. hh       BASE.... shhhh

                I/O Statistics:
                Connect. hhhhhhhh Pending. hhhhhhhh Discon.. hhhhhhhh
                CUQ..... hhhhhhhh DAO..... hhhhhhhh Devbsy.. hhhhhhhh
                ICMR.... hhhhhhhh StartCt. hhhhhhhh SamplCt. hhhhhhhh
                ZTotdev. hhhhhhhh ZDefer.. hhhhhhhh ZCUQ.... hhhhhhhh
                ZDevBsy. hhhhhhhh ZDAO.... hhhhhhhh
                IntrDly. hhhhhhhh IOSQTim. hhhhhhhh
```

IOCS records represent an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility.

```
IOCS.... shhhh  ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                IRB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh SENSE... hhhh     FLA..... hh
                OPT..... hh       DVRID... hh       IOSLVL.. hh
                UCBLVL.. hh       UCBWGT.. hh       BASE.... shhhh

                I/O Statistics:
                Connect. hhhhhhhh Pending. hhhhhhhh Discon.. hhhhhhhh
                CUQ..... hhhhhhhh DAO..... hhhhhhhh Devbsy.. hhhhhhhh
                ICMR.... hhhhhhhh StartCt. hhhhhhhh SamplCt. hhhhhhhh
                ZTotdev. hhhhhhhh ZDefer.. hhhhhhhh ZCUQ.... hhhhhhhh
                ZDevBsy. hhhhhhhh ZDAO.... hhhhhhhh
                IntrDly. hhhhhhhh IOSQTim. hhhhhhhh
```

PCI records represent a program-controlled interruption.

```
PCI..... hhhh   ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                IRB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh SENSE... hhhh     FLA..... hh
                OPT..... hh       DVRID... hh       IOSLVL.. hh
                UCBLVL.. hh       UCBWGT.. hh       BASE.... shhhh

                I/O Statistics:
                Connect. hhhhhhhh Pending. hhhhhhhh Discon.. hhhhhhhh
                CUQ..... hhhhhhhh DAO..... hhhhhhhh Devbsy.. hhhhhhhh
                ICMR.... hhhhhhhh StartCt. hhhhhhhh SamplCt. hhhhhhhh
                ZTotdev. hhhhhhhh ZDefer.. hhhhhhhh ZCUQ.... hhhhhhhh
                ZDevBsy. hhhhhhhh ZDAO.... hhhhhhhh
                IntrDly. hhhhhhhh IOSQTim. hhhhhhhh
```

EOS shhhh
INTG shhhh
IO shhhh
IOCS shhhh
PCI hhhh

The device number from the UCBCHAN field of the unit control block (UCB), which includes the subchannel set identifier when appropriate.

ASCB {hhhhhhhh|U/A}

One of the following:

hhhhhhhh

Address of the address space control block (ASCB) for the address space that started the I/O operation.

U/A

Unavailable because the I/O supervisor block (IOSB) control block is unavailable.

CPU hhhh

Address of the processor on which the interruption occurred.

JOBN {cccccccc|N/A|U/A}

One of the following:

cccccccc

Name of the job associated with the task that requested the I/O operation.

N/A

Not applicable.

U/A

Unavailable because the IOSB control block is unavailable.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh

Program status word (PSW) stored when the interruption occurred.

IRB (see explanation)

For the EOS, IO, and PCI trace records, this field contains the first five words, in hexadecimal, of the interruption response block (IRB) operand of the Test Subchannel (TSCH) instruction.

For the IOCS trace record, this field contains the first 16 words, in hexadecimal, of the interruption response block operand of the TSCH instruction. (Note that this IRB is not the interruption request block indicated as `**IRB***` in a DSP trace record.)

TCB {hhhhhhhh|N/A|U/A}

One of the following:

hhhhhhhh

Address of the TCB for the task that requested the I/O operation.

N/A

Not applicable.

U/A
Unavailable because the IOSB control block is unavailable.

SENSE {hhhh|N/A|U/A}
One of the following:

hhhh
First 2 sense bytes from the IOSSNS field of the IOSB.

N/A
Not applicable.

U/A
Unavailable because the IOSB control block is unavailable.

FLA {hh|U/A}
One of the following:

hh
Flag byte from the IOSFLA field of the IOSB.

U/A
Unavailable because the IOSB control block is unavailable.

OPT {hh|U/A}
One of the following:

hh
IOSB options byte from the IOSOPT field of the IOSB.

U/A
Unavailable because the IOSB control block is unavailable.

DVRID {hh|U/A}
One of the following:

hh
Driver identifier from the IOSDVRID field of the IOSB.

U/A
Unavailable because the IOSB control block is unavailable.

IOSLVL {hh|U/A}
One of the following:

hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

U/A
Unavailable because the IOSB control block is unavailable.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

Connect hhhhhhhh
The device connect time for this I/O request, in units of 0.5 microseconds.

Pending hhhhhhhh
The function pending time for this I/O request, in units of 0.5 microseconds.

Discon hhhhhhhh
The device disconnect time for this I/O request, in units of 0.5 microseconds.

CUQ hhhhhhhh
The control unit queueing time for this I/O request, in units of 0.5 microseconds.

DAO hhhhhhhh
The device active only time for this I/O request, in units of 0.5 microseconds.

Devbsy hhhhhhhh
The device busy time for this I/O request, in units of 0.5 microseconds.

ICMR hhhhhhhh
The initial command response time for this I/O request, in units of 0.5 microseconds.

StartCt hhhhhhhh
The number of SSCH/RSCH instructions that were issued for the device while GTF trace was active.

SamplCt hhhhhhhh
The number of SSCH/RSCH instructions for which data was collected for the device.

ZTotdev hhhhhhhh
The total device time for this I/O request, in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZDefer hhhhhhhh
The control unit defer time for this I/O request, in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZCUQ hhhhhhhh
The control unit queue time for this I/O request, in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZDevBsy hhhhhhhh
The device busy time for this I/O request, in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

ZDAO hhhhhhhh
The device active only time for this I/O request, in units of 1 microseconds. This information is reported only for zHPF I/O requests when it is provided by the device.

IntrDly hhhhhhhh
The total interrupt delay time for all I/O requests, in units of 128 microseconds.

IOSQTim hhhhhhhh
The measured I/O queuing time for this I/O request, in units of 1 microseconds.

EXT trace records

An EXT record represents a general external interruption.

Minimal Trace Record Format

```
EXT  CODE.... hhhh  ASCB.... hhhhhhhh CPU..... hhhh
                    PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    TCB..... hhhhhhhh ccc-TCB. hhhhhhhh
```

Comprehensive Trace Record Format

```
EXT..... hhhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
                OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh TQE FIELDS:       FLAGS... hhhh
                EXTADDR. hhhhhhhh TCB..... hhhhhhhh
EXT..... hhhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
                OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh TQE FIELDS:       FLAGS... hhhh
                EXTADDR. hhhhhhhh ASCB.... hhhhhhhh TCB..... hhhhhhhh
EXT..... hhhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
                OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh PARM.... hhhhhhhh SIG-CPU. hhhh
```

EXT CODE hhhh
EXT hhhh
External interruption code.

ASCB hhhhhhhh
Address of ASCB for the address space that was current when the interruption occurred.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the interrupted task

N/A
The record is for a system or started task

PPPPPPPP
A page fault occurred

****
An internal error occurred

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
OLD-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word stored when the interruption occurred.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the TCB for the interrupted task

N/A
Not applicable, as in the case of an interrupted SRB routine

INT-TCB hhhhhhhh
TQE-TCB hhhhhhhh
Address of the TCB. This interruption is indicated by interruption codes 12hh.

TQE FIELDS
Indicates a clock comparator or CPU timer interruption. These interruptions are indicated by interruption codes X'1004' or X'1005'. The following fields contain information from the timer queue element (TQE):

FLAGS hhhh
The flags from the TQEFLGS field.

EXTADDR hhhhhhhh
The first four hexadecimal digits are the contents of the TQEFLGS field; the last four hexadecimal digits are the contents of the TQEEXIT field.

ASCB hhhhhhhh
One of the following:

hhhhhhhh
Contents of the TQEASCB field.

PPPPPPPP
A page fault occurred

****
An internal error occurred

The TQEASCB field is present only for a clock comparator interruption. TQEASCB contains the address of the ASCB for the address space in which the timer exit routine will be run.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Contents of the TQETCB field.

N/A
The record is for a system or started task

PPPPPPPP
A page fault occurred

****
An internal error occurred

TQETCB contains the address of the TCB for the task under which the timer exit routine will run.

PARM hhhhhhhh
Signal passed on a signal processor interruption, which is indicated by interruption codes 12hh.

SIG-CPU hhhh
Address of the processor on which a signal processor interruption occurred.

FRR trace records

An FRR record represents the return to the recovery termination manager (RTM) from a functional recovery routine (FRR). All fields, except the processor address, are gathered from the system diagnostic word area (SDWA) that was passed to the FRR.

Minimal Trace Record Format

```
FRR   ASCB.... hhhhhhhh CPU..... hhhh
                        PSW..... hhhhhhhh hhhhhhhh  hhhhhhhh hhhhhhhh
                        CC...... hhhhhhhh FLG1.... hhhhhhhh FLG2.... hhhhhhhh
                        RETRY... hhhhhhhh
```

Comprehensive Trace Record Format

```
 FRR           ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
               NAME.... cccccccc
               PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               ABCC.... hhhhhhhh ERRT.... hhhhhhhh FLG..... hhhhhh
               RC...... hh       RTRY.... hhhhhhhh
```

ASCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the ASCB for the address space in which the error occurred.

PPPPPPPP
A page fault occurred

****
An internal error occurred

CPU hhhh
Address of the processor associated with the error.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the error

N/A
The record is for a system or started task

PPPPPPPP
A page fault occurred

****
An internal error occurred

NAME cccccccc
Name of the FRR routine.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
One of the following:

hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word at the time of the error

PPPPPPPP
A page fault occurred

****
An internal error occurred

CC hhhhhhhh
ABCC hhhhhhhh
One of the following:

hhhhhhhh
The first three digits are the system completion code and the last three digits are the user completion code

U/A
Unavailable because the system diagnostic work area (SDWA) was unavailable

****
An internal error occurred

FLG1 hhhhhhhh
FLG hhhhhh
ERRT hhhhhhhh
Error-type flags from the SDWAFLGS field of SDWA.

FLG2 hhhhhh
Additional flags from the SDWAMCHD and SDWAACF2 fields of SDWA. The flags are contained in the two low-order bytes of this printed field; the high order byte is meaningless.

RC hh
Return code

RETRY hhhhhhhh
RTRY hhhhhhhh
One of the following:

hhhhhhhh
Retry address supplied by the FRR

N/A
Not applicable, indicating an FRR return code other than 4

PPPPPPPP
A page fault occurred

****
An internal error occurred

RTCA hhhhhhhh
Indicates if the recovery routine was a STAE or ESTAE.

HEXFORMAT, SUBSYS, and SYSTEM trace records

HEXFORMAT, SUBSYS, and SYSTEM records represent events for which GTF could not format the records.

```
HEXFORMAT   AID hh FID hh EID hh hhhhhhhh hhhhhhh ...

SUBSYS      AID hh FID hh EID hh hhhhhhhh hhhhhhh ...

SYSTEM      AID hh FID hh EID hh hhhhhhhh hhhhhhh ...
```

HEXFORMAT
Indicates an event signalled by a GTRACE macro. The macro specified no formatting routine (FID=00).

SUBSYS
Indicates an event signalled by a GTRACE macro. The macro specified a formatting routine (FID=hh) that could not be found.

SYSTEM
Indicates a system event. The trace record could not be formatted for one of the following reasons:

- If EEEE hex appears in bytes 0-1 or 8-9 of the recorded data, an unrecoverable error occurred in a GTF data-gathering routine. Message AHL118I is written on the console, identifying the module that caused the error and the action taken. (The message indicates that GTF will no longer trace this type of event. No more records for this type of event will appear in the trace output.)
- If EEEE hex does not appear in bytes 0-1 or 8-9 of the recorded data, the record could not be formatted because the GTF formatting routine could not be found.

AID hh
Application identifier, which should always be AID FF.

FID hh
Format identifier of the routine (AMDUSRhh or AMDSYShh) that was to format this record.

EID hh
Event identifier, which uniquely identifies the event that produced the record.

hhhhhhhh hhhhhhhh ...
Recorded data (256 bytes maximum).

IOX trace records

IOX records represent an input/output (I/O) interruption for a completed channel program and a summary of a complete channel program for the I/O operation.

```
IOX....shhhh ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              DEVN.... hhhh     SID..... hhhh     DRID.... hh
              TVSN.... hh       ECNO.... hhhh     DVCLS... hh
              DSTAT... hh       AERRC... hh       FLAG0... hh
              VOLSER.. cccccc   UCBTYP.. hhhhhhhh
              DSNAME.. cccc.cccccc.cccccc.ccccc
              NSSCH... hhhhhhhh DSSCH... hhhhhhhh SDCON... hhhhhhhh
              SRPEN... hhhhhhhh SDISC... hhhhhhhh SCUQU... hhhhhhhh
              IODTS... hhhhhhhh hhhhhhhh
              AONLY... hhhhhhhh DVBSY... hhhhhhhh ICMR.... hhhhhhhh


  CCW SECTION   SQNO.... hh       FGS1.... hh       FGS2.... hh
                RCNT.... hh       BLKR.... hhhh     BLKW.... hhhh
                BTRD.... hhhhhhhh BTWR.... hhhhhhhh DCHN.... hhhh
                CCHN.... hhhh     DEGA.... hh       DEGE.... hh
                DEEE.... hhhh     SEEKLOCC hh       CCHHR... hh
                LROP.... hh       LRSECT.. hh       LREXOP.. hh
                LREXPM.. hhhh
```

IOX shhhh
IOX identifies the beginning of an IOX record where hhhh is the device number and s is the subchannel set identifier.

ASCB {hhhhhhhh|U/A}
One of the following:

hhhhhhhh
Address of the address space control block (ACSB) for the address space that started the I/O operation.

U/A
Unavailable because the I/O supervisor block (IOSB) control block is unavailable.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBN {cccccccc|N/A|U/A}
One of the following:

cccccccc
Name of the job associated with the task that requested the I/O operation.

N/A
Not applicable.

U/A
Unavailable because the IOSB control block is unavailable.

DEVN shhhh
Device number with the subchannel set identifier when appropriate.

SID hhhh
System ID

DRID hh
Driver ID from IOSB

TVSN hh
Trace version

ECNO hhhh
Record count

DVCLS hh
Device class

DSTAT hh
Device status

AERRC hh
Error codes found during CCW analysis. See “CCW error codes” on page 307 for a description.

FLAG0 hh
Flag byte

VOLSER ccccccc
Volume Serial

UCBTYP hhhhhhhh
UCB type

DSNAME cccc.cccccc.cccccc.ccccc
44-byte data set name

NSSCH hhhhhhhh
Number of SSCH instructions.

DSSCH hhhhhhhh
Number of SSCH instructions for which data was collected.

SDCON hhhhhhhh
Summation of device connect times.

SRPEN hhhhhhhh
Summation of function pending times.

SDISC hhhhhhhh
Summation of device disconnect times.

SCUQU hhhhhhhh
Summation of control unit queuing times

IODTS hhhhhhhh
Time stamp from IOD

AONLY hhhhhhhh
Summation of device active only times

DVBSY hhhhhhhh
Summation of device busy times

ICMR hhhhhhhh
Summation of initial command response times

SQNO hh
Orientation Sequence Number

FGS1 hh
Flag byte 1

FGS2 hh
Flag byte 2

RCNT hh
Count of erase

BLKR hhhh
Number of blocks read

BLKW hhhh
Number of block written

BTRD hhhhhhhh
Number of bytes read

BTWR hhhhhhhh
Number of bytes written

DCHN hhhh
Number of data chain CCWs

CCHN hhhh
Number of COM chain CCWs

DEGA hh
Definition of exterior global attribute

DEGE hh
Definition of exterior global attribute extended

DEEE hhhhhhh
Definition of exterior end of extend CCH

SEEKLOCC hh
The command code that performed the seek or locate record operation.

CCHHR hhhhhhhh
CCHHR seek or search address

LROP hh
The locate record operation code.

LRSECT hh
The locate record sector number.

LREXOP hh
The locate record extended operation code.

LREXPM hhhh
The locate record extended parameters.

LSR trace records

An LSR record represents dispatching of a local supervisor routine in an address space.

Minimal Trace Record Format

```
LSR   ASCB.... hhhhhhhh CPU..... hhhh
                        PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
                        R1...... hhhhhhhh
```

Comprehensive Trace Record Format

```
 LSR          ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              LSR-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
              TCB..... hhhhhhhh
```

ASCB hhhhhhhh
Address of the address space control block.

CPU hhhh
Address of the processor on which the routine will be dispatched.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
LSR-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word under which the routine receives control.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the routine being dispatched

N/A
Not applicable

PPPPPPPP
A page fault occurred

****
An internal error occurred

TCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the task control block associated with this routine (if the routine is run as part of a task)

N/A
Not applicable

R15 hhhhhhhh
R0 hhhhhhhh
R1 hhhhhhhh
One of the following:

hhhhhhhh
Data that will appear in general registers 15, 0, and 1 when the local supervisor routine is dispatched

PPPPPPPP
A page fault occurred

****
An internal error occurred

MSCH trace records

An MSCH record represents a modify subchannel operation.

```
MSCH.... shhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
               SID..... hhhhhhhh CC...... hh       OPT..... hh
               OPT2.... hh       IOSLVL.. hh       SCHIB1.. hhhhhhhh
                        hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        hhhhhhhh UCBLVL.. hh       SCHIB2.. hhhhhhhh
                        hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        hhhhhhhh
                        hhhhhhhh UCBWGT.. hh       BASE.... shhhh
```

MSCH shhhh
Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that started the modify subchannel operation.

CPU hhhh
Address of the processor on which the modify subchannel started.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the task that requested the modify subchannel operation

N/A
Not applicable

SID hhhhhhhh
Subchannel ID from the UCBSID field of the UCB.

CC hh
MSCH condition code in bits 2 and 3.

OPT hh
IOSB option bytes from the IOSOPT field of the IOSB.

OPT2 hh
IOSB option bytes from the IOSOPT field of the IOSB.

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

SCHIB1 hhhhhhhh ... hhhhhhhh
First 7 words of the subchannel information block. Input from the caller of modify subchannel instruction. SCHIB address from the IOSSCHIB field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

SCHIB2 hhhhhhhh ... hhhhhhhh
First 7 words of the subchannel information block resulting from the modify subchannel instruction.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

PCIDMX trace records

PCIDMX records represent a PCIE de-multiplexing request.

```
PCIDMX.. hhhhhhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                  DEVTYPE. hhhhhhhh DMXAD... hhhhhhhh CALLBK@. hhhhhhhh
                  PARMS... hhhhhhhh hhhhhhhh
```

PCIDMX hhhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

ASCB hhhhhhhh
Address of the ASCB for the address space that was active when the de-multiplexing event occurred.

CPUID hhhh
Address of the processor on which the de-multiplexing operation occurred.

JOBN cccccccc
Name of the job associated with the address space that was active when the de-multiplexinng event occurred.

DEVTYPE hhhhhhhh
PCIE device type.

DMXAD hhhhhhhh
Address of the program that is performing the de-multiplexing.

CALLBK@ hhhhhhhh
Address of the callback routine.

PARMS hhhhhhhh hhhhhhhh
First 8 bytes of the callback parameters.

PCILG trace records

A PCILG record represents a PCI load instruction.

```
PCILG... hhhhhhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                  TCB..... hhhhhhhh CC...... hh       REQAD... hhhhhhhh
                  TRCID... hhhhhhhh
                  DATA.... hhhhhhhh hhhhhhhh          HANDLE.. hhhhhhhh
                  STATUS.. hh       PCIAS... hh       LENGTH.. hh
                  OFFSET.. hhhhhhhh hhhhhhhh
```

PCILG hhhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

ASCB hhhhhhhh
Address of the ASCB for the address space that issued the PCI load instruction.

CPUID hhhh
Address of the processor on which the PCI load instruction was issued.

JOBN cccccccc
Name of the job associated with the address space that issued the PCI load operation.

TCB hhhhhhhh
Address of task control block or zero if not running under a task.

CC hh
Condition code from the PCI load instruction.

REQAD hhhhhhhh
Address of the program that requested the PCI load instruction to be issued.

TRCID hhhhhhhh
Program defined trace identifier that can be used to determine why the PCI load instruction is being issued.

DATA hhhhhhhh hhhhhhhh
The data that was loaded.

HANDLE hhhhhhhh
PCIE function hardware handle.

STATUS hh
Error status information.

PCIAS hh
PCIE address space associated with the request.

LENGTH hh
Length of the data that was loaded.

OFFSET hhhhhhhh hhhhhhhh
Offset of the data within the PCIE address space that was loaded.

PCISTG trace records

A PCISTG record represents a PCI store instruction.

```
PCISTG... hhhhhhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                   TCB..... hhhhhhhh CC...... hh       REQAD... hhhhhhhh
                   TRCID... hhhhhhhh
                   DATA.... hhhhhhhh hhhhhhhh          HANDLE.. hhhhhhhh
                   STATUS.. hh       PCIAS... hh       LENGTH.. hh
                   OFFSET.. hhhhhhhh hhhhhhhh
```

PCILG hhhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

ASCB hhhhhhhh
Address of the ASCB for the address space that issued the PCI store instruction.

CPUID hhhh
Address of the processor on which the PCI store instruction was issued.

JOBN cccccccc
Name of the job associated with the address space that issued the PCI store operation.

TCB hhhhhhhh
Address of task control block or zero if not running under a task.

CC hh
Condition code from the PCI store instruction.

REQAD hhhhhhhh
Address of the program that requested the PCI store instruction to be issued.

TRCID hhhhhhhh
Program defined trace identifier that can be used to determine why the PCI store instruction is being issued.

DATA hhhhhhhh hhhhhhhh
The data that was stored.

HANDLE hhhhhhhh
PCIE function hardware handle.

STATUS hh
Error status information.

PCIAS hh
PCIE address space associated with the request.

LENGTH hh
Length of the data that was stored.

OFFSET hhhhhhhh hhhhhhhh
Offset of the data within the PCIE address space that was stored.

PGM and PI trace records

PGM and PI records represent program interruptions.

Minimal Trace Record Format

```
PI    CODE.... hhh    ASCB.... hhhhhhhh CPU..... hhhh
                      PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                      TCB..... hhhhhhhh VPH..... hhhhhhhh VPA..... hhhhhhhh
                      R15..... hhhhhhhh R1...... hhhhhhhh
```

Comprehensive Trace Record Format

```
PGM..... hhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
              TCB..... hhhhhhhh VPH..... hhhhhhhh VPA..... hhhhhhhh
              MODN.... cccccccc
               64-bit GRs
  Left halves of all registers contain zeros
   0-3  hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh
   4-7  hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh
   8-11 hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh
  12-15 hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh

  Access register values
    0-3  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
    4-7  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
    8-11 aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
   12-15 aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
```

or

```
PGM..... hhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
              OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
              TCB..... hhhhhhhh VPH..... hhhhhhhh VPA..... hhhhhhhh
              MODN.... cccccccc
               64-bit GRs
   0-1  hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
   2-3  hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
   4-5  hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
   6-7  hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
   8-9  hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
  10-11 hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
  12-13 hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh
  14-15 hhhhhhhh_hhhhhhhh  hhhhhhhh_hhhhhhhh

  Access register values
    0-3  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
    4-7  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
    8-11 aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
   12-15 aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
```

PI CODE hhh
PGM hhh
: Program interruption code, in decimal.

ASCB hhhhhhhh
: Address of ASCB for the address space in which the interruption occurred.

CPU hhhh
: Address of the processor on which the interruption occurred.

JOBN cccccccc
: One of the following:

  cccccccc
  : Name of the job associated with the interruption

  N/A
  : Not applicable

  PPPPPPPP
  : A page fault occurred

****
An internal error occurred

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
OLD-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word stored when the interruption occurred.

TCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the TCB for the interrupted task

N/A
Not applicable as in the case of an interrupted SRB routine

VPH hhhhhhhh
VPA hhhhhhhh

VPH hhhhhhhh

Bytes 0-3 of the translation exception ID (TEID) , displayed only when these bytes are not all zero.

VPA hhhhhhhh
Bytes 4-7 of the translation exception ID. In cases defined by principles of operation, bits 0-51 of the TEID with 12 zero bits appended on the right are the translation exception address which is the virtual address of the page, the reference to which resulted in the program exception.

MODN cccccccc
cccccccc is one of the following:

mod_name
The name of a module that will receive control when the task is dispatched.

WAITTCB
Indicates that the system wait task was interrupted.

SVC-T2
Indicates that a type 2 SVC routine resident in the nucleus was interrupted.

SVC-RES
Indicates that a type 2 SVC routine or the first load module of a type 4 SVC routine was interrupted. The routine is located in the pageable link pack area (PLPA).

SVC-ccc
Indicates that the second or subsequent load module of a type 4 SVC routine was interrupted. The module is located in the fixed or pageable link pack area (LPA). The last four characters of the load module name are cccc.

****IRB*****
Indicates that an asynchronous routine with an associated interrupt request block was interrupted. No module name is available.

***ccccccc**
Indicates that an error recovery module was in control. The last seven characters of the module name are ccccccc.

****
An internal error occurred

x-y hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Bits 32 - 63 of general registers *x* through *y* when the interruption occurred.

x-y hhhhhhhh_hhhhhhhh hhhhhhhh_hhhhhhhh
Bits 0 - 63 of general registers *x* through *y* when the interruption occurred.

x-y aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Access registers *x* through *y* when the interruption occurred.

RNIO trace records

An RNIO record represents a VTAM remote network input/output event. For trace information, see *z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures*.

Minimal Trace Record Format

```
RNIO  ASCB.... hhhhhhhh CPU..... hhhh     R0...... hhhhhhhh
```

Comprehensive Trace Record Format

```
 RNIO          ASCB.... hhhhhhhh CPU..... hhhh     JOB..... cccccccc
               IN...... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               R0...... hhhhhhhh

 RNIO          ASCB.... hhhhhhhh CPU..... hhhh     JOB..... cccccccc
               OUT..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               R0...... hhhhhhhh
```

ASCB hhhhhhhh
Address of the ASCB for the address space of the application associated with the event.

CPU hhhh
Address of the processor that ran the I/O instruction.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the IO event

N/A
Not applicable

PPPPPPPP
A page fault occurred

****
An internal error occurred

IN hhhhhhhh ... hhhhhhhh
OUT hhhhhhhh ... hhhhhhhh
IN indicates that the I/O is from NCP to VTAM; OUT indicates that the direction of the I/O is from VTAM to NCP. The hexadecimal data is:

- For IN events: the transmission header, the response header, and the response unit.
- For OUT events: the transmission header, the request header, and the request unit.

R0 hhhhhhhh
Contents of general register 1 when the event occurred.

RSCH trace records

An RSCH record represents a resume subchannel operation.

```
RSCH.... shhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
               RST..... hhhhhhhh VST..... hhhhhhhh DSID.... hhhhhhhh
               CC...... hh       SEEKA... hhhhhhhh hhhhhhhh
               GPMSK... hh       OPT..... hh       FMSK.... hh
               DVRID... hh       IOSLVL.. hh       UCBLVL.. hh
               UCBWGT.. hh       BASE.... shhhh
```

RSCH shhhh
Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that started the I/O operation.

CPU hhhh
Address of the processor on which the I/O operation resumed.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the I/O operation

N/A
Not applicable

RST hhhhhhhh
Address of the channel program. This value comes from the contents of the IOSRST field of the IOSB.

VST hhhhhhhh
Virtual address of the channel program. This value comes from the contents of the IOSVST field of the IOSB.

DSID hhhhhhhh
Request identifier used by purge. Contents of the IOSDID field of the IOSB (address of the DEB or another control block used by purge).

CC hh
RSCH condition code in bits 2 and 3.

SEEKA hhhhhhhh hhhhhhhh
Dynamic seek address from the IOSEEKA field of the IOSB.

GPMSK hh
Guaranteed device path mask for GDP requests from the IOSEEKA field of the IOSB.

OPT hh
IOSB options byte from the IOSOPT field of the IOSB.

FMSK hh
Mode set/file mask from the IOSFMSK field of the IOSB.

DVRID hh
Driver ID from the IOSDVRID field of the IOSB.

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

SLIP trace records

A SLIP record represents a SLIP program event interruption. GTF writes four types of SLIP records:

- SLIP standard trace record
- SLIP stand/user trace record
- SLIP user trace record
- SLIP debug trace record

SLIP standard trace record

A SLIP standard (STD) trace record represents a slip trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP.

```
    SLIP STD     ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
                 TID..... cccc     ASID.... hhhh     JSP..... cccccccc
                 TCB..... hhhhhhhh MFLG.... hhhh     EFLG.... hhhh
                 SFLG.... hh       DAUN.... hhhh     MODN.... cccccccc
                 OFFS.... hhhhhhhh IADR.... hhhhhhhh hhhhhhhh
                 INS..... hhhhhhhh hhhh
                 EXSIAD.. hhhhhhhh hhhhhhhh EXSINS.. hhhhhhhh hhhh
                 BRNGH... hhhhhhhh BRNGA... hhhhhhhh BRNGD... hhhhhhhh
                 OPSW.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                 ILC/PIC. hhhhhhhh PERC.... hh       TYP..... hh
                 PKM..... hhhh     SASID... hhhh     AX...... hhhh
                 PASID... hhhh     ASC..... c
                 SA-SPACE ccccccccc     cccc         DATX.... hh
```

ASCB hhhhhhhh
The address of the ASCB for the current address space.

CPU hhhh
The processor identifier (ID).

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the SLIP trap

N/A
Not applicable

TID cccc
The trap ID.

ASID hhhh
The identifier of the current address space.

JSP cccccccc
One of the following:

cccccccc
Job step program name

N/A
Not applicable

U/A
Unavailable

TCB hhhhhhhh
One of the following:

hhhhhhhh
TCB address

N/A
Not applicable

MFLG hhhh
System mode indicators that indicate the status of the system. The indicators correspond to the SLWACW field in the SLWA. For a description of the SLWA, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

EFLG hhhh
Error bytes that indicate the error status of the system. These bytes correspond to SDWAERRA in the SDWA. For a description of the SDWA, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

SFLD hh
SLIP status flags.

DAUN hhhhhhhh
A counter representing the number of times data was unavailable for the DATA keyword test.

The following fields apply to PER interruptions only. For other than PER interruptions, these fields are not applicable and contain: N/A, N/, or N.

MODN cccccccc
One of the following:

cccccccc
Load module name in which the interruption occurred

N/A
Not applicable

U/A
Unavailable

OFFS hhhhhhhh
One of the following:

hhhhhhhh
Offset into the load module containing the instruction that caused the interruption

N/A
Not applicable

U/A
Unavailable

IADR hhhhhhhh hhhhhhhh
Address of the instruction that caused the interruption.

INS hhhhhhhh hhhh
Instruction content: the instruction that caused the PER interruption.

EXSIAD hhhhhhhh hhhhhhhh
One of the following:

hhhhhhhh hhhhhhhh
Target instruction address if the INS field is an Execute instruction

N/A
Not applicable

U/A
Unavailable

EXSINS hhhhhhhhh hhhh
One of the following:

hhhhhhhh hhhh
Target instruction content if an INS field is an Execute instruction: 6 bytes of data beginning at the target instruction address

N/A
Not applicable

U/A
Unavailable

BRNGH hhhhhhhh
BRNGA hhhhhhhh
One of the following:

hhhhhhhh
The beginning range virtual address if the SLIP command specified SA. BRNGH identifies the high 4 bytes. BRNGA identifies the low 4 bytes.

N/A
Not applicable

BRNGD hhhhhhhh
One of the following:

hhhhhhhh
Four bytes of storage starting at the beginning range virtual address if SA was specified

N/A
Not applicable

U/A
Unavailable

OPSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
The program old PSW.

ILC/PIC hhhhhhhh
The instruction length code and program interruption code.

PERC hh
The PER interruption code.

TYP hh
The PER trap mode.

PKM hhhh
The PSW key mask.

SASID hhhh
The identifier of the secondary address space.

AX hhhh
The authorization index.

PASID hhhh
The identifier of the primary address space.

ASC c
The PSW ASC mode indicator:

c
Meaning

0
Primary addressing mode

1
Access register addressing mode

2
Secondary addressing mode

3
Home addressing mode

SA-SPACE cccccccccccc
Storage alteration space identifier, as follows:

- The ASID, for an address space
- The owning ASID and the data space name, for a data space

DATX hh
The DATA filter mismatch count due to an event that occurred in transactional execution mode.

SLIP standard/user trace record

The SLIP standard/user trace record represents a slip trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP and TRDATA=parameters.

```
SLIP S+U   ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
           TID..... cccc     ASID.... hhhh     JSP..... cccccccc
           TCB..... hhhhhhhh MFLG.... hhhh     EFLG.... hhhh
           SFLG.... hh       DAUN.... hhhh     MODN.... cccccccc
           OFFS.... hhhhhhhh IADR.... hhhhhhhh hhhhhhhh
           INS..... hhhhhhhh hhhh     EXSIAD.. hhhhhhhh hhhhhhhh
           EXSINS.. hhhhhhhh BRNGH... hhhhhhhh BRNGA... hhhhhhhh
           BRNGD... hhhhhhhh
           OPSW.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           ILC/PIC. hhhhhhhh PERC.... hh       TYP..... hh
           PKM..... hhhh     SASID... hhhh     AX...... hhhh
           PASID... hhhh     ASC..... c        SA-SPACE ccccccccc        cccc
           DATX.... hh
           GENERAL PURPOSE REGISTER VALUES
           0-3..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           4-7..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           8-11.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           12-15... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           GPR HIGH HALF VALUES
           0-3..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           4-7..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           8-11.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           12-15... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
            ACCESS REGISTER VALUES
           0-3..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           4-7..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           8-11.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
           12-15... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
```

ASCB hhhhhhhh . . . DATX hh
: These fields are the same as the fields in the SLIP standard trace record.

GENERAL PURPOSE REGISTER VALUES
GPR HIGH HALF VALUES
ACCESS REGISTER VALUES
: Contents of the general purpose registers and access registers at the time of the error or interruption, if REGS is specified in TRDATA on the SLIP command. The GPR high half values will only be traced in z/Architecture mode.

SLIP user trace record

The SLIP user record represents a SLIP trap match when the SLIP command specifies ACTION=TRACE or ACTION=TRDUMP and TRDATA=parameters.

```
SLIP USR    CPU..... hhhh     EXT..... hhhh     CNTLN... hh
      hhhh  hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh | cccccccccccccccc |
```

CPU hhhh
: Processor ID.

EXT hhhh
: Extension number.

CNTLN hh
: Continuation length.

hhhh
: Length for the single range in the SLIP command. If hhhh is zero, either the range was not available or the range was not valid, so that GTF did not collect data for the range. GTF would consider the range not valid if, for example, the ending range address precedes the beginning range address.

hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh | cccccccccccccccc |
: User-defined data fields that are specified by TRDATA on the SLIP command. The length and data fields may be repeated.

For a SLIP command, the trace contains as many user records and user continuation records as needed to trace the data ranges specified in the TRDATA parameter on the SLIP command. The header in each record contains the processor ID and the extension number. When a record is filled enough so that the next data range cannot fit, GTF writes the partially filled record to the GTF trace table. GTF builds another record; its extension number is increased by one and the continuation length is set to zero.

When the length of data from a range is greater than 249 bytes, the excess data is put in user continuation records. After writing the SLIP USR record, GTF builds a user continuation record. GTF increases the extension number by one and sets the continuation length to the number of bytes of data to be put in the continuation record. If more than 251 bytes of data are left, GTF copies 248 bytes into the record and places it in the GTF trace table. GTF builds user continuation records until all the data from a range is traced.

SLIP debug trace record

The SLIP debug record represents a SLIP trap match when the SLIP command specifies DEBUG.

```
SLIP S+U     ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
             TID..... cccc     ASID.... hhhh     JSP..... cccccccc
             TCB..... hhhhhhhh MFLG.... hhhh     EFLG.... hhhh
             SFLG.... hh       DAUN.... hhhh     MODN.... cccccccc
             OFFS.... hhhhhhhh IADR.... hhhhhhhh hhhhhhhh
             INS..... hhhhhhhh hhhh
             EXSIAD.. hhhhhhhh hhhhhhhh          EXSINS.. hhhhhhhh
             BRNGH... hhhhhhhh BRNGA... hhhhhhhh BRNGD... hhhhhhhh
             OPSW.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
             ILC/PIC. hhhhhhhh PERC.... hh       TYP..... hh
             PKM..... hhhh     SASID... hhhh     AX...... hhhh
             PASID... hhhh     ASC..... h        SA-SPACE hhh
             hh00              DATX.... hh
```

ASCB hhhhhhhh . . . DATX hh
These fields are the same as the fields in the SLIP standard trace record. The high order bit in the SFLG field is set to 1 to indicate a debug record.

hh00
Two bytes of debug-produced data. The first byte indicates which keyword failed, the second byte contains zeros.

Byte 1 (decimal)
Keyword That Failed

01
DATA test failed

03
ASID

04
JOBNAME

05
JSPGM

06
PVTMOD

0 7
LPAMOD

08
ADDRESS

09
MODE

0A
ERRTYP

0D
RANGE

0E
DATA

14
ASIDSA

16
REASON CODE

17
NUCMOD

18
PSWASC

1A
DSSA

SRB trace records

An SRB record represents dispatching of an asynchronous routine represented by a service request block (SRB).

Minimal Trace Record Format

```
SRB   ASCB.... hhhhhhhh CPU..... hhhh
                        PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        R15..... hhhhhhhh SRB..... hhhhhhhh R1...... hhhhhhhh
                        TYPE.... cccccccccccccccccccccccccc
```

Comprehensive Trace Record Format

```
SRB          ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
             SRB-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
             SRB..... hhhhhhhh
                        TYPE.... cccccccccccccccccccccccccc
```

ASCB hhhhhhhh
Address of the ASCB for the address space in which the SRB routine is dispatched. This may or may not be the address space in which the SRB was created.

CPU hhh
Address of the processor on which the SRB routine is dispatched.

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
SRB-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word under which the SRB routine receives control.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the SRB being dispatched

N/A
Not applicable, as in the case of a global SRB, which is indicated in the TYPE field

****
An internal error occurred

SRB hhhhhhhh
One of the following:

hhhhhhhh
Address of the service request block (SRB)

****
An internal error occurred

R15 hhhhhhhh
R1hhhhhhhh
One of the following:

hhhhhhhh
Data that will appear in general registers 15 and 1 when the SRB routine is dispatched

****
An internal error occurred

PARM hhhhhhhh
One of the following:

hhhhhhhh
Four-byte parameter or the address of a parameter field to be passed to the SRB routine

N/A
Not applicable, as in the case of a suspended SRB, which is indicated in the TYPE field

TYPE cccccccccccccccccccccccccc
Indicates the type of SRB routine, as follows:

SUSPENDED
Denotes an SRB routine that was dispatched earlier and was subsequently interrupted (for example, by I/O operations or by a request for a lock). The routine is about to be re-dispatched.

INITIAL DISPATCH OF SRB
Denotes an SRB routine selected from the service priority list that is about to be dispatched for the first time.

REDISPATCH OF SUSPENDED SRB
Denotes an SRB routine that was dispatched earlier and was subsequently interrupted (for example, by I/O operations or by a request for a lock). The routine is about to be re-dispatched.

SRM trace records

An SRM record represents an entry to the system resources manager (SRM).

Minimal Trace Record Format

```
SRM   ASCB.... hhhhhhhh CPU..... hhhh   R15..... hhhhhhhh R0...... hhhhhhhh
                        R1...... hhhhhhhh
```

Comprehensive Trace Record Format

```
SRM          ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
             R15..... hhhhhhhh R0...... hhhhhhhh R1...... hhhhhhhh
```

ASCB hhhhhhhh
One of the following:

hhhhhhhh
Address of the ASCB for the address space that was current when SRM was entered

****
An internal error occurred

CPU hhhh
Address of the processor used by the system resources manager.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with the entry to SRM

N/A
Not applicable

****
An internal error occurred

R15 hhhhhhhh
R0 hhhhhhhh
R1 hhhhhhhh
Data that was contained in general registers 15, 0, and 1 when the system resources manager passed control to GTF. The data includes the SYSEVENT code in the low-order byte of register 0.

SSCH trace records

An SSCH record represents a start subchannel operation.

```
SSCH.... shhhh ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
               RST..... hhhhhhhh VST..... hhhhhhhh DSID.... hhhhhhhh
               CC...... hh       SEEKA... hhhhhhhh hhhhhhhh
               GPMSK... hh       OPT..... hh       FMSK.... hh
               DVRID... hh       IOSLVL.. hh       UCBLVL.. hh
               UCBWGT.. hh       BASE.... shhhh
               ORB..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                        hhhhhhhh hhhhhhhh hhhhhhhh
```

SSCH shhhh
Device number from the UCBCHAN field of the UCB with subchannel set identifier when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that started the I/O operation.

CPU hhhh
Address of the processor on which the I/O operation started.

JOBN cccccccc
One of the following:

cccccccc
Name of the job associated with I/O operation

N/A
Not applicable

RST hhhhhhhh
Address of the channel program. This value comes from the contents of the IOSRST field of the IOSB.

VST hhhhhhhh
Virtual address of the channel program. This value comes from the contents of the IOSVST field of the IOSB.

DSID hhhhhhhh
Request identifier used by purge. This identifier is in the IOSDID field of the IOSB and is the address of the DEB or another control block used by PURGE.

CC hh
SSCH condition code in bits 2 and 3.

SEEKA hhhhhhhh hhhhhhhh
Dynamic seek address from the IOSEEKA field of the IOSB.

GPMSK hh
Guaranteed device path mask for GDP requests from the IOSGPMSK field of the IOSB.

OPT hh
IOSB options byte from the IOSOPT field of the IOSB.

FMSK hh
Mode Set/File mask from the IOSFMSK field of the IOSB.

DVRID hh
Driver ID from the IOSDVRID field of the IOSB.

IOSLVL hh
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT hh
Flags from the UCBWGT field of the UCB.

BASE shhhh
Device number from the UCBCHAN field of the UCB, which includes the subchannel set identifier when appropriate.

ORB hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Contents of the operation request block (ORB).

STAE trace records

A STAE record represents return to the recovery termination manager (RTM) from a STAE or ESTAE routine.

Minimal Trace Record Format

```
STAE  PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh CC...... hhhhhhhh
                        RF...... hhhhhhhh TYCA.... hhhhhhhh hhhhhhhh
```

Comprehensive Trace Record Format

```
STAE           ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc
               ESTN.... cccccccc
               ERR-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               ABCC.... hhhhhhhh ERRT.... hhhhhhhh FLG..... hhhhhh
               RC...... hh       RTRY.... hhhhhhhh RTCA.... hhhhhhhh
```

ASCB hhhhhhhh
Address of the ASCB for the address space involved in the recovery.

CPU hhhh
Address of the processor.

JOBN cccccccc
One of the following:

cccccccc
Name of the job involved in the recovery

N/A
Not applicable

****
An internal error occurred

ESTN cccccccc
One of the following:

cccccccc
ESTAE routine name

U/A
Unavailable because the routine did not supply a name

****
An internal error occurred

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
ERR-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
One of the following:

hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word at the time of the error

U/A
Unavailable because the system diagnostic work area (SDWA) was unavailable

****
An internal error occurred

CC hhhhhhhh
ABCC hhhhhhhh
One of the following:

hhhhhhhh
The first four digits are the system completion code and the last four digits are the user completion code

U/A
Unavailable because the system diagnostic work area (SDWA) was unavailable

****
An internal error occurred

TYCA hhhhhhhh hhhhhhhh
Retry address (see RTRY hhhhhhhh following) and an indication of whether the routine was a STAE or ESTAE (see RTCA hhhhhhhh following).

RF hhhhhhhh
FLG hhhhhh
ERRT hhhhhhhh
Error flags from the SDWAFLGS field of the SDWA.

RC hh
Return code

RTRY hhhhhhhh
One of the following:

hhhhhhhh
The address supplied by the FRR

N/A
Not applicable, indicating an FRR return code other than 4

PPPPPPPP
A page fault occurred

****
An internal error occurred

RTCA hhhhhhhh
Indicates if the recovery routine was a STAE or ESTAE.

SVC and SVCR trace records

An SVC record represents a supervisor call (SVC) interruption. An SVCR record represents an exit from a supervisor call. SDSP interruptions also build SVC exit records with label SDSP. When both DSP and SVC options are in effect, the SVCR format of trace record is produced by IPCS.

If the trace data contains an SVC exit record, the label that appears in the formatted output will depend on the options selected during IPCS.

1. If the SVC option is selected in the IPCS dialog, the SVC exit record and the SVC number will appear with the label SVCR.
2. If only the DSP option is chosen in the IPCS dialog, the formatted output record will remain unchanged; DSP and SDSP labels will appear in the formatted output and no SVC number will be present.
3. If both DSP and SVC options are active in IPCS, the SVCR along with SVC number will appear.

It can be concluded, if SVC is one of the options selected during IPCS formatting, all SVC exit records will appear with label SVCR along with SVC number.

The format of an SVC and SVCR trace record depends on the SVC interruption being traced. For a break down of the information that GTF collects for each SVC, see the *SVC Summary* chapter of *z/OS MVS Diagnosis: Reference*. The formats shown are typical.

Minimal Trace Record Format

```
SVC   CODE.... hhh  ASCB.... hhhhhhhh CPU..... hhhh
                    PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
                    R1...... hhhhhhhh
           SVCR  CODE.... hhh  ASCB.... hhhhhhhh CPU..... hhhh
                    PSW..... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    TCB..... hhhhhhhh R15..... hhhhhhhh R0...... hhhhhhhh
                    R1...... hhhhhhhh
```

Comprehensive Trace Record Format

```
SVC..... hhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBNAME. cccccccc
               OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
               TCB..... hhhhhhhh MODN.... yyyyyyyy R15..... hhhhhhhh
               R0..... hhhhhhhh R1..... hhhhhhhh
           SVCR..... hhh   ASCB.... hhhhhhhh CPU..... hhhh     JOBNAME. cccccccc
                OLD-PSW. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                TCB..... hhhhhhhh MODN.... yyyyyyyy R15..... hhhhhhhh
                R0..... hhhhhhhh R1..... hhhhhhhh
```

SVC CODE hhh
SVC hhh

- For SVC, and for SVCR when not X'FFxx': SVC interruption code, which is also called the SVC number.
- For SVCR when X'FF00': completion of the system-initiated processing involved with ATTACH, LINK, XCTL or SYNCH processing prior to the target routine getting control.
- For SVCR when X'FF01': initial system-initiated processing involved with XCTL processing prior to the target routine getting control.

ASCB hhhhhhhh
Address of the ASCB for the address space in which the interruption occurred.

CPU hhhh
Address of the processor on which the interruption occurred.

JOBNAME cccccccc
One of the following:

cccccccc
Name of the job associated with SVC interruption

SSSSSSSS
Unavailable; GTF cannot provide data for the SVC due to security considerations.

****
An internal error occurred

PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
OLD-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
DSP-PSW hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Program status word stored when the interruption occurred.

- For SVC, and for SVCR when SVC Code is not X'FFxx': Program status word stored when the interruption occurred.
- For SVCR when SVC Code is X'FF00': PSW of the target routine that will get control as a result of ATTACH, LINK, XCTL or SYNCH processing.
- For SVCR when SVC Code is X'FF01': PSW of a system routine that will get control as a result of initial processing involved with XCTL.

TCB hhhhhhhh
Address of the TCB for the interrupted task, that is, the task that issued the SVC instruction.

R15 hhhhhhhh
R0 hhhhhhhh
R1 hhhhhhhh
Data in general registers 15, 0, and 1 when the SVC instruction ran.

MODN cccccccc
cccccccc is one of the following:

mod_name
The name of a module that will receive control when the task is dispatched.

SVC–T2
Indicates a type 2 SVC routine resident in the nucleus.

SVC–RES
Indicates a type 3 SVC routine or the first load module of a type 4 SVC routine. The routine is located in the pageable link pack area (PLPA).

SVC cccc
Indicates the second or subsequent load module of a type 4 SVC routine. The routine is located in the fixed or pageable link pack area (LPA). The last four characters of the load module name are cccc.

****IRB******
Indicates an asynchronous routine with an associated interruption request block. No module name is available.

***cccccc**
Indicates an error recovery module. The last seven characters of the load module name are cccccc.

PPPPPPPP
A page fault occurred

****
An internal error occurred

DDNAM cccccc
Name of the DD statement associated with the SVC, if applicable.

additional fields
Vary with the SVC number. These fields are described for the SVC in the *z/OS MVS Diagnosis: Reference*.

SYNS and SYNE trace records

A SYNS trace entry represents the start of a synchronous I/O (zHyperLink) operation. A SYNE trace entry represents the end of a synchronous I/O (zHyperLink) operation.

```
SYNS.... shhhh   ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                 RST..... hhhhhhhh VST..... hhhhhhhh SEARCHA. hhhhhhhh hh
                 DSID.... hhhhhhhh DVRID... hh       UCBSFLS. hhhh
```

```
                 UCBLVL.. hh       PFID.... hhhhhhhh HANDLE.. hhhhhhhh
                 OPCODE.. hh       SYNTYPE. cccccccc LEN1.... hhhh
                 LEN2.... hhhh     RECCNT.. hh       DIAGTKN. hhhhhhhh


SYNE.... shhhh   ASCB.... hhhhhhhh CPUID... hhhh     JOBN.... cccccccc
                 TCB..... hhhhhhhh DVRID... hh       SEARCHA. hhhhhhhh hh
                 UCBSFLS. hhhh     UCBLVL.. hh       PFID.... hhhhhhhh
                 HANDLE.. hhhhhhhh SYNTYPE. cccccccc OPCODE.. hh
                 LEN1.... hhhh     LEN2.... hhhh     RECCNT.. hh
                 RESP.... hhhh     RCQ..... hhhh     DIAGTKN. hhhhhhhh
                 ELAPSED. hhhhhhhh
                 DSNAME.. cccccccccccccccccccccccccccccccccccccccccccc
                 DEVSTAT. hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
```

SYNS shhhh or SYNE shhhh
Device number from the UCBCHAN field of the UCB with subchannel set identifier, when appropriate.

ASCB hhhhhhhh
Address of the ASCB for the address space that performed the I/O operation.

CPUID hhhh
Address of the processor on which the I/O operation was performed.

JOBNAME {cccccccc | NA}
One of the following:

cccccccc
Name of the job associated with task that requested the I/O operation.

N/A
Not applicable.

TCB {hhhhhhhh | N/A}
One of the following:

hhhhhhhh
Address of the TCB for the task that requested the I/O operation.

N/A
Not applicable.

RST hhhhhhhh
Real address of the channel program. This value comes from the contents of the IOSRST field of the IOSB.

VST hhhhhhhh
Virtual address of the channel program. This value comes from the contents of the IOSVST field of the IOSB.

DSID hhhhhhhh
Request identifier used by purge. This identifier is in the IOSDSID field of the IOSB and is the address of the DEB or another control block used by PURGE.

DVRID hh
Driver identifier from the IOSDVRID field of the IOSB.

SEARCHA hhhhhhhh hh
Search address from the channel program.

UCBSFLS hhhh
Contents of the UCB startability flags (UCBSFLS) from the UCB.

UCBLVL hh
UCB level value from the UCBLEVEL field of the UCB.

PFID hhhhhhhh
PCIE function identifier (PFID) for PCIE device/function.

HANDLE hhhhhhhh
PCIE function hardware handle.

SYNTYPE cccccccc
The type of synchronous I/O request:

- Initiate - This is an initiate type request. An initiate request is used when multiple synchronous I/O requests need to be performed in parallel. There is one initiate request issued for each synchronous I/O to be started in parallel. Normally, the system traces only the start of an initiate request (SYNS event). However, if the initiate request fails, the end of the initiate request is also traced (SYNE event).
- Complete – This is a complete type request. A complete request is used to wait for previously initiated requests to complete. There is one complete request issued for each synchronous I/O that was initiated successfully. The system traces only the end of the complete request (SYNE event).
- Only – This is a standalone (only) request. The system traces the start (SYNS event) and end (SYNE event) of a standalone request.

OPCODE hh
Operation code for the operation to be performed.

LEN1 hhh
First data length field.

LEN2 hhh
Second data length field.

RECCNT hh
Record count for this I/O operation.

RESP hhhh
Response code.

RCQ hhhh
Response code qualifier.

DIAGTKN hhhhhhhh
An internal diagnostic token that represents the synchronous I/O request.

ELAPSED hhhhhhhh
Elapsed time in 0.5 microsecond units.

DSNAME cccccccccccccccccccccccccccccccccccccccccccc
The data set name associated with the I/O request, if any.

DEVSTAT hhhhhhhh hhhhhhhh...
Device specific status (optional).

SYNCH I/O trace records

A SYNCH I/O record represents the processing of a synchronous I/O (zHyperLink) request. SYNCH I/O trace records appear following a SYNE trace record; they do not appear alone.

```
FORMAT 0 cccc                        SYNCH I/O REQ     DEV..... hhhhh
                 ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... cccccccc

CLP Request Block at vvvvvvvv_vvvvvvvv
  LEN.............. hhhh                      CmdCode.......... hhhh
  Fmt.............. hh                        OpCode........... hh
  Flag............. hh                        Key.............. hh
  Func Handle...... hhhhhhhh                  SID.............. hhhhhhhh
  WWNN............. hhhhhhhh hhhhhhhh
  Device Spec...... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh

CLP Response Block at vvvvvvvv_vvvvvvvv
  LEN.............. hhhh                      RespCod.......... hhhh
  RCQ.............. hhhh                      Flag.............. hh
  RecNo............ hh                        TimeStamp........ hhhhhhhh hhhhhhhh
  DiagID........... hhhhhhhh hhhhhhhh
  Device Status.... hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
                    hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
```

```
                    hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Data at rrrrrrrr_rrrrrrrr
 dddddddd dddddddd dddddddd dddddddd | ................ |
 dddddddd dddddddd dddddddd dddddddd | ................ |
 **** 0020X CONSECUTIVE BYTES ARE 00X
 *** Back half of split data ***
 dddddddd dddddddd dddddddd dddddddd | ................ |
 dddddddd dddddddd dddddddd dddddddd | ................ |
IOSB at vvvvvvvv
  formatted iosb data

IOBE at vvvvvvvv
  formatted iobe data
```

FORMAT d cccc
Format (d) and type of trace event (cccc): SYNE. Format is zero.

DEV hhhhh
The device number qualified with the subchannel set identifier.

ASCB hhhhhhhh
Same as the ASCB field in the SYNE base record.

CPU hhhh
Same as the CPU ID field in the SYNE base record.

JOBN cccccccc
Same as the job named (JOBN) field in the SYNE base record.

CLP Request Block at vvvvvvvv_vvvvvvvv
The request block at virtual address vvvvvvvv_vvvvvvvv. The formatted request block follows.

Len hhhhh
Request block length

CmdCode hhhh
Command code

Fmt hh
Request block format

OpCode hh
Operation code for the operation to be performed

Flag hh
Request block flags

Key hh
Channel program key in the first four bits

Func Handle hhhhhhhh
PCIE function hardware handle

SID hhhhhhhh
Subsystem id for the associated subchannel/device

WWNN hhhhhhhh hhhhhhhh
World-wide node name for the control unit

Device Spec hhhhhhhh...
Device specific parameters

CLP Response Block at vvvvvvvv_vvvvvvvv
The response block at virtual address vvvvvvvv_vvvvvvvv. The formatted response block follows.

Len hhhhh
Request block length

RespCode hhhh
Response code

RCQ hhhh
Response code qualifier

Flag hh
Response flags

RecNo hh
Record number to which the status pertains or zero

TimeStamp hhhhhhhh hhhhhhhh
Control unit time stamp at the time the status condition was detected

DiagId hhhhhhhh hhhhhhhh
Diagnostic information identifier

Device Status hhhhhhhh...
Device specific status (optional)

Data at rrrrrrrr_rrrrrrrr
Data buffer at real address rrrrrrrr_rrrrrrrr

dddddddd dddddddd dddddddd dddddddd
Data transferred to or from the data buffer. If there is not a series of dashes in this field, then all transferred data are displayed in four byte sections.

***** Back half of split data *****
Indicates there were more bytes of information transferred than were specified on the START command. The default value is 20 bytes, but you can specify the number of bytes to be shown. The specified value is halved; for an odd number, the larger section is shown first. The first section of data displayed comes from the beginning of the buffer from which the data was transferred. The last section comes from the end of the buffer.

IOSB vvvvvvvv
Fullword virtual address of the IOSB followed by the formatted contents of the IOSB

IOBE vvvvvvvv
Fullword virtual address of the IOBE followed by the formatted contents of the IOBE.

TCW trace records

A TCW record represents the processing of a zHPF channel program. TCW trace records appear following INTG, IOCS, IO, SSCH, and XSCH trace records; they do not appear alone.

```
FORMAT d   cccc             TCW CHAIN        DEV..... hhhhh
           ASCB.... hhhhhhhh CPU..... hhhh     JOBN.... hhhhhh
```

```
TCW at rrrrrrrr (vvvvvvvv)
  Format........... hh                  Flag1............ hh
  Flag2............ hh                  Flag3............ hh
  TCCBL/R/W........ hh
  Output Address... rrrrrrrr  rrrrrrrr
  Input Address.... rrrrrrrr  rrrrrrrr
  TSB Address...... rrrrrrrr  rrrrrrrr
  TCCB Address..... rrrrrrrr  rrrrrrrr
  Output Count..... hhhhhhhh         Input Count...... hhhhhhhh
  Interrogate TCW.. rrrrrrrr
```

```
tsbtype TSB at rrrrrrrr_rrrrrrrr
  Length.......... hh       Flags........... hh
  DCW Offset...... hhhh     Count........... hhhhhhhh
  TotalDevTime.... hhhhhhhh  DeferTime....... hhhhhhhh
  CUQueueTime..... hhhhhhhh  DevBusyTime..... hhhhhhhh
  DevActOnlyTime.. hhhhhhhh
  Sense Data...... hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh
                   hhhhhhhh  hhhhhhhh  hhhhhhhh  hhhhhhhh
```

```
TCCB TIDAW at rrrrrrrr_rrrrrrrr
  Flags... hh  Count... hhhhhhhh  Addr... rrrrrrrr  rrrrrrrr
```

```
TCA Header at rrrrrrrr_rrrrrrrr
  Format.......... hh       TCALen......... hh
  Serv Act Code... hhhh     Priority....... hh
```

```
DCW at rrrrrrrr_rrrrrrrr
  Command.. hh  Flags.. hh  CD Count.. hh  Count.. hhhhhhhh
```

```
DCW Control Data at rrrrrrrr_rrrrrrrr
  dddddddd dddddddd dddddddd dddddddd | ................ |
  dddddddd dddddddd dddddddd dddddddd | ................ |
```

```
Data TIDAW at rrrrrrrr_rrrrrrrr
  Flags... hh  Count... hhhhhhhh  Addr... rrrrrrrr  rrrrrrrr
```

```
Data at rrrrrrrr_rrrrrrrr
  dddddddd dddddddd dddddddd dddddddd | ................ |
  dddddddd dddddddd dddddddd dddddddd | ................ |
  **** '0020'X CONSECUTIVE BYTES ARE '00'X
  *** Back half of split data ***
  dddddddd dddddddd dddddddd dddddddd | ................ |
  dddddddd dddddddd dddddddd dddddddd | ................ |
```

```
TCA Trailer at rrrrrrrr_rrrrrrrr
  Transport Count... hhhhhhhh
= or =
TCA Trailer at rrrrrrrr_rrrrrrrr
  Write Count... hhhhhhhh Read Count... hhhhhhhh
```

```
TCAX TIDAW at rrrrrrrr_rrrrrrrr
  Flags... hh Count... hhhhhhhh Addr... rrrrrrrr_rrrrrrrr
```

```
DCW (TCAX) at rrrrrrrr_rrrrrrrr
  Command.. hh Flags.. hh CD Count.. hh Count.. hhhhhhhh
```

```
DCW Control Data (TCAX) at rrrrrrrr_rrrrrrrr
  dddddddd dddddddd dddddddd dddddddd | ................ |
  dddddddd dddddddd dddddddd dddddddd | ................ |
```

```
IOSB at vvvvvvvv
  formatted iosb data
```

```
IOBE at vvvvvvvv
  formatted iobe data
```

```
EWA at vvvvvvvv
  formatted ewa data
```

FORMAT d cccc
Format (d) and type of trace event (cccc): INTG, IO, IOCS, SSCH, or XSCH. Format is zero.

DEV hhhhh
The device number qualified with the subchannel set identifier.

ASCB hhhhhhhh
Same as the ASCB field in the INTG, IO, IOCS, SSCH, or XSCH base record.

CPU hhhh
Same as the CPU ID field in the INTG, IO, IOCS, SSCH, or XSCH base record.

JOBN cccccccc
Same as the job named (JOBN) field in the INTG, IO, IOCS, SSCH, or XSCH base record.

TCW at rrrrrrrr (vvvvvvvv)
The Transport Control Word (TCW) at real address rrrrrrrr and virtual address vvvvvvvv. The formatted TCW follows. Fields designated as "rrrrrrrr" or "rrrrrrrr_rrrrrrrr" are real addresses.

tsbtype TSB at rrrrrrrr_rrrrrrrr
The Transport Status Block (TSB) at real address rrrrrrrr_rrrrrrrr. The formatted TSB follows. The TSB is only formatted for I/O interruptions (trace events INTG, IO, and IOCS).

tsbtype
Describes the type of TSB. It can be one of the following:

- I/O status - This is a TSB for an I/O completion.
- Interrogate - This is a TSB for the completion of an interrogate operation.
- Program Check - This is a TSB for an I/O completion with status indicating a device detected program check.
- Unknown - The TSB type is not recognized. In this case, the TSB is formatted as hexadecimal data.

TCCB TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the Transport Command Control Block (TCCB) at real address rrrrrrrr_rrrrrrrr. The formatted TIDAW follows.

TCA Header at rrrrrrrr_rrrrrrrr
The Transport Control Area Header (TCAH) at real address rrrrrrrr_rrrrrrrr. The formatted TCAH follows.

DCW at rrrrrrrr_rrrrrrrr
A Device Command Word (DCW) at real address rrrrrrrr_rrrrrrrr. The formatted DCW follows.

DCW Control Data at rrrrrrrr_rrrrrrrr
The control data (command parameters) for the preceding DCW at rrrrrrrr_rrrrrrrr. The control data is formatted as hexadecimal data.

Data TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the input or output data buffers at real address rrrrrrrr_rrrrrrrr. The formatted TIDAW follows.

Data at rrrrrrrr_rrrrrrrr
Data transferred by the preceding DCW at real address rrrrrrrr_rrrrrrrr.

dddddddd dddddddd dddddddd dddddddd
Data transferred by the DCW. If there is not a series of dashes in this field, then all transferred data are displayed in four byte sections.

***** Back half of split data *****
Indicates there were more bytes of information transferred than were specified on the START command. The default value is 20 bytes, but you can specify the number of bytes to be shown. The specified value is halved; for an odd number, the larger section is shown first. The first section of data displayed comes from the beginning of the buffer from which the data was transferred. The last section comes from the end of the buffer.

TCA Trailer at rrrrrrrr_rrrrrrrr
The Transport Control Area Trailer (TCAT) at real address rrrrrrrr_rrrrrrrr. The formatted TCAT follows.

Transport Count hhhhhhhh
Fullword count of total data transferred.

Write Count hhhhhhhh
Fullword count of total write data transferred.

Read Count hhhhhhhh
Fullword count of total read data transferred.

TCAX TIDAW at rrrrrrrr_rrrrrrrr
A Transport Indirect Address Word (TIDAW) for the Transport Control Area Extension (TCAX) at real address rrrrrrrr_rrrrrrrr. The formated TIDAW follows.

DCW (TCAX) at rrrrrrrrr_rrrrrrrr
A Device Command Word (DCW) in the Transport Control Area Extension (TCAX) at real address rrrrrrrrr_rrrrrrrr. The formatted DCW follows.

DCW Control Data (TCAX) at rrrrrrrrr_rrrrrrrr
The control data (command parameters) for the preceding DCW in the TCAX at rrrrrrrr_rrrrrrrr. The control data is formatted as hexadecimal data.

IOSB vvvvvvvv
Fullword virtual address of the IOSB followed by the formatted contents of the IOSB.

IOBE vvvvvvvv
Fullword virtual address of the IOBE followed by the formatted contents of the IOBE.

EWA vvvvvvvv
Fullword virtual address of the error recovery procedure work area (EWA), followed by the formatted contents of EWA.

USR trace records

The USR record represents processing of a GTRACE macro. A user-supplied formatting routine (AMDUSRhh) formats the record. If a routine is not supplied, GTF prints the record without formatting.

This topic shows the unformatted and formatted records, then shows the following examples of USR records created by GTRACE macros in IBM-components:

- USRF9 trace records for VSAM
- USRFE trace records for BSAM, QSAM, BPAM, and BDAM
- USRFF trace records for open, close, and end-of-volume (EOV)

The USRFD trace records for VTAM are described in *z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures*.

USR records contain the following information useful for identifying the user program, MVS component, or IBM product producing the record and the routine you can use to format the record:

- Event identifiers (EIDs) identify the event that produced the record. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for USR trace records. Because each EID for USR records start with an E, unformatted USR records show just the last three numbers of the EID after the E.
- Format identifiers (FIDs) identify the routine that the system used to format the USR trace record. See "Format Identifiers (FIDs) for USR trace records" on page 296 for a list of the FIDs and associated routines.

Unformatted USR trace record

An unformatted user trace record represents processing of a GTRACE macro when a formatting routine is not supplied.

```
USR AID hh FID hhhh EID hhhh hhhhhhhh hhhhhhhh ....
```

AID hh
Application identifier, which should always be AID FF.

FID hhhh
Format identifier of the routine (AMDUSRhh) that was to format this record. See "Format Identifiers (FIDs) for USR trace records" on page 296 for a list of the FIDs and associated formatting routines for user trace records.

EID hhhh
Event identifier, which identifies the event that produced the record. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for USR trace records.

hhhhhhhh hhhhhhhh ....
Recorded data (268 bytes maximum). The data are as follows:

- Bytes 0-3: ASCB address
- Bytes 4-11: jobname
- Bytes 12-256: user data

Formatted USR trace record

A formatted user trace record represents processing of a GTRACE macro when an AMDUSRhh formatting routine is supplied.

```
USRhh hhh ASCB hhhhhhhh JOBN cccccccc
          xxxx ...
```

USRhh
Identifies the user-supplied formatting routine (AMDUSRhh). The following USR records are generated and formatted by system components, and are described in the following topics:

- USRF9 Trace Records for VSAM
- USRFD Trace Records for VTAM
- USRFE Trace Records for BSAM, QSAM, BPAM, and BDAM
- USRFF Trace Records for Open/Close/EOV

hhh
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for USR trace records.

ASCB hhhhhhhh
Address of the ASCB for the address space that created the record.

JOBN cccccc
Name of the job associated with the address space.

xxxx ...
User-formatted trace data.

USRF9 trace record for VSAM

The USRF9 trace record represents opening or closing of a VSAM data set.

```
USRF9 FF5   ASCB hhhhhhhh JOBN cccccc
            JOB NAME cccccc STEP NAME cccccc
            TIOT ENT hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh

            ACB      hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...

            AMBL     hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...

            AMB      hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...

            AMDSB    hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...

            AMB      hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...

            AMDSB    hhhhhhhh hhhhhhhh hhhhhhhh...
                     hhhhhhhh hhhhhhhh hhhhhhhh...
```

USRF9
Identifies VSAM's trace-record formatting routine (AMDUSRF9).

FF5
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for USR trace records.

ASCB hhhhhhhh
Address of the ASCB for the address space in which the event occurred.

JOBN cccccc
JOB NAME cccccc
Name of the job.

STEP NAME cccccc
Name of the job step during which the event occurred.

TIOT ENT hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
Data set entry from the task I/O table (TIOT).

ACB hhhhhhhh ...
Contents of the data set's access method control block (ACB).

AMBL hhhhhhhh ...
Contents of the AMB list (AMBL).

AMB hhhhhhhh ...
Contents of the access method block (AMB). The first AMB is for data, the second for the index.

AMDSB hhhhhhhh ...
Contents of the access method statistics block (AMDSB). The first AMDSB is for data, the second for the index.

USRFD trace record for VTAM

See *z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures* for samples of the USRFD trace records.

USRFE trace record for BSAM, QSAM, BPAM, and BDAM

The USRFE trace record represents abnormal termination of an access method routine for basic sequential access method (BSAM), queued sequential access method (QSAM), basic partitioned access method (BPAM), or basic direct access method (BDAM).

```
USRFE hhh ASCB hhhhhhhh JOBN cccccc

      BSAM/QSAM/BPAM/BDAM TRACE RECORD DDNAME cccccc ABEND CODE hh

      cccc...[AT LOCATION hhhhhhhh]
      hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh ...
      hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh ...
```

USRFE
Identifies the trace-record formatting routine (AMDUSRFE).

hhh
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See “Event Identifiers (EIDs) for USR trace records” on page 294 for a list of the EIDs and associated products for USR trace records. The event identifier (EID) corresponds to the system completion code as follows:

EID	Code
FF3	002
FF4	008
FF6	112
FF7	215
FF8	119
FF9	235
FFA	239
FFB	145
FFC	251
FFD	451
FFE	169

ASCB hhhhhhhh
Address of the ASCB for the address space in which the abnormal termination occurred.

JOBN cccccc
Name of the job associated with the address space.

BSAM/QSAM/BPAM/DBAM TRACE RECORD
Record identification provided by the AMDUSRFE formatting routine.

DDNAME cccccc
Name of the DD statement for the data set being processed.

ABEND CODE hhh
System completion code for the abnormal termination of the task.

RETURN CODE hh
Return code from the module that detected the error condition.

TIME=dd.dd.dd
Time (hour.minute.second) when the GTRACE macro was processed or blank, if the time is not available.

ccc...[AT LOCATION hhhhhhhh]
hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh ...
Data area name, or name and address, followed by the data area contents.

USRFF trace record for open/close/EOV abnormal end

This USRFF trace record represents an abnormal end during open, close, or end-of-volume (EOV).

```
USRFF   FFF    ASCB hhhhhhhh JOBN cccccc
               xxxx ...
```

USRFF
Identifies the Open/Close/EOV trace record formatting routine (IMDUSRFF).

FFF
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for USR trace records.

xxxx ...
Unformatted RRCBSA's (recovery routine control block save areas).

USRFF trace record for user requested work area

This USRFF trace record represents a user request for a work area trace.

```
USRFF      FFF     ASCB hhhhhhhh JOBN cccccc
DCB        xxxx ...
WKAREA1    xxxx ...
WKAREA2    xxxx ...
WKAREA3    xxxx ...
WKAREA4    xxxx ...
WKAREA5    xxxx ...
WTG TBL    xxxx ...
```

USRFF
Identifies the Open/Close/EOV trace record formatting routine (IMDUSRFF).

FFF
Last three numbers of the event identifier (EID) specified in the GTRACE macro. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for user trace records.

DCB
Data control block.

WKAREA1
Volume labels, file labels, DSCBs or message area. See *z/OS DFSMS Using Magnetic Tapes*.

WKAREA2
Job file control block.

WKAREA3
Internal control blocks for Open/Close/EOV. These blocks are the data control block (DCB), data extent block (DEB), and the input/output block (IOB).

WKAREA4
WKAREA5
Where-to-go-table used in transferring control among CSECTs of Open/Close/EOV.

XSCH trace record

An XSCH record represents a cancel subchannel operation. For zHPF I/O operations, a cancel subchannel can be used to initiate an interrogate operation to query the status of the I/O operation at the device.

```
XSCH.... ddddd     ASCB.... aaaaaaaa CPUID... cccc     JOBN.... jjjjjjjj
                  SID..... ssssssss CC...... cc       DVRID... dd
                  IOSLVL.. ll       UCBLVL.. ll       UCBWGT.. ww
                  BASE.... sbbbb    INTTCW.. aaaaaaaa
```

XSCH sdddd
Device number with the subchannel set identifier that the XSCH was issued for.

ASCB aaaaaaaa
Address of the ASCB.

CPU cccc
Address of the processor.

JOBN jjjjjjjj
Name of the job associated with I/O operation.

SID ssssssss
Subchannel ID from the UCBSID field of the UCB

CC cc
The condition code of the XSCH request.

DVRID dd
The IOSB driver ID field (IOSDVRID) of the request that is attempting to be cancelled.

IOSLVL ll
Function level to provide serialization of I/O requests. This value comes from the IOSLEVEL field of the IOSB.

UCBLVL ll
UCB level value from the UCBLEVEL field of the UCB.

UCBWGT ww
Flags from the UCBWGT field of the UCB.

BASE sbbbb
The base device number and subchannel set id if the device is a PAV.

INTTCW
The virtual address of the interrogate TCW, if the XSCH was used to initiate an interrogate operation, or zero.

Event Identifiers (EIDs) for USR trace records

The event identifier (EID) in GTF trace records is a 2-byte hexadecimal number that identifies the event producing the record. You can use it to identify the product that produced the record. Table 41 on page 295 shows the full 2-byte EID, but because EIDs for USR records start with an E, often unformatted USR records show just the last three numbers of the EID after the E. If you have a three number EID, such as FF5, look for EFF5 in the table.

Table 41. Event identifiers for USR trace records		
EID (hex)	**Symbolic Name**	**Issued by**
E000-E3FF		GTF user program
E400-E5F0		Reserved for IBM use
E5F1		PVM
E5F2-E5F3		Reserved for IBM use
E5F4-E5F5		NetView® Session Monitor
E5F6		NetView
E5F7	IMDOMGM	Omegamon for DB2
E5F8-E5F9		Reserved for IBM use
E5FA-E5FB		ALCS
E5FC-EF0F		Reserved for IBM use
EF10-EF1C		Reserved for IBM use
EF1D-EF1F		MVS Job Management - Dynamic Allocation (SVC 99)
EF20-EF3F		LANRES
EF40-EF41		Reserved for IBM use
EF42	IMDEF42	IBM Client I/O Sockets
EF43	IMDEF43	MVS System logger
EF44-EF45		RACF
EF46		Reserved for IBM use
EF47	IMDEF47	OSI
EF48		IOS
EF49		BDT
EF4A-EF51		Reserved for IBM use
EF52		Netview
EF53		OSI
EF54-EF5D		FSI
EF5E		Reserved for IBM use
EF5F		DB2
EF60		JES3
EF61		VSAM Buffer Manager
EF62		Dynamic output SVC installation exit
EF63		Converter/Interpreter installation exit
EF64		APPC/VM VTAM Support (AVS)
EF65		GETMAIN FREEMAIN STORAGE trace (MVS)
EF66-EF6B		Reserved for IBM use
EF6C		CICS
EF6D-EF8C		Netware
EF8D-EF8F		Reserved for IBM use

Table 41. Event identifiers for USR trace records (continued)		
EID (hex)	**Symbolic Name**	**Issued by**
EF90	IMDVTMDS	VTAM
EF91-EF9F		Reserved for IBM use
EFA0-EFA9		TCAM
EFAA		VTAM VM/SNA Console Services (VSCS)
EFAB		DFSMS Media Manger
EFAC		NetSpool
EFAD-EFAE		VM
EFAF-EFCF		Reserved for IBM use
EFD0-EFD4		Print Service Facility
EFD5-EFE0		Reserved for IBM use
EFE1	ISTVIEID	VTAM
EFE2	ISTTHEID	VTAM
EFE3	ISTTREID	VTAM
EFE4	ISTTDEID	VTAM
EFE5-EFEE		JES2
EFEF	ISTTPEID	VTAM
EFEF	ISTTPEID	VTAM
EFF0	ISTRPEID	VTAM
EFF1	ISTCLEID	VTAM
EFF2	ISTLNEID	VTAM
EFF3	IGGSP002	SAM/PAM/DAM
EFF4	IGGSP008	SAM/PAM/DAM
EFF5	IDAAM01	VSAM
EFF6	IGGSP112	SAM/PAM/DAM
EFF7	IGGSP215	SAM/PAM/DAM
EFF8	IGGSP119	SAM/PAM/DAM
EFF9	IGGSP235	SAM/PAM/DAM
EFFA	IGGSP239	SAM/PAM/DAM
EFFB	IGGSP145	SAM/PAM/DAM
EFFC	IGGSP251	SAM/PAM/DAM
EFFD	IGGSP451	SAM/PAM/DAM
EFFE	IGGSP169	SAM/PAM/DAM
EFFF	IHLMDMA1	OPEN/CLOSE/EOV

Format Identifiers (FIDs) for USR trace records

As Table 42 on page 297 shows, the format identifier (FID) in GTF trace records is a one-byte hexadecimal number that is used to determine the name of the GTFTRACE module you can use to format USR records. See *z/OS MVS IPCS Customization* for information about the GTFTRACE formatting appendage for formatting USR trace records.

Table 42. Format identifiers for USR trace records

FID (hex)	EID	Issued by	Optional format module
00	E000-EFE4	User/component	CSECT AHLFFILT in AHLFINIT
01-50	E000-E3FF	User	IMDUSR or AMDUSR (01-50)
57	EF44-EF45	RACF	AMDUSR57
81		VMSI	IMDUSR81 or AMDUSR81
84		VMSI/VTAM	IMDUSR84 or AMDUSR84
DC		PVM	IMDUSRDC or AMDUSRDC
E2-E3		PSF/MVS	IMDUSRE2-IMDUSER3 or AMDUSRE2-AMDUSER3
E6		OSI	IMDUSRE6 or AMDUSRE6
E8		FSI	IMDUSRE8 or AMDUSRE8
E9		DB2/VSAM	IMDUSRE9 or AMDUSRE9
EB		APPC/VM VTAM Support (AVS)	IMDUSREB or AMDUSREB
EC		VTAM	IMDUSREC or AMDUSREC
F5		VTAM/VSCS	IMDUSRF5 or AMDUSR5
F9	EFF5	VSAM	IMDUSRF9 or AMDUSRF9
FA	EFAB	DFSMS Media Manager	IMDUSRFA or AMDUSRFA
FD	EFEF-EFF2	VTAM	IMDUSRFD or AMDUSRFD
FE	EFF3-EFF4, EFF6-EFFE	SAM/PAM/DAM	IMDUSRFE or AMDUSRFE

Unformatted GTF trace output

This topic describes GTF output records that are not formatted by IPCS or other routines. You can use this information to write your own formatting or analysis routines.

Note: When GTF cannot obtain the data normally placed in fields of the following records, it signals this by placing one of the following values in the field

- C'U/A'. Blanks are added on the right to fill out the field.
- C'*'. Asterisks are replicated to fill out the field.

There are several types of output records:

- Control records, see “Control records” on page 298.
- Lost data records, see “Unformatted lost event records” on page 299.
- User data record, see “User data records” on page 299.
- System data records, see “System data records” on page 301.

The lost data, user data and system data records all contain optional fields, which are fields that only appear under certain conditions. The conditions are covered in the explanation for the fields. Make sure that your formatting or analysis routine takes these variable fields into account.

This section also describes the GTF system data records for individual events. See “CCW trace record” on page 302 through “SVC minimal trace record” on page 318.

Control records

GTF creates a control record at the start of each block of trace output. The control record can be followed by lost data, user data, and system data records. If this trace output was merged from multiple systems using the IPCS COPYTRC subcommand, then the control record reflects the combined GTF options in effect from all the systems. See *z/OS MVS IPCS Commands* for more information about the COPYTRC subcommand. Figure 100 on page 298 shows the format of a control record.

```
|length| res | AID | FID |time zone|time stamp|options| ... source descriptors ...
   2      2     1     1      4          8         8
 bytes  bytes  byte  byte  bytes      bytes     bytes
```

Figure 100. Unformatted control record

The fields in the control record contain the following information:

length
Total length of the record, in bytes.

res
Two bytes of zeroes. Reserved for IBM use.

AID
Application identifier, which is always zero for control records.

FID
Format identifier of the routine that will format the record, which is always X'01' for a control record.

time zone
Value showing the difference between local time and Greenwich mean time (GMT) in binary units of 1.048576 seconds when tracing began.

time stamp
Time stamp showing the eight-byte Greenwich mean time (GMT) when the control record was created.

options
An eight-byte field containing the following: The first five bytes identify the GTF options in effect for a block of trace output. See mapping macro AHLZGTO in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The remaining 3 bytes contain the following important flags, in bit ranges 0-7:

GTWCFSID - Byte 6, Bit 6
1, if the individual trace records have SIDs (system identifiers) indicating that the GTF trace data from multiple systems was merged using the IPCS COPYTRC command. In this case, there is multiple source descriptors, one for each system. The source descriptors are ranged in order by system identifier (SID). Use the value in the SID field as an array index to locate the source descriptor for a particular system.

The source descriptor information is identical in all control records within a single trace data set.

0, if the trace records have no SIDs.

GTWCFNEW - Byte 6, Bit 7
1.

Source descriptors
One or more arrays of information about the origins of the records in this block of trace data, such as the release level of the system issuing the trace data and the GTF options in effect. If GTF trace data was merged from multiple systems, there are multiple source descriptors, one for each system. Use the value in the SID field as an array index to locate the source descriptor for a particular system.

See mapping macro AHLZGTS in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary), and check the format of the source descriptor information.

Unformatted lost event records

A lost event record indicates that GTF lost the trace records for one or more events because of an error or overflow of the trace buffer. Figure 101 on page 299 shows the format of a lost event record.

```
|length| res | AID | FID |time zone|time stamp|count|  SID    |
   2      2     1     1      4          8        4      2
 bytes  bytes  byte  byte  bytes      bytes    bytes  bytes
                                                    (optional)
```

Figure 101. Unformatted Lost Event Record

The fields in the lost event record contain the following information:

length
Total length of the record, in bytes.

res
Two bytes of zeroes. Reserved for IBM use.

AID
Application identifier, which is always zero for lost event records.

FID
Format identifier. The value of FID is one of the following:

- X'02', if some trace records are missing because of an error or an overflow of the trace buffer.
- X'03', if an entire block of trace records is missing because of an error or an overflow of the trace buffer.

time zone
Value showing the difference between local time and Greenwich mean time (GMT) in binary units of 1.048576 seconds when tracing began.

time stamp
Time stamp showing the eight-byte Greenwich mean time (GMT) when the control record was created.

count
If the FID is X'02', indicating that some trace records are missing, this field contains the number of trace events that are lost.

If the FID is X'03', indicating that an entire block of trace data is missing, this field contains zeros.

SID
The system identifier of the system where this trace record was created. This 2-byte field only exists when GTF trace data from multiple systems was merged using the IPCS COPYTRC command. When present, the SID is an array index you can use to locate the source descriptor information for a particular system. For example, if the SID value for a record is 3, the source descriptor information for the system issuing the record is the third source descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple systems, look in the control record for the **options** field and see if the GTWCFSID bit is set on. See "Control records" on page 298 for the **options** field.

User data records

This topic describes the format of user trace records requested using the GTRACE macro.

If the application using GTRACE specifies more than 256 bytes of data, the user records may be split. If a user trace record is a split record, the AID will contain a value of X'F0', X'F1', or X'F3'. Split records contain the optional **sequence** and **total length** fields.

The records have the general format shown in Figure 102 on page 300.

```
|length| res | AID | FID |time stamp| EID |   SID    | sequence |total length| ASCB|job name| … data …
   2      2     1     1       8        2       2          2           4          4      8
 bytes  bytes  byte  byte   bytes    bytes   bytes      bytes       bytes      bytes  bytes
                                          (optional) (optional)  (optional)
```

Figure 102. Unformatted User trace record Format

The fields in the record contain the following information:

length
Total length of the record, in bytes.

res
Two bytes of zeros. Reserved for IBM use.

AID
Application identifier, which is one of the following:

- X'FF'-- Non-split record
- X'F0'-- The first record of a series of split records
- X'F1'-- A middle record in a series of split records
- X'F3'-- The last record in a series of split records

FID
Format identifier of the routine that will format the trace record. See "Format Identifiers (FIDs) for USR trace records" on page 296 for a list of FIDs and associated formatting routines.

time stamp
Time stamp showing the eight-byte Greenwich mean time (GMT) when the record was created.

EID
Event identifier, which identifies the event that produced the trace record. See "Event Identifiers (EIDs) for USR trace records" on page 294 for a list of the EIDs and associated products for user trace records.

SID
System identifier, which identifies the system where the record was produced. This 2-byte field only exists in the following cases:

- GTF trace data from multiple systems was merged using the IPCS COPYTRC command.
- The record is a split one. If the trace data containing this split record was not merged from multiple systems, the SID field for the split record contains zeros.

You can use the SID from merged trace data as an array index to locate the source descriptor information for a particular system. For example, if the SID value for a record is 3, the source descriptor information for the system issuing the record is the third source descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple systems, look in the control record for the **options** field and see if the GTWCFSID bit is set on. See "Control records" on page 298 for the **options** field.

sequence
Sequence number, in hexadecimal, of this split record. This field only exists for split records.

total length
Total length of the split trace data. This field only exists for split records.

ASCB
The address of the address space control block (ASCB) for the address space where the GTRACE macro was issued.

jobname
The name of the job associated with the task where the GTRACE macro was issued.

data

Contains the trace data gathered for the requested event. The length of this field varies according to the event being traced. The number of bytes of data in the data field for user records is equal to the number of bytes specified on the GTRACE macro.

System data records

GTF creates trace records for each system event you select when requesting GTF tracing. The header portion of system data records for events is shown in Figure 103 on page 301. Individual event record formats The format of individual system data records are shown in "Unformatted trace records for events" on page 301 in alphabetical order. Note that this section does not include all system events.

```
|length| res | AID | FID |time stamp| EID |   SID    | ... data ...
   2      2     1     1       8        2       2
 bytes  bytes  byte  byte   bytes    bytes   bytes
                                          (optional)
```

Figure 103. Header for Unformatted System trace record Format

The fields in the record contain the following information:

length

Total length of the record, in bytes.

res

Two bytes of zeros. Reserved for IBM use.

AID

Application identifier, which is always X'FF' for system data records.

FID

Format identifier of the routine that will format the trace record.

time stamp

Time stamp showing the eight-byte Greenwich mean time (GMT) when the record was created.

EID

Event identifier, which identifies the event that produced the trace record.

SID

System identifier, which identifies the system where the record was produced. The SID field contains zeros when the record is a split record. This 2-byte field is only created when GTF trace data from multiple systems was merged using the IPCS COPYTRC command. When present, the SID is an array index you can use to locate the source descriptor information for a particular system. For example, if the SID value for a record is 3, the source descriptor information for the system issuing the record is the third source descriptor in the control record.

To check to see whether trace data for a block of output comes from multiple systems, look in the control record for the **options** field and see if the GTWCFSID bit is set on. See "Control records" on page 298 for the **options** field.

data

Trace data gathered for the requested event. The length of this field varies according to the event being traced. The data portions for individual system trace records are shown starting on "Unformatted trace records for events" on page 301.

Unformatted trace records for events

This topic presents the records for a selection of system events in alphabetical order. It shows the unformatted layout of the data for individual event records. See "System data records" on page 301 to see the header section for the records. Note that not all system events are included in this topic. Fields in a trace record may contain the following special indicators:

N/A

Not applicable. The field does not apply in this record. In a 2-byte field, not applicable appears as `N/`.

U/A
Unavailable. GTF could not gather the information. In a 2-byte field, unavailable appears as `U/`.

The offsets for all the data records are relative and do not reflect the actual number of bytes into the record for each field. The offsets begin at the start of the data portion of the each record because the header section varies in length, depending on whether the optional SID field is present.

ADINT trace record

For a complete mapping of the adapter interruption trace record, see the AHLPCIE (PCIE trace record formats) data area in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

CCW trace record

For a complete mapping of the AHLMCWRC data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

DSP comprehensive trace record

GTF builds a DSP record when an entry is made to the dispatcher to dispatch a unit of work and TRACE=DSP is the GTF option in effect.

The FID for the DSP comprehensive trace record is X'00'. The EID is one of the following:

- X'0001' - indicates SRB dispatching.
- X'0002' - indicates LSR dispatching.
- X'0003' - indicates TCB dispatching.
- X'0004' - indicates exit prolog dispatching.

Table 43. DSP trace record offset, size, and description

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.
14 (E)	16	Resume PSW for new task.
30 (1E)	4	New TCB (for LSR and TCB, SRB for SRB).
For TCB only:		
34 (22)	8	CDE name.
For SRB only:		
34 (22)	4	Parm address.
38 (26)	1	For SRB only, SRB routine type indicator. S for a suspended SRB that is about to be re-dispatched. I for an SRB that is about to be dispatched for the first (initial) time.

DSP minimal trace record

GTF builds a DSP minimal record when an entry is made to the dispatcher to dispatch a unit of work and both `TRACE=SYSM,DSP` are the GTF options in effect.

The FID for the DSP minimal trace record is `X'03'`. The EID is one of the following values:

- `X'0001'` - indicates SRB dispatching.

- `X'0002'` - indicates LSR dispatching.
- `X'0003'` - indicates TCB dispatching.
- `X'0004'` - indicates exit prolog dispatching.

Table 44. Values for DSP minimal trace record

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	16	Resume PSW for work unit that is being dispatched.
22 (16)	4	Current TCB or N/A (for TCB and LSR only).
26 (1A)	4	Register 15.
30 (1E)	4	Register 0 or SRB.
34 (22)	4	Register 1.
38 (26)	1	For SRB only. The SRB routine type indicator. S for a suspended SRB that is about to be redispatched. I for an SRB that is about to be dispatched for the first (initial) time.

EXT comprehensive trace record

GTF builds a EXT comprehensive record when an external interruption occurs and either TRACE=SYS or TRACE=EXT are the GTF options in effect.

The FID for the EXT comprehensive trace record is X'02'. The EID is X'6201'. interruption.

Table 45. EXT comprehensive trace record offset, size, and description

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.
14 (E)	16	External old PSW.
30 (1E)	4	Old TCB.
For SIGP interrupt:		
34 (22)	4	PARMFIELD.
38 (26)	2	CPUID.
For clock comparator interrupt:		
34 (22)	2	Reserved for IBM use.
36 (24)	4	TQE exit.
40 (28)	4	TQE ASCB.
For CPU timer interrupt:		
34 (22)	2	Reserved for IBM use.
36 (24)	4	TQE exit.

EXT minimal trace record

GTF builds an EXT minimal record when an external interruption occurs and TRACE=SYSM is the GTF option in effect.

The FID for the EXT minimal trace record is X'03'. An EID of X'6201' indicates an external interruption.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	16	External old PSW.
22 (16)	4	TCB of interrupted task or N/A.
26 (1A)	4	NTQE TCB or INT CPU or N/A.

I/O summary trace record

GTF builds an I/O summary record when an I/O interruption occurs and TRACE=IOX or TRACE=IOXP is a GTF option in effect. To trace PCI I/O interruptions, TRACE=PCI must also be in effect.

The FID for the I/O summary record is X'08'. The EID is one of the following:

- X'2100' - indicates a PCI I/O interruption
- X'5107' - indicates an EOS I/O interruption
- X'5202' - indicates an I/O interruption with a valid UCB
- X'5203' - indicates a IOCS I/O interruption. It indicates an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility.

The I/O summary record will always contain a header section, followed by a section header and a common section. The section header describes the type and length of the following section and an indicator if this is the last section of the record.

A typical I/O summary record for a dasd device would have a header section, a common section, a data set section, a CMB section and, probably, one or more CCW sections. If an I/O summary record has to be extended the following extension records would consist of a header section with the header record number greater than 1, a common section and one or more CCW sections.

Header section

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.
14 (E)	2	Device number.
16 (10)	4	System ID.
20 (14)	1	Driver ID.
21 (15)	1	Trace version.
22 (16)	2	Record count.

Section header

Offset	Size	Description
0 (0)	1	Section type: • FL1'0': Common section • FL1'1': CMB section • FL1'3': CCW Orientation section • FL1'4': Data set section
1 (1)	1	Flag identifiers.
	1...	Last section of this record.
2 (2)	2	Section length.

Common section

Offset	Size	Description
0 (0)	1	Device class.
1 (1)	1	Device status.
2 (2)	1	Error codes: indicate errors found during CCW analysis. See "CCW error codes" on page 307 for a description.
3 (3)	1	Flag byte.
	1...	Last trace recordof this I/O event.
	.1..	Reserve (conditional or unconditional).
	..1.	Release.
	...1	At least one search CCW.
4 (4)	6	Volume serial.
10 (A)	4	Device type.

Data set section

Offset	Size	Description
0 (0)	1	Data set type.
1 (1)	1	Name length.
2 (2)	44	Data set name.

CMB Section

Offset	Size	Description
0 (0)	2	Number of SSCH instructions.
2 (2)	2	Number of SSCH instructions for which data was collected.
4 (4)	4	Sum of device connect times.
8 (4)	4	Sum of SSCH request pending times.
12 (C)	4	Sum of subchannel disconnect times.
16 (10)	4	Sum of control unit queueing times.
20 (14)	8	Time stamp for the start of this I/O request.

Offset	Size	Description
28 (1C)	4	Device active only time.
32 (20)	4	Number of SSCH instructions
36 (24)	4	Number of SSCH instructions for which data was collected
40 (28)	4	Sum of device busy times
44 (2C)	4	Sum of initial command response times

CCW orientation section

Offset	Size	Description
0 (0)	1	Orientation sequence number.
1 (1)	1	Flag byte 1.
	1...	At least one SILI bit on.
	.1..	At least one suspend bit on.
	..1.	At least one PCI.
	...1	At least one skip bit on.
	.... 1...	Read record zero or read home address.
	.... .1..	Reserved.
	.... ..1.	Read multiple CKD or track.
	1	At least one erase CCW.
2 (2)	1	Flag byte 2.
	1...	End of file read or written.
	.1..	At least one format write.
	..1.	This record is for an FCX (zHPF) channel program
3 (3)	1	Count of erase, read MCKD.
4 (4)	2	Number of blocks read.
6 (6)	2	Number of blocks written.
8 (8)	4	Number of bytes read.
12 (C)	4	Number of bytes written.
16 (10)	2	Number of data chain CCWs.
18 (12)	2	Number of COM chain CCWs.
20 (14)	1	External global attribute.
	11..	'11' (only allowed value).
	..1.	CKD conversion mode.
	...1	Subsystem operation mode.
	.... ..1.	Cache fast write.
	1	Inhibit DASD fast write.
21 (15)	1	External global attribute extended.
22 (16)	4	External end of extent CCH.
26 (1A)	1	Seek/locate code.

Offset	Size	Description
27 (1B)	5	CCHHR (search ID equal).
32 (20)	1	Operation code from locate parameter.
33 (21)	1	Sector number from parameter.
34 (22)	1	Extended operation code.
35 (23)	2	Extended parameters.

CCW error codes

Table 46. CCW error codes

Error code (Hex)	Error Code Name	Description
03	IONOCCW	Bad CCW
04	IOINSPAC	Insufficient space in table
05	IOX4K	Target crosses 4K boundary
06	IOLASTCC	Last CCW not trace
07	IOSRCHCD	Search CCW with data chain
08	IOINSCNT	Insufficient count
09	IOCPUNAV	Channel program unavailable
0A	IORCKD8	Read CKD with count <8
0B	IONOIDAW	No IDAW where needed
0C	IOCSWINV	Invalid CSW
0D	IOTICBAD	Invalid TIC address
0E	IOEBIDA	IDAL across page bound
0F	IOEBIDAL	IDAL not aligned on correct boundary
10	IOMIDNOQ	MIDAW not aligned on correct boundary
11	IOMIDZCT	MIDAW contains a zero count
12	IOMIDPBD	MIDAW crosses a page boundary (ignored if the skip bit is on)
13	IOMIDIDA	Both the IDAW and MIDAW bits are on
14	IONOTCW	Failure accessing TCW
15	IONOTCAH	Failure accessing TCA header
16	IONOTSB	Failure accessing TSB
17	IONOTIDA	Failure accessing TIDAW
18	IONODCW	Failure accessing DCW
19	IOINCMDL	TCW command length in error
1A	IOCmdParmInvalid	Command dependent parameters contain invalid information

I/O trace record

GTF builds an I/O record when an I/O interruption occurs and TRACE=SYSM, TRACE=SYS, TRACE=IO, or TRACE=IOP are the GTF options in effect. To trace PCI I/O interruptions, TRACE=PCI must also be in effect.

The FID for the I/O trace record is X'07'. The EID is one of the following:

- X'2100' - indicates a PCI I/O interruption.
- X'5101' - indicates an EOS I/O interruption.
- X'5200' - indicates an I/O interruption with a valid UCB.
- X'5201' - indicates a IOCS I/O interruption. It indicates an I/O interruption that also contains concurrent sense information, for devices that support the concurrent sense facility.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.
14 (E)	2	Device number.
16 (10)	16	I/O old PSW.
32 (20)	20	IRB words 0-4.
52 (34)	4	TCB.
56 (38)	2	Sense bytes.
58 (3A)	1	IOSB Flag (IOSFLA).
59 (3B)	1	IOSB Option (IOSOPT).
60 (3C)	1	IOS Driver ID.
61 (3D)	1	IOS level.
62 (3E)	1	UCB level.
63 (3F)	1	Flags (UCBGWT).
64 (40)	2	Base device number.
66 (42)	44	IRB words 5–15 (for EID X'5201' only).

PCIDMX trace record

For a complete mapping of the PCIE de-multiplexing trace record, see the AHLPCIE (PCIE trace record formats) data area in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

PCILG trace record

For a complete mapping of the PCILG trace record, see the AHLPCIE (PCIE trace record formats) data area in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

PCISTG trace record

For a complete mapping of the PCISTG trace record, see the AHLPCIE (PCIE trace record formats) data area in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

PI comprehensive trace record

GTF builds a PI comprehensive record when a program interruption occurs and either TRACE=PI or TRACE=SYS are the GTF options in effect.

The FID for the PI comprehensive trace record is X'00'. The EID is one of the following:

- X'6101' - indicates a program interruption with codes 1-17, 19, and 128.

- X'6200' - indicates a program interruption with code 18.

Offset	Size	Description
0 (0)	4	ASCB
4 (4)	2	CPUID
6 (6)	8	Job name
14 (E)	16	Program old PSW
30 (1E)	4	INT TCB
34 (22)	4	Virtual page address low half
38 (26)	8	RB or CDE name
46 (2E)	64	GRs 0 - 15 bits 32 - 63
110 (6E)	64	GRs 0 - 15 bits 0 - 31
174 (AE)	64	ARs 0 - 15
238 (EE)	4	Virtual page address high half

PI minimal trace record

GTF builds a PI minimal record when a program interruption occurs and TRACE=SYSM is the GTF option in effect.

The FID for the PI minimal trace record is X'03'. The EID is one of the following:

- X'6101' - indicates a program interruption with codes 1-17, 19, and 128.
- X'6200' - indicates a program interruption with code 18.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	16	Program old PSW.
22 (16)	4	Old TCB.
26 (1A)	4	Virtual page address low half.
30 (1E)	4	Register 15.
34 (22)	4	Register 1.
38 (26)	4	Virtual page address high half.

RR comprehensive trace record

GTF builds an RR comprehensive record when a recovery routine is invoked and TRACE=SYS or TRACE=RR are the GTF options in effect.

The FID for the RR comprehensive trace record is X'04'. The EID is one of the following:

- X'4002' - indicates STAE/ESTAE invocation.
- X'4003' - indicates FRR invocation.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.

Offset	Size	Description
14 (E)	8	Name of recovery routine or U/A.
22 (16)	16	PSW current when error occurred.
38 (26)	4	Completion code.
42 (2A)	8	Reserved for IBM use.
50 (32)	4	Retry address or N/A.
54 (36)	4	Address of SDWA (STAE/ESTAE only).

RR minimal trace record

GTF builds an RR minimal record when a recovery routine is invoked and TRACE=SYS is the GTF option in effect.

The FID for the RR minimal trace record is X'03'. The EID is one of the following:

- X'4002' - indicates STAE/ESTAE invocation.
- X'4003' - indicates FRR invocation.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	16	Error PSW.
22 (16)	4	Completion code.
26 (1A)	4	Reserved for IBM use.
30 (1E)	3	Reserved for IBM use.
33 (21)	1	Return code from recovery routine (STAE/ESTAE only).
34 (22)	4	Retry address or N/A. **Note:** If no return code at offset 33, begin retry address at offset 33.
38 (26)	4	Address of SDWA (STAE/ESTAE only). **Note:** If retry address begins at offset 33, SDWA address begins at offset 37.

SLIP trace records

GTF builds a SLIP trace record when TRACE=SLIP is the GTF option in effect and:

- A SLIP trap has matched and either TRACE or TRDUMP has been specified on the SLIP command.
- A SLIP trap is in DEBUG mode (specified on the SLIP command) and is inspected by the SLIP processor as a result of any SLIP event.

The SLIP trace records are:

- SLIP Standard Trace Record
- SLIP Standard/User Trace Record
- SLIP User Trace Record
- SLIP DEBUG Trace Record

SLIP standard trace record

The FID for the SLIP standard trace record is X'04'. The EID is X'4004'.

A field will contain asterisks if an error occurred when attempting to obtain data or the data is unavailable because it is paged out.

Offset	Size	Description
0 (0)	4	ASCB address.
4 (4)	2	CPUID. (**Note:** When SLIP is entered from RTM2, the CPUID recorded may be different from the CPUID when RTM2 was running.)
6 (6)	8	Jobname from current address space (or N/A).
14 (E)	4	SLIP trap ID.
18 (12)	2	ASID of current address space.
20 (14)	8	Job step program name (or U/A or N/A).
28 (1C)	4	TCB address (or N/A).
32 (20)	1	System mode indicators, byte 1:
	1...	Supervisor control mode.
	.1..	Disabled for I/O and external interrupts.
	..1.	Global spin lock held.
	...1	Global suspend lock held.
	.... 1...	Local lock held.
	1	Type 1 SVC in control.
	.... ..1.	SRB mode.
	1	TCB mode.
33 (21)	1	System mode indicators, byte 2:
	1...	Recovery routine in control (always zero if a PER interrupt).
	.1..	Problem program state.
	..1.	Supervisor state.
	...1	System key.
	.... 1...	Problem program key.
	.... .1..	Any global lock held.
	.... ..1.	Any lock held.
34 (22)	1	Error byte 1 (or zeros if a PER interrupt):
	1...	Program check interrupt.
	.1..	Restart interrupt.
	..1.	SVC error.
	...1	Abend; task issued SVC 13.
	.... 1...	Paging I/O error.
	.... .1..	Dynamic address translation error.
	.... ..1.	Software error caused by machine check.
	1	Abnormal address space termination.
35 (23)	1	Error byte 2 (or zeros if a PER interrupt):
	1...	Memterm.
36 (24)	1	SLIP flags:
	1...	DEBUG record.

Offset	Size	Description
	.1..	Registers collected.
37 (25)	2	Data unavailable counter (or zeros if DATA was not specified for the trap).

The following fields apply only to PER interrupts otherwise set to N/A (or N for one-byte fields).

Offset	Size	Description
39 (27)	8	Load module name in which the interrupt occurred (or U/A or N/A).
47 (2F)	4	Offset in load module (or U/A or N/A).
51 (33)	8	Address of the instruction that caused the PER interrupt.
59 (3B)	6	Instruction content (six bytes of data beginning at the address of the instruction that caused the PER interrupt).
65 (41)	8	Target instruction address if EXECUTE instruction (or N/A or U/A).
73 (49)	6	Target instruction content if EXECUTE instruction (six bytes of data beginning at the target instruction address), or (N/A or U/A).
79 (4F)	4	Beginning range virtual address if SA (storage-alteration) specified on SLIP command (or N/A).
83 (53)	4	Four bytes of storage starting at beginning range virtual address if SA specified (or N/A or U/A).
87 (57)	16	Program old PSW.
103 (67)	4	Program interrupt code (PIC) and instruction length code.
107 (6B)	1	PER interrupt code:
	1..	Successful-branch event (SB).
	.1..	Instruction-fetch event (IF).
	..1.	Storage-alteration event (SA).
108 (6C)	1	PER trap mode:
	1...	Successful-branch monitoring (SB).
	.1..	Instruction-fetch monitoring (IF).
	..1.	Storage-alteration monitoring (SA).
	...x	Reserved.
	.... 1...	PER trap.
	.... .1..	Recovery specified.
	.... ..1.	Message flag.
	1	Message flag.
109 (6D)	2	Key mask.
111 (6F)	2	SASID.
113 (71)	2	Authorization index.
115 (73)	2	PASID.

Offset	Size	Description
117 (75)	1	PSW ASC mode indicator • F0: primary addressing mode • F1: access register addressing mode • F2: secondary addressing mode • F3 home addressing mode
118 (76)	13	Storage Alteration Space Identifier • For an address space: contains the ASID • For a data space: contains the owning ASID and the dataspace name
131 (83)	4	High-half of begin range
135 (87)	1	Transactional execution DATA filter mismatch count

SLIP standard + user trace record

The FID for the SLIP Standard + User trace record is X'04'. The EID is X'4005'.

Offset	Size	Description
0 (0)	136	
through 135		Fields are identical to the SLIP standard record.
136 (88)	2	Length of user-defined data.
138 (8A)	variable	User-defined data (specified through the TRDATA parameter on the SLIP command).

SLIP user trace record

The FID for the SLIP user trace record is X'04'. The EID is X'4006'.

Offset	Size	Description
0 (0)	2	CPUID.
2 (2)	2	Extension number.
4 (4)	1	Continuation length.
5 (5)	2	Length of the user defined data.
7 (7)	variable	User-defined data (specified through the TRDATA parameter on the SLIP command).

Note:

1. If the SLIP user requests registers to be placed in the SLIP user record, they are the first field in the record.
2. A length field of zero indicates that the user-defined data was not available (for example, the data is paged out).
3. The TRDATA parameter on the SLIP command specifies one or more data ranges. The number of records needed to trace these ranges depends on the size of the ranges specified. The trace contains a standard plus (+) user record from the next range or a user record followed by as many user records and user continuation records as needed to trace the ranges specified. The header for each record contains the CPUID and extension number to help correlate the output (extension numbers apply only to user and user continuation records). When a record is partially filled and the data from the next range will not fit in the remaining space; the partially filled record is written to the trace data set. Another user record is built, the extension number is increased by one, and the continuation length is set to zero. The data length and data is then copied into this record.

When the length of the data from a range is greater than 249 bytes, the excess data is put in user continuation records in the following manner. The data length and first 248 bytes are put in a user record. After writing that record a user continuation record is built. The extension number is increased by one and the continuation length is set to the number of bytes of data to be put in this record. If more than 251 bytes of data are left, 248 bytes are copied into record, and it is written. User continuation records are built until all the data in from that range is traced.

SLIP DEBUG trace record

The FID for the SLIP DEBUG trace record is X'04'. The EID is X'4005'.

Offset	Size	Description
0 (0)	136	
through 135		Fields are identical to the SLIP standard record.
136 (88)	1	DEBUG byte; indication of which keyword failed: • Decimal 2 indicates COMP keyword • Decimal 3 indicates ASID keyword • Decimal 4 indicates JOBNAME keyword • Decimal 5 indicates JSPGM keyword • Decimal 6 indicates PVTMOD keyword • Decimal 7 indicates LPAMOD keyword • Decimal 8 indicates ADDRESS keyword • Decimal 9 indicates MODE keyword • Decimal 10 indicates ERRTYP keyword • Decimal 13 indicates RANGE keyword • Decimal 14 indicates DATA keyword • Decimal 20 indicates ASIDSA keyword • Decimal 22 indicates REASON CODE keyword • Decimal 23 indicates NUCMOD keyword • Decimal 24 indicates PSWASC keyword • Decimal 26 indicates DSSA keyword
137 (89)	1	Reserved.

Note: The high-order bit in the SLIP flags (SFLG) field (at offset X'34') is set on to indicate a DEBUG record.

SRM comprehensive trace record

GTF builds an SRM comprehensive record when system resource manager is invoked and TRACE=SYS or TRACE=SRM are the trace options in effect.

The FID for the SRM trace record is X'04'. The EID is X'4001'.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname
14 (E)	4	Register 15.
18 (12)	4	Register 0.
22 (16)	4	Register 1.

SRM minimal trace record

GTF builds an SRM minimal record when system resource manager is invoked and TRACE=SYSM is the GTF option in effect.

The FID for the SRM minimal trace record is X'03'. The EID is X'4001'.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	4	Register 15.
10 (A)	4	Register 0.
14 (E)	4	Register 1.

SSCH trace record

GTF builds an SSCH record when an SSCH event occurs and TRACE=SYSM, TRACE=SYS, TRACE=SYSP, TRACE=SSCH or TRACE=SSCHP are the GTF options in effect.

The FID for the SSCH trace record is X'00'. The EID is X'5105'.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Jobname.
14 (E)	2	Device number.
16 (10)	4	Real address of channel program.
20 (14)	4	Virtual address of channel program.
24 (18)	4	Reserved for IBM use.
28 (1C)	1	Condition code.
29 (1D)	12	Reserved for IBM use.
41 (29)	8	Dynamic seek address.
49 (31)	1	Reserved for IBM use.
50 (32)	1	Reserved for IBM use.
51 (33)	1	Reserved for IBM use.
52 (34)	1	Reserved for IBM use.
53 (35)	1	Reserved for IBM use.
54 (36)	1	Reserved for IBM use.

SVC comprehensive trace records

GTF builds SVC comprehensive records when an SVC interruption occurs and either the TRACE=SYS or TRACE=SVC GTF option is in effect. All SVC records contain the basic data described below; however, many SVC numbers invoke additional data recording, described following the basic data.

The FID for the SVC comprehensive trace record is X'010'. The EID is X'1000'.

Table 47. Basic SVC comprehensive trace record

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	8	Job name.
14 (E)	16	SVC old PSW. The seventh and eighth bytes contain the SVC number.
30 (1E)	4	Old TCB.
34 (22)	8	CDE name.
42 (2A)	4	Register 15.
46 (2E)	4	Register 0.
50 (32)	4	Register 1.

GTF builds only a basic comprehensive trace record for the following SVCs:

Name	Number	Name	Number
EXIT	3	TEST	97
GETMAIN/FREEMAIN	10	SUBMIT	100
TIME	11	QTIP	101
SYNCH	12	XLATE	103
MGCR	34	TOPCTL	104
WTL	36	IMBLIB	105
TTROUTER	38	REQUEST	106
CIRB	43	MODESET	107
TTIMER	46	None	109
TTOPEN	49	DSTATUS	110
NOP	50	JECS	111
OLTEP	59	RELEASE	112
TSAV	61	SIR	113
CHATR	72	BLKPAGE	115
(IFBSTAT)	76	None	116
STATUS	79	None	117
SMFWTM	83	DSSPATCH	118
(IGC084)	84	TESTAUTH	119
SWAP	85	GETMAIN/FREEMAIN	120
EMSERV	89	None	121
VOLSTAT	91	LINK, LOAD, XCTL	122
TPUT/TGET	93	PURGEDQ	123
TSO terminal control	94	TPIO	124
SYSEVENT	95		

Basic SVC comprehensive trace record with parameter list information

For detailed information about data gathered for the following SVCs, see *z/OS MVS Diagnosis: Reference*.

Name	Number	Name	Number
EXCP	0	STIMER	47
WAIT/WAITR	1	DEQ	48
POST	2	SNAP	51
GETMAIN	4	RESTART	52
FREEMAIN	5	RELEX	53
LINK	6	DISABLE	54
XCTL	7	EOV	55
LOAD	8	ENQ/RESERVE	56
DELETE	9	FREEDBUF	57
ABEND	13	RELBUF/REQBUF	58
SPIE	14	STAE	60
ERREXCP	15	DETACH	62
PURGE	16	CHKPT	63
RESTORE	17	RDJFCB	64
BLDL/FIND	18	BTAMTEST	66
OPEN	19	BSP	69
CLOSE	20	GSERV	70
STOW	21	ASGNBFR/BUFINQ/ RLSEBFR	71
OPEN TYPE = J	22	SPAR	73
CLOSE TYPE = T	23	DAR	74
DEVTYPE	24	DQUEUE	75
TRKBAL	25	LSPACE	78
CATLG	26	GJP	80
OBTAIN	27	SETPRT	81
SCRATCH	29	ATLAS	86
RENAME	30	DOM	87
FEOB	31	MOD88	88
ALLOC	32	TCBEXCP	92
WTO/WROR	35	PROTECT	98
SEGLD/SEGWT	37	Dynamic allocation	99
LABEL	39	EXCPVR	114
EXTRACT	40		
IDENTIFY	41		
ATTACH	42		
CHAP	44		

Name	Number	Name	Number
OVLYBRCH	45		

SVC minimal trace record

GTF builds an SVC minimal record when an SVC interruption occurs and TRACE=SYSM is the GTF option in effect. The FID for the SVC minimal trace record is X'010'. The EID is X'1000'.

Offset	Size	Description
0 (0)	4	ASCB.
4 (4)	2	CPUID.
6 (6)	16	SVC old PSW. The seventh and eighth bytes contain the SVC number.
22 (16)	4	Old TCB.
26 (1A)	4	Register 15.
30 (1E)	4	Register 0.
34 (22)	4	Register 1.

TCW trace record

For a complete mapping of the AHLFCXG (FCX/zHPF channel program) data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Chapter 11. The generic tracker facility

Overview

The intended purpose of the generic tracker is to be a migration aid and to help assess exploitation of old and new function. For example, you can assess use of interfaces you intend to make obsolete, and assess exploitation of old and new function in general. For functions that are already obsolete in a new release, you can use the generic tracker to assess their use on previous releases.

The generic tracker is a tracking facility that provides these services:

- A callable tracking service that users can instrument code with:
  - The caller passes an event address and other related information to the service.
  - The service attempts to resolve the event address to a program name and store it with the other information for later analysis.
- A callable query service that extracts previously stored track records and tracking facility information.
- Operator commands to display and maintain tracking facility information and configuration details.
- Support for parmlib members for easy reuse of tracking facility configuration statements.
- More tools to aid in the use of the tracking facility and the information it stores.

The generic tracker replaces the Console Tracking Facility, which was available in releases before z/OS V2R1. The Console Tracking Facility operator commands are available only in releases before z/OS V2R1. Macro CNZTRKR continues to be supported, but the recommendation is to use macro GTZTRACK instead. Any information CNZTRKR collects is stored in the new tracking facility. Do not continue to use service CNZTRKR. When possible, replace existing CNZTRKR invocations with GTZTRACK invocations.

The generic tracker consists of the following parts:

- A system parameter named GTZ - see *z/OS MVS Initialization and Tuning Guide*.
- Support for GTZPRMxx parmlib members. See *z/OS MVS Initialization and Tuning Guide*
  - Shipped with GTZPRM00, which includes an exclusion list.
- Operator commands. See *z/OS MVS System Commands*.
  - SET GTZ – add GTZPRMxx members
  - SETGTZ – control the tracking facility
  - DISPLAY GTZ – report tracking facility information.
- Callable macro services. See *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.
  - GTZTRACK – track an event
    - Associated mapping macro GTZZTRK
  - GTZQUERY – report tracking facility information
    - Associated mapping macros for Assembler (GTZZQRY) and METAL-C (GTZHQRY)
- REXX callable functions. See "GTZLQRY REXX callable function" on page 322.
  - GTZLQRY – report tracking facility information
- Utility programs
  - GTZPRINT – report tracking facility information
    - Associated mapping macro GTZZPRT
  - GTZCNIDT – create GTZPRMxx from CNIDTRxx
    - Associated mapping macro GTZZCNI

  - GTZSMFU2 and GTZSMFU3 – retrieve GTZ data from standard SMF record backend stores and format the data in text form
- Samples – see SYS1.SAMPLIB
  - GTZCNIDJ – start utility program GTZCNIDT
  - GTZPRNTJ – start utility program GTZPRINT
  - GTZSHCK – a sample health check to report tracked events
  - GTZSHCKJ – build health check GTZSHCK
  - GTZSMFJ – use of utility programs GTZSMFU2 and GTZSMFU3

The generic tracker is a system address space that starts automatically during the IPL of the system. A restart is not required or recommended unless it is explicitly requested, for example to apply service. The source of the IBM supplied started procedure GTZ can be modified to configure the MEMLIMIT, which determines how much information the tracking facility can store.

Actual tracking (that is, recording track events) is disabled by default. When tracking is disabled, invocations of GTZTRACK are allowed, but ignored by the system. Use the SETGTZ TRACKING=ON operator command to enable tracking.

Besides storing track data in the dynamic storage of the GTZ address space, you can also enable the tracking facility to persist track data using SMF records. For more information, see "Data persistence" on page 341.

Before you enable tracking and allow the tracking facility to store data and report it back to the users, consider protecting the DISPLAY GTZ and SETGTZ operator commands. As mentioned in *z/OS MVS System Commands*, these GTZ commands can be protected by RACF resource profiles for the OPERCMDS class. The security administrator can, for example, take the following steps to give a user profile access to the MVS.SETGTZ.GTZ resource:

1. Ensure that the OPERCMDS class is active:

```
SETROPTS CLASSACT(OPERCMDS) RACLIST(OPERCMDS)
```

2. Create the MVS.SETGTZ.GTZ resource profile and require explicit access to be given:

```
RDEFINE OPERCMDS MVS.SETGTZ.GTZ UACC(NONE)
```

3. Grant individual access to the resource:

```
PERMIT MVS.SETGTZ.GTZ CLASS(OPERCMDS) ID(user) ACCESS(UPDATE)
```

4. Refresh the OPERCMDS class to activate the changes:

```
SETROPTS RACLIST(OPERCMDS) REFRESH
```

All SETGTZ operator commands can be used in GTZPRMxx parmlib members. For example, you can put a TRACKING=ON statement into a GTZPRMxx parmlib member.

System parameter GTZ can be set to one or more suffixes, which identify the GTZPRMxx parmlib members that the tracking facility must process at startup. See the description of system parameter GTZ in *z/OS MVS Initialization and Tuning Reference*. It contains requirements and details on how to grant the tracking facility permission to access the parmlib members at start-up. If you do not specify system parameter GTZ, or specify PARM='GTZPRM=*NONE' in procedure GTZ, the tracking facility starts without reading any parmlib members initially.

You can add more GTZPRMxx parmlib members later with the operator command SET GTZ.

You can use the SETGTZ CLEAR operator command to remove configuration data previously added by using GTZPRMxx members and to remove previously recorded track data.

To prevent certain known track events from recording any data, you can add EXCLUDE statements. The tracking facility ignores any future track data that matches such EXCLUDE filters and it removes any matching previously recorded track data. Typically EXCLUDE statements are provided in GTZPRMxx parmlib members, and can be specified by using the SETGTZ EXCLUDE operator command too.

A standard list of EXCLUDE statements for known track events is shipped in parmlib member GTZPRM00. You can download it from z/OS downloads (www.ibm.com/systems/z/os/zos/downloads). It is recommended that you use the GTZPRMxx parmlib member as is and enable it by default by adding the 00 suffix to the system parameter GTZ in IEASYSxx using GTZ=(00,*your_suffixes*). Keeping GTZPRM00 separate allows for easier updates when z/OS releases a new GTZPRM00 version. More GTZPRMxx parmlib members would then have any additional EXCLUDEs and, for example, a TRACKING=ON, if wanted.

If you used the Console Tracking Facility and its CNIDTRxx parmlib members for exclusion lists before, you can use the GTZCNIDT utility program to create GTZPRMxx parmlib members from existing CNIDTRxx parmlib members. The sample JCL GTZCNIDJ explains how to start GTZCNIDT.

A unique tracked event is identified by the following fields, which are referenced by the EXCLUDE filters. The tracking facility increments an associated occurrence count if a GTZTRACK call results in an identical set of unique fields:

- The program name and program offset as resolved from the passed in event address.
- The name of the owner and source of the track event.
- The event description text.
- The (binary) event data.
- Job names and ASIDs associated with the track event.
- The authorization state that is associated with the track event.

For events that are difficult to track, where the program name cannot be determined by the system from the passed in event address, you can define DEBUG filters using the SETGTZ DEBUG operator command or in a GTZPRMxx parmlib member. The DEBUG statement allows, for example, to trigger a non-percolating ABEND, code E77, with custom reason codes, that you can define SLIP traps for to collect dumps.

You can use the DISPLAY GTZ operator command to display:

- General status and statistics for the tracking facility – DISPLAY GTZ[,STATUS].
- All or a subset of the track data that is currently stored in the tracking facility – DISPLAY GTZ,TRACKDATA[(*filters*)].
- All the EXCLUDE statements – DISPLAY GTZ,EXCLUDE.
- All the DEBUG statements – DISPLAY GTZ,DEBUG.

Using the utility program GTZPRINT, you can collect information in a data set instead of reading it on the console or in the system LOG. The sample JCL GTZPRNTJ explains how to use GTZPRINT.

Here are some typical scenarios

- Use GTZPRINT to preserve tracking information for later analysis. This helps when you must submit the information to IBM, or an independent software vendor (ISV) product to have them analyze the output or diagnose problems.
- Use DISPLAY GTZ for quick interactive access. Certain command line and output length limits might apply.
- Use callable macro service GTZQUERY when you want to access tracking facility information from a program. For example, for a custom reporting tool or for a health check for the IBM Health Checker for z/OS. See the sample health check GTZSHCK shipped in SAMPLIB. This can be built with the sample JCL GTZSHCKJ. While local health checks run authorized, other uses of GTZQUERY require the calling user ID to have access to the `GTZ.`*`sysname`*`.QUERY` RACF profile that is used similar to how the GTZ operator commands are protected.

GTZLQRY REXX callable function

REXX callable function GTZLQRY allows REXX programs access to data that is provided by the existing macro service GTZQUERY.

The GTZLQRY function uses corresponding request types that are provided by REXX (input) variable GTZLQRY_REQUEST to support all four published request types of the GTZQUERY macro service:

- GTZLQRY_REQUEST = "STATUS" “GTZLQRY "STATUS" interface” on page 322
- GTZLQRY_REQUEST = "TRACKDATA" “GTZLQRY "TRACKDATA" interface” on page 326
- GTZLQRY_REQUEST = "EXCLUDE" “GTZLQRY "EXCLUDE" interface” on page 333
- GTZLQRY_REQUEST = "DEBUG" “GTZLQRY "DEBUG" interface” on page 337

The requested data is provided as REXX STEM output variables.

Like the GTZQUERY macro service, the TRACKDATA request supports additional filters that allow you to limit the resulting data to only a subset of the potentially very many available track data entries.

For TRACKDATA entries, EXCLUDE statements, and DEBUG statements, which are the three request types that return lists of items, GTZLQRY supports additional OPTION input parameters to further limit the result sets.

GTZLQRY "STATUS" interface

Accepts input using a single variable. The following required statement requests to return status information:

```
GTZLQRY_REQUEST="STATUS"
```

GTZLQRY with REQUEST="STATUS" returns output with STEM variable GTZQUAAS. Diagnostic information such as the return code is provided with separate variables.

Table 48. REQUEST="STATUS" output variable names and descriptions

Output variable name	Variable description
GtzQuaaS.TrackEnabled	1, if tracking is currently enabled, otherwise "0"
GtzQuaaS.Full	1, if the tracking facility is full, otherwise "0"
GtzQuaaS.ExcludeNoPrm	1, if some EXCLUDE statements did not originate from GTZPRMxx, otherwise "0"
GtzQuaaS.DebugNoPrm	1, if some DEBUG statements did not originate from GTZPRMxx, otherwise "0"
GtzQuaaS.GtzPrmFull	1, if some GTZPRMxx could not be recorded centrally, otherwise "0"
GtzQuaaS.ClearedALL	1, if a CLEAR=ALL has been used at some time for this tracker instance, otherwise "0"
GtzQuaaS.ClearedTRACKDATA	1, if a CLEAR=TRACKDATA or a CLEAR=ALL removed trackdata at some time for this tracker instance, otherwise "0"
GtzQuaaS.ClearedEXCLUDE	1, if a CLEAR=EXCLUDE or a CLEAR=ALL removed EXCLUDE statements at some time for this tracker instance, otherwise "0"
GtzQuaaS.ClearedDEBUG	1, if a CLEAR=DEBUG or a CLEAR=ALL removed DEBUG statements at some time for this tracker instance, otherwise "0"

Table 48. REQUEST="STATUS" output variable names and descriptions (continued)	
Output variable name	**Variable description**
GtzQuaaS.PersistSMF	ON ('1'b), if data persistence by SMF records is enabled.
GtzQuaaS.EnabledCount	Number of times the tracking facility moved from disabled to enabled. An unsigned number in the single-word (32-bit) range.
GtzQuaaS.TrackDataEntriesAvailable	Total number of unique tracked instances currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.ExcludeEntriesAvailable	Total number of exclusion statements currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.DebugEntriesAvailable	Total number of DEBUG statements currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.TrackDataEntriesEncountered	Total number of non-unique tracked instances currently known to the tracking facility. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.ExcludeRejectCount	Total number of GTZTRACK requests rejected due to a matching EXCLUDE statement. This counter is reset when the EXCLUDE statements are cleared. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.DebugActionCount	Total number of GTZTRACK requests which triggered a DEBUG action as specified by a matching DEBUG statement (with its LIMIT not exceeded yet). This counter is reset when the DEBUG statements are cleared. An unsigned number in the double-word (64-bit) range.
GtzQuaaS.GtzPrmSuffixes.0	Number of GTZPRMxx members currently known to the tracking facility. An unsigned number in the half-word (16-bit) range.
GtzQuaaS.GtzPrmSuffixes	The i=1..GtzQuaaS.GtzPrmSuffixes.0 GTZPRMxx 2-character suffixes currently known to the tracking facility.
GtzQuaaS.GtzPrmIplSuffixes.0	Number of GTZPRMxx members specified at IPL time from system parameter GTZ. The currently known GTZPRMxx suffix list might be different than this IPL-time list, if suffixes have been added or cleared in between. An unsigned number in the half-word (16-bit) range.
GtzQuaaS.GtzPrmIplSuffixes	The i=1..GtzQuaaS.GtzPrmIplSuffixes.0 GTZPRMxx 2-character suffixes specified at IPL time.

Table 48. REQUEST="STATUS" output variable names and descriptions (continued)

Output variable name	Variable description
GtzQuaaS.MemAvailPercent	A number between 0 and 100, indicating the percentage of how much of the tracking facility's total dynamic memory is still available to store data for tracking entries, EXCLUDE statements, etc. When GtzQuaaS.Full is "1" then GtzQuaaS.MemAvailPercent is zero.
GtzQuaaS.SystemName	The name of the system the (unique per system) tracking facility is running on. An up to 8 character long string.
GtzQuaaS.EnabledTOD	The time stamp when tracking was last enabled (if GtzQuaaS.TrackEnabled="1"), or when tracking was last disabled (if GtzQuaaSTrackEnabled = "0"), where for the latter this might be the timeof when the facility started, if tracking has not been enabled since. See also GtzQuaaS.EnabledCount. This value is provided in STCK format. See, for example, BLSUXTOD to format this as human readable time stamp.
GTZLQRY_RC	Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured from, for example, RC = GTZLQRY(). This is a decimal text string so that, for example, RC=12 is reported as "12", not "0C". The possible return codes are as follows: **0** **Meaning:** The GTZLQRY request completed successfully. **4** **Meaning:** The GTZLQRY request completed successfully, but issued a warning. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **8** **Meaning:** The GTZLQRY request encountered an error and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **12 (X'C')** **Meaning:** The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **16 (X'10')** **Meaning:** The GTZLQRY request encountered an internal, component error and did not complete successfully. **Action:** Contact IBM Service.

Table 48. REQUEST="STATUS" output variable names and descriptions (continued)	
Output variable name	**Variable description**
GTZLQRY_RSN	(See the description following this table.)
GTZLQRY_SYSTEMDIAG	Additional diagnostic information for some nonzero return codes. An up to 100-character string.

Description for output variable GTZLQRY_RSN: Reason code. For nonzero return codes. This is a hexadecimal text string so that, for example, RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"
Meaning: The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrrr0801"
Meaning: The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"
Meaning: An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure that you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"
Meaning: An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure that you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero, then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"rrrr0C01"
Meaning: The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"
Meaning: GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"
Meaning: GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code E77 with the following possible ABEND reason codes:

Table 49. REQUEST="STATUS" ABEND reason codes and descriptions	
ABEND reason code	**Reason code description**
X'rrrr0C27'	**Meaning:** GTZLQRY could not write any output REXX variables. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A nonzero "rrrr" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values.
X'xxxx0C29'	**Meaning:** An error occurred while accessing the internal GTZLQRY parameter list. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions.
X'xxxx10xx'	**Meaning:** An unexpected error occurred. **Action:** Contact IBM Service.

GTZLQRY "TRACKDATA" interface

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return track data entries:

```
GTZLQRY_REQUEST="TRACKDATA"
```

The following optional statement requests to return only up to maxresults number of trackdata entries values:

```
GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }
```

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

Optionally enter one or more filter variables, all starting with GTZLQRY_FILTER. Note that all filter values, except for SOURCETYPE and PROGRAMTYPE, also accept values including wild card characters. For text values, an asterisk ('*') matches zero or more characters of any type, and a question mark ('?') matches a single character of any type. For numeric or binary values, such as EVENTDATA or EVENTASID, only a single asterisk ('*') is supported, as a way to explicitly specify the otherwise implied default value of "match all or match any value".

The following optional statement requests to return only such tracked instances that have a matching OWNER value.

```
GTZLQRY_FILTER.OWNER = "owner"
```

The value is a 1-16 character string.

The following optional statement identifies what type of tracked instance source values to match.

```
GTZLQRY_FILTER.SOURCETYPE = { "ALL" | "NOPATH" | "PATH" }
```

The following optional statement requests to return only those tracked instances that have a matching SOURCE value.

```
GTZLQRY_FILTER.SOURCE = "source"
```

The value is a 1-8 character string that also requires the specification of GTZLQRY_FILTER.SOURCETYPE = "NOPATH".

The following optional statement requests to return only that tracked instances that have a matching SOURCEPATH value.

```
GTZLQRY_FILTER.SOURCEPATH = "sourcepath"
```

The value is a 1-1024 character string that also requires the specification of GTZLQRY_FILTER.SOURCETYPE = "PATH".

The following optional statement requests to return only those tracked instances that have a matching EVENTDESC value.

```
GTZLQRY_FILTER.EVENTDESC = "eventdesc"
```

The value is a 0-64 character string.

The following optional statement requests to return only those tracked instances that have a matching EVENTDATA value.

```
GTZLQRY_FILTER.EVENTDATA = {"eventdata" | "*"}
```

The value, if not "*", is a 16 byte string with a binary representation that is used for comparisons.

The following optional statement requests to return only those tracked instances that have a matching EVENTASID value.

```
GTZLQRY_FILTER.EVENTASID = {"eventASID" | "*"}
```

The value is a number between 0 and 65535 ('FFFF'X)

Tip: REXX function X2D can help specify the number in hex format in the REXX source.

The following optional statement requests to return only those tracked instances that were associated with a matching job name of the address space identified by the EVENTASID value.

```
GTZLQRY_FILTER.EVENTJOB = "eventjob"
```

The value is a 1-8 character string.

The following optional statement identifies the type of tracked instance program values to match.

```
GTZLQRY_FILTER.PROGRAMTYPE = { "ALL" | "NOPATH" | "PATH" }
```

The following optional statement requests to return only those tracked instances that have a matching PROGRAM value.

```
GTZLQRY_FILTER.PROGRAM = "program"
```

The value is a 1-8 character string that also requires the specification of GTZLQRY_FILTER.PROGRAMTYPE = "NOPATH".

The following optional statement requests to return only those tracked instances that have a matching PROGRAMPATH values.

```
GTZLQRY_FILTER.PROGRAMPATH = "programpath"
```

The value is a 1-1024 character string that also requires the specification of GTZLQRY_FILTER.PROGRAMTYPE = "PATH".

The following optional statement requests to return only those tracked instances that have a matching PROGRAMOFFSET value.

```
GTZLQRY_FILTER.PROGRAMOFFSET = { "programoffset" | "*"}
```

The value is an unsigned number in the double-word (64-bit) range.

Tip: REXX function X2D can help specify the number in hex format in the REXX source.

The following optional statement requests to return only those tracked instances that have a matching HOMEASID value.

```
GTZLQRY_FILTER.HOMEASID = { "homeASID" | "*"}
```

The value is a number between 0 and 65535 ('FFFF'X).

Tip: REXX function X2D can help specify the number in hex format in the REXX source.

The following optional statement requests to return only those tracked instances that have a matching HOMEJOB value.

```
GTZLQRY_FILTER.HOMEJOB = "homejob"
```

The value is a 1-8 character string.

Specifying GTZLQRY with REQUEST="TRACKDATA" returns its output with the variables that are described in the following table.

Table 50. REQUEST="TRACKDATA" output variable names and descriptions

Output variable name	**Variable description**
GTZQUAAT.TrackDataEntriesAvailable	Number of TRACKDATA entries which are known to the tracking facility and which match the (optional) filter.
GTZQUAAT.TrackDataEntriesProvided	Number (n) of TRACKDATA entries provided, if n > 0, from variables GTZQUAAT.*i*. with i=1..n. This number n is determined as MIN(GTZQUAAT.TrackDataEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).
GtzQuaaT.*i*.isSourcePath	1, if GtzQuaaT.*i*.Source represents a SOURCE, otherwise "0", to indicate a SOURCEPATH
GtzQuaaT.*i*.isProgramPath	1, if GtzQuaaT.Program represents a PROGRAM, otherwise "0", to indicate a PROGRAMPATH
GtzQuaaT.*i*.isAuthorized	1, if the tracked event ran authorized, otherwise "0"
GtzQuaaT.*i*.FirstTOD	The timestamp when the first instance of this (unique) tracked instance was recorded (all others just had the occurrence count Incremented). This value is provided in STCK form. See, for example, for BLSUXTOD to format this as human readable time stamp.
GtzQuaaT.*i*.Count	How often this (unique) tracked instance was recorded.
GtzQuaaT.*i*.Owner	OWNER value, an up to 16-character string
GtzQuaaT.*i*.Source	SOURCE or SOURCEPATH value, depending on GtzQuaaT.*i*.isSourcePath, an up to 8-character resp. 1024-character string
GtzQuaaT.*i*.EventDesc	EVENTDESC value, an up to 64-character string
GtzQuaaT.*i*.EventData	EVENTDATA value, a 16-byte binary string

Table 50. REQUEST="TRACKDATA" output variable names and descriptions (continued)

Output variable name	Variable description
GtzQuaaT.*i*.EventJob	The derived EVENTJOB-name value, an up to 8-character string
GtzQuaaT.*i*.HomeJob	The derived HOMEJOB-name value, an up to 8-character string
GtzQuaaT.*i*.Program	The derived PROGRAM or PROGRAMPATH value, depending on GtzQuaaT.*i*.isProgramPath, an up to 8-character resp. 1024-character string
GtzQuaaT.*i*.ProgramOffset	The derived PROGRAMOFFSET value, a decimal unsigned number in the double-word (64-bit) range
GtzQuaaT.*i*.EventASID	EVENTASID value, a decimal number in the range 0 through 65535 (X'FFFF')
GtzQuaaT.*i*.HomeASID	HOMEASID value, a decimal number in the range 0 through 65535 (X'FFFF')
GTZLQRY_RC	Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured from, for example, RC = GTZLQRY(). This is a decimal text string so that, for example, RC=12 is reported as "12", not "0C". The possible return codes are as follows: **0** **Meaning:** The GTZLQRY request completed successfully. **4** **Meaning:** The GTZLQRY request completed successfully, but issued a warning. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **8** **Meaning:** The GTZLQRY request encountered an error and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **12 (X'C')** **Meaning:** The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **16 (X'10')** **Meaning:** The GTZLQRY request encountered an internal, component error and did not complete successfully. **Action:** Contact IBM Service.
GTZLQRY_RSN	(See the description following this table.)

Table 50. REQUEST="TRACKDATA" output variable names and descriptions (continued)	
Output variable name	**Variable description**
GTZLQRY_SYSTEMDIAG	Additional diagnostic information for some nonzero return codes. An up to 100-character string.

Description for output variable GTZLQRY_RSN: Reason code. For nonzero return codes. This is a hexadecimal text string so that, for example, RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"
: **Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

: **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrrr0801"
: **Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

: **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"
: **Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

: **Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"
: **Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

: **Action:** Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"rrLL0804"
: **Meaning:** An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.

: **Action:** Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"xxxx0805"
: **Meaning:** No SOURCETYPE or an invalid SOURCETYPE has been specified from input variable GTZLQRY_FILTER.SOURCETYPE.

: **Action:** When GTZLQRY_FILTER.SOURCE is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.SOURCETYPE of "NOPATH".

"xxxx0806"
: **Meaning:** No PROGRAMTYPE or an invalid PROGRAMTYPE has been specified from input variable GTZLQRY_FILTER.PROGRAMTYPE.

: **Action:** When GTZLQRY_FILTER.PROGRAM is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.PROGRAMTYPE of "NOPATH".

"xxxx0807"
: **Meaning:** No SOURCETYPE or an invalid SOURCETYPE has been specified from input variable GTZLQRY_FILTER.SOURCETYPE.

: **Action:** When GTZLQRY_FILTER.SOURCEPATH is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.SOURCETYPE of "PATH".

"xxxx0808"
: **Meaning:** No PROGRAMTYPE or an invalid PROGRAMTYPE has been specified from input variable GTZLQRY_FILTER.PROGRAMTYPE.

: **Action:** When GTZLQRY_FILTER.PROGRAMPATH is specified on input, ensure that you also specify a matching GTZLQRY_FILTER.PROGRAMTYPE of "PATH".

"rrLL0809"

Meaning: An unrecognized GTZLQRY_FILTER.SOURCETYPE value has been found.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCETYPE values ("ALL", "PATH", or "NOPATH"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCETYPE input variable.

"rrLL080A"

Meaning: An unrecognized GTZLQRY_FILTER.PROGRAMTYPE value has been found.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAMTYPE values ("ALL", "PATH", or "NOPATH"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAMTYPE input variable.

"rrLL080B"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.SOURCE input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCE values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCE input variable.

"rrLL080C"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.PROGRAM input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAM values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAM input variable.

"rrLL080D"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.SOURCEPATH input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.SOURCEPATH values (up to 1024 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.SOURCEPATH input variable.

"rrLL080E"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.PROGRAMPATH input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAMPATH values (up to 1024 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAMPATH input variable.

"rrLL080F"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.EVENTDESC input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.EVENTDESC values (up to 64 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTDESC input variable.

"rrLL0810"

Meaning: When attempting to retrieve the GTZLQRY_FILTER.EVENTDATA input value an error was encountered.

Action: Ensure you only specify valid GTZLQRY_FILTER.EVENTDATA values (16 bytes long, or '*'). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTDATA input variable.

"rrLL0811"
: **Meaning:** When attempting to retrieve the GTZLQRY_FILTER.EVENTJOB input value an error was encountered.

  Action: Ensure you only specify valid GTZLQRY_FILTER.EVENTJOB values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTJOB input variable.

"rrLL0812"
: **Meaning:** When attempting to retrieve the GTZLQRY_FILTER.HOMEJOB input value an error was encountered.

  Action: Ensure you only specify valid GTZLQRY_FILTER.HOMEJOB values (up to eight characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.HOMEJOB input variable.

"rrLL0813"
: **Meaning:** When attempting to retrieve the GTZLQRY_FILTER.OWNER input value an error was encountered.

  Action: Ensure you only specify valid GTZLQRY_FILTER.OWNER values (up to 16 characters long). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.OWNER input variable.

"rrLL0814"
: **Meaning:** An unrecognized GTZLQRY_FILTER.PROGRAMOFFSET value has been found.

  Action: Ensure you only specify valid GTZLQRY_FILTER.PROGRAMOFFSET values (double-word number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.PROGRAMOFFSET input variable.

"rrLL0815"
: **Meaning:** An unrecognized GTZLQRY_FILTER.EVENTASID value has been found.

  Action: Ensure you only specify valid GTZLQRY_FILTER.EVENTASID values (half-word number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.EVENTASID input variable.

"rrLL0816"
: **Meaning:** An unrecognized GTZLQRY_FILTER.HOMEASID value has been found.

  Action: Ensure you only specify valid GTZLQRY_FILTER.HOMEASID values (half-word number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_FILTER.HOMEASID input variable.

"rrrr0C01"
: **Meaning:** The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

  Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"
: **Meaning:** GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

  Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"
: **Meaning:** GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

  Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code E77 with the following possible ABEND reason codes:

Table 51. REQUEST="TRACKDATA" ABEND reason codes and descriptions	
ABEND reason code	**Reason code description**
X'rrrr0C27'	**Meaning:** GTZLQRY could not write any output REXX variables. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A nonzero "rrrr" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values.
X'xxxx0C29'	**Meaning:** An error occurred while accessing the internal GTZLQRY parameter list. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions.
X'xxxx10xx'	**Meaning:** An unexpected error occurred. **Action:** Contact IBM Service.

GTZLQRY "EXCLUDE" interface

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return EXCLUDE statements:

```
GTZLQRY_REQUEST="EXCLUDE"
```

The following optional statement requests to return only up to maxresults number of EXCLUDE statements:

```
GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }
```

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

Note: This is provided mainly for consistency among the list requests such as REQUEST="TRACKDATA". Unlike REQUEST="TRACKDATA", which potentially can return tens of thousands or more entries, REQUEST="EXCLUDE" is expected to return at most a few dozen entries in a typical configuration.

Specifying GTZLQRY with REQUEST="EXCLUDE" returns its output with the variables described in the following table.

Table 52. REQUEST="EXCLUDE" output variable names and descriptions	
Output variable name	**Variable description**
GTZQUAAE.ExcludeEntriesAvailable	Number of EXCLUDE statements known to the tracking facility.
GTZQUAAE.ExcludeEntriesProvided	Number (n) of EXCLUDE statements provided, if n > 0, from variables GTZQUAAE.*i*. with i=1..n. This number n is determined as MIN(GTZQUAAE.ExcludeEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).

Table 52. REQUEST="EXCLUDE" output variable names and descriptions (continued)	
Output variable name	**Variable description**
GtzQuaaE.*i*.OriginType	Indicates where this EXCLUDE originated from: PARMLIB - A GTZPRMxx parmlib member, COMMAND - A SETGTZ EXCLUDE command, PROGRAM - A program interface
GtzQuaaE.*i*.OriginSuffix	If this EXCLUDE was specified from a GTZPRMxx parmlib member, as indicated by a GtzQuaaE.*i*.OriginType value of "PARMLIB", this field here contains the 2-character xx suffix.
GtzQuaaE.*i*.Filter.	A set of variables indicating what TRACKDATA entries this EXCLUDE statement is supposed to filter out. For all but GtzQuaaE.*i*.Filter.SourceType and GtzQuaaE.*i*.Filter.ProgramType this might be a single wildcard character "*", when not filtering for a specific value, indicating that all values for the respective field match this filter.
GtzQuaaE.*i*.Filter.SourceType	Indicates what type of source this filter is defined to match: SOURCE, "SOURCEPATH", or "ALL"
GtzQuaaE.*i*.Filter.ProgramType	Indicates what type of program this filter is defined to match: PROGRAM, "PROGRAMPATH", or "ALL"
GtzQuaaE.*i*.Filter.Owner	OWNER filter value. An up to 16-character string.
GtzQuaaE.*i*.Filter.Source	SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaE.*i*.Filter.SourceType: For SourceType="SOURCE" it is a source filter, for SourceType="SOURCEPATH" it is a source path filter, and for SourceType="ALL" it is always "*" (match all). An up to 8-character resp. 1024-character string
GtzQuaaE.*i*.Filter.EventDesc	Event description filter value. An up to 64-character string.
GtzQuaaE.*i*.Filter.EventData	Event data filter value. A 16-byte binary string or "*".
GtzQuaaE.*i*.Filter.EventJob	Event job filter value. An up to 8-character string.
GtzQuaaE.*i*.Filter.HomeJob	Home job filter value. An up to 8-character string.
GtzQuaaE.*i*.Filter.Program	PROGRAM or PROGRAMPATH filter value, depending on the value of GtzQuaaE.*i*.Filter.ProgramType: For ProgramType="PROGRAM" it is a source filter, for ProgramType="PROGRAMPATH" it is a program path filter, and for ProgramType="ALL" it is always "*" (match all). An up to 8-character resp. 1024-character string
GtzQuaaE.*i*.Filter.ProgramOffset	PROGRAMOFFSET filter value. A decimal unsigned number in the double-word (64-bit) range
GtzQuaaE.*i*.Filter.EventASID	Event ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')

Table 52. REQUEST="EXCLUDE" output variable names and descriptions (continued)	
Output variable name	**Variable description**
GtzQuaaE.*i*.Filter.HomeASID	Home ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')
GTZLQRY_RC	Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured from, for example, RC = GTZLQRY(). This is a decimal text string so that, for example, RC=12 is reported as "12", not "0C". The possible return codes are as follows: **0** **Meaning:** The GTZLQRY request completed successfully. **4** **Meaning:** The GTZLQRY request completed successfully, but issued a warning. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **8** **Meaning:** The GTZLQRY request encountered an error and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **12 (X'C')** **Meaning:** The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **16 (X'10')** **Meaning:** The GTZLQRY request encountered an internal, component error and did not complete successfully. **Action:** Contact IBM Service.
GTZLQRY_RSN	(See the description following this table.)
GTZLQRY_SYSTEMDIAG	Additional diagnostic information for some nonzero return codes. An up to 100-character string.

Description for output variable GTZLQRY_RSN: Reason code. For nonzero return codes. This is a hexadecimal text string so that, for example, RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"
: **Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

: **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrrr0801"
: **Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

: **Action:** Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"

Meaning: An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"

Meaning: An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0804"

Meaning: An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.

Action: Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"rrrr0C01"

Meaning: The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"

Meaning: GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"

Meaning: GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code of E77 with the following possible ABEND reason codes:

Table 53. REQUEST="EXCLUDE" ABEND reason codes and descriptions

ABEND reason code	Reason code description
X'rrrr0C27'	**Meaning:** GTZLQRY could not write any output REXX variables. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A nonzero "rrrr" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values.
X'xxxx0C29'	**Meaning:** An error occurred while accessing the internal GTZLQRY parameter list. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions.
X'xxxx10xx'	**Meaning:** An unexpected error occurred. **Action:** Contact IBM Service.

GTZLQRY "DEBUG" interface

Accepts input using multiple variables. See also the description of corresponding parameters on the GTZQUERY macro service for more details on single parameters. The following required statement requests to return DEBUG statements:

```
GTZLQRY_REQUEST="DEBUG"
```

The following optional statement requests to return only up to maxresults number of DEBUG statements:

```
GTZLQRY_OPTION.MAXRESULTS = { "ALL" | "maxresults" }
```

Enter special value "ALL" or an unsigned number in the double-word (64-bit) range.

Note: This is provided mainly for consistency among the list requests such as REQUEST="TRACKDATA". Unlike REQUEST="TRACKDATA", which potentially can return tens of thousands or more entries, REQUEST="DEBUG" is expected to return at most a few dozen entries in a typical configuration.

Specifying GTZLQRY with REQUEST="DEBUG" returns its output with the variables described in the following table.

Table 54. REQUEST="DEBUG" output variable names and descriptions

Output variable name	Variable description
GTZQUAAD.DebugEntriesAvailable	Number of DEBUG statements known to the tracking facility.
GTZQUAAD.DebugEntriesProvided	Number (n) of DEBUG statements provided, if n > 0, from variables GTZQUAAD.*i*. with i=1..n. This number n is determined as MIN(GTZQUAAD.DebugEntriesAvailable, GTZLQRY_OPTION.MAXRESULTS).
GTZQUAAD.*i*.OriginType	Indicates where this DEBUG originated from: PARMLIB - A GTZPRMxx parmlib member, COMMAND - A SETGTZ DEBUG command, PROGRAM - A program interface
GTZQUAAD.*i*.OriginSuffix	If this DEBUG was specified from a GTZPRMxx parmlib member, as indicated by a GtzQuaaD.*i*.OriginType value of "PARMLIB", this field here contains the 2-character xx suffix.
GtzQuaaD.*i*.Reason	The DEBUG REASON code. A decimal number in the range 0 through 65535 (X'FFFF')
GtzQuaaD.*i*.Action	The ACTION requested when this DEBUG is matched: "ABEND" or "DUMP".
GtzQuaaD.*i*.ActionLimit	How often the system is allowed to trigger the action for this DEBUG statement (when matched by a new tracked instance candidate). A value of "NOLIMIT" means "no limit", otherwise a decimal number in the range 1 through 65535.
GtzQuaaD.*i*.ActionCount	How often this DEBUG statement triggered an action so far. A decimal number in the range 0 through 65535 which counts towards the ActionLimit, unless "NOLIMIT".

Table 54. REQUEST="DEBUG" output variable names and descriptions (continued)

Output variable name	Variable description
GtzQuaaD.*i*.Filter.	A set of variables indicating what TRACKDATA entries this DEBUG statement is supposed to trigger for. For all but GtzQuaaD.*i*.Filter.SourceType and GtzQuaaD.*i*.Filter.ProgramType this might be a single wildcard character "*", when not filtering for a specific value, indicating that all values for the respective field match this filter.
GtzQuaaD.*i*.Filter.Owner	OWNER filter value. An up to 16-character string.
GtzQuaaD.*i*.Filter.Source	SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaD.*i*.Filter.SourceType: For SourceType="SOURCE" it is a source filter, for SourceType="SOURCEPATH" it is a source path filter, and for SourceType="ALL" it is always "*" (match all). An up to 8-character resp. 1024-character string
GtzQuaaD.*i*.Filter.EventDesc	Event description filter value. An up to 64-character string.
GtzQuaaD.*i*.Filter.EventData	Event data filter value. A 16-byte binary string or "*".
GtzQuaaD.*i*.Filter.EventJob	Event job filter value. An up to 8-character string.
GtzQuaaD.*i*.Filter.HomeJob	Home job filter value. An up to 8-character string.
GtzQuaaD.*i*.Filter.Program	PROGRAM or PROGRAMPATH filter value, depending on the value of GtzQuaaD.*i*.Filter.ProgramType: For ProgramType="PROGRAM" it is a source filter, for ProgramType="PROGRAMPATH" it is a program path filter, and for ProgramType="ALL" is always "*" (match all). An up to 8-character resp. 1024-character string
GtzQuaaD.*i*.Filter.ProgramOffset	PROGRAMOFFSET filter value. A decimal unsigned number in the double-word (64-bit) range
GtzQuaaD.*i*.Filter.EventASID	Event ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')
GtzQuaaD.*i*.Filter.HomeASID	Home ASID filter value. A decimal number in the range 0 through 65535 (X'FFFF')
GtzQuaaD.*i*.Filter.Owner	OWNER filter value. An up to 16-character string.
GtzQuaaD.*i*.Filter.Source	SOURCE or SOURCEPATH filter value, depending on the value of GtzQuaaD.*i*.Filter.SourceType: For SourceType="SOURCE" it is a source filter, for SourceType="SOURCEPATH" it is a source path filter, and for SourceType="ALL" it is always "*" (match all). An up to 8-character resp. 1024-character string

Table 54. REQUEST="DEBUG" output variable names and descriptions (continued)

Output variable name	Variable description
GTZLQRY_RC	Return code. Same as the REXX supplied output variable RESULT which REXX sets if the function result is not explicitly captured from, for example, RC = GTZLQRY(). This is a decimal text string so that, for example, RC=12 is reported as "12", not "0C". The possible return codes are as follows: **0** **Meaning:** The GTZLQRY request completed successfully. **4** **Meaning:** The GTZLQRY request completed successfully, but issued a warning. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **8** **Meaning:** The GTZLQRY request encountered an error and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **12 (X'C')** **Meaning:** The GTZLQRY request encountered a severe error, typically related to the environment in which GTZLQRY was invoked, and did not complete successfully. **Action:** Refer to the action under the individual reason code returned in GTZLQRY_RSN. **16 (X'10')** **Meaning:** The GTZLQRY request encountered an internal, component error and did not complete successfully. **Action:** Contact IBM Service.
GTZLQRY_RSN	(See the description following this table.)
GTZLQRY_SYSTEMDIAG	Additional diagnostic information for some nonzero return codes. An up to 100-character string.

Description for output variable GTZLQRY_RSN: Reason code. For nonzero return codes. This is a hexadecimal text string so that, for example, RSN=X'00000841' is reported as "00000841", not "2113". The reason codes are as follows:

"rrrr0401"
: **Meaning:** The underlying GTZQUERY service reported a return code of 4 and reason code "rrrr".

  Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrrr0801"
: **Meaning:** The underlying GTZQUERY service reported a return code of 8 and reason code "rrrr".

  Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"rrLL0802"
: **Meaning:** An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0803"

Meaning: An unrecognized GTZLQRY_REQUEST value has been found.

Action: Ensure you only specify valid GTZLQRY request types ("STATUS", "TRACKDATA", "EXCLUDE", or "DEBUG"). If "LLLL" is not zero then it is the last two bytes of the length as reported by service IRXEXCOM used to retrieve the GTZLQRY_REQUEST input variable.

"LLLL0804"

Meaning: An unrecognized GTZLQRY_OPTION.MAXRESULTS value has been found.

Action: Ensure you only specify valid GTZLQRY_OPTION.MAXRESULTS values ("ALL" or a valid number). If "rrLL" is not zero then "rr" is the return code and "LL" is the last byte of the length of the variable as reported by service IRXEXCOM used to retrieve the GTZLQRY_OPTION.MAXRESULTS input variable.

"rrrr0C01"

Meaning: The underlying GTZQUERY service reported a return code of 12 (X'C') and reason code "rrrr".

Action: Refer to the action under reason code "rrrr" in the documentation for service GTZQUERY.

"xxxx0C02"

Meaning: GTZLQRY was invoked with a NUL REXX ENVBLOCK address in register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

"xxxx0C03"

Meaning: GTZLQRY was invoked with an unrecognized ENVBLOCK, pointed to by register 0 on entry.

Action: Only invoke GTZLQRY from within a REXX program/environment, ensuring standard REXX calling conventions.

In addition, GTZLQRY can issue an ABEND with system completion code of E77 with the following possible ABEND reason codes:

Table 55. REQUEST="DEBUG" ABEND reason codes and descriptions

ABEND reason code	Reason code description
X'rrrr0C27'	**Meaning:** GTZLQRY could not write any output REXX variables. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, which includes GTZLQRY being called with a valid ENVBLOCK in register 0 on entry. A nonzero "rrrr" value provides additional diagnostic information. Check the GTZLQRY_RSN variable description for possibly related values.
X'xxxx0C29'	**Meaning:** An error occurred while accessing the internal GTZLQRY parameter list. **Action:** Ensure that GTZLQRY is called within a valid REXX environment, ensuring standard REXX linkage conventions.
X'xxxx10xx'	**Meaning:** An unexpected error occurred. **Action:** Contact IBM Service.

Data persistence

Use the generic tracker service GTZTRACK to provide persisted records of tracked events in SMF type 125 records. With this function, you can retrieve historical information about tracked events whenever needed.

The generic tracker keeps track of many different types of data, including the following data types:

- An "exclusion list" made up of one or more EXCLUDE statements
- A set of DEBUG statements
- A set of TRACKDATA instances
- STATUS information

Of all of these, only TRACKDATA is enabled for persistence using SMF records.

This persistence of TRACKDATA is done in parallel to the TRACKDATA that is kept in the dynamic memory of the GTZ address space. The TRACKDATA that is persisted as SMF records is meant to serve as a backup, or long-term storage for later analysis, of such data. This is because the data that is kept in dynamic memory is only available until the GTZ address space shuts down, at the latest before the next system restart. You can still achieve GTZ data persistence across those system restarts without using SMF records by instead using the existing tools GTZPRINT, GTZQUERY, or DISPLAY GTZ. These tools can explicitly extract the data from the live GTZ address space.

The external interface for the TRACKDATA persistence using SMF records consists of the following pieces:

- SETGTZ operator subcommand PERSIST. SETGTZ PERSIST enables or disables recording using SMF records.
- GTZPRMxx parmlib member statement PERSIST, which is equivalent to the SETGTZ PERSIST operator command.
- Operator command DISPLAY GTZ, STATUS reports the current enablement status of SMF recording for GTZ data.
- Macro service GTZQUERY REQUEST=STATUS reports the current enablement status of SMF recording for GTZ data.
- REXX callable function GTZLQRY with GTZLQRY_REQUEST="STATUS" reports the current enablement status of SMF recording for GTZ data among other information.
- Programs GTZSMFU2 and GTZSMFU3 allow for the retrieval of GTZ data from standard SMF record backend stores (data set or jobstream).
- Mapping macro GTZZSMF1 for the new top-level SMF record type 125, which is assigned to GTZ for its data persistence.
- Macro GTZZSMFU for return and reason codes from programs GTZSMFU2 and GTZSFMU3.
- Sample member GTZSMFJ in SYS1.PARMLIB demonstrates the use of SMF dump program exit routines GTZSMFU2 and GTZSMFU3.

SMF already provides the following dump programs to retrieve raw binary data from different SMF backend stores for SMF records:

- Program IFASMFDP for SMF records that are stored in data sets
- Program IFASMFDL for SMF data records that are stored in log streams.

However, these existing dump programs also support the use of user exit routines to filter and process individual SMF records from the overall set of SMF records that are stored in the targeted SMF record store (data set or logstream).

The generic Tracker provides exit routines for two of those dump program user exits in order to extract and format, in text form that is very similar to the existing DISPLAY GTZ,TRACKDATA output format, any GTZ SMF records that are contained in the SMF record source:

- Program GTZSMFU2, to be used as an exit routine for exit USER2, is given control only when the SMF data set dump program selects a record to be written. GTZSMFU2 gains control for each record that is selected by any optional general filtering that is specified through the existing supported input to the SMF dump programs. GTZSMFU2 ignores any non-GTZ SMF records and formats GTZ SMF records as text in a new output DD, named GTZOUT, of the SMF dump program.

- Program GTZSMFU3, to be used as an exit routine for exit USER3, is given control after the output data set is closed. Pair it with USER2 exit routine GTZSMFU2 so that GTZSMFU3 can close any GTZ output data sets that GTZSMFU2 opened initially and shared across multiple calls for different SMF records that are fed into GTZSMFU2.

In addition to GTZOUT as output DD, the exit routines GTZSMFU2 and GTZSMFU3 also expect a DD GTZPRINT as target for additional messages from the GTZ processing. Both GTZ DDs require LRECL=80.

Note: Usage details are also described in the prologue of the GTZSMFJ sample job.

On exit, the GTZOUT DD contains text similar to that in the following example:

```
/* GENERATED BY GTZSMFU2 (HBB77AZ-14260)  2014-09-17 19:40:09 */

----------------------------------------------------------------------
 INSTANCE:      1                   COUNT:         1
 EVENTDESC:     'GTZFCT neutral event description'
 OWNER:         IBM_GTZ_FCT         SOURCE:        MYSOURCE
 EVENTDATA:     xC7E3E9C6C3E3C2C5   xC5C6F1F500000004
 PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000064
 HOMEJOB:       MAINASID            HOMEASID:      x003A
 EVENTJOB:      MAINASID            EVENTASID:     x003A
 AUTHORIZED:    YES                 FIRST TIME:    2014-09-17 18:38:04
 SYSPLEX:       PLEX1               SYSTEM:        SY39
 SMF SID:       SY39
----------------------------------------------------------------------
 INSTANCE:      2                   COUNT:         1
 EVENTDESC:     'GTZTRACK DIAGNOSE'
 OWNER:         IBMGTZ              SOURCE:        GTZKCTRL
 EVENTDATA:     x0000000000000002   x4040404040404040
 PROGRAM:       GTZKPVT             PROGRAMOFFSET: x000000000002A762
 HOMEJOB:       *MASTER*            HOMEASID:      x0001
 EVENTJOB:      *MASTER*            EVENTASID:     x0001
 AUTHORIZED:    YES                 FIRST TIME:    2014-09-17 19:10:09
 SYSPLEX:       PLEX1               SYSTEM:        SY39
 SMF SID:       SY39

/* GTZSMFU2 END TIME 2014-09-17 19:40:10 */
```

Mapping macro GTZZSMF1 provides the mapping for SMF records of type 125 and all supported subtypes. There is currently only one subtype, for TRACKDATA.

Mapping macro GTZZSMF1 is also referenced in the standard SMF record mapping macro IFASMFRD, which covers record types 124-127 and is referenced by the main include IFASMFR. That way customers can also create mappings for the GTZ SMF record type 125 using standard SMF include protocol 'IFASMFR 125'.

Use new operator command SETGTZ PERSIST or new statement PERSIST in the GTZPRMxx parmlib member to enable or disable track data persistence. For more information, see the SETGTZ PERSIST command in *z/OS MVS System Commands* and GTZPRMxx (the Generic Tracker parameters) in *z/OS MVS Initialization and Tuning Reference*.

Generic tracker exploiters

DFSMS tracking

- Event description starting with **GDGLIMIT**
  - EVENTDESC: 'GDGLIMIT jobname stepname procname'

    The field name GDGLIMIT was used in a call to Catalog Management and did not also include GDGLIMTE. This might indicate that a program or utility is not capable of handling extended Generation Data Groups (GDGs). *jobname* is the name of the user application job, *stepname*, and *procname* are also listed if they are not blanks.

An example of tracked usage reported by the catalog for using GDGLIMIT (and not also using GDGLIMTE) could appear as the following:

```
----------------------------------------------------------------------
INSTANCE:      1                   COUNT:         1
 EVENTDESC:     'GDGLIMIT CSIRXJCL STEP1            '
 OWNER:         IBMDFSMS            SOURCE:        IGG0CLFA
 EVENTDATA:     x0000000000000000   x0000000000000000
 PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
 HOMEJOB:       CATALOG             HOMEASID:      x002F
 EVENTJOB:      CATALOG             EVENTASID:     x002F
 AUTHORIZED:    YES                 FIRST TIME:    2014-10-29 09:06:46

----------------------------------------------------------------------
```

Figure 104. Example of tracked usage reported by the catalog for using GDGLIMIT (and not also using GDGLIMTE)

- Event description starting with **SMS-**
  - See the “EAV migration assistance tracker” section in *z/OS DFSMSdfp Advanced Services*.

JES3 control statement tracking

JES3 is instrumented to report the use of JES3 control statements (JECL) in jobs that have been submitted to the system. Occurrences of JES3 JECL statements in a job is reported during input service using the Generic Tracker macro, GTZTRACK. When GTZ tracking is enabled, JES3 records GTZ data identifying the JES3 JECL statements found within a job stream.

JES3 generated GTZ records are identified with the OWNER field set to IBMJES3. Information in the JES3 record aids in identifying the source of the job stream and the JES3 control statements found in the job stream.

```
----------------------------------------------------------------------
INSTANCE:      1                   COUNT:         1
EVENTDESC:     '|01110000 000xxxxx| INTRDR   CRJOB002 IBMUSER '
OWNER:         IBMJES3             SOURCE:        IATISLG
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
HOMEJOB:       JES3                HOMEASID:      x001E
EVENTJOB:      JES3                EVENTASID:     x001E
AUTHORIZED:    YES                 FIRST TIME:    2014-04-11 08:30:23
----------------------------------------------------------------------
INSTANCE:      2                   COUNT:         1
EVENTDESC:     '|00000001 000xxxxx| RDR011   CRJOB001 RDR011 '
OWNER:         IBMJES3             SOURCE:        IATISRI
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
HOMEJOB:       JES3                HOMEASID:      x001E
EVENTJOB:      JES3                EVENTASID:     x001E
AUTHORIZED:    YES                 FIRST TIME:    2014-04-11 08:30:52
----------------------------------------------------------------------
INSTANCE:      3                   COUNT:         1
EVENTDESC:     '|00110010 000xxxxx| RDR011   CRJOB001 '
OWNER:         IBMJES3             SOURCE:        IATISLG
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
HOMEJOB:       JES3                HOMEASID:      x001E
EVENTJOB:      JES3                EVENTASID:     x001E
AUTHORIZED:    YES                 FIRST TIME:    2014-04-11 08:30:53
----------------------------------------------------------------------
INSTANCE:      4                   COUNT:         1
EVENTDESC:     '|00110010 000xxxxx| RDR011   CRJOB0X1 '
OWNER:         IBMJES3             SOURCE:        IATISLG
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
HOMEJOB:       JES3                HOMEASID:      x001E
EVENTJOB:      JES3                EVENTASID:     x001E
AUTHORIZED:    YES                 FIRST TIME:    2014-04-11 08:30:53
----------------------------------------------------------------------
```

Figure 105. Examples of JES3 control statement GTZ records

Description of the GTZ record fields with data provided by JES3:

- OWNER is the string IBMJES3 and identifies the JES3 subsystem as the source of the GTZ record.
- SOURCE identifies the JES3 module which identified the occurrence of a JES3 control statement in the job stream. This is either IATISLG or IATISRI.
- EVENTDATA is set to zeros.
- PROGRAM is *UNKNOWN and PROGRAMOFFSET is zeros as JES3 provides no program specific information.
- EVENTDESC is a 46 character string in which JES3 provides information about the job stream and the JES3 control statement usage within the job stream. The contents of EVENTDESC by character position is:

  1 = A starting delimiter, which is the vertical bar character, for the
  JES3 control statement usage indicators.
  2-18 = Each character identifies whether a specific JES3 control
  statement is used in the job stream. 0=Not used. 1=Used.
  2 = JES3 command statement
  3 = //*DATASET statement
  4 = //*FORMAT statement
  5 = //*MAIN statement
  6 = //*NET statement
  7 = //*NETACCT statement
  8 = //*OPERATOR statement
  9 = //*PAUSE statement
  10 = Blank character
  11 = //*PROCESS statement
  12 = //*ROUTE statement
  13 = //*SIGNON statement
  14-18 = Reserved
  19 = An ending delimiter which is the vertical bar character.
  20 = Blank character
  21-28 = JES3 point of entry for the job stream.
  29 = Blank character
  30-37 = Job name when SOURCE=IATISLG.
  DDNAME of the reader when SOURCE=IATISRI.
  38 = Blank character
  39-46 = Submitting TSO user ID when SOURCE=IATISLG.
  DDNAME of the reader when SOURCE=IATISRI.

Data for the remaining GTZ record fields is supplied by the generic tracker.

JES2 control statement tracking

JES2 is instrumented to report the use of JES2 and JES3 control statements (JECL) in jobs that have been submitted to the system. Note that any JES3 JECL that is tracked by JES2 still came from a JES2 input stream, which is possible now with JES2 supporting many JES3 JECL statements.

Occurrences of JES2 and JES3 JECL statements in a job is reported during input service using the Generic Tracker service GTZTRACK. When GTZ tracking is enabled, JES2 records GTZ data identifying the JES2 and JES3 JECL statements found within a job stream.

JES2 generated GTZ records are identified with the OWNER field set to IBMJES2. Information in the JES2 record aids in identifying the source of the job stream and the JES2 and JES3 control statements found in the job stream.

```
INSTANCE:      1                    COUNT:         2
EVENTDESC:     '|00000000 0|00010000 0| INTRDR     TESTJOB  USER0S1'+
                 '  JES2       '
OWNER:         IBMJES2              SOURCE:        HASCINJR
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN             PROGRAMOFFSET: x0000000000000000
HOMEJOB:       USER0S1              HOMEASID:      x008E
EVENTJOB:      USER0S1              EVENTASID:     x008E
AUTHORIZED:    YES                  FIRST TIME:    2018-11-13 12:46:49
```

Figure 106. Example of a JES2 control statement GTZ record

Description of the GTZ record fields with data provided by JES2:

- OWNER is the string IBMJES2 and identifies the JES2 subsystem as the source of the GTZ record.
- SOURCE identifies the JES2 module which identified the occurrence of a JES2 or JES3 control statement in the job stream. This is either HASCINJR, HASCNJJR, or HASPRDR.
- EVENTDATA is set to zeros.
- PROGRAM is *UNKNOWN and PROGRAMOFFSET is zeros as JES2 provides no program specific information.
- EVENTDESC is a 46 character string in which JES2 provides information about the job stream and the JES2 and JES3 control statement usage within the job stream. The contents of EVENTDESC by character position is:

1 = A starting delimiter, which is the vertical bar character, for the
JES2 and JES3 control statement usage indicators.
2-22 = Each character identifies whether or not a specific JES2 or JES3 control
statement is used in the job stream. 0=Not used. 1=Used.
2 = //*DATASET statement (JES3)
3 = //*FORMAT statement (JES3)
4 = //*MAIN statement (JES3)
5 = //*NET statement (JES3)
6 = //*NETACCT statement (JES3)
7 = //*OPERATOR statement (JES3)
8 = //*PAUSE statement (JES3)
9 = //*PROCESS statement (JES3)
10 = Blank character
11 = //*ROUTE statement (JES3)
12 = Delimiter, vertical bar character
13 = /*JOBPARM statement (JES2)
14 = /*MESSAGE statement (JES2)
15 = /*OUTPUT statement (JES2)
16 = /*ROUTE statement (JES2)
17 = /*SETUP statement (JES2)
18 = /*XEQ statement (JES2)
19 = /*NETACCT statement (JES2)
20 = /*NOTIFY statement (JES2)
21 = Blank character
22 = /*XMIT statement (JES2)
23 = Delimiter, vertical bar character
24 = Blank character
25-34 = JES2 device name for point of entry for the job stream
35 = Blank character
36-43 = Job name
44 = Blank character
45-52 = Submitting TSO userid when SOURCE=HASCINJR

53 = Blank character
54-57 = JES2 susbsystem name

Data for the remaining GTZ record fields is supplied by the generic tracker.

If you do not want to have JES2 generated tracking records included when GTZ tracking is enabled, consider adding the following type of GTZ EXCLUDE statements to your GTZPRMxx parmlib members:

```
EXCLUDE(EVENTDESC='|??????????!??????????|*'
     OWNER=IBMJES2 SOURCETYPE=NOPATH SOURCE=HASCINJR)
EXCLUDE(EVENTDESC='|??????????!??????????|*'
     OWNER=IBMJES2 SOURCETYPE=NOPATH SOURCE=HASCNJJR)
EXCLUDE(EVENTDESC='|??????????!??????????|*'
     OWNER=IBMJES2 SOURCETYPE=NOPATH SOURCE=HASPRDR)
```

JES2 PUT-UPDATE tracking

JES2 uses GTZTRACK to report instances of jobs performing PUT-UPDATEs to SPOOL data sets. When GTZ tracking is enabled, JES2 records data that identifies the step name and program name, as specified on the PGM= keyword, when the PUT-UPDATE is performed.

GTZ records that are generated by JES2 are identified with the OWNER field set to IBMJES2. Information in the JES2 record aids in identifying the source of the PUT-UPDATE. This track is useful on systems that use SPOOL encryption in a future release, where data sets updated using PUT-UPDATE are not eligible for encryption/compression.

Descriptions of the GTZ record fields with data that is provided by JES2 are:

- OWNER is set to the string IBMJES2 and identifies the JES2 subsystem as the source of the GTZ record.
- SOURCE identifies the JES2 module that identified the occurrence of a JES2 PUT-UPDATE use. This is set to the string HASCPHAM.
- EVENTDATA is set to zeros.
- PROGRAM is set to the string *OMITTED and PROGRAMOFFSET is set to zeros.
- EVENTDESC is a 64 character string that JES2 provides information about the job step and the Put Update usage within the job step. The contents of EVENTDESC are organized using a header that is ended by a colon, followed by the supplied value. A comma followed by a space separates these fields.

  The information and headers that are provided are:

  'DDNAME:'
  The name of the data set the PUT-UPDATE is performed against.

  'STEP:'
  The job step active at the time of the PUT-UPDATE.

  'PROGRAM:'
  The name of the program that performs the PUT-UPDATE.

  'SUBSYSTEM:'
  The name of the subsystem where the job executes.

Data for the remaining GTZ record fields is supplied by the Generic Tracker Facility.

MVS Allocation tracking

Event description 'IEFALC 01: <step name> <DD name>'

- See message IEF384I “WARNING: VOLUME NOT RETRIEVED FROM CATALOG”.

  Note that the ALLOCxx parmlib member option SYSTEM VERIFY_UNCAT(FAIL|TRACK|MSGTRACK|LOGTRACK) determines whether message IEF348I is issued and where:

  – Option SYSTEM VERIFY_UNCAT(TRACK) enables tracking, but does not issue message IEF348I.

  – Option SYSTEM VERIFY_UNCAT(MSGTRACK) specifies that message IEF348I is issued only to the job log.
  – Option SYSTEM VERIFY_UNCAT(LOGTRACK) specifies that message IEF348I is issued to both the job log and hardcopy-only WTO.

  The default for SYSTEM VERIFY_UNCAT is FAIL.

```
D GTZ,TRACKDATA
 GTZ1002I  23.42.10 GTZ TRACKDATA 239
 FOUND 1 MATCHING TRACKED INSTANCE(S)
 ----------------------------------------------------------------------
  INSTANCE:      1                   COUNT:         1
  EVENTDESC:     'IEFALC 01: STEP1    DD1'
  OWNER:         IBMCNZ              SOURCE:        CNZTRKR
  EVENTDATA:     x0000000000000000   x0000000000000000
  PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
  HOMEJOB:       BEANSZZ             HOMEASID:      x001D
  EVENTJOB:      BEANSZZ             EVENTASID:     x001D
  AUTHORIZED:    YES                 FIRST TIME:    2013-09-28 23:35:31
```

Figure 107. Example of MVS allocation tracking

SDSF tracking

- Event description: SDSF NOPARM FALLBACK: ISFPRMXX NOT ACTIVE
- Event description: SDSF MENU TABLE DISABLED: ISFMIGMN ALLOCATED

  For these events:

  – OWNER is IBMSDSF
  – EVENTDATA is set to zeros
  – PROGRAM is the SDSF module that detected the event

  See *z/OS SDSF Operation and Customization*.

```
D GTZ,TRACKDATA
GTZ1002I  17.44.01 GTZ TRACKDATA 649
FOUND 1 MATCHING TRACKED INSTANCE(S)
----------------------------------------------------------------------
 INSTANCE:      1                   COUNT:         1
 EVENTDESC:     'SDSF NOPARM FALLBACK: ISFPRMXX NOT ACTIVE'
 OWNER:         IBMSDSF             SOURCE:        ISFINITC
 EVENTDATA:     x0000000000000000   x0000000000000000
 PROGRAM:       ISFVTBL             PROGRAMOFFSET: x000000000020C15E
 HOMEJOB:       KIMURA              HOMEASID:      x0067
 EVENTJOB:      KIMURA              EVENTASID:     x0067
 AUTHORIZED:    YES                 FIRST TIME:    2017-10-08 17:43:21

D GTZ,TRACKDATA
GTZ1002I  00.39.55 GTZ TRACKDATA 863
FOUND 1 MATCHING TRACKED INSTANCE(S)
----------------------------------------------------------------------
 INSTANCE:      1                   COUNT:         1
 EVENTDESC:     'SDSF MENU TABLE DISABLED: ISFMIGMN ALLOCATED'
 OWNER:         IBMSDSF             SOURCE:        ISFINITC
 EVENTDATA:     x0000000000000000   x0000000000000000
 PROGRAM:       ISFVTBL             PROGRAMOFFSET: x000000000020C15E
 HOMEJOB:       KIMURA              HOMEASID:      x003A
 EVENTJOB:      KIMURA              EVENTASID:     x003A
 AUTHORIZED:    YES                 FIRST TIME:    2017-09-20 00:39:43
```

Figure 108. Examples of SDSF tracking

TSO/E tracking

Event description: `'MVSSERV CALL'`

The MVSSERV command was executed to invoke the Enhanced Connectivity Facility function.

An example of tracked usage reported by TSO/E could appear as the following:

```
INSTANCE:  1                 COUNT:         1
EVENTDESC: 'MVSSERV CALL'
OWNER:     IBMTSO            SOURCE:        MVSSERV
EVENTDATA: x0000000000000000 x0000000000000000
PROGRAM:   CHSTCPS           PROGRAMOFFSET: x0000000000000000
HOMEJOB:   IBMUSER           HOMEASID:      x0022
EVENTJOB:  IBMUSER           EVENTASID:     x0022
AUTHORIZED:NO                FIRST TIME:    2016-06-09 06:13:02
```

Figure 109. Example of tracked usage by TSO/E

VSM tracking

Event description: `'JOB REQUESTED V=R REGION (ADDRSPC=REAL)'`

An event with event description `'JOB REQUESTED V=R REGION (ADDRSPC=REAL)'` indicates that a job ran with V=R (ADDRSPC=REAL) settings. See Verify that the new default value for REAL is acceptable in *z/OS Upgrade Workflow* for more information.

For this event:

- OWNER is always IBMVSM
- SOURCE is always IGVGRRGN
- EVENTDATA is always zero
- PROGRAM and PROGRAMOFFSET are not relevant and undefined
- The JOB and ASID fields identify an actual job that used ADDRSPC=REAL.

An example of a tracked event reported by VSM would look like the following:

```
INSTANCE:      4                   COUNT:         1
EVENTDESC:     'JOB REQUESTED V=R REGION (ADDRSPC=REAL)'
OWNER:         IBMVSM              SOURCE:        IGVGRRGN
EVENTDATA:     x0000000000000000   x0000000000000000
PROGRAM:       *UNKNOWN            PROGRAMOFFSET: x0000000000000000
HOMEJOB:       REAL0JB             HOMEASID:      x0039
EVENTJOB:      REAL0JB             EVENTASID:     x0039
AUTHORIZED:    YES                 FIRST TIME:    2017-03-02 12:29:34
```

Figure 110. Example of an event tracked by VSM

Chapter 12. Component trace

The component trace service provides a way for MVS components to collect problem data about events. Each component that uses the component trace service has set up its trace in a way that provides the unique data needed for the component. A component trace provides data about events that occur in the component. The trace data is used by the IBM Support Center, which uses the trace data to:

- Diagnose problems in the component
- See how the component is running.

You will typically use component trace while recreating a problem.

Usage of system resources: Some component traces use minimal system resources, especially while tracing a small number of events. These minimal traces often run anytime the component is running. Other traces use significant system resources, especially when many kinds of events are traced. These large traces are usually requested only when the IBM Support Center asks for them.

Run concurrent traces: You can run more than one component trace at a time; you can run component traces:

- Concurrently for several components on one system.
- Concurrently for one or more components on some or all of the systems in a sysplex.
- Concurrently for one component on a system. Multiple concurrent traces for a component are ***sublevel traces***.
- Concurrently for several components on some or all of the systems in a sysplex and with sublevel traces.

The following topics describe tasks for component traces:

- "Planning for component tracing" on page 350 tells you about the tasks needed to plan component tracing.
- "Obtaining a component trace" on page 358 tells you how to request a specific component trace that is needed to diagnose a problem. The tasks depend on where you plan to put trace output and if you are running traces on multiple systems in a sysplex; therefore, requesting traces is presented in three topics:
  - "Request component tracing to address space or data space trace buffers" on page 358
  - "Request writing component trace data to trace data sets" on page 361
  - "Request component tracing for systems in a sysplex" on page 365
- "Verifying component tracing" on page 367 tells how an operator can check that a requested trace is running and that a component trace writer is running.
- "Viewing the component trace data" on page 369 tells you how to format the trace output.

This topic uses tables to show the similarities and differences in the individual traces from different components. Table 56 on page 349 summarizes each BCP component trace that uses the component trace service.

Table 56. Summary of BCP component traces that use the component trace service

Component	Reference
Advanced Program-to-Program Communication/MVS (APPC/MVS)	"SYSAPPC component trace" on page 371
Base control program internal interface (BCPii)	"SYSBCPII component trace" on page 387
Basic HyperSwap® socket support	"SYSBHI component trace" on page 390
Common Event Adapter (CEA)	"SYSCEA component trace" on page 394

Table 56. Summary of BCP component traces that use the component trace service (continued)	
Component	**Reference**
Cross-system coupling facility (XCF)	"SYSXCF component trace" on page 494
Cross-system extended services (XES)	"SYSXES component trace" on page 498
Data lookaside facility (DLF) of VLF	"SYSDLF component trace" on page 397
Distributed function of SOMobjects	"SYSDSOM component trace" on page 399
Global resource serialization	"SYSGRS component trace" on page 407
Allocation Component	"SYSIEFAL component trace" on page 417
IOS Component Trace	"SYSIOS component trace" on page 422
JES common coupling services	"SYSJES component trace" on page 428
JES2 rolling trace table	"SYSjes2 component trace" on page 435
Library lookaside (LLA) of contents supervision	"SYSLLA component trace" on page 443
z/OS UNIX System Services (z/OS UNIX)	"SYSOMVS component trace" on page 450
Operations services (OPS)	"Requesting a SYSOPS trace" on page 461
Resource recovery services (RRS)	"SYSRRS component trace" on page 465
Real storage manager (RSM)	"SYSRSM component trace" on page 471
SDUMP	"SYSDUMP component trace" on page 401
Service processor interface (SPI)	"SYSSPI component trace" on page 486
System logger	"SYSLOGR component trace" on page 443
System REXX	"SYSAXR component trace" on page 383
Transaction trace (TTRC)	"SYSTTRC transaction trace" on page 488
Virtual lookaside facility (VLF)	"SYSVLF component trace" on page 488
Workload management (WLM)	"SYSWLM component trace" on page 491

A program product or application, if authorized, can also use the component trace service to provide an ***application trace***. See the documentation for the program product or application for information about its trace.

- See *z/OS MVS Initialization and Tuning Reference* for the CTncccxx parmlib member.
- See *z/OS MVS System Commands* for the TRACE CT command.
- See *z/OS MVS IPCS Commands* for the COPYDUMP, COPYTRC, CTRACE, GTFTRACE, and MERGE subcommands.
- For a description of these messages, see *MVS System Messages*.
- See *z/OS MVS Programming: Authorized Assembler Services Guide* for information on creating an ***application trace***.

Planning for component tracing

Planning for component tracing consists of the following tasks, which are performed by the system programmer:

- "Create CTncccxx parmlib members for some components" on page 351:
  - "Specify buffers" on page 353
- "Select the trace options for the component trace" on page 356
- "Decide where to collect the trace records" on page 357

Create CTncccxx parmlib members for some components

Table 57 on page 351 shows if a component has a parmlib member, if the member is a default member needed at system or component initialization, and if the component has default tracing. Some components run default tracing at all times when the component is running; default tracing is usually minimal and covers only unexpected events. Other components run traces only when requested.

When preparing your production SYS1.PARMLIB system library, do the following:

1. Make sure the parmlib contains all default members identified in the table. If parmlib does not contain the default members at initialization, the system issues messages.

   Make sure that the IBM-supplied CTIITT00 member is in the parmlib. PARM=CTIITT00 can be specified on a TRACE CT command for a component trace that does not have a parmlib member; CTIITT00 prevents the system from prompting for a REPLY after the TRACE CT command. In a sysplex, CTIITT00 is useful to prevent each system from requesting a reply.
2. Decide if each default member meets the needs of your installation. If it does not, customize it.
3. Decide if the buffer size specified in the default members meets the needs of your installation. Some component traces do not allow buffer size change after initialization. Change the buffer size, if needed.

Most components can run only one component trace at a time; some components can run concurrent traces, called ***sublevel traces***. Each sublevel trace is identified by its sublevel trace name. For some components, you need to identify the component's CTncccxx member in another parmlib member; the components with this requirement have the other parmlib member listed in the default member column in Table 57 on page 351.

For example, for XCF specify CTIXCF00 on the CTRACE parameter in the COUPLExx parmlib member.

```
COUPLE SYSPLEX( ...
     CTRACE(CTIXCF00)
     ...
```

Table 57. Determining if a component has a parmlib member

Trace	Parmlib member	Default member	Default tracing beginning at initialization	Sublevel traces
SYSAPPC	CTnAPPxx (see "CTnAPPxx parmlib member" on page 371)	No	No; cannot turn trace ON or OFF in CTnAPPxx	No
SYSAXR	CTIAXRnn (see "CTIAXRnn parmlib member" on page 384).	CTIAXR00	Yes	No
SYSBCPII	CTIHWI00	CTIHWI00	Yes; minimal diagnostic tracing is always in effect. The presence of a valid CTIHW100 parmlib at BCPii startup can modify these default trace options.	No
SYSBHI	CTIBHIxx	CTIBHI00	Yes; minimal; unexpected events	No
SYSCEA	CTICEAnn (see "CTICEAnn parmlib member" on page 395)	CTICEA00	Yes	No
SYSDLF	None	N/A	Yes; always on when DLF is running	No
SYSDSOM	None	N/A	No	Yes
SYSDUMP	CTIDMPxx	CTIDMP00	Yes, full tracing	No

Table 57. Determining if a component has a parmlib member (continued)

Trace	Parmlib member	Default member	Default tracing beginning at initialization	Sublevel traces
SYSGRS	CTnGRSxx (see "CTnGRSxx parmlib member" on page 407).	CTIGRS00, which is specified in GRSCNF00 member	Yes, if global resource serialization is active; CONTROL and MONITOR options	No
SYSIEFAL	CTIIEFxx (see "CTIIEFxx parmlib member" on page 418)	CTIIEFAL	Yes	No
SYSIOS	CTnIOSxx (see "CTnIOSxx parmlib member" on page 424)	No	Yes; minimal; unexpected events	No
SYSJES	CTnJESxx (see "CTnJESxx parmlib member" on page 429)	CTIJES01, CTIJES02, CTIJES03, CTIJES04 You should also receive and rename members IXZCTION and IXZCTIOF supplied in SYS1.SAMPLIB to CTIJESON and CTIJESOF.	Yes; full tracing for sublevels XCFEVT and FLOW; minimal tracing of unexpected events for sublevels USRXIT and MSGTRC	Yes
SYS*jes2*	None	N/A	Yes; always on when JES2 is running	Yes
SYSLLA	None	N/A	Yes; always on when LLA is running	No
SYSLOGR	CTnLOGxx (see "CTnLOGxx parmlib member" on page 446).	CTILOG00, which can be specified in IXGCNFxx member.	Yes; activated during system logger (IXGLOGR) address space initialization	No
SYSOMVS	CTnBPXxx (see "CTnBPXxx parmlib member" on page 451)	CTIBPX00, which must be specified in BPXPRM00 member	Yes; minimal; unexpected events	No
SYSOPS	CTnOPSxx (see "CTnOPSxx parmlib member" on page 461)	CTIOPS00, which must be specified in CONSOLxx member	Yes; minimal; unexpected events	No
SYSRRS	CTnRRSxx (see "CTnRRSxx parmlib member" on page 466)	None, but member ATRCTRRS supplied in SYS1.SAMPLIB can be used. See SET-UP and ACTIVATION instructions in the ATRCTRRS sample.	Yes; minimal; unexpected events	No
SYSRSM	CTnRSMxx (see "CTnRSMxx parmlib member" on page 472)	No	No	No
SYSSPI	None	N/A	No	No
SYSTTRC	N/A	N/A	No	No
SYSVLF	None	N/A	Yes; minimal; unexpected events	No
SYSWLM	None	N/A	Yes; minimal; unexpected events	No
SYSXCF	CTnXCFxx (see "CTnXCFxx parmlib member" on page 494)	CTIXCF00, which can be specified in COUPLE00 member	Yes; minimal; unexpected events	No

Table 57. Determining if a component has a parmlib member (continued)				
Trace	**Parmlib member**	**Default member**	**Default tracing beginning at initialization**	**Sublevel traces**
SYSXES	CTnXESxx (see "CTnXESxx parmlib member" on page 500)	CTIXES00, which can be specified in COUPLE00 member	Yes; minimal; unexpected events	Yes

Specify buffers

Each component determines the buffer size and how it is specified. Depending on the component, you may or may not be able to change the buffer size. You may be able to change the size only at system or component initialization, or when the trace is started, or at any time, including while the trace is running. Table 58 on page 353 shows how the buffers specifications.

The buffer size determines whether you get all the records needed for diagnosis; when the buffer is full, the system wraps the buffer, overwriting the oldest records. To change the size of the buffer, specify an nnnnK or nnnnM parameter on the TRACE CT operator command or a BUFSIZE parameter in the parmlib member.

Usually you should increase the size of the trace buffer when you increase the amount of tracing. However, if you plan to place a component's trace records in a trace data set, you can probably leave the buffer at its original size. Many component traces do not allow you to change the buffer size after initialization; the table indicates those component traces. If you increase the amount of tracing for one of these traces, specify use of a trace data set, if the component supports its use.

Table 58. Component trace options				
Trace	**Default size and size range**	**Size set by**	**Change size after IPL**	**Buffer location**
SYSAPPC	512 KB 64 KB - 32 MB	CTnAPPxx member or REPLY for TRACE CT command	Yes, while a trace is running	Data space. A TRACE CT,OFF command requests a dump, which includes the trace buffers
SYSAXR	2 MB 1 MB - 2 GB	CTIAXRnn parmlib member or REPLY to TRACE CT command	Yes, when restarting a trace after stopping it	AXR private; AXR trace dataspace
SYSBCPII	4 MB 4 MB	MVS system	No	Data space. A SLIP or DUMP command can always be issued to capture the trace buffers for the BCPii address space. (Specify ASID=(BCPii's ASID),DSPNAME=('HWIBCPII'.*)) In addition, if CTrace for SYSBCPII is ON, a Trace CT,OFF command requests a dump, which includes the trace buffers.
SYSBHI	4 MB 4 MB - 64 MB	CTIBHIxx parmlib member or REPLY to TRACE CT command	Yes, while trace is running	64-bit Common Service Area (ECSA)
SYSCEA	2 MB 1 MB - 2 GB	CTICEAnn parmlib member or REPLY to TRACE CT command	Yes, when restarting a trace after stopping it	CEA private; CEA trace dataspace

Table 58. Component trace options (continued)

Trace	Default size and size range	Size set by	Change size after IPL	Buffer location
SYSDLF	N/A	MVS system	No	Data space. In the REPLY for the DUMP command, specify DSPNAME= ('DLF'.CCOFGSDO)
SYSDSOM	N/A	MVS system	No	Private address space
SYSDUMP	4 MB 4 MB - 32 MB	CTIDMPxx parmlib member or TRACE CT command	Yes	64-bit common storage area
SYSGRS	16 MB 128 KB - 2047 MB (System rounds size up to nearest 64 KB boundary.)	CTnGRSxx member	Yes.	In the GRS address space above the bar which means it will not constrain GRS virtual storage. Options such as FLOW, REQUEST, and MONITOR produce a large number of entries in a short period of time. When dumping by SDUMP, specify ASID=GRS's asid and SDATA=(RGN,NUC). The RGN is needed for blocks that address the ctrace buffer. Note that SDATA=GRSQ does not collect GRS CTRACE.
SYSIEFAL	4 MB 256 KB - 8 MB	CTIIEFxx member	Yes.	In the component address space
SYSIOS	324 KB 324 KB - 1.5 MB	CTnIOSxx member or REPLY for TRACE CT command	Yes	Extended system queue area (ESQA). **Note:** Full buffers are copied to an IOS data space to allow for more data capture. For information about specifying an IOS data space size at IPL, see "OPTIONS parameter" on page 425.
SYSJES	N/A	MVS system	No	In the component address space
SYS*jes2*	N/A	JES2	No	In the component address space
SYSLLA	N/A	MVS system	No	In the component address space

Table 58. Component trace options (continued)				
Trace	**Default size and size range**	**Size set by**	**Change size after IPL**	**Buffer location**
SYSLOGR	2 MB 2 MB - 2047 MB	MVS system, CTnLOGxx member, or REPLY for TRACE CT command.	Yes	Data space. In the REPLY for the DUMP command, specify DSPNAME=('IXLOGR'.*). See "Obtaining a dump of system logger information" on page 444.
SYSOMVS	4 MB 16 KB - 64 MB	CTxBPXxx member or REPLY for TRACE CT command	Yes, when initializing z/OS UNIX.	Data space. In the REPLY for the DUMP command, specify DSPNAME=(asid.SYSZBPX2) where asid is the ASID for z/OS UNIX.
SYSOPS	2 MB 64 KB - 16 MB	CTnOPSxx member or REPLY for TRACE CT command	Yes, when restarting a trace after stopping it	Console address space (private).
SYSRRS	1 MB 1 MB - 2045 MB	CTxRRSxx member or REPLY for TRACE CT command	Yes, when restarting a trace after stopping it	Data space and component address space. In the REPLY for the DUMP command, specify DSPNAME=('RRS' .ATR TRACE) and SDATA=RGN.
SYSRSM	3 buffers of 132 pages 2 - 7 page-fixed primary buffers, 4 - 262,144 pages per buffer 1 - 2047 MB for secondary buffers	CTnRSMxx member or REPLY for TRACE CT command	Yes, when starting a trace	Common service area (LIKECSA) or, if specified in CTnRSMxx, high virtual private storage of the RASP address space.
SYSSPI	64KB	MVS system	Yes, when starting a trace	In the component address space
SYSTTRC	1 MB 16K - 999K 1MB - 32MB	MVS system	Yes	Data space owned by the system trace address space
SYSVLF	N/A	MVS system	No	Data space. Enter DISPLAY J,VLF to identify the VLF data spaces. In the REPLY for the DUMP command, specify DSPNAME=('VLF'.Dclsname, 'VLF'.Cclsname), where *clsname* is a VLF class name.

Table 58. Component trace options (continued)				
Trace	**Default size and size range**	**Size set by**	**Change size after IPL**	**Buffer location**
SYSWLM	64KB 64KB - 16M	MVS system	Yes, when starting a trace	Extended common service area (ECSA)
SYSXCF	4MB 16KB - 16MB (System rounds size up to a multiple of 72 bytes.)	CTnXCFxx member	No	Extended local system queue area (ELSQA) of the XCF address space
SYSXES	336KB 16KB - 16MB	CTnXESxx member or REPLY for TRACE CT command	Yes, while a trace is running.	64-bit common storage above the 2 GB bar. In the REPLY for the DUMP command, specify SDATA=XESDATA.

Select the trace options for the component trace

If the IBM Support Center requests a trace, the Center might specify the options, if the component trace uses an OPTIONS parameter in its parmlib member or REPLY for the TRACE CT command. The options are:

Table 59. Trace request options for component traces	
Trace	**Trace request OPTIONS parameter**
SYSAPPC	See "OPTIONS parameter" on page 372.
SYSAXR	See "OPTIONS parameter" on page 385.
SYSBCPII	See "OPTIONS parameter" on page 388.
SYSBHI	See "OPTIONS parameter" on page 392.
SYSCEA	See "OPTIONS parameter" on page 396.
SYSDLF	None.
SYSDSOM	None.
SYSDUMP	See "OPTIONS parameter" on page 396.
SYSGRS	See "OPTIONS parameter" on page 408.
SYSIEFAL	See "OPTIONS parameter" on page 419.
SYSIOS	See "OPTIONS parameter" on page 425.
SYSJES	None.
SYS*jes2*	None.
SYSLLA	None.
SYSLOGR	See "OPTIONS parameter" on page 448.
SYSOMVS	See "OPTIONS parameter" on page 452.
SYSOPS	See "OPTIONS parameter" on page 462.
SYSRRS	See "OPTIONS parameter" on page 467.

Table 59. Trace request options for component traces (continued)	
Trace	**Trace request OPTIONS parameter**
SYSRSM	See "OPTIONS parameter" on page 474.
SYSTTRC	None.
SYSSPI	None.
SYSVLF	None.
SYSWLM	None.
SYSXCF	See "OPTIONS parameter" on page 495.
SYSXES	See "OPTIONS parameter" on page 501.

You must specify all options you would like to have in effect when you start a trace. Options specified for a previous trace of the same component do not continue to be in effect when the trace is started again.

If the component has default tracing started at initialization by a parmlib member without an OPTIONS parameter, you can return to the default by doing one of the following:

- Stopping the tracing with a TRACE CT,OFF command.
- Specifying OPTIONS() in the REPLY for the TRACE CT command or in the CTncccxx member.

For XCF, the IBM Support Center identifies the options needed to diagnose a particular problem as both of the following:

```
SERIAL
STATUS
```

Decide where to collect the trace records

As Table 60 on page 357 shows, depending on the component, the potential locations of the trace data are:

- In address-space buffers, which are obtained in a dump
- In data-space buffers, which are obtained in a dump
- In a trace data set or sets, if supported by the component trace

Table 60. Location of trace buffers for components			
Component	**Address-space Buffer**	**Data-space Buffer**	**Trace data set**
SYSAPPC	No	Yes	No
SYSAXR	Yes	Yes	Yes
SYSBCPII	No	Yes	No
SYSBHI	Yes	No	Yes
SYSCEA	Yes	Yes	Yes
SYSDLF	Yes	Yes	No
SYSDSOM	Yes	No	Yes
SYSDUMP	Yes	No	Yes
SYSGRS	Yes	No	Yes
SYSIEFAL	Yes	No	Yes

Table 60. Location of trace buffers for components (continued)			
Component	**Address-space Buffer**	**Data-space Buffer**	**Trace data set**
SYSIOS	Yes	Yes	Yes
SYSJES	Yes	No	Yes
SYS*jes2*	Yes	No	No
SYSLLA	Yes	No	No
SYSLOGR	Yes	Yes	Yes
SYSOMV	No	Yes	Yes
SYSOPS	Yes	No	Yes
SYSRRS	Yes	Yes	Yes
SYSRSM	Yes	Yes	Yes
SYSTTRC	No	Yes	No
SYSSPI	Yes	No	No
SYSVLF	Yes	Yes	No
SYSWLM	Yes	No	Yes
SYSXCF	Yes	No	Yes
SYSXES	No	Yes	Yes

If the trace records of the trace you want to run can be placed in more than one location, you need to select the location. For a component that supports trace data sets, you should choose trace data sets for the following reasons:

- Because you expect a large number of trace records
- To avoid interrupting processing with a dump of the trace data
- To keep the buffer size from limiting the amount of trace data
- To avoid increasing the buffer size

Depending on the component, you might also want to dump the address-space trace buffers and data-space trace buffers.

Note: You may need to consider the amount of auxiliary storage required to back data space buffers. In general, most components which use data space buffers establish a small default value less than 500 kilobytes of virtual storage. Some components allow you to specify values up to 2 gigabytes. The SYSIOS component trace uses a default of 512 megabytes for data space buffers. You should consider SYSIOS and other component data space buffers to ensure that the potential cumulative effect of all CTRACE data space buffers for your system can be accommodated by the local page data sets that you have allocated. For more information on auxiliary storage, refer to *z/OS MVS Initialization and Tuning Guide*.

Obtaining a component trace

To obtain a specific component trace, use one of the following procedures:

- "Request component tracing to address space or data space trace buffers" on page 358
- "Request writing component trace data to trace data sets" on page 361
- "Request component tracing for systems in a sysplex" on page 365

Request component tracing to address space or data space trace buffers

This topic describes how to obtain component trace records in dumps. The trace records are in address-space or data-space trace buffers. The topic contains information about how to:

- “Prepare for a specific component trace to trace buffers” on page 359
- “Perform component tracing to trace buffers” on page 360.

Prepare for a specific component trace to trace buffers

The system programmer performs the tasks.

1. Select How the Operator Is to Request the Trace: For most component traces, the request is made by:

- A TRACE CT operator command without a PARM parameter, followed by a reply containing the options
- A TRACE CT operator command with a PARM parameter that specifies a CTncccxx parmlib member containing the options

If you do not use a parmlib member, tell the operator the options.

2. Create a parmlib member, if used: If you use a parmlib member, create the member and place it on SYS1.PARMLIB. Use a parmlib member if the options are complicated and you have access to the SYS1.PARMLIB data set, or if a parmlib member is required by the component, or if you had already set up a parmlib member with the needed options. Use a REPLY for simple options. See “Create CTncccxx parmlib members for some components” on page 351. For XCF, for example, you can create CTWXCF03 to specify the options.

```
TRACEOPTS
   ON
   OPTIONS('SERIAL','STATUS')
```

3. Determine the dump to be used to obtain the trace records: Table 61 on page 359 shows how to request SVC dumps for the component traces. Possible ways of requesting SVC dumps are:

- By a DUMP operator command
- By a SLIP trap
- By the component

For the following failures, use another type of dump:

- Failure of an application program or program product: The program requests a SYSMDUMP dump.
- The system waits, hangs, or enters a loop: The operator requests a stand-alone dump.

Table 61. How to request SVC dumps for component traces

Trace	Request of SVC dump
SYSAPPC	By the component when the operator stops SYSAPPC tracing with a TRACE CT,OFF command
SYSAXR	By DUMP or SLIP command
SYSBCPII	By DUMP or SLIP command, or by the component, if Ctrace is ON for SYSBCPII and a TRACE CT,OFF command is issued.
SYSBHI	By DUMP or SLIP command if the Basic HyperSwap Management address space or one of the BHIHSRV address spaces are to be included in the dump.
SYSCEA	By DUMP or SLIP command
SYSDLF	By DUMP or SLIP command
SYSDSOM	By DUMP or SLIP command
SYSGRS	By DUMP or SLIP command
SYSIEFAL	By DUMP or SLIP command
SYSIOS	By DUMP or SLIP command, or by the component • In the REPLY for the DUMP command, specify the IOS address space to be dumped

Table 61. How to request SVC dumps for component traces (continued)	
Trace	**Request of SVC dump**
SYSJES	By the component
SYS*jes2*	By DUMP or SLIP command or component
SYSLLA	By the component
SYSLOGR	By DUMP or SLIP command
SYSOMVS	By DUMP or SLIP command
SYSOPS	By DUMP or SLIP command
SYSRRS	By DUMP or SLIP command
SYSRSM	• By DUMP or SLIP command • Through the DMPREC option on the CTnRSMxx parmlib member or on the REPLY for the TRACE CT command when RSM enters recovery processing (default) • Through the DMPOFF option of CTnRSMxx or the TRACE CT reply when SYSRSM tracing is turned off
SYSTTRC	Automatically dumped by the Tailored SVC Dump Exits function
SYSSPI	By the component
SYSVLF	By DUMP or SLIP command or when SYSVLF full tracing is turned off
SYSWLM	By DUMP or SLIP command
SYSXCF	By DUMP or SLIP command
SYSXES	By DUMP or SLIP command

4. Make Sure the component trace Buffers Will Be Dumped: The location of the address-space and data-space buffers depends on the component being traced. See the table in "Specify buffers" on page 353 for the location of the buffers. When the component being traced requests an SVC dump, the dump will contain the address-space and/or data-space trace buffers.

Perform component tracing to trace buffers

The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. Start the component trace: The operator enters a TRACE operator command on the console with MVS master authority. The operator replies with the options that you specified.

In the following example, the TRACE CT command does not specify a parmlib member.

```
trace ct,on,comp=sysxcf
* 21 ITT006A ....
r 21,options=(serial,status),end
```

This next example requests the same trace using parmlib member CTWXCF03. When TRACE CT specifies a parmlib member, the system does not issue message ITT006A.

```
trace ct,on,comp=sysxcf,parm=ctwxcf03
```

2. Verify that the Trace Is Running: See "Verifying component tracing" on page 367.

3. Obtain the Dump Containing the component trace Records: The operator obtains the dump that contains the component trace records: an SVC dump, a stand-alone dump, or a dump requested by the component when a problem occurs or when the operator stops the tracing.

This example shows a DUMP operator command entered on the console with MVS master authority. The system issues message IEE094D in response to the DUMP command. If you requested, the operator enters dump options in the reply to IEE094D. SDATA options are needed to obtain the trace buffers An address space identifier (ASID) should be specified for the XCF address space; in the example, XCF is in address space 6.

```
dump  comm=(dump for xcf component trace)
* 32 IEE094D ...
r 32,sdata=(couple,sqa,lsqa),asid=6,end
 .
 .
 .
IEA911E ...
```

The system identifies the data set containing the dump in message IEA911E. If an installation exit moves the dump, the operator should look for a message identifying the data set containing the moved dump and tell you the name of the dump and the data set containing it.

If desired, the operator can request more than one dump while a component trace is running.

4. Stop the component trace: The operator enters a TRACE CT,OFF command on the console with MVS master authority. For some component traces, the command requests a dump, which contains the trace records.

The following example shows how to specify the TRACE CT,OFF command.

```
trace ct,off,comp=sysxcf
```

Request writing component trace data to trace data sets

The following topics describe only the component traces that can write to trace data sets. You can also change the trace data sets that are in use without stopping the trace. See "Change trace data sets" on page 365.

Prepare for a specific component trace to trace data sets

The system programmer performs the following tasks:

1. Determine the dispatching priority required for the external writer started task, the server address space for the component's trace:

While component trace runs under the master scheduler address space, you need to verify that the priority of the external writer is at least equal to, and preferably greater than the priority of the component being traced. For example, if you are tracing COMP(SYSXES) for JOBNAME(IRLMA), the dispatching priority of the external writer should be equal to or greater than that assigned to IRLMA. See *z/OS MVS Initialization and Tuning Guide* for more information on setting priorities.

2. Select How the Operator Is to Request the Trace:

For most component traces, the request can be made by:

- A TRACE CT operator command without a PARM parameter, followed by a reply containing the options
- A TRACE CT operator command with a PARM parameter that specifies a CTncccxx parmlib member containing the options.

If you do not use a parmlib member, tell the operator the options.

3. Create Source JCL for the External Writer:

Create source JCL to invoke an external writer, which will send the component trace output to one or more trace data sets. Add a procedure to the SYS1.PROCLIB system library or a job as a member of the data set in the IEFJOBS or IEFPDSI concatenation.

An external writer is not specific for a component but can be used by any application. So you can use the same source JCL again for other tracing later, if needed.

Concurrent traces for different components must use separate source JCL to place their traces in separate data sets.

Because the writer source JCL specifies data sets, use a different set of source JCL for each system in a sysplex. Several systems cannot share the same data set; attempting to share the same data set will lead to contention problems. If your sysplex uses a common SYS1.PROCLIB, you need to specify a unique writer procedure for each system or use a unique job as the source JCL, when tracing the same component on several systems.

The procedure shown in Figure 111 on page 362 places trace data on two DASD data sets. The procedure is placed in member WTRD2 of SYS1.PROCLIB.

```
//WTDASD2  PROC
//IEFPROC  EXEC PGM=ITTTRCWR,REGION=32M
//TRCOUT01 DD  DSNAME=SYS1.CTRACE1,VOL=SER=TRACE6,UNIT=DASD,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD  DSNAME=SYS1.CTRACE2,VOL=SER=TRACE7,UNIT=DASD,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
```

Figure 111. Example: Cataloged Procedure for an External Writer

Rules for the Source JCL for an External Writer:

- The name specified on the TRACE CT command or CTncccxx parmlib member is the member name of the source JCL; in the preceding example, WTRD2. The name is 1 to 7 alphanumeric or national characters, with the first character alphabetic or national. National characters are represented by the following hexadecimal codes (in other languages, the codes represent different characters):

  Code
  : **US English EBCDIC Character**

  X'5B'
  : $

  X'7B'
  : #

  X'7C'
  : @

- The procedure must invoke the external writer program ITTTRCWR. Code the REGION= keyword on the EXEC statement to specify the maximum storage size required by the external writer.
- The source JCL can specify up to 16 trace data sets. The DD statements have ddnames of TRCOUTxx, where xx is 01 through 16.
- The trace data sets must be sequential data sets. You can use extended format sequential data sets as dump data sets for trace output. Extended format sequential data sets have the following features:
  - Have a greater capacity than sequential data sets.
  - Support striping
  - Support compression
- To help you manage the trace data sets, establish a naming convention so that the data set name indicates the component trace, the date, and so on.
- All of the data sets must be on DASD or tape. Do not mix device classes, such as tape and DASD.

  Within a device class, IBM recommends that you do not mix several types of devices, such as 3380 and 3390. In a mix of device types, the system would use the smallest block size for all the data sets.
- Do **not** specify the following DCB parameters:
  - BLKSIZE. The system uses the optimal block size, which is 4096 or larger.
  - RECFM. The system uses VB.
  - LRECL. The system uses BLKSIZE minus 4.
- Do **not** specify DISP=SHR.

- Do **not** concatenate trace data sets.
- Use a separate member for each component's trace, even though you *can* connect more than one trace to the same member.
- Use the same member for all the sublevel traces for a component. This approach reduces the number of data sets you must manage.
- Use a separate member for each system's component trace, when a component trace runs in two or more of the systems of a sysplex. If the component traces specify the same cataloged procedure in a shared SYS1.PROCLIB, they use the same data set or group of data sets; in this case, contention might develop for the data set or sets.
- System security may require that you have RACF SYSHIGH authority to access the trace data sets.

Here are some types of external media supported:

- BASIC, LARGE, Extended Format Sequential
- VSAM linear data sets

  GTF and CTRACE accept a single VSAM linear data set as output. VSAM's support for striping can increase data rate without the complexity associated with the use of distinct data sets.

For the details of the external data sets guidelines, see "Guidelines for defining GTF trace output data sets in a cataloged procedure" on page 217.

Wrapping DASD Trace Data Sets: If the WTRSTART parameter on the CTncccxx parmlib member or TRACE CT operator command specifies:

- WRAP or omits the parameter: When the system reaches the end of the data set or group of data sets, it writes over the oldest data at the start of the data set or first data set.

  The system also uses only the primary extent or extents for the data set or sets. To obtain the maximum degree of control over the number of trace entries for which space will be allocated, specify space allocation in units of the BLKSIZE of your trace data set, no secondary space, and use the option for contiguous allocation. For example, if your BLKSIZE is 8192, code the SPACE keyword as follows:

  ```
  SPACE=(8192,(500,0),,CONTIG)
  ```

- NOWRAP: When the data set or sets are full, the system stops writing trace records to the data set or sets. The system continues writing trace records in the address-space buffers.

  The system also uses the primary and secondary extents of the data set or sets.

Note: Wrapping is not supported for Extended Format Sequential data sets, which are treated as NOWRAP even if WRAP is specified.

Tape Data Sets: CTRACE writes an end-of-file record. The tape is rewound and unloaded, then a new volume is mounted on the tape unit. If CTRACE has only one tape data set and only one unit for the data set, CTRACE does not write trace records while the tape is unavailable, thus losing trace data. CTRACE can write to multiple tape units in the way that multiple TRCOUTxx DD statements can specify tape data sets. When CTRACE fills one data set, it changes to the next data set.

Note: GTF and CTRACE support placement of NOWRAP traces in cylinder-managed space. WRAP traces placed in VSAM linear data sets can reside in cylinder-managed space too. WRAP traces in non-VSAM data sets cannot be placed in large format data sets, extended format data sets, or cylinder-managed space.

Multiple Trace Data Sets: Use multiple data sets to capture all the trace records, even during spikes of activity. For a SYSRSM trace, which typically produces large numbers of trace records, use multiple data sets to keep from losing records. Multiple trace data sets by using different DASD devices can improve performance. To view the trace records in chronological sequence, the system programmer can:

- Combine the trace records into one data set, by using an IPCS COPYTRC subcommand, then use the CTRACE subcommand to format the records from the data set.

- Use an IPCS MERGE subcommand to format the records from multiple data sets.

The system places component trace records into each trace data set in sequence. For example, for three data sets, the system places:

- Record 1 into data set 1.
- Record 2 into data set 2.
- Record 3 into data set 3.
- Record 4 into data set 1.
- Record 5 into data set 2.
- And so on

Lost Trace Data: Ctrace will give an indication in the next successfully written record of any trace data that did not reach the output medium. If no further records are written, the following message is displayed when the external writer is stopped:

```
ITT120I SOME CTRACE DATA HAS BEEN LOST.
LAST nnn BUFFERS NOT WRITTEN.
```

Create a parmlib member

If you use a parmlib member, create the member and place it on SYS1.PARMLIB. Use a parmlib member if the options are complicated and you have access to the SYS1.PARMLIB data set, or if a parmlib member is required by the component, or if you had already set up a parmlib member with the needed options. Use a REPLY for simple options. See "Create CTncccxx parmlib members for some components" on page 351 for more information.**Example: CTWXCF04 parmlib member** For XCF, create CTWXCF04. Notice the two statements for the writer; the WTRSTART statement starts the writer and the WTR statement connects the writer to the component.

```
TRACEOPTS
   WTRSTART(WTDASD2)
   ON
   WTR(WTDASD2)
   OPTIONS('SERIAL','STATUS')
```

Perform component tracing to trace data sets

The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. Start the Writer and component trace: The operator enters TRACE operator commands on the console with MVS master authority and replies with the options specified by the system programmer.

Example: TRACE CT command not specifying a parmlib member: The second TRACE CT command starts the SYSXCF trace; the trace options were selected in a previous example. Notice the two writer operands; the WTRSTART operand starts the writer and the WTR operand connects the writer to the component.

```
trace ct,wtrstart=wtdasd2
trace ct,on,comp=sysxcf
* 44 ITT006A ....
r 44,wtr=wtdasd2,options=(serial,status),end
```

Example: TRACE CT command specifying a parmlib member: This example requests the same trace using parmlib member CTWXCF04.

```
trace ct,on,comp=sysxcf,parm=ctwxcf04
```

2. Verify that the Trace and the Writer Are Running: See "Verifying component tracing" on page 367.

3. Stop the component trace: The operator enters a TRACE CT command on the console with MVS master authority.

Example 1: TRACE CT,OFF Command

```
trace ct,off,comp=sysxcf
* 56 ITT006A ....
r 56,end
```

Example 2: TRACE CT Command to Disconnect the Writer: To stop sending trace records to the trace data set, but keep the trace running, the operator can enter the following when the trace is currently running.

```
trace ct,on,comp=sysxcf
* 56 ITT006A ....
r 56,wtr=disconnect,end
```

The operator should stop the external writer.

4. Stop the External Writer: The operator enters a TRACE CT command on the console with MVS master authority.

Example: TRACE CT,WTRSTOP Command:

```
trace ct,wtrstop=wtdasd2
```

Change trace data sets

If you are running a component trace to a trace data set or sets, you can determine if you have the needed records without stopping the trace. Ask the operator to perform the following actions:

1. Enter a **TRACE CT,WTRSTART** command for a different set of source JCL for each external writer to trace data sets.
2. Enter a **TRACE CT** command that starts the trace with the different source JCL for the writer.

The new source JCL sends the trace records to the new data set or sets.

Note: You might lose one or more trace records.

You can view the previous data set or sets to verify the collected trace records, then continue or stop the trace, as needed.

Example: Changing the trace data sets

1. Enter the following command to start the new external writer NewWtr:

   ```
   TRACE CT,WTRSTART=NewWtr
   ```

2. Enter the following commands to disconnect SYSxxx CTRACE from the current writer CurrWtr:

   ```
   TRACE CT,ON,COMP=SYSxxx
   REPLY xx,WTR=DISCONNECT,END
   ```

3. Enter the following commands to connect the new external writer NewWtr to SYSxxx CTRACE:

   ```
   TRACE CT,ON,COMP=SYSxxx
   REPLY xx, WTR=NewWtr, END
   ```

4. Enter the following command to stop the current external writer CurrWtr:

   ```
   TRACE CT,WTRSTOP=CurrWtr
   ```

Request component tracing for systems in a sysplex

The following topics describe one way to obtain traces for a component on more than one system in a sysplex. The approach is to obtain a trace in the dump of each system and merge the traces from the dumps, using an IPCS MERGE subcommand. To be useful for diagnosis, the traces must cover the same time period and end at the same time.

You can also trace to data sets, if each system uses a unique source JCL for each external writer, so that the trace for each system goes to its own data set. If your installation has a shared SYS1.PROCLIB system library, use a unique parmlib member for each system; each unique parmlib member must specify a unique set of source JCL. If the source JCL is shared, all systems will write trace records on one data set, possibly causing contention problems.

Prepare for specific component traces on systems in a sysplex

The system programmer performs the tasks.

1. Create a Parmlib Member to Start the Traces: Create a parmlib member to start the traces of the component. Place the member in the shared SYS1.PARMLIB for the sysplex or in the parmlib for each system to be traced. If a parmlib member is used for each system, give it the same name so that one TRACE CT command can start all the component traces on the systems. See “Create CTncccxx parmlib members for some components” on page 351.

Example: CTWXCF33 to Start XCF Trace: For XCF, create CTWXCF33 to start the trace.

```
TRACEOPTS
   ON
   OPTIONS('SERIAL','STATUS')
```

The directions for the task assume that a parmlib member can be used. If the component to be traced does not have a parmlib member, the operator can start it with a TRACE CT command in a ROUTE command. The operator has to enter a reply for each system. (The ROUTE command can be used only on MVS systems with JES2.)

2. Make Sure the component trace Buffers Will Be Dumped: The location of the address-space and data-space trace buffers depends on the component being traced. For XCF, the extended local system queue area (ELSQA) of the XCF address space contains the XCF component trace buffers.

Example: Obtaining XCF and XES Trace Buffers:

- For XCF, the operator should specify SQA and LSQA on the REPLY for the DUMP command.
- For XES, the operator should specify SDATA=(XESDATA).

Perform component tracing on the systems in the sysplex

The operator performs the tasks. Note that these tasks are for a specific component trace, rather than for a trace started by the system at initialization.

1. Start the component traces: On a console with MVS master authority on one system in the sysplex, the operator enters a ROUTE command containing a TRACE CT command. (The ROUTE command can be used only on MVS systems with JES2.)

The command specifies a parmlib member with the same name in each system being traced. Note that, if parmlib members are not specified, all systems issue message ITT006A to prompt for options. If the component to be traced does not have a parmlib member, specify the IBM-supplied CTIITT00 member to avoid the prompts.

Example 1: Command to start traces in all systems: The command starts the trace in all systems in the sysplex.

```
route *all,trace ct,on,comp=sysxcf,parm=ctwxcf33
```

Example 2: Command to start traces in some systems: The command starts the trace in a subset of systems. Both commands specify the CTWXCF33 parmlib member on each system being traced.

```
route subs2,trace ct,on,comp=sysxcf,parm=ctwxcf33
```

Example 3: Command for a component without a parmlib member: The following command turns on tracing for a SYSVLF trace in the systems of a sysplex, without prompts for replies to the TRACE command. The SYSVLF component trace has no parmlib member.

```
route *all,trace ct,on,comp=sysvlf,parm=ctiitt00
```

2. Verify that the traces Are running: See "Verifying component tracing" on page 367.

3. Obtain the dumps containing the component trace records: The operator requests an SVC dump for each system being traced.

Example: DUMP command for systems in a sysplex: The example shows the DUMP operator command entered on a console with MVS master authority on one system in the sysplex. The reply requests dumps on all of the systems named in the pattern of S*. The example assumes that the systems being traced have the following names: S1, S2, S3, and S4; any other systems in the sysplex have names that do not fit the pattern, such as B1 or T2.

```
dump  comm=(dump for xcf component trace)
* 32 IEE094D ...
r 32,remote=(syslist=(s*)),end
 .
 .
 .
IEA911E ...
```

The system identifies the data set containing the dump in message IEA911E. If an exit moves a dump, the operator should look for a message identifying the data set containing the moved dump and tell you the name of the dump and the data set containing it.

4. Stop the component traces: On a console with MVS master authority on one system in the sysplex, the operator enters a ROUTE command containing a TRACE CT,OFF command to stop the traces. (The ROUTE command can be used only on MVS systems with JES2.)

Example 1: Command to stop traces in all systems: The command stops the traces in all systems in the sysplex.

```
route *all,trace ct,off,comp=sysxcf
```

Example 2: Command to Stop Traces in Some Systems: The command stops the traces in a subset of systems in the sysplex.

```
route subs2,trace ct,off,comp=sysxcf
```

Verifying component tracing

The operator should do the following tasks after starting a component trace to make sure that it started successfully. How the task is done depends on whether the component trace has sublevels and whether an external writer is used.

Verify tracing for component traces without sublevels

The operator should do one of the following:

- Identify all current tracing by entering the following DISPLAY TRACE command on the console with MVS master authority. The response, in message IEE843I, gives the status in short form of all current component traces.

  ```
  display trace
  IEE843I ...
  ```

- Identify current tracing and the options for the traces by entering one of the following DISPLAY TRACE commands on the console with MVS master authority. The first command requests the status for all current component traces; the second command requests it for one component trace, such as XCF. The response, in message IEE843I, gives full information about the status.

```
display trace,comp=all
IEE843I ...

display trace,comp=sysxcf
IEE843I ...
```

Verify tracing for component traces with sublevels

The commands for verification depend on the component trace.

To verify a SYSJES component trace, the operator enters the following command to verify a SYSJES trace; the system will show the 4 sublevel traces.

```
display trace,comp=sysjes,sublevel,n=4
```

When a SYSXES component trace has multiple sublevel traces, a DISPLAY command shows only one sublevel. To verify a SYSXES component trace, the operator needs to enter multiple DISPLAY commands to see the multiple sublevels.

A SYSXES component trace has structures, address spaces, and connections. The following examples show the DISPLAY (D) command entered by the operator and the type of information that the system returns.

1. To see how the SYSXES component trace is set up.

```
 D TRACE,COMP=SYSXES

 IEE843I 15.24.40  TRACE DISPLAY 213
         SYSTEM STATUS INFORMATION
  ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
   COMPONENT     MODE BUFFER HEAD SUBS
  -----------------------------------------------------
   SYSXES        ON   0168K  HEAD    2
      ASIDS      *NOT SUPPORTED*
      JOBNAMES   *NOT SUPPORTED*
      OPTIONS    LOCKMGR
      WRITER     *NONE*
```

2. To display the structure level trace for each structure and the number of subtraces available.

```
D TRACE,COMP=SYSXES,SUB=(LT01),N=99

IEE843I 15.25.00  TRACE DISPLAY 216
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
  TRACENAME
  =========
  SYSXES
                      MODE BUFFER HEAD SUBS
                      =====================
                      ON   0168K  HEAD    2
     ASIDS      *NOT SUPPORTED*
     JOBNAMES   *NOT SUPPORTED*
     OPTIONS    LOCKMGR
     WRITER     *NONE*
  SUBTRACE            MODE BUFFER HEAD SUBS
 ----------------------------------------------------------
   LT01                            HEAD    1
      LIKEHEAD
  -----------------------------------------------------
   GLOBAL
      LIKEHEAD
```

3. To display the address space level trace for each structure. (The ASID specified is the asid in hex of the address space of the connector.)

```
D TRACE,COMP=SYSXES,SUB=(LT01.ASID(19)),N=99

IEE843I 15.25.39  TRACE DISPLAY 221
```

```
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
  TRACENAME
  =========
  SYSXES.LT01
                      MODE BUFFER HEAD SUBS
                      =====================
                      ON   0168K  HEAD    1
     LIKEHEAD
     ASIDS      *NOT SUPPORTED*
     JOBNAMES   *NOT SUPPORTED*
     OPTIONS    LOCKMGR
     WRITER     *NONE*

  SUBTRACE            MODE BUFFER HEAD SUBS
 -------------------------------------------------------------
  ASID(0019)                    HEAD    8
     LIKEHEAD
```

4. To display the external writer and the buffer size and options associated with the connection.

```
D TRACE,COMP=SYSXES,SUB=(LT01.ASID(19).A1),N=99

IEE843I 15.25.56  TRACE DISPLAY 224
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
  TRACENAME
  =========
  SYSXES.LT01.ASID(0019)
                      MODE BUFFER HEAD SUBS
                      =====================
                      ON   0168K  HEAD    8
     LIKEHEAD
     ASIDS      *NOT SUPPORTED*
     JOBNAMES   *NOT SUPPORTED*
     OPTIONS    LOCKMGR
     WRITER     *NONE*
 SUBTRACE            MODE BUFFER HEAD SUBS
-------------------------------------------------------------
 A1                  ON   0100K
    ASIDS      *NOT SUPPORTED*
    JOBNAMES   *NOT SUPPORTED*
    OPTIONS    CONNECT,RECOVERY
    WRITER     *NONE*
```

Verify that the writer is active

If an external writer is being used, the operator should verify that the writer is active for the trace by entering one of the following DISPLAY TRACE commands on the console with MVS master authority. The first command requests writer status for all current component traces; the second command requests it for one writer by specifying the membername of the source JCL for the writer, such as WTDASD2. The response is in message IEE843I.

```
display trace,wtr=all
IEE843I ...

display trace,wtr=wtdasd2
IEE843I ...
```

The operator should verify that the source JCL for the writer in this display is the same as the source JCL for the writer that was started for the trace. If the membernames do not match, the component trace data is lost. The operator should stop the writer job identified in the display and the component trace; then start the correct writer source JCL and start the trace again.

Viewing the component trace data

During diagnosis, the system programmer performs the tasks, using IPCS. See *z/OS MVS IPCS Commands* for the COPYDUMP, COPYTRC, CTRACE, GTFTRACE, and MERGE subcommands.

1. If your trace is in a dump in a SYS1.DUMPxx data set, enter a COPYDUMP subcommand to move the dump to another data set. Use option 5.3 of the IPCS dialog to select the COPYDUMP subcommand.

2. For all traces on trace data sets, use a COPYTRC subcommand to reorder component trace records that are out of chronological sequence. Use option 5.3 of the IPCS dialog to select the COPYTRC subcommand.

3. If your trace is on multiple data sets, do one of the following to view the trace records in one chronological sequence, which is needed to understand what was happening when the problem occurred. The input data sets can be component trace data sets, SVC dumps, and stand-alone dumps.

- Use the COPYTRC subcommand to combine the records on several data sets into a chronological sequence on one data set. Use this data set as input to the CTRACE subcommand, which formats the trace records. Use option 5.3 of the IPCS dialog to select the COPYTRC subcommand.
- Use a MERGE subcommand to format trace records from one or more input data sets. MERGE lets you combine and format the following:
  - Component traces
  - GTF traces
  - Sublevel traces from one component on one trace data set
  - Sublevel traces from one component on separate trace data sets

  For sublevel traces, MERGE groups together the trace records for each sublevel.

  Use option 2.7 of the IPCS dialog to select the MERGE subcommand. MERGE allows you to issue individual CTRACE or GTFTRACE subcommands for each input data set.

4. Use the subcommands in* Table 62 on page 370 *when formatting the component trace records. See *z/OS MVS IPCS Commands* for the SHORT, SUMMARY, FULL, and TALLY report type keywords and other keywords for the CTRACE subcommand.

Table 62. Subcommands that format component trace records

Trace	IPCS subcommand	CTRACE OPTIONS parameter
SYSAPPC	CTRACE COMP(SYSAPPC)	See “Formatting a SYSAPPC trace” on page 375.
SYSAXR	CTRACE COMP(SYSAXR)	See “Formatting a SYSAXR trace” on page 386.
SYSBCPII	CTRACE COMP(SYSBCPII)	See “SYSBCPII component trace” on page 387.
SYSBHI	CTRACE COMP(SYSBHI)	See “SYSBHI component trace” on page 390.
SYSCEA	CTRACE COMP(SYSCEA)	See “Formatting a SYSCEA trace” on page 396.
SYSDLF	CTRACE COMP(SYSDLF)	None.
SYSDSOM	CTRACE COMP(SYSDSOM)	See “SYSDSOM component trace” on page 399.
SYSDUMP	CTRACE COMP(SYSDUMP)	See “SYSDUMP component trace” on page 401.
SYSGRS	CTRACE COMP(SYSGRS)	None.
SYSHZS	CTRACE COMP(SYSHZS)	See “SYSHZS component trace” on page 412.
SYSIEFAL	CTRACE COMP(SYSIEFAL)	None.
SYSIOS	CTRACE COMP(SYSIOS)	See “Formatting a SYSIOS trace” on page 426.
SYSJES	CTRACE COMP(SYSJES)	See “Formatting a SYSJES trace” on page 432.
SYS*jes2*	CTRACE COMP(SYS*jes2*)	None.
SYSLLA	CTRACE COMP(SYSLLA)	None.
SYSLOGR	CTRACE COMP(SYSLOGR)	See “Formatting a SYSLOGR trace” on page 449.
SYSOMVS	CTRACE COMP(SYSOMVS)	See “Formatting a SYSOMVS trace” on page 453.
SYSOPS	CTRACE COMP(SYSOPS)	See “Formatting a SYSOPS trace” on page 463.
SYSRRS	CTRACE COMP(SYSRRS)	See “Formatting a SYSRRS trace” on page 469.

Table 62. Subcommands that format component trace records (continued)		
Trace	**IPCS subcommand**	**CTRACE OPTIONS parameter**
SYSRSM	CTRACE COMP(SYSRSM)	None.
SYSSPI	CTRACE COMP(SYSSPI)	None.
SYSTTRC	CTRACE COMP(SYSTTRC)	None.
SYSVLF	CTRACE COMP(SYSVLF)	None.
SYSWLM	CTRACE COMP(SYSWLM)	None.
SYSXCF	CTRACE COMP(SYSXCF)	See "Formatting a SYSXCF trace" on page 497.
SYSXES	CTRACE COMP(SYSXES)	See "Formatting a SYSXES trace" on page 502.

If some of the output in a combined or merged trace data set is for a GTF trace, use a GTFTRACE subcommand to format the GTF records and a CTRACE subcommand to format the component trace records. See Chapter 10, "The Generalized Trace Facility (GTF)," on page 211 for GTF tracing.

This example shows the CTRACE subcommand for a SYSXCF component trace, when the SERIAL and STATUS options are requested in the OPTIONS parameter.

```
ctrace comp(sysxcf) options((serial,status))
```

SYSAPPC component trace

Table 63 on page 371 summarizes information for requesting a SYSAPPC component trace for APPC/MVS.

Table 63. Requesting SYSAPPC component trace for APPC/MVS	
Information	**For SYSAPPC:**
Parmlib member	CTnAPPxx; there is no default member
Default tracing	No; cannot turn trace ON or OFF in CTnAPPxx
Trace request OPTIONS parameter	In CTnAPPxx or REPLY for TRACE command
Buffer	• Default: 32 MB • Range: 64 KB - 256 MB • Size set by: CTnAPPxx member or REPLY for TRACE command • Change size after IPL: Yes, while a trace is running • Location: In data space. A TRACE CT,OFF command requests a dump, which includes the trace buffers.
Trace records location	Dataspace buffer
Request of SVC dump	By the component when the operator stops SYSAPPC tracing with a TRACE CT,OFF command
Trace formatting by IPCS	CTRACE COMP(SYSAPPC)
Trace format OPTIONS parameter	NO

Requesting a SYSAPPC trace

Specify options for requesting a SYSAPPC component trace on a CTnAPPxx parmlib member or on the reply for a TRACE CT command.

CTnAPPxx parmlib member

You can specify the parameters listed in Table 64 on page 372 on a CTnAPPxx parmlib member.

Table 64. CTnAPPxx parameters	
Parameters	**Allowed on CTnAPPxx?**
ON or OFF	No
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	No
WTRSTART or WTRSTOP	No

TRACE and REPLY commands

Table 65 on page 372 and Table 66 on page 372 summarize the parameters you can specify on TRACE CT commands and a REPLY.

Table 65. Parameters allowed on TRACE CT	
Parameters	**Allowed on TRACE CT for Trace?**
ON, nnnnK, nnM, or OFF	One is required
COMP	Required
SUB	No
PARM	Yes

Table 66. Parameters allowed on REPLY	
Parameters	**Allowed on REPLY for Trace?**
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	No

Automatic Dump: The component requests an SVC dump when the operator stops the trace.

OPTIONS parameter

APPC trace request options are **hierarchical**. Figure 112 on page 373 shows the hierarchy of SYSAPPC trace options. Each option traces its own events, plus all the events of the options below it. For example, if you specify the SCHEDULE trace option, the system also traces ENQWORK, DEQWORK, and ASMANAGE events.

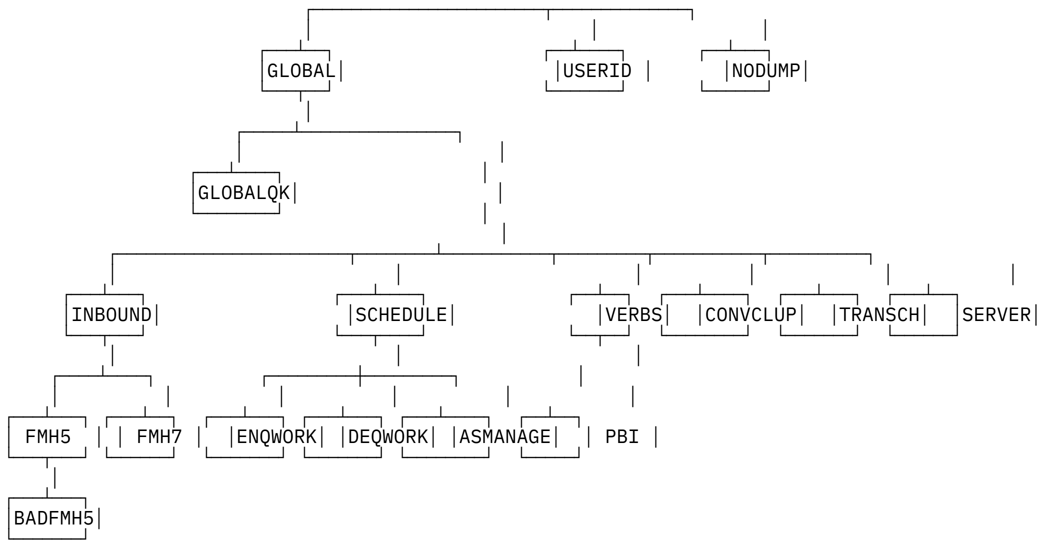


Figure 112. Hierarchy of SYSAPPC Component Trace Options

SYSAPPC tracing always includes all exception (error) events. If no trace options are specified, the trace output includes only the exception events.

If you do not know where the errors are occurring, use the GLOBAL trace option to catch the full range of APPC/MVS events. GLOBAL can slow performance, but you will catch the error in one re-create.

The values for the OPTIONS parameter for the CTnAPPxx parmlib member and reply for a TRACE command, in alphabetical order, are:

ASMANAGE
: Traces events related to the creation and deletion of the APPC/MVS transaction scheduler's subordinate address space. ASMANAGE is a subset of SCHEDULE events.

BADFMH5
: Traces events related to incorrect FMH-5s. BADFMH5 is a subset of FMH-5 events.

CONVCLUP
: Traces events related to conversation cleanup. CONVCLUP is a subset of GLOBAL events.

DEQWORK
: Traces the process of removing work requests from an APPC/MVS scheduler queue. DEQWORK is a subset of SCHEDULE events.

ENQWORK
: Traces the process of adding work requests to an APPC/MVS scheduler queue. ENQWORK is a subset of SCHEDULE events.

FMH5
: Traces FMH-5 events. FMH5 is a subset of INBOUND events.

FMH7
: Traces FMH-7 events. FMH7 is a subset of INBOUND events.

GLOBAL
: Traces the full range of APPC/MVS events.

GLOBALQK
: Traces a subset of important GLOBAL trace events.

INBOUND
: Traces inbound transaction processor (TP) requests. INBOUND is a subset of GLOBAL events.

NODUMP
Specifies no dumping when the operator stops the SYSAPPC component trace. Otherwise, component trace requests an SVC dump with the trace data when the operator stops tracing with a TRACE CT,OFF command.

IBM does not recommend the NODUMP option because the option makes obtaining the trace buffers difficult. The operator would have to identify the data space containing the buffers and specify it in a SLIP command or the reply for a DUMP command.

PBI
Traces events related to protocol boundary. PBI is a subset of VERBS events.

RR
Traces events related to the participation of APPC/MVS in resource recovery for protected conversations. RR is a subset of VERBS events.

SERVER
Traces events related to the APPC/MVS servers. SERVER is a subset of GLOBAL events.

SCHEDULE
Traces events related to the APPC/MVS transaction scheduler. SCHEDULE is a subset of GLOBAL events.

TRANSCH
Traces events related to APPC/MVS transaction scheduler interface support. TRANSCH is a subset of GLOBAL events.

USERID=(*userid*,*userid*)
Traces events for only the specified userid or userids. Specify the TSO/E userid of the person reporting a problem with an APPC/MVS application. Specify from 1 through 9 userids.

VERBS
Traces events related to outbound TPs or LU services. VERBS is a subset of GLOBAL events.

Examples of requesting SYSAPPC traces

This section contains examples of how to request SYSAPPC traces.

- Example 1: CTnAPPxx Member

  The member requests SERVER and VERBS options for the address space or spaces for the TSO/E userid JOHNDOE.

  ```
  TRACEOPTS
     OPTIONS('SERVER','VERBS','USERID=(JOHNDOE)')
  ```

- Example 2: TRACE command specifying a parmlib member

  The example specifies that options are to be obtained from the parmlib member CTWAPP03.

  ```
  trace ct,on,comp=sysappc,parm=ctwapp03
  ```

- Example 3: TRACE Command with Options Specified in a REPLY

  The example requests the same trace as Example 2, but specifies all options in the REPLY.

  ```
  trace ct,on,comp=sysappc
  * 15 ITT006A ...
  reply 15,options=(server,verbs,userid=(johndoe)),end
  ```

- Example 4: TRACE Command Requesting GLOBAL Options

  The example requests GLOBAL options for all address spaces using APPC/MVS.

  ```
  trace ct,on,comp=sysappc
  * 14 ITT006A ...
  reply 14,options=(global),end
  ```

Formatting a SYSAPPC trace

Format the trace with an IPCS CTRACE COMP(SYSAPPC) subcommand. Its OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

The options follow. The first option is either FILTER or CORRELATE, which are mutually exclusive; the first option controls how the other options select the records.

FILTER

The FILTER option selects the trace records that match only one of the specified options. The options that are valid with the FILTER option are:

- AQTOKEN
- CONVCOR
- CONVID
- FUNCID
- INSTNUM
- LUNAME
- LUWID
- NETNAME
- SEQNUM
- SESSID
- TPIDPRI
- TPIDSEC
- URID
- USERID

The formats of the OPTION parameter with FILTER are:

```
OPTION((FILTER,option))
OPTION((FILTER,option,option, ... ,option))
```

CORRELATE

The CORRELATE option selects the trace records that match a specified option and, for an unspecified option, uses the option's default values. The DEFAULTS keyword defines how default values are found for the unspecified options. The options that are valid with the CORRELATE option are:

- AQTOKEN
- CONVCOR
- CONVID
- DEFAULTS
- INSTNUM
- LUNAME
- LUWID
- NETNAME
- SEQNUM
- SESSID
- TPIDPRI
- TPIDSEC

- URID

The formats of the OPTION parameter with CORRELATE are:

```
OPTION((CORRELATE,option))
OPTION((CORRELATE,option,option, ... ,option))
```

AQTOKEN(*allocate-queue-token*)
Use with either the FILTER or CORRELATE option to specify an allocate queue token. The *allocate-queue-token* is an 8-byte hexadecimal string.

CONVCOR(*conversation-correlator*)
Use with either the FILTER or CORRELATE option to specify a conversation correlation. The *conversation-correlator* is an 8-byte hexadecimal string.

CONVID(*conversation-id*)
Use with either the FILTER or CORRELATE option to specify a conversation identifier. The *conversation-id* is a 4-byte hexadecimal string.

||DEFAULTS(NONEANYEXACT)
Use only with the CORRELATE option to specify the values to be used for matching unspecified options.

NONE
Tells component trace to format only the trace records that match one or more of the specified options. NONE is the default.

ANY
Tells component trace to format:

- Trace records matching one or more of the specified options.
- Related trace records that match default values established for the unspecified options. Component trace derives the defaults from the values for unspecified options found in the first records that match **any** of the specified options.

EXACT
Tells component trace to format:

- Trace records matching one or more of the specified options.
- Related trace records matching default values established for the unspecified options. Component trace derives the defaults from the values for unspecified options found in the first records that match **all** of the specified options.

FUNCID(*function-id*)
Use only with the FILTER option to specify the APPC/MVS subcomponent trace records to format. Specify one *function-id*:

01
Recovery

02
Verb services

03
FMH-5 manager

04
Conversation manager

05
System data file manager (SDFM)

06
VTAM exits

07
LU manager

08
State machine

09
Test enablement

10
APPC/MVS scheduler (ASCH)

11
Transaction scheduler interface

12
Allocate queue services

INSTNUM(*instance-number*)
Use with either the FILTER or CORRELATE option to specify the instance number for a logical unit of work. The *instance-number* is a 6-byte hexadecimal string.

LUNAME(*local-luname*)
Use with either the FILTER or CORRELATE option to specify the LU name for the local logical unit of work. The *local-luname* is an 8-byte EBCDIC character string.

LUWID(*logical-unit-of-work-id*)
Use either the FILTER or CORRELATE option to specify a logical unit of work identifier, which represents the processing a program performs from one sync point to the next. To specify the *logical-unit-of-work-id*, enter the hexadecimal string as it appears in the CTRACE report, without including blank spaces.

NETNAME(*network-name*)
Use with either the FILTER or CORRELATE option to specify the network name for a logical unit of work. The *network-name* is an 8-byte EBCDIC character string, which is the same as the network-ID portion of a network-qualified LU name.

SEQNUM(*sequence-number*)
Use with either the FILTER or CORRELATE option to specify the sequence number for a logical unit of work. The *sequence-number* is a 2-byte hexadecimal string.

SESSID(*session-id*)
Use with either the FILTER or CORRELATE option to specify the session identifier. The *session-id* is an 8-byte hexadecimal string.

TPIDPRI(*tp-id*)
Use with either the FILTER or CORRELATE option to specify the primary TP identifier. The *tp-id* is an 8-byte hexadecimal string.

TPIDSEC(*tp-id*)
Use with either the FILTER or CORRELATE option to specify the secondary TP identifier, which is used for multi-trans TPs. The *tp-id* is an 8-byte hexadecimal string.

URID(*unit-of-recovery-id*)
Use with either the FILTER or CORRELATE option to specify a unit of recovery identifier, which represents part of a TP's processing for a protected conversation. The *unit-of-recovery-id* is a 32-byte hexadecimal string.

USERID(*userid*)
Use only with the FILTER option to specify a userid as a filter. The *userid* is an 8-byte EBCDIC character string.

Examples of subcommands to format a SYSAPPC trace

- Example 1: CTRACE subcommand to view all trace entries

  To view all the SYSAPPC trace records, enter:

  ```
  CTRACE COMP(SYSAPPC)
  ```

- Example 2: CTRACE Subcommand to view exception entries

To format abnormal SYSAPPC events, such as abends or VTAM return codes, enter:

```
CTRACE COMP(SYSAPPC) EXCEPTION
```

- Example 3: CTRACE subcommand for subcomponent

  To format all the records for one APPC/MVS subcomponent, enter the following subcommand. Use this subcommand to locate an error if you have narrowed the problem down to one subcomponent.

```
CTRACE COMP(SYSAPPC) OPTIONS((FILTER,FUNCID(nn)))
```

- Example 4: CTRACE subcommand to view a userid's entries

  To format all the records for userid JOHNDOE, who is experiencing problems, enter the following subcommand. If you specified the USERID option when requesting the trace, this formatting option is redundant.

```
CTRACE COMP(SYSAPPC) OPTIONS((FILTER,USERID(JOHNDOE)))
```

Output from a SYSAPPC Trace

The following topics contain examples of different types of output produced by a SYSAPPC trace.

CTRACE COMP(SYSAPPC) SHORT subcommand output

The SHORT parameter shows one line of output for each trace record. Figure 113 on page 378 shows an example of SYSAPPC component trace output formatted with the SHORT parameter.

```
   VEFMH5XT  00004101  155705.182844  VEFMH-5 RECEIVED
   VEFMH5ER  00000101  155705.367233  VEFMH-5 IN TPEND
```

Figure 113. CTRACE COMP(SYSAPPC) SHORT subcommand output

The fields in each SHORT report line are:

Mnemonic
For example, VEFMH5XT.

Entry ID
The identifier for the trace record. For example, 00004101.

Time
The time in hh:mm:ss.tttttt format. For example, 15:57:05.182844.

Title
The title of the record. For example, VE:FMH-5 RECEIVED. Each title begins with a prefix that indicates the APPC/MVS subcomponent that wrote the trace record. For example, VE, which represents the VTAM exits subcomponent. Table 67 on page 378 relates the title prefixes to their APPC/MVS subcomponents.

Table 67. Summary of the title prefixes and APPC/MVS subcomponents

Prefix	Subcomponent
AMI	Verb services
ASCH	APPC/MVS scheduler (ASCH)
CM	Conversation manager
ERROR	Recovery
FMH5	FMH-5 manager
LUM	LU manager

Table 67. Summary of the title prefixes and APPC/MVS subcomponents (continued)

Prefix	Subcomponent
PC	Protected conversations
SDFM	MVS system data file manager (SDFM)
SF	Allocate queue services
SM	State machine
TE	Test enablement
TSI	Transaction scheduler interface
VC	Verb services
VE	VTAM exits
VS	Verb services

CTRACE COMP(SYSAPPC) SUMMARY subcommand output

The SUMMARY parameter gives the line in the SHORT report and most fields in each trace record. An example of SYSAPPC component trace output formatted with the SUMMARY parameter follows in Figure 114 on page 379.

```
SY1       PCESC     00007802  13:07:29.491950  PC:ENTRY STATE CHECK EXIT
      FUNCID... 02
      USERID... IBMUSER             JOBNAME.. APPC
      ASIDHOME. 001C                ASIDPRI.. 001C
      TPIDPRI.. 00000000  TPIDSEC.. 00000000
      SESSID... E723ED63  AAB04BDF  CONVID... 01000014
      CONVCOR.. 063313F8  0000000D  AQTOKEN.. 00000000
      LUWID.... 10E4E2C9  C2D4E9F0  4BE9F0C3  F0C1D7F0  F36FDB2A  C0220700  01
        NETNAME.. USIBMZ0             LUNAME... Z0C0AP03
        INSTNUM.. 6FDB2AC0  2207      SEQNUM... 0001
      URID..... AD355FDB  7EEFB000  00000007  01010000
```

Figure 114. CTRACE COMP(SYSAPPC) SUMMARY subcommand output

The fields in the SUMMARY report, after the first line, follow. See the SHORT report for the first line.

FUNCID
An identifier of the APPC/MVS subcomponent that wrote the trace record. See the FUNCID option for the identifiers.

USERID
The system was processing work for this userid when the trace record event occurred.

JOBNAME
The name of the job that the system was processing when the trace record event occurred.

ASIDHOME
The address space identifier (ASID) of the primary address space the system was processing when the trace record event occurred.

TPIDPRI
_The TP identifier of a primary TP. (Multitrans TPs have a primary and a secondary TP.)

TPIDSEC
_The TP identifier for a secondary TP. (Multitrans TPs have a primary and a secondary TP.)

SESSID
The identifier for a session.

CONVID
The identifier for a conversation.

AQTOKEN
The identifier for an allocate queue.

LUWID
The identifier for a logical unit of work. The following fields refer to the logical unit of work: If the LUWID is either all zeros or not valid,* the fields contain asterisks ().

NETNAME
The network name for the logical unit of work.

LUNAME
The name of the local LU.

INSTNUM
The instance number for the logical unit of work.

SEQNUM
The sequence number for the logical unit of work.

URID
The identifier for a unit of recovery.

CTRACE COMP(SYSAPPC) FULL subcommand output

The FULL parameter gives all the data in the trace records. It contains the line in the SHORT report, the fields in the SUMMARY report, and KEY and ADDR fields. An example of SYSAPPC component trace output formatted with the FULL parameter follows in Figure 115 on page 380.

```
SY1       PCESC     00007802  13:07:29.491950  PC:ENTRY STATE CHECK EXIT
     FUNCID... 02
     USERID... IBMUSER             JOBNAME.. APPC
     ASIDHOME. 001C                ASIDPRI.. 001C
     TPIDPRI.. 00000000  TPIDSEC.. 00000000
     SESSID... E723ED63  AAB04BDF  CONVID... 01000014
     CONVCOR.. 063313F8  0000000D  AQTOKEN.. 00000000
     LUWID.... 10E4E2C9  C2D4E9F0  4BE9F0C3  F0C1D7F0  F36FDB2A  C0220700  01
       NETNAME.. USIBMZ0             LUNAME... Z0C0AP03
       INSTNUM.. 6FDB2AC0  2207      SEQNUM... 0001
     URID..... AD355FDB  7EEFB000  00000007  01010000
     KEY...... 0015                ADDR..... 066F26DA
         E4E2C9C2  D4E9F04B  E9F0C3F0  C1D7F0F3  | USIBMZ0.Z0C0AP03 |
     KEY...... 001A                ADDR..... 066F26EB
         E4E2C9C2  D4E9F04B  E9F0C3F0  C1D7F0F4  | USIBMZ0.Z0C0AP04 |
     KEY...... 0039                ADDR..... 066F26A8
         E3D9C1D5  D7C1D940                    | TRANPAR          |
     KEY...... 0054                ADDR..... 064162E4
         00000000                              | ....             |
     KEY...... 00A1                ADDR..... 066F2640
         00000000                              | ....             |
     KEY...... 00A3                ADDR..... 055683D4
         00000000                              | ....             |
     KEY...... 00A3                ADDR..... 066F263C
         00000000                              | ....             |
     KEY...... 00A2                ADDR..... 066F263A
         00                                    | .                |
```

Figure 115. CTRACE COMP(SYSAPPC) FULL subcommand output

See the SHORT report and the SUMMARY report for the fields in the report. IBM might need the KEY and ADDR fields for diagnosis.

FMH-5 trace data

FMH-5 trace records contain information useful for tracking TP flow and diagnosing the following types of problems:

- Persistent verification problems
- Password maintenance problems
- APPC/MVS security problems

To obtain FMH-5 data, request the SYSAPPC component trace with an FMH5, INBOUND, or GLOBAL option. To isolate the FMH-5 records in the trace output, enter the following IPCS subcommand:

```
CTRACE COMP(SYSAPPC) OPTIONS((FILTER,FUNCID(03))) FULL
```

Table 68 on page 381 gives the mnemonic and title for each FMH-5 trace record and explains the record. Most of the trace records have FMH-5 itself formatted in KEY field X'0012'.

For the format of the FMH-5, see:

- *z/OS Communications Server: SNA Programmer's LU 6.2 Guide*
- *z/OS Communications Server: SNA Programmer's LU 6.2 Reference*.

Table 68. FMH-5 trace entries in the SYSAPPC component trace

Mnemonic	Title	Description/Action
FMH5BDSC	FMH5:BAD SECURITY COMBINATION	APPC/MVS found an incorrect security option or security subfields or both. Contact the IBM Support Center.
FMH5ERCV	FMH5:FMH-5 RECEIVE FAILURE	An FMH-5 was not successfully received by the local MVS LU. Contact the IBM Support Center.
FMH5INCD	FMH5:FMH-5 COMMAND IS NOT VALID	APPC/MVS detected an incorrect FMH-5 command. Contact the IBM Support Center.
FMH5LUNA	FMH5:LU IS NOT ACTIVE	An LU is not active. See the LUNAME field in the trace output. Enter a DISPLAY APPC command to find the status of this LU.
FMH5NOTP	FMH5:TP NAME IS NOT RECOGNIZED	The TP name was not specified correctly in the FMH-5.
FMH5NSCH	FMH5:NOT SERVED AND NO SCHEDULER	The TP cannot be scheduled because no scheduler is associated with the LU.
FMH5PFST	FMH5:FMH-5 PROFILE IS NOT VALID	The FMH-5 profile is incorrect; it is greater than 8 characters.
FMH5PIP	FMH5:PIP DATA PRESENT IN FMH-5	APPC/MVS found profile initialization parameters (PIP) data in the FMH-5; PIP data is not valid in FMH-5 for APPC/MVS.
FMH5PWCC	FMH5:PW CONV CLEANUP FAILED	Internal error. Contact the IBM Support Center.
FMH5PWDE	FMH5:PW DEALLOCATE FAILED	Internal error. Contact the IBM Support Center.
FMH5PWDF	FMH5:PW DEQUE REQUEST FAILED	An attempt to attach the SIGNON/Change password TP failed. Contact the IBM Support Center.
FMH5PWQF	FMH5:PW QUEUE REQUEST FAILED	Internal error. Contact the IBM Support Center.
FMH5PWRF	FMH5:QW RACF REQUEST REJECTED	Internal error. Contact the IBM Support Center.
FMH5PWR1 FMH5PWR2	FMH5:PW RECEIVE DATA FAILED 1 FMH5:PW RECEIVE DATA FAILED 2	The __SIGNON/Change password TP attempted to perform a ReceiveandWait call for a GDS variable. See the following KEY fields: • **KEY X'007E'** contains the status received__ value returned to the SIGNON/Change password TP by ReceiveandWait. • **KEY X'007F'** contains the data received value__ returned to the SIGNON/Change password TP by ReceiveandWait. • **KEY X'003F'** contains the return code from__ ReceiveandWait. Make sure that your GDS variable was sent correctly. If you cannot resolve the problem, contact the IBM Support Center.

Mnemonic	Title	Description/Action
Table 68. FMH-5 trace entries in the SYSAPPC component trace (continued)		
FMH5PWSD	FMH5:PW SEND DATA FAILED	A SIGNON/Change password TP SendData call failed. Verify that your TP has a valid conversation established with the SIGNON/Change password TP. If you cannot resolve the problem, contact the IBM Support Center.
FMH5PWSF	FMH5:PW SEND MESSAGE FAILED	A persistent verification signoff flow to the partner LU failed. The __SIGNEDONTO list in the partner LU may not be in sync with the local __SIGNEDONFROM list. See the following KEY fields: • **KEY X'0026'** contains the TCB address. • **KEY X'001A'** contains the name of the partner LU. • **Key X'002F'** contains the userid of the user whose SIGNOFF failed. If you cannot resolve the problem, contact the IBM Support Center.
FMH5PWSM	FMH5:PW SEND MESSAGE	APPC/MVS could not attach the X'30F0F5F2' expired password notification program to notify a partner system user that the user's password expired. See the following KEY fields: • **KEY X'0026'** contains the TCB address. • **KEY X'001A'** contains the name of the partner LU. • **KEY X'002F'** contains the USERID of the user whose attach request failed. If you cannot resolve the problem, contact the IBM Support Center.
FMH5PWSR	FMH5:PW SIF RESERVE FAILURE	Internal error. Contact the IBM Support Center.
FMH5PWST	FMH5:FMH-5 PASSWORD IS NOT VALID	The FMH-5 password is incorrect; it is greater than 8 characters.
FMH5QMFL	FMH5:FMFP QUEUE MANAGER FAILURE	Internal error. Contact the IBM Support Center
FMH5RECV	FMH5:FMH-5 SUCCESSFULLY RECEIVED	An FMH-5 was successfully received by the local MVS LU.
FMH5RFRJ	FMH5:RACF REQUEST REJECTED	The system received a bad return code from one of the RACF services. See KEY X'0053' for a code identifying the RACF service that failed. The code can be one of the following: **1** RACROUTE REQUEST=VERIFY **2** RACROUTE REQUEST=SIGNON TYPE=SIGNIN **3** RACROUTE REQUEST=SIGNON TYPE=QSIGNON **4** RACROUTE REQUEST=SIGNON TYPE=SIGNOFF See the following KEY fields: • **KEY X'0054'** contains the return code for the RACF service request. • **KEY X'0055'** contains the reason code for the RACF service request. • **KEY X'0021'** contains the security authorization facility (SAF) return code for the service.

Table 68. FMH-5 trace entries in the SYSAPPC component trace (continued)		
Mnemonic	**Title**	**Description/Action**
FMH5SERF	FMH5:APPC/MVS SERVICE FAILURE	APPC/MVS internal failure. Contact the IBM Support Center.
FMH5SFAL	FMH5:SEND MESSAGE FAILED	Persistent verification signoff flow to the partner LU failed. Make sure you have valid sessions established. See the following KEY fields: • **KEY X'0026'** contains the TCB address. • **KEY X'001A'** contains the name of the partner LU.
FMH5SOFF	FMH5:SIGNOFF FLOW	Persistent verification signoff flow to the partner LU completed. See the following KEY fields: • **KEY X'0026'** contains the TCB address. • **KEY X'001A'** contains the name of the partner LU.
FMH5SVFC	FMH5:ACCEPTED BY SRVR FACILITIES	APPC/MVS placed the inbound request on an allocate queue to await later processing by an APPC/MVS server.
FMH5TEST	FMH5:FMH5 ACCEPTED FOR TESTING	An FMH-5 is accepted for testing.
FMH5TPAD	FMH5:TP PROFILE ACCESS DENIED	TP profile access denied. Request=AUTH failed.
FMH5TPNA	FMH5:TP PROFILE IS NOT ACTIVE	The TP profile is not active. Get the TP name from the FMH-5 formatted at KEY X'0012' in this trace record. Then use the SDFM utility to look at the TP profile.
FMH5TPRQ	FMH5:TP PROFILE IS REQUIRED	The system found no TP profile for the requested TP. The scheduler associated with the TP requires a TP profile. The error is probably due to an SDFM problem. Look for trace records with a prefix of SDFM.
FMH5UIST	FMH5:FMH-5 USERID IS NOT VALID	The FMH-5 user ID is incorrect; it is greater than 8 characters.
FMH5VALD	FMH5:FMH5 SUCCESSFULLY VALIDATED	An FMH-5 has been successfully validated.
FMH5XLNF	FMH5:EXCHANGE LOG NAME FAILED	APPC/MVS rejected the protected conversation because required log-name exchange processing did not occur.
QMANFAIL	FMH5:FMAX QUEUE MANAGER FAILURE	Internal error. Contact the IBM Support Center.
RESVFAIL	FMH5:SIF RESERVE FAILURE	Internal error. Contact the IBM Support Center.

SYSAXR component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

The following summarizes information for requesting a SYSAXR component trace for System REXX component.

Information	For SYSAXR:
Parmlib member	CTIAXRnn. Default member: CTIAXR00
Default tracing	Yes; error; error events
Trace request OPTIONS parameter	In CTIAXRxx and REPLY for TRACE command

Information	For SYSAXR:
Buffer	• Default: 2MB • Range: 1MB - 2GB • Size set by: CTIAXRnn parmlib member or REPLY to TRACE CT command • Change size after IPL: Yes, when restarting a trace after stopping it • Location: System REXX trace data space.
Trace records location	Address-space buffer; System REXX trace data space; trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSAXR)
Trace format OPTIONS parameter	Yes

Requesting a SYSAXR trace

Specify options for requesting a SYSAXR component trace in a CTIAXRxx parmlib member or on the reply for a TRACE CT command. Changing SYSAXR trace options after AXR has started requires stopping and restarting the trace.

CTIAXRnn parmlib member

The following table indicates the parameters you can specify in a CTIAXRnn parmlib member.

Parameters	Allowed on CTIAXRnn?
ON or OFF	No
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
MOD	No
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

Note: The buffer size can be changed after IPL. To become effective the System REXX address space (AXR) must be restarted. Specify the new buffer size in the BUFSIZE parameter in the CTIAXRnn member being used.

The IBM supplied CTIAXR00 parmlib member initializes error tracing as soon as the System REXX address space starts. The contents of CTIAXR00 are:

```
TRACEOPTS
   ON
   OPTIONS('ERROR')
   BUFSIZE(2M)
```

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, OFF or nnnnM	One is required
nnnnK or nnnnM	No
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter in the CTIAXRxx parmlib member and reply for a TRACE command are:

ALL
Trace everything.

AXRCMD
Trace Command function package events.

AXRMLWTO
Trace multiline WTO function package events.

AXRWTO
Trace WTO function package events.

AXRWAIT
Trace wait function package events.

AXRINFO
Trace information function package events.

GETRXLIB
Trace AXREXX REQUEST=GETREXXLIB events.

CANCEL
Trace AXREXX REQUEST=CANCEL events.

COMMAND
Trace System REXX command events.

ERROR
Trace error events.

EXEC
Traces only events that occur under the specified exec name.

REXXARGS
Trace all events associated with REXX arguments.

REXXVARS
Trace all events associated with REXX variables.

RXCLIENT
Trace events that occur under the invoker of AXREXX.

RXSERVER
Trace all server events.

Formatting a SYSAXR trace

Format the trace with an IPCS CTRACE COMP(SYSAXR) subcommand.

Output from a SYSAXR Variables Trace

Each trace record has a header associated with it, as Figure 116 on page 386 shows. This header is consistent among ALL trace records, although not all fields are filled in (for example, there is no TracePrefixReqToken for command processing).

Note: This is NOT an interface and should only be used for diagnostic purposes.

```
                                            DSECT
TracePrefixType                   DS   OD
TracePrefixASID                   DS   H    Primary ASID when trace record cut
TracePrefixAXREXXInvokersASID     DS   H    Primary ASID when AXREXX invoked
TracePrefixJobname                DS   D    Jobname when trace record cut
TracePrefixAXREXXInvokersJobname  DS   D    Jobname when AXREXX was invoked
TracePrefixTcb@                   DS   A    TCB Address when AXREXX was invoked
TracePrefixExecName               DS   D    REXX exec name
TracePrefixAXREXXInvokersHomeAsid DS   A    Home ASID when AXREXX was invoked
                                  DS   CL2  Reserved
TracePrefixReqToken               DS   4F   Trace request token type
```

Figure 116. SYSAXR variables trace record header

The following shows the formatted IPCS output produced from the CTRACE COMP(SYSAXR) subcommand after running a REXX exec invoked using the AXREXX programming interface with the CTRACE REXXVARS option enabled.

```
SY1       REXXVARS  04130D06  20:40:59.969289  REXX VAR NAME

   00330032  C1E7D9D4  C1C9D540  C1E7D9D4  | ....AXRMAIN AXRM |
   C1C9D540  006EAE88  C8C1D9D9  C9E2E540  | AIN .>.hHARRISV  |
   00320000  00005000  00000001  BF8C7970  | ......&.......`. |
   95690DB8  00000001  D4E8E5C1  D94BF1    | n.......MYVAR.1  |
```

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The remainder of the trace entry contains the value of the variable name.

```
SY1       REXXVARS  04130D02  20:40:59.969292  REXX VAR BEFORE EXEC

   00330032  C1E7D9D4  C1C9D540  C1E7D9D4  | ....AXRMAIN AXRM |
   C1C9D540  006EAE88  C8C1D9D9  C9E2E540  | AIN .>.hHARRISV  |
   00320000  00005000  00000001  BF8C7970  | ......&.......`. |
   95690DB8  00000001  0000000B  40404040  | n...........     |
   40404040  4040F1                        |       1          |
```

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The next 4 bytes contain the length of the value. The remainder contains the value of the variable on input to the exec.

```
SY1       REXXVARS  04140D04  22:01:15.525516  REXX VAR AFTER EXEC

   00150032  C1E7D9D4  C1C9D540  C1E7D9D4  | ....AXRMAIN AXRM |
   C1C9D540  006EAE88  C8C1D9D9  C9E2E540  | AIN .>.hHARRISV  |
   00320000  00005000  00000000  BF8C8B61  | ......&......../ |
   09A8845A  00000001  00000003  F1F0F1    | .yd!........101  |
```

The 1st 4 bytes following the header contains the index of the variable in the variable list (AXRArgLst). The next 4 bytes should contain the length of the output. The next set of bytes should contain the output variable. If the result was truncated it will contain the truncated result.

The input/output variable contained 1 on entry to the exec and its final value when the exec completed was 101.

SYSBCPII component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following table summarizes information for requesting a SYSBCPII component trace for base control program internal interface (BCPii).

Information	For SYSBCPII:
Parmlib member	CTIHWI00 Default and only member: CTIHWI00. If no valid CTIHWI00 member exists, minimal tracing is activated at BPCPii address space initialization.
Default tracing	Minimal tracing is always in effect for SYSBCPII. If no valid CTIHWI00 member exists, if CTrace is turned OFF, or if CTrace is ON with no format OPTIONS specified, minimal tracing occurs.
Trace request OPTIONS parameter	In CTIHWI00 and REPLY for TRACE command
Buffer	• Size: 4M12M • Size set by: BCPii address space • Change size after IPL: No • Location: In the component data space
Trace records location	Data space buffer
Request of SVC dump	By DUMP or SLIP command, or by the component if trace is ON for SYSBCPII and a TRACE CT,OFF command is issued.
Trace formatting by IPCS	CTRACE COMP(SYSBCPII)
Trace format OPTIONS parameter	MIN, ALL

Requesting a SYSBCPII trace

Specify options for requesting a SYSBCPII component trace in a CTIHWI00 parmlib member or on a reply to a TRACE CT ,ON command.

You can change options for SYSBCPII tracing while the trace is running.

CTIHWI00 parmlib member

The following table indicates the parameters you can specify on a CTIHWI00 parmlib member.

Parameters	Allowed on CTIHWI00?
ON or OFF	Yes
OPTIONS	Yes

You cannot change the SYSBCPII component trace buffer size of 4M.

The IBM-supplied CTIHWI00 parmlib member initializes minimal error tracing as soon as the HWIBCPII address space starts.

The contents of CTIHWI00 are:

```
TRACEOPTS ON OPTIONS('MIN')
```

It is suggested that you use these default settings in the CTIHWI00 parmlib member, unless the IBM Support Center requests different tracing options for BCPii. If the CTIHWI00 parmlib member cannot be found during BCPii initialization, or if it is in error, minimal tracing will be activated.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON or OFF	One is required
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on REPLY for trace?
OPTIONS	Yes

You can change options while a SYSBCPII trace is running.

OPTIONS parameter

The values for the OPTIONS parameter for the CTIHWI00 parmlib member and reply for a TRACE command, in an alphabetical order, are:

ALL
: Traces events listed for all the options, including module flow and tracing for every request in both success and failure paths.

MIN
: Traces events related to BCPii component recovery, abnormal conditions, and other non-mainline paths.

Examples of requesting SYSBCPII traces

- The CTIHWI00 member requests ALL options.

  ```
  TRACEOPTS
     ON
     OPTIONS('ALL')
  ```

- The TRACE command specifying a Parmlib Member

  ```
  trace ct,on,comp=sysbcpii,parm=ctihwi00
  ```

- The TRACE command with Options Specified in a REPLY

  ```
  trace ct,on,comp=sysbcpii
  * 8 ITT006A ...
  reply 8,options=(all),end
  ```

- The TRACE command to Stop Tracing

  ```
  trace ct,off,comp=sysbcpii
  ```

Formatting a SYSBCPII trace

Format the trace with an IPCS CTRACE COMP(SYSBCPII) subcommand.

Output from a SYSBCPII trace

Figure 117 on page 389 is an example of SYSBCPII component trace records formatted with the CTRACE COMP(SYSBCPII) SHORT subcommand:

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSBCPII)
**** 07/25/2008

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP      DESCRIPTION
-------   --------  --------  ---------------   -------------

SY1       Message   10012130  13:50:09.359769   HWIPHCPI About to call
HWIPHRES
SY1       Request   100321C0  13:50:09.359963   HWIPHRES HSDB created

SY1       State     10002007  13:50:09.361439   HWIPHARI Appl Extension
located
SY1       Request   1000200E  13:50:09.368871   HWIPHARI Session Elem defined

SY1       Request   10002011  13:50:09.369177   HWIPHARI Request Elem created

SY1       Request   10002005  13:50:09.369328   HWIPHARI Request Elem queued

SY1       Request   10052220  13:50:09.374042   HWIPHSPI EDB Mds_MU created

SY1       State     10072351  13:50:09.427487   HWIPHMNX Appl Extension located

SY1       Request   10072352  13:50:09.427678   HWIPHMNX MDS_MU Req received
```

Figure 117. Example: SYSBCPII component trace records formatted with CTRACE COMP(SYSBCPII) SHORT

CTRACE COMP(SYSBCPII) FULL subcommand output

Figure 118 on page 390 is an example of SYSBCPII component trace records formatted with the CTRACE COMP(SYSBCPII) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSBCPII)
**** 07/25/2008

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP      DESCRIPTION
-------   --------  --------  ---------------   -------------

SY1       Message   10012130  13:50:09.359769   HWIPHCPI About to call HWIPHRES

  ASIDHOME. 0018 ASIDPRI.. 0018
  JOBNAME.. HWIBCPII TCBADDR.. 005DFA48

  KEY...... 0004 LEN...... 0026 COUNT.... 0001

    BCPii About to pass HSDB to HWIPHRES.

SY1       Request   100321C0  13:50:09.359963   HWIPHRES HSDB created

  ASIDHOME. 0018 ASIDPRI.. 0018
  JOBNAME.. HWIBCPII TCBADDR.. 005DFA48

  KEY...... 0005 LEN...... 003C COUNT.... 0001

    C8E2C4C2 080100B4 7ED57A54 00000001 | HSDB....=N:..... |
    40000000 00000000 00000000 C8E6C9E2 |  ...........HWIS |
    C5D9E540 00000000 00000000 5C404040 | ERV ........*    |
    40404040 5C404040 40404040          |     *            |

SY1       State     10002007  13:50:09.361439   HWIPHARI Appl Extension located

  ASIDHOME. 0018 ASIDPRI.. 0018
  JOBNAME.. HWIBCPII TCBADDR.. 005DFA48

  KEY...... 0005 LEN...... 0060 COUNT.... 0001

    C8E6C1E7 01100060 00F9000F 00000000 | HWAX...-.9...... |
    00000000 00000000 00000000 00000000 | ................ |
    00000000 00000000 00000000 00000000 | ................ |
    00000000 00000000 00000000 C8E6C9E2 | ............HWIS |
    C5D9E540 00000000 00000000 0000001C | ERV ............ |
    7F548228 00000000 00000000 00000000 | ".b............. |

SY1       Request   1000200E  13:50:09.368871   HWIPHARI Session Elem defined

  ASIDHOME. 0018 ASIDPRI.. 0018
  JOBNAME.. HWIBCPII TCBADDR.. 005DFA48

  KEY...... 0005 LEN...... 0080 COUNT.... 0001

    C8E2C540 01100080 C2BE9BC7 013E74E4 | HSE ....B..G...U |
    00000000 00000000 7ED54FA0 00000018 | ........=N|..... |
    00FB2A00 005DFA48 00000060 00000001 | .....)......-.... |
    0000000D 005DFA48 E4E2C9C2 D4E2C340 | .....)..USIBMSC  |
    E2C3E9D7 F9F0F140 00000000 00000000 | SCZP901 ........ |
    00000000 00000000 00000000 00000000 | ................ |
    C8E6C9F0 F0F0F0F1 00000000 00C80000 | HWI00001.....H.. |
```

Figure 118. Example: SYSBCPII component trace records formatted with CTRACE COMP(SYSBCPII) FULL

SYSBHI component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

Basic HyperSwap socket support component trace is described by the following attributes:

- Trace buffers reside in 64-bit common ECSA. Size is controlled by the TRACE CT operator command.
- Minimal and unexpected event tracing is activated during Basic HyperSwap management address space initialization.
- Component trace buffers externalized through:
  - DUMP or SLIP operator command when the Basic HyperSwap management address space or a BHIHSRV address space is requested to be dumped.

  - SVC dumps issued by Basic HyperSwap management address space or the BHIHSRV address space.
  - MVS component trace (CTRACE) external writer.
- Trace options revert to minimal event and exception tracing when the operator turns the trace off.

The following summarizes information for requesting a SYSBHI component trace for Basic HyperSwap.

Information	**For SYSBHI:**
Parmlib member	CTIBHIxx; Default member: CTIBHI00 (IBM provides a sample in SYS1.SAMPLIB)
Default tracing	Yes; error; error events
Trace request OPTIONS parameter	In CTIBHIxx and REPLY for TRACE command
Buffer	• Default: 4MB • Range: 4MB - 64MB • Size set by: CTIBHIxx parmlib member or REPLY to TRACE CT command • Change size after IPL: Yes • Location: 64-bit Common Storage Area (ECSA)
Trace records location	Address-space buffer; trace data sets
Request of SVC dump	By DUMP or SLIP command when dumping the HyperSwap Management address space or one of the BHIHSRV address spaces
Trace formatting by IPCS	CTRACE COMP(SYSBHI)
Trace format OPTIONS parameter	Yes

Requesting a SYSBHI trace

Specify options for requesting a SYSBHI component trace in a CTIBHIxx parmlib member or on the reply for a TRACE CT command

CTIBHIxx parmlib member

The following table indicates the parameters you can specify in a CTIBHIxx parmlib member.

Parameters	**Allowed on CTICEAnn?**
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
MOD	No
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

The IBM supplied CTIBHI00 parmlib member in SYS1.SAMPLIB can be used as a starting point for defining BHI CTRACE options. The contents of CTIBHI00 are:

```
TRACEOPTS
   ON
   BUFSIZE(4M)
   OPTIONS('MINIMUM')
```

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, OFF or nnnnM	One is required
nnnnK or nnnnM	No
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter in the CTIBHIxx parmlib member and reply for a TRACE command are:

ALL
Trace everything.

FLOW
Trace the flow of all requests through their processing.

INITTERM
Trace the initialization and termination process of the BHIHSRV address spaces and tasks.

MINIMUM
Trace errors and unusual events.

Formatting a SYSBHI trace

Format the trace with an IPCS CTRACE COMP(SYSBHI) subcommand.

Output from a SYSBHI trace

Figure 119 on page 393 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSBHI) SHORT subcommand.

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSBHI)
**** 12/20/2012

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

SY1       InitMsg   00000001  19:20:28.105817  CTrace Initialized

SY1       TskStrtd  00000002  19:20:28.105858  Task has been started

SY1       BPXRslt   0000002B  19:20:28.106110  BPX Service Results

SY1       BPXRslt   0000002B  19:20:28.107162  BPX Service Results

SY1       RACFRslt  00000035  19:20:56.880588  Racf Results

SY1       LognRejt  00000036  19:20:56.880589  Logon Rejection

SY1       BPXRslt   0000002B  19:20:56.880626  BPX Service Results
```

Figure 119. Example: formatted IPCS output formatted with CTRACE COMP(SYSBHI) SHORT

Figure 120 on page 394 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSBHI) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSBHI)
**** 12/20/2012

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

SY1       InitMsg   00000001  19:20:28.105817  CTrace Initialized

    ASID..... 0025      IssueMod. IOSHMSTR  TCB...... 005D6048
    HSAITIdx. 00000000
    MiscEbcd. CTrace Initialization Complete
SY1       TskStrtd  00000002  19:20:28.105858  Task has been started

    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000
SY1       BPXRslt   0000002B  19:20:28.106110  BPX Service Results

    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000
    BPXServ.. BPX1QDB - Querying DUB Status
    RetValue. 00000008
    RetCode.. 00000000
    RsnCode.. 00000000
SY1       BPXRslt   0000002B  19:20:28.107162  BPX Service Results

    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000
    BPXServ.. BPX1ENV - Reg USS Shutdown Exit
    RetValue. 00000000
    RetCode.. 00000000
    RsnCode.. 00000000
SY1       RACFRslt  00000035  19:20:56.880588  Racf Results

    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000
    MiscEbcd. Req=Verify Envir=Create
    RetCode.. 00000008
    RetCode.. 00000008
    RsnCode.. 00000000
SY1       LognRejt  00000036  19:20:56.880589  Logon Rejection

    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000

    --- TOH Start ---
    Acronym.. TOH       Version.. 01        Size..... 00000038
    Function Code: 000000C8 (Logon Request)
    User Token
    D4A8E396  92859540  40404040  40404040  | MyToken          |
    BuffSize. 00000000  Buff_Off. 0000
    Return Code: 00000390 (SAF or RACF not available or error)
    Reason Code: 05080800
                 RACROUTE Function    : Req=Verify Envir=Create
                 RACROUTE Return Code: 08
                 RACF Return Code     : 08   RACF Reason Code: 00
    --- TOH End ---
SY1       BPXRslt   0000002B  19:20:56.880626  BPX Service Results
    ASID..... 0025      IssueMod. IOSHMSRT  TCB...... 005D6048
    HSAITIdx. 00000000
    BPXServ.. BPX1SND - Reject Logon
    RetValue. 00000038
    RetCode.. 00000000
    RsnCode.. 00000000
    SockTokn. 23B00000
```

Figure 120. Example: formatted IPCS output formatted with CTRACE COMP(SYSBHI) FULL

SYSCEA component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSCEA component trace for common event adapter component.

Information	For SYSCEA:
Parmlib member	CTICEAnn; default member: CTICEA00
Default tracing	Yes; error; error events
Trace request OPTIONS parameter	In CTICEAxx and REPLY for TRACE command
Buffer	• Default: 2MB • Range: 1MB - 2GB • Size set by: CTICEAnn parmlib member or REPLY to TRACE CT command • Change size after IPL: Yes, when restarting a trace after stopping it • Location: common event adapter trace data space.
Trace records location	Address-space buffer; common event adapter trace data space; trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSCEA)
Trace format OPTIONS parameter	Yes

Requesting a SYSCEA trace

Specify options for requesting a SYSCEA component trace in a CTICEAxx parmlib member or on the reply for a TRACE CT command. Changing SYSCEA trace options after CEA has started requires stopping and restarting the trace.

CTICEAnn parmlib member

The following table indicates the parameters you can specify in a CTICEAnn parmlib member.

Parameters	Allowed on CTICEAnn?
ON or OFF	No
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
MOD	No
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

Note: The buffer size can be changed after IPL. To become effective, the common event adapter address space (CEA) must be restarted. Specify the new buffer size in the BUFSIZE parameter in the CTICEAnn member being used.

The IBM supplied CTICEA00 parmlib member initializes error tracing as soon as the common event adapter address space starts. The contents of CTICEA00 are:

```
TRACEOPTS
   ON
   BUFSIZE(2M)
   OPTIONS('ERROR')
```

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, OFF or nnnnM	One is required
nnnnK or nnnnM	No
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter in the CTICEAxx parmlib member and reply for a TRACE command are:

ALL
Trace everything.

CNTLFLOW
Trace CNTLFLOW events.

EVNTFLOW
Trace EVNTFLOW events.

JOBSFLOW
Trace JOBSFLOW events.

PDWBFLOW
Trace PDWBFLOW events.

ERROR
Trace error events.

Formatting a SYSCEA trace

Format the trace with an IPCS CTRACE COMP(SYSCEA) subcommand.

Output from a SYSCEA trace

Figure 121 on page 397 shows an example of the formatted IPCS output produced from the CTRACE COMP(SYSCEA) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSCEA)
**** 06/09/2008

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP      DESCRIPTION
-------   --------  --------  ---------------  -------------

SY1       ALL       05030001  11:37:11.181208  CEA CTRACE DEBUG INFO

   FFF0003E  7F47A000  00000000  00000000  | .0..".......... |
   00000000  00000000  00000000  00028000  | ................ |
   00000000  00000000  7F74ACA0  01000000  | ........"....... |
   00000000  00000000  00000000  0000      | ..............   |
SY1       CNTLFLOW  04010001  11:37:11.181208  CEAS SERVER

   00160000  C3C5C140  40404040  008E2E88  | ....CEA     ...h |
   C3C5C1E2  6DC9D5C9  E3C9C1D3  C9E9C5C4  | CEAS_INITIALIZED |
   C3C5C1E2  6DC9D5C9  E3C9C1D3  C9E9C5C4  | CEAS_INITIALIZED |
   00000000  00000000  00000000  00000000  | ................ |
   00000000  00000000  00000000            | ............     |
SY1       PDWBFLOW  04400001  11:38:01.616152  ICIN--COMPONENT ID TABLE LOAD

   00160000  C3C5C140  40404040  008E2E88  | ....CEA     ...h |
   00000001  00000000                      | ........         |
SY1       CNTLFLOW  04010001  11:38:01.616156  CEAS SERVER

   00160000  C3C5C140  40404040  008E2E88  | ....CEA     ...h |
   E4E2E26D  C9E26DE4  D7404040  40404040  | USS_IS_UP        |
   00000000  00000000  00000133  00000000  | ................ |
   00000000  00000000  00000000            | ............     |
SY1       PDWBFLOW  044F0001  11:45:26.522624  PDWB--DONE WITH GETINCIDENT

   00160000  D4C5C7C1  F3404040  008FF1D8  | ....MEGA3   ..1Q |
   00000000  00000000  00000000  00000000  | ................ |
```

Figure 121. Example: formatted IPCS output formatted with CTRACE COMP(SYSCEA) FULL

SYSDLF component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSDLF component trace for the data lookaside facility (DLF).

Information	**For SYSDLF:**
Parmlib member	None
Default tracing	Yes; always on when DLF is running
Trace request OPTIONS parameter	None
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: Data space. In the REPLY for the DUMP command, specify DSPNAME=('DLF'.CCOFGSDO)
Trace records location	Address-space buffer, data-space buffer
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSDLF)
Trace format OPTIONS parameter	None

Requesting a SYSDLF trace

The trace runs whenever DLF is in control. No actions are needed to request it.

Formatting a SYSDLF trace

Format the trace with an IPCS CTRACE COMP(SYSDLF) subcommand. The subcommand has no OPTIONS values.

Output from a SYSDLF trace

Figure 122 on page 398 is an example of DLF component trace records formatted with a CTRACE COMP(SYSDLF) FULL subcommand. It shows formatted exception records from the trace buffers.

```
                     DLF COMPONENT TRACE FULL FORMAT

    COFRCVRY  00000000  15:47:40.397545  DLF RECOVERY ENTRY
      HASID... 000E     SASID... 000E     CPUID... FF170067 30900000
      MODNAME. COFMCON2 ABEND... 840C1000 REASON.. 00000001
      EPTABLE. CON2  EST2  ....  ....  ....  ....  ....  ....  ....
    COFRCVRY  00000001  15:47:40.397625  DLF RECOVERY EXIT
      HASID... 000E     SASID... 000E     CPUID... FF170067 30900000
      MODNAME. COFMCON2 ABEND... 840C1000 REASON.. 00000001
      RETCODE. 0000002C RSNCODE. 0000C200 FTPRTS.. C0000000 DATA.... 00000000
```

Figure 122. Example: formatted IPCS output formatted with CTRACE COMP(SYSDLF) FULL

The fields in the report are:

COFRCVRY
The name or identifier of the trace record.

00000000
The identifier in hexadecimal.

15:47:40.397545
The time stamp indicating when the record was placed in the trace table.

HASID... 000E
The home address space identifier.

SASID... 000E
The secondary address space identifier.

CPUID... FF170067 30900000
The identifier of the processor that placed the record in the trace table.

CALLER
The address of the routine that issued a DLF service request.

MODNAME COFMCON2
The name of the module that was running.

ABEND... 840C1000
The abend that occurred and caused DLF to enter recovery.

REASON.. 00000001
The reason code associated with the abend.

EPTABLE. CON2 EST2
Information used for diagnosis by IBM.

RETCODE. 0000002C
The return code that was issued by the module that is exiting.

RSNCODE. 0000C200
The reason code that was issued by the module that is exiting.

FTPRTS.. C0000000
Information used for diagnosis by IBM.

DATA.... 00000000
Information used for diagnosis by IBM.

SYSDSOM component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSDSOM component trace for distributed SOMobjects (DSOM).

Information	For SYSDSOM:
Parmlib member	None
Default tracing	No
Trace request OPTIONS parameter	In REPLY for TRACE command
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: Address space
Trace records location	Address-space buffer
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSDSOM)
Trace format OPTIONS parameter	Yes

Requesting a SYSDSOM trace

Request the trace by specifying any non-zero value on the SOMDTRACELEVEL DSOM environment variable.

Formatting a SYSDSOM trace

Format the trace with an IPCS CTRACE COMP(SYSDSOM) subcommand. The subcommand has the following OPTIONS values:

SKIPID
Omits the jobname, ASID, and thread identifier from the output.

LONGFORM
Tells the system to display detailed output. In the output, each trace function might have multiple trace elements, each on a separate line. If you do not specify LONGFORM, the default is SHORTFORM. Do not specify both LONGFORM and SHORTFORM on the OPTIONS parameter.

SHORTFORM
Tells the system to display abbreviated output. In the output, each trace function is on one line. SHORTFORM is the default value. Do not specify both LONGFORM and SHORTFORM on the OPTIONS parameter.

Output from a SYSDSOM trace

The output shown in Figure 123 on page 400 is an example of DSOM component trace records formatted with a CTRACE COMP(SYSDSOM) FULL OPTIONS((SKIPID,SHORTFORM)) subcommand. It shows formatted exception records from the trace buffers.

```
                     DSOM COMPONENT TRACE FULL FORMAT

      KESYS522  METRETRN  00000004  21:31:34.864277  Return from method
           Entry to method:        ImplRepository::somInit
```

Figure 123. Example: DSOM component trace records formatted with CTRACE COMP(SYSDSOM) FULL OPTIONS((SKIPID,SHORTFORM))

Figure 124 on page 400 is an example of DSOM component trace records formatted with a CTRACE COMP(SYSDSOM) FULL subcommand. It shows formatted exception records from the trace buffers.

```
   COMPONENT TRACE FULL FORMAT
   SYSNAME(KESYS522)
   COMP(SYSDSOM)
   **** 09/29/1995
   SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
   -------   --------  --------  ---------------  -------------
   KESYS522  GETBUFF   00000001  21:31:30.198289  Get new trace buffer
        JOBNAME. KREPROC  ASID.... 0029     THREADID 04233100 00000000
        Buffer address:         7F672508
   KESYS522  METDEBUG  00000004  21:31:39.410681  Method debug
        JOBNAME. KREPROC  ASID.... 0029     THREADID 04233100 00000000
        Entry to method:        SOMOA::somInit
   KESYS522  METRETRN  00000005  21:31:40.000019  Return from method
        JOBNAME. KREPROC  ASID.... 0029     THREADID 04233100 00000000
        Exiting method:         SOMOA::somInit, RC(hex)=00000000, RSN=00000000
```

Figure 124. Example: DSOM component trace records formatted with CTRACE COMP(SYSDSOM) FULL

The output shown in Figure 125 on page 400 is an example of DSOM component trace records formatted with a CTRACE COMP(SYSDSOM) FULL OPTIONS((SKIPID)) DSN('dsom.trace.dsn') subcommand. It shows formatted exception records from the trace buffers.

```
   COMPONENT TRACE FULL FORMAT
   SYSNAME(KESYS522)
   COMP(SYSDSOM)
   OPTIONS((SKIPID))
   **** 09/29/1995
   SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
   -------   --------  --------  ---------------  -------------
   KESYS522  GETBUFF   00000001  21:31:30.198289  Get new trace buffer
        Buffer address:         7F672508
   KESYS522  METDEBUG  00000004  21:31:39.410681  Method debug
        Entry to method:        SOMOA::somInit
   KESYS522  METRETRN  00000005  21:31:40.000019  Return from method
        Exiting method:         SOMOA::somInit, RC(hex)=00000000, RSN=00000000
```

Figure 125. Example: DSOM component trace records formatted with CTRACE COMP(SYSDSOM) FULL OPTIONS((SKIPID)) DSN('dsom.trace.dsn')

The fields in the report are:

KESYS522
The name of the system.

METDEBUG
The name of the trace event.

00000004
The decimal identifier of the trace event.

21:31:39.410681
The time stamp indicating when the record was placed in the trace table.

ASID
The ASID of the job listed in the JOBNAME field.

JOBNAME. KREPROC
The job name.

THREADID 04233100 00000000
The POSIX thread identifier.

Entry to method
The entry to the somInit method in class SOMOA.

Exiting Method
The exit from the somInit method in class SOMOA.

SYSDUMP component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

SDUMP component trace is described by the following attributes:

- Trace buffers reside in 64-bit common storage area. Size is controlled by the TRACE CT operator command.
- Tracing is activated during DUMPSRV address space initialization.
- SDUMP CTRACE is captured at the end of the SDUMP capture phase and dumped at the end of the dump. If the trace data is needed at other times, you must request a new dump to collect the SDUMP CTRACE.
- The MVS component trace (CTRACE) external writer can also capture SDUMP CTRACE.
- Trace options revert to minimal event tracing when the operator turns the trace off.

The following table summarizes the information required to request a SYSDUMP component trace for SDUMP.

Information	For SYSDUMP:
Parmlib member	CTIDMPxx; Default member: CTIDMP00
Default tracing	Yes; full tracing
Trace request OPTIONS parameter	In CTIDMPxx and REPLY for TRACE CT command
Buffer	• Default: 4MB • Range: 4MB – 32MB • Size set by: CTIDMPxx parmlib member or TRACE CT command • Change size after IPL: Yes • Location: 64-bit Common Storage Area
Trace records location	64-bit Common Storage Area; trace data sets
Externalizing trace data	Included in every SDUMP; can also use external writer to write data to a trace data set
Trace formatting by IPCS	CTRACE COMP(SYSDUMP)
IPCS trace format OPTIONS parameter	None
Maximum trace entry size	4050 bytes

Requesting a SYSDUMP trace

Specify options for requesting a SYSDUMP component trace in a CTIDMPxx parmlib member or on the reply for a TRACE CT command.

CTIDMPxx parmlib member

The following table indicates the parameters you can specify in a CTIDMPxx parmlib member.

Parameters	Allowed on CTIDMPxx?
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
MOD	No
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

You can use the IBM supplied CTIDMP00 parmlib member in SYS1.PARMLIB as a starting point for defining SDUMP CTRACE options. The contents of CTIDMP00 are as follows:

```
TRACEOPTS
ON
BUFSIZE(4M)
OPTIONS('ALL')
```

TRACE and REPLY commands

The following tables indicate the parameters that you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, OFF or nnnnM	One is required
nnnnK or nnnnM	No
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is used

Parameters	Allowed on REPLY for Trace?
ASID	No
JOBNAME	No
OPTIONS	Yes

Parameters	Allowed on REPLY for Trace?
WTR	Yes

OPTIONS parameter

The following values are available for the OPTIONS parameter in the CTIDMPxx parmlib member and reply for a TRACE command:

ALL
Trace everything.

MINIMUM
Trace errors.

Formatting a SYSDUMP trace

Format the trace with an IPCS CTRACE COMP(SYSDUMP) subcommand.

Output from a SYSDUMP trace

Figure 126 on page 404 is an example of the formatted IPCS output that is produced from the CTRACE COPM(SYSDUMP) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSDUMP)
**** 08/20/2014

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

S790      WtDSVSST  0000007F  17:59:02.744031  Wait for DSVSSTECB to be post

    ASID..... 0005      IssueMod. IEAVTSST  TCB...... 004DFD90
    RetnAddr. 89B0258A

S790      LockSdmp  00000024  18:03:35.343314  Sdump is locked

    ASID..... 001B      IssueMod. IEAVAD00  TCB...... 004FF200
    RetnAddr. 862D28EC
    CVTSDBF.. 80950FF8
    RTCTSDPL. 00000000
    JobName.. DCMTESTC
S790      DmpStrtd  00000002  18:03:35.343317  Sdump started

    ASID..... 001B      IssueMod. IEAVAD00  TCB...... 004FF200
    RetnAddr. 862D29D4
    RTCTSDPL. 894CF620

S790      ClrStor   00000029  18:03:35.343782  Storage cleared for new dump

    ASID..... 001B      IssueMod. IEAVAD00  TCB...... 004FF200
    RetnAddr. 862D2A78

S790      EntyTSPR  00000013  18:03:35.343788  Entry at IEAVTSPR

    ASID..... 001B      IssueMod. ........  TCB...... 004FF200
    RetnAddr. 812D6184
    RTSDFNCD. 0001

S790      EntyTSPR  00000013  18:03:35.343795  Entry at IEAVTSPR

    ASID..... 001B      IssueMod. ........  TCB...... 004FF200
    RetnAddr. 812D6184
    RTSDFNCD. 0005

S790      SdumpPml  00000004  18:03:35.343813  Sdump PList after calling TSPR

    ASID..... 001B      IssueMod. IEAVAD00  TCB...... 004FF200
    RetnAddr. 862D3120
    RTCTSDPL. 023CCF40
           Sdump ParmList:
    +0000  10A19010  00000000  00000000  023CC838  | .~...........H. |
    +0010  00000000  00000000  00000000  00000000  | ................ |
    +0020  00000000  00000000  00C00003  25504000  | .........{...& . |
    +0030  00000000  00000000  00000000  00000000  | ................ |
    +0040  00000000  00000000  00000000  00000000  | ................ |
    +0050  00000000  00000000  090B03F0  00000000  | ...........0.... |
    +0060  00000000  00000000  00000000  00000000  | ................ |
    +0070  00000000  00000000  00000000  00000000  | ................ |
    +0080  00000000  00000000  025B4190  00000000  | .........$...... |
    +0090  00000000  00000000  00000000  00000000  | ................ |
    +00A0  00000000  00000000  00000000  00000000  | ................ |
    +00B0  80000000  00000000                      | ........         |
    Title.... DMPTESTC:  Test case dump for >2G local capture
                      optimization.................

S790      DumpSupp  00000006  18:03:35.343820  Dump is not being suppressed

    ASID..... 001B      IssueMod. IEAVAD00  TCB...... 004FF200
    RetnAddr. 862D33CC
```

Figure 126. Example: SDUMP component trace records formatted with CTRACE COMP(SYSDUMP) FULL

Viewing SDUMP CTRACE in IPCS Active

You can view SDUMP CTRACE data in IPCS from active system storage. Ensure that the dump source is set to 'ACTIVE' and that the user has authorized READ access to the FACILITY class resource BLSACTV.ADDRSPAC. See "Active Storage Processing" in *z/OS MVS IPCS User's Guide* for more information.

SYSGLZ component trace

The following summarizes information for requesting a SYSGLZ component trace for z/OS Container Extensions (zCX).

Table 69. zCX Information for SYSGLZ

Information Type	SYSGLZ Specifications
Parmlib member	CTIGLZnn (Default member is CTIGLZ00)
Default tracing	Yes
Trace request `OPTIONS` parameter	In CTIGLZnn and REPLY for TRACE command
Buffer	• Default: 64 MB • Range: 1 MB - 64 MB • Size set by CTIGLZnn parmlib member or TRACE CT command • Size change after IPL when restarting a trace after stopping it • Location: zCX instance address spaces
Trace records location	Address-space buffer, trace data set for external writer
Request of SVC dump	By `DUMP` or `SLIP` command
Trace formatting by IPCS	`CTRACE COMP(SYSGLZ) SUB((ASID(xxxx)))`
Trace format `OPTIONS` parameter	Yes

Requesting a SYSGLZ trace

Specify options for requesting a SYSGLZ component trace in a CTIGLZxx parmlib member or on the reply for a TRACE CT command. The following table indicates the parameters in a CTIGLZnn parmlib member.

GCTIGLZnn parmlib member

The following table indicates the parameters you can specify on a GCTIGLZnn parmlib member.

Parameter	Allowed specification on CTIGLZnn
`ON` or `OFF`	Yes
`ASID`	No
`JOBNAME`	No
`BUFSIZE`	Yes (can be changed after IPL)
`OPTIONS`	Yes
`MOD`	No
`SUB`	No
`PRESET`	No
`LIKEHEAD`	No
`WTR`	Yes
`WTRSTART` or `WTRSTOP`	Yes

The IBM-supplied CTIGLZ00 parmlib member initializes tracing as soon as a zCX instance address space starts. The contents of CTIGLZ00 are:

```
TRACEOPTS
          ON
          BUFSIZE(64M)
          OPTIONS('DISK','NET','SIE')
```

TRACE and REPLY commands

The following tables indicate the parameters that can be specified on TRACE CT commands and a REPLY.

Table 70. Parameters for TRACE CT for Trace

Parameter	Allowed specification on TRACE CT for Trace
ON, OFF, or nnnnM	One is required
nnnnK or nnnnM	Yes
SUB	Required; specified as SUB=(ASID(xxx)), where xxxx is the 4-digit hexadecimal ASID of the zCX instance the command is targeting
PARM	Yes

Table 71. Parameters for TRACE CT for Write

Parameter	Allowed specification on TRACE CT for Write
WTRSTART or WTRSTOP	One is required if a writer is being used

Table 72. Parameters on REPLY for Trace

Parameter	Allowed specification on REPLY for Trace
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter in the CTIGLZxx parmlib member and reply for a TRACE command are in the following table.

Table 73. Values for the OPTIONS parameter in the CTIGLZxx parmlib member

Value	Meaning
ALL	Trace everything
DISK	Trace Disk events
NET	Trace Network events
SIE	Trace SIE events
MIN	Trace events related to zCX component recovery, abnormal conditions, and other non-mainline paths.

Format the trace with an IPCS CTRACE COMP(SYSGLZ) SUB((ASID(xxxx))) subcommand.

SYSGRS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

The following summarizes information for requesting a SYSGRS component trace for global resource serialization.

Information	For SYSGRS:
Parmlib member	CTnGRSxx; default member: CTIGRS00 specified in GRSCNF00 member
Default tracing	Yes, if global resource serialization **ring** is active; CONTROL and MONITOR options
Default tracing	Yes, if global resource serialization **star** is active; CONTROL1, CONTROL2, SIGNAL0 and MONITOR options
Trace request OPTIONS parameter	In CTnGRSxx and REPLY for TRACE command
Buffer	• Default: 16MB • Range: 128KB - 2047MB (System rounds size up to nearest 64KB boundary.) • Size set by: CTnGRSxx member • Change size after IPL: Yes, when restarting a trace after stopping it • Location: In the GRS address space above the bar.
Trace records location	Address-space buffer, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	“SYSGRS component trace” on page 407
Trace format OPTIONS parameter	FLOW, CONTROL, MONITOR, REQUEST, SIGNAL, and RSA. See “OPTIONS parameter” on page 408 for details on sub-options.

Requesting a SYSGRS trace

Specify options for requesting a SYSGRS component trace on a CTnGRSxx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSGRS tracing while the trace is running.

CTnGRSxx parmlib member

The following table indicates the parameters you can specify on a CTnGRSxx parmlib member.

Parameters	Allowed on CTnGRSxx?
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No

Parameters	Allowed on CTnGRSxx?
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM-supplied GRSCNF00 parmlib member specifies CTIGRS00 as the default.

IBM-supplied CTIGRS00 parmlib member:

- Specifies that GRS tracing is begun at IPL
- The parmlib member contains:

  TRACEOPTS OFF
  This parameter turns off all SYSGRS tracing options except for the minimum options (MINOPS).
- See the “Specify buffers” on page 353 section for information about the buffer size default and possible sizes.

IBM recommends that you use the CTIGRS00 parmlib member, unless the IBM Support Center requests different tracing for global resource serialization.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON, nnnnK, nnnnM, or OFF	One is required. The buffer size can be changed only when the trace is OFF or the trace is ON.
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

You can change options while a SYSGRS trace is running. However, to change the buffer size, you have to stop the trace and restart it with the new buffer size.

OPTIONS parameter

The values for the OPTIONS parameter for the CTnGRSxx parmlib member and reply for a TRACE command are listed below. The sub-options on the CONTROL, REQUEST, MONITOR, SIGNAL and FLOW, allow you to refine the set of events traced for the major option. When you select the major option, all events pertaining to that option are traced. However, you can select one or more of the sub-options instead of the major option and thus limit the trace to only those events included in the sub-options specified. A major option, such as MONITOR, and all of its sub-options (in this case MONITOR0, MONITOR1, and MONITOR2 through MONITORF) is referred to as an option group. In alphabetical order the values for the OPTIONS parameter are:

CONTROL

Traces unusual events and events related to the establishment, modification, or termination of the control structure needed for processing such as:

- Dynamic RNL changes
- Error events
- XCF services used when setting up for processing

When you specify CONTROL, all of the following sub-options are traced.

CONTROL0

Traces dynamic RNL changes only.

CONTROL1

Traces events related to the establishment of or termination of membership in the global resource serialization group connection to the global resource serialization coupling facility structures.

CONTROL2

Traces global resource serialization recovery processing only.

CONTROL3

Traces global resource serialization resource manager events for abnormal task and ASID termination only.

CONTROL4-CONTROLE

Reserved for IBM use.

CONTROLF

Traces all other unusual events not included in sub-options CONTROL0 through CONTROL3.

FLOW

Traces the flow of control from one entry point to another.

FLOW0

Traces GRS Star system server processing only.

FLOW1

Traces GQSCAN processing only.

FLOW2

Traces cross-system communications processing only.

FLOW3

Traces command processing only.

FLOW4

Traces storage manager services only.

FLOW5

Traces coupling facility processing only.

FLOW6

Traces initialization processing only.

FLOW7

Traces contention monitor processing only.

FLOW8

Traces general ENQ/DEQ processing only.

FLOW9

Traces entry to GQSCAN/ISGQUERY only.

FLOWA

Traces GRS Latch Manager processing only.

FLOWB-FLOWD

Reserved for IBM use.

FLOWE
Activates extended tracing for the GRS Storage Manager. Do not turn on this option without direction from IBM Service.

FLOWF
Reserved for IBM use.

Monitor
Traces events for selected global resource serialization invocations of monitoring and communication services provided by other components.

MONITOR0
Traces use of XES services.

MONITOR1
Traces use of XCF services.

MONITOR2-MONITORF
Reserved for IBM use.

REQUEST
Traces events for global ENQ, DEQ, GQSCAN, and RESERVE macro requests, and GRS command processing.

REQUEST0
Traces ENQ/RESERVE requests only.

REQUEST1
Traces DEQ requests only.

REQUEST2
Traces GQSCAN only.

REQUEST3
Traces IXLLOCK only.

REQUEST4
Traces command processing only.

REQUEST5
Traces lock structure (ISGLOCK) rebuild processing only.

REQUEST6-REQUESTF
Reserved for IBM use.

RSA
Traces events for RSA control information.

SIGNAL
Traces events for selected global resource serialization invocations of cross-system coupling facility (XCF) signalling service processing.

SIGNAL0
Traces migration signals only.

SIGNAL1
Traces GQSCAN signals only.

SIGNAL2
Traces ENQ/DEQ signals, including RNL change signals only.

SIGNAL3
Traces contention monitor signals only.

SIGNAL4-SIGNALF
Reserved for IBM use.

Examples of requesting SYSGRS traces

- Example 1: CTnGRSxx member

The member requests CONTROL, MONITOR, and RSA options and doubles the default buffer size.

```
TRACEOPTS
   ON
   OPTIONS('CONTROL','MONITOR','RSA')
   BUFSIZE(32M)
```

- Example 2: TRACE command

  The example requests a trace of CONTROL, MONITOR, and REQUEST trace events.

```
trace ct,on,comp=sysgrs
* 17 ITT006A ...
reply 17,options=(control,monitor,request),end
```

Formatting a SYSGRS trace

Format the trace with an IPCS CTRACE COMP(SYSGRS) subcommand. It is possible to use the OPTIONS subcommand for COMP (SYSGRS) with values of FLOW, CONTROL, REQUEST, MONITOR, SIGNAL, and RSA for filtering.

Output from a SYSGRS trace

Figure 127 on page 411 is an example of SYSGRS component trace records formatted with the CTRACE COMP(SYSGRS) SHORT subcommand.

```
                    SYSGRS COMPONENT TRACE SHORT FORMAT

  GRPXCNTL  00000030  13:05:59.858746  GROUP EXIT IN CONTROL
  DISRUPT   0000000E  13:05:59.858780  RING DISRUPTION TRIGGERED
  MAINRF1   0000000B  13:05:59.860196  MAIN RING FAILURE
  CEXBCI1   0000000C  13:05:59.860324  CONTROL EXITED FROM ISGBCI
  SETUS     00000035  13:06:00.031243  CALL TO XCF SETUS SERVICE
  GRPXCNTL  00000030  13:06:00.141669  GROUP EXIT IN CONTROL
  STAXIN    00000033  13:06:00.160559  STATUS EXIT IN CONTROL
  .
  .
  .
```

Figure 127. Example: SYSGRS component trace records formatted with CTRACE COMP(SYSGRS) SHORT

Figure 128 on page 412 is an example of SYSGRS component trace records formatted with the CTRACE COMP(SYSGRS) TALLY subcommand.

```
COMPONENT TRACE TALLY REPORT
COMP(SYSGRS)
    TRACE ENTRY COUNTS AND AVERAGE INTERVALS (IN MICROSECONDS)

FMTID     COUNT     INTERVAL      MNEMONIC  DESCRIBE
--------  --------  ------------  --------  --------------------------------
00000001         0                RSAIN1    RSA has no CMD area no QWB data
00000002         0                RSAIN2    RSA has QWB data but no CMD area
00000003         0                RSAIN3    RSA has CMD area but no QWB data
00000004         0                RSAIN4    RSA has CMD area and QWB data
00000005         0                RSAOUT1   RSA has no CMD area no QWB data
00000006       898     2,037,854  RSAOUT2   RSA has QWB data but no CMD area
00000007         0                RSAOUT3   RSA has CMD area but no QWB data
00000008         8   149,924,356  RSAOUT4   RSA has CMD area and QWB data
00000009         5   328,792,726  INVBBE1   ISGBBE - QMERGE
0000000A         0                INVBBE2   ISGBBE - not QMERGE
0000000B         4   490,847,959  MAINRF1   Main Ring Failure
0000000C        13   149,346,135  CEXBCI1   Control Exited from ISGBCI
0000000D         0                CLNQSCD   Cleanup after Quiesced from ring
.                                 .         .
.                                 .         .
.                                 .         .

00000058         0                UEVENT8   Recovery during write
00000059         0                UEVENT9   Recovery during read
0000005A         0                UEVENTA   Remote discarded message

Total trace entries:    1,120
```

Figure 128. Example: SYSGRS component trace records formatted with CTRACE COMP(SYSGRS) TALLY

SYSHZS component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSHZS component trace for IBM Health Checker for z/OS.

Information	**For SYSHZS:**
Parmlib member	CTIHZS00
Default tracing	Yes
Trace request OPTIONS parameter	In CTIHZS00 and REPLY for TRACE command
Buffer	• Default: 4MB • Range: 16KB - 4MB (System rounds size up to nearest 64KB boundary.) • Size set by: CTIHZS00 member • Change size after IPL: Yes, when restarting a trace • Location: In the IBM Health Checker for z/OS address space
Trace records location	Address-space buffer
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSHZS) - see "Output from a SYSHZS trace" on page 414
Trace format OPTIONS parameter	None

Requesting a SYSHZS trace

Specify options for requesting a SYSHZS component trace on a CTIHZS00 parmlib member or on the reply for a TRACE CT command.

You can change options for SYSHZS tracing while the trace is running.

CTIHZS00 parmlib member

The following table indicates the parameters you can specify on a CTIHZS00 parmlib member.

Parameters	Allowed on CTIHZS00
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM supplies the CTIHZS00 parmlib member, which specifies the IBM Health Checker for z/OS tracing begun at IPL. The contents of CTIHZS00 are:

```
TRACEOPTS
         OFF
```

This parameter turns off all SYSHZS tracing options.

If additional SYSHZS tracing options are turned on, additional buffer space might be required.

The default trace buffer size is 4MB.

IBM recommends that you use the CTIHZS00 parmlib member, unless the IBM Support Center requests different tracing for IBM Health Checker for z/OS.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON, nnnnK, nnnnM, or OFF	One is required
COMP	Required
SUB	No
PARM	No

Parameters	Allowed on TRACE CT for writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

You can change options while a SYSHZS trace is running. However, to change the buffer size, you have to stop the trace and restart it with the new buffer size.

OPTIONS parameter

The values for the OPTIONS parameter for the CTnHZSxx parmlib member and reply for a TRACE command are listed below:

CHECKS
Traces unusual events and events related to IBM Health Checker for z/OS checks.

COMMANDS
Traces information about the F *hzsproc* command, the HZSPRMxx parmlib member and the HZSCHECK macro.

STORAGE
Traces information about storage used by the IBM Health Checker for z/OS address space.

LOGGER
Traces information about the IBM Health Checker for z/OS log stream.

MISC
Traces miscellaneous information.

ALL
Traces all events for IBM Health Checker for z/OS. ALL is the default.

Examples of requesting SYSHZS traces

- Example 1: CTIHZS00 member

  The member requests ALL IBM Health Checker for z/OS component tracing:

```
TRACEOPTS
   ON
   OPTIONS('ALL')
   BUFSIZE(4M)
```

- Example 2: TRACE command

  The example requests a trace of ALL trace events.

```
trace ct,on,comp=syshzs
* 17 ITT006A ...
reply 17,options=(all),end
```

Formatting a SYSHZS trace

Format the trace with an IPCS CTRACE COMP(SYSHZS) FULL subcommand.

Output from a SYSHZS trace

Figure 129 on page 415 is an example of SYSHZS component trace records formatted with the CTRACE COMP(SYSHZS) FULL subcommand.

```
B7VB0038  CHECKS    00000001  21:23:37.960765

    ASID..0028    ModID..0201    TCB..004E3B58    Stack..7F11A000
    Event..Candidat  Function..N/A
    Owner..IBMRSM
    Name...RSM_MEMLIMIT
    PQEAddr..7FFD4000  Result..00000000  Diag..00000000  00000000

B7VB0038  STORAGE   00000003  21:23:37.960793  GET/FREE

    ASID..0028    ModID..0105    TCB..004E3B58    Stack..7F11A000
    Oper..Get       Type..PQE       CModID..0201  Area@..7FFD5000
 IPCS OUTPUT STREAM ----------------------------------- FOUND: LINE 5367 COL 12
 Command ===>                                                 SCROLL ===> CSR
   B7VB0038  STORAGE   00000003  21:24:09.757964  GET/FREE

       ASID..0028    ModID..0105    TCB..004E3B58    Stack..7EEDA000
       Oper..Get       Type..CMDI      CModID..0116  Area@..7FFE0000

   B7VB0038  COMMANDS  00000002  21:24:09.757966

       ASID..0028    ModID..0116    TCB..004E3B58    Stack..7EEDA000
       Command..Display
       Keywords..00000100  00000010  00000000  00000000
       Owner..................
       Name...................................
       PolStmt..N/A

   B7VB0038  STORAGE   00000003  21:24:09.762583  GET/FREE

       ASID..0028    ModID..0105    TCB..004E38C8    Stack..7F13E000
       Oper..Free      Type..CMDI      CModID..0703  Area@..7FFE0000

   B7VB0038  STORAGE   00000003  21:28:25.209146  GET/FREE

       ASID..0028    ModID..0105    TCB..004E3B58    Stack..7EEDA000
       Oper..Get       Type..CMDI      CModID..0116  Area@..7FFE0000

   B7VB0038  COMMANDS  00000002  21:28:25.209148

       ASID..0028    ModID..0116    TCB..004E3B58    Stack..7EEDA000
       Command..Display
       Keywords..00040100  00000000  00000000  00000000
       Owner..................
       Name...................................
       PolStmt..N/A

   B7VB0038  MISC      00000004  21:28:41.204631

       ASID..0028    ModID..0304    TCB..004E38C8    Stack..7F13E000
      +0000  D9C5C3E5  C8E9E2E3  D2C4C9E2  840E0000  | RECVHZSTKDISd... |
      +0010  00000028
```

Figure 129. Example: SYSHZS component trace records formatted with CTRACE COMP(SYSHZS) FULL

SYSIEAVX component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSIEAVX component trace for PC/Auth.

.

Information	For SYSIEAVX:
Parmlib member	CTIIEAVX, but no changes to that member are supported
Default tracing	Yes; always on
Trace request OPTIONS parameter	None

Information	For SYSIEAVX:
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: Above 2G Common storage
Trace records location	Address-space buffer
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSIEAVX)
Trace format OPTIONS parameter	None

Requesting a SYSIEAVX trace

No actions are needed to request it.

Formatting a SYSIEAVX trace

Format the trace with an IPCS CTRACE COMP(SYSIEAVX) subcommand. The subcommand has no OPTIONS values.

Output from a SYSIEAVX trace

Figure 130 on page 416 is an example of IEAVX component trace records formatted with a CTRACE COMP(SYSIEAVX) FULL subcommand. It shows formatted exception records from the trace buffers.

```
                    IEAVX COMPONENT TRACE FULL FORMAT

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY1       Add       00000001  13:30:06.093525  ALESERV Add
    HASN..... 002E      PASN..... 002E      WU....... 005F8588
    ALET..... 002B0003  ALE...... 002B0001  00000000  1584BF00  00000001
    RC....... 00        PSWA..... 00000000  000070E0
    STOKEN... 80001201  00000012            OPTIONS.. 01000000  CEAX..... 0000
SY1       Delete    00000003  14:50:03.167286  ALESERV Delete
    HASN..... 002E      PASN..... 002E      WU....... 005F8588
    ALET..... 002B0003  ALE...... 002B0001  00000000  1584BF00  00000001
    RC....... 00        PSWA..... 00000000  0000711C
    STOKEN... 00000000  00000000            OPTIONS.. 03000000  CEAX..... 0000
SY1       AddPASN   00000002  14:50:03.167287  ALESERV AddPASN
    HASN..... 002E      PASN..... 002E      WU....... 005F8588
    ALET..... 002C0003  ALE...... 002C0000  00000000  1FF44B80  00000001
    RC....... 00        PSWA..... 00000000  00007150
    STOKEN... 00000000  00000000            OPTIONS.. 02000000  CEAX..... 0000
```

Figure 130. Example: SYSIEAVX component trace records formatted with CTRACE COMP(SYSIEAVX) FULL

The fields in the report are:

The Function
: Add ... 00000001 ... ALESERV Add

  AddPASN ... 0000002 ... ALESERV AddPASN

  Delete ... 00000003 ... ALESERV Delete

  ART ... 00000004 ... AR Translation Exception

The time (xx:yy:zz.wwwwww)
: The time when the record was placed into the trace buffer.

HASN..... xxxx
The home address space number.

PASN..... xxxx
The primary address space number.

WU....... xxxxxxxx
Work unit address for TCB, WEB address for SRB.

ALET..... xxxxxxxx
Access List Entry Token (ALET)

ALE...... xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
Access List Entry

RC....... xx
The return code (not relevant when the function is ART).

PSWA..... xxxxxxxx xxxxxxxx
8-byte PSW address of the caller.

STOKEN... xxxxxxxx xxxxxxxx
The space token (STOKEN) for the Add function.

OPTIONS.. xxxxxxxx
ALESERV service options in hexadecimal (not relevant when the function is ART).

CEAX..... xxxx
The caller's extended authorization index (EAX) (not relevant when the function is ART).

SYSIEFAL component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSIEFAL component trace for Allocation.

Information	For SYSIEFAL:
Parmlib member	CTIIEFxx; default member: CTIIEFAL
Default tracing	Yes; FLOW0, FLOW1, FLOW6, FLOWF, DATA, CONTROL0, CONTROL1, CONTROL6, CONTROLF, SERIAL1, and SERIALF options
Trace request OPTIONS parameter	In CTIIEFxx or REPLY for TRACE command
Buffer	• Default: 8M • Range: 256KB - 64MB • Size set by: CTIIEFxx member • Change size after IPL: Yes • Location: In the component address space
Trace records location	Address-space buffer
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSIEFAL)
Trace format OPTIONS parameter	Yes; FLOW, CONTROL, SERIAL, DATA, MISC, and ERROR

Requesting a SYSIEFAL trace

Specify options for requesting a SYSIEFAL component trace on a CTIIEFxx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSIEFAL tracing while the trace is running.

CTIIEFxx parmlib member

The following table indicates the parameters you can specify on a CTIIEFxx parmlib member.

Parameters	Allowed on CTIIEFxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM supplies the CTIIEFAL parmlib member, which specifies the Allocation tracing begun at IPL. The contents of CTIIEFAL are:

```
TRACEOPTS
       ON
OPTIONS(
        'FLOW0'
       ,'FLOW1'
       ,'FLOW6'
       ,'FLOWF'
       ,'SERIAL1'
       ,'SERIALF'
       ,'DATA'
       ,'CONTROL0'
       ,'CONTROL1'
       ,'CONTROL6'
       ,'CONTROLF'
       )
BUFSIZE(8M)
```

If additional SYSIEFAL tracing options are turned on, additional buffer space may be required.

The default trace buffer size is 8M.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON, nnnnK, nnnnM, or OFF	One is required
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

You can change options while a SYSIEFAL trace is running. However, to change the buffer size immediately, you have to stop the trace and restart it with the new buffer size. If the trace is not stopped and restarted, the buffer size will be changed when the current set of buffers has filled up and a new set is acquired.

OPTIONS parameter

The values for the OPTIONS parameter for the CTIIEFxx parmlib member and reply for a TRACE command are listed. The suboptions on the CONTROL, DATA, FLOW, and SERIAL, allow you to refine the set of events traced for the major option. When you select the major option, all events pertaining to that option are traced. However, you can select one or more of the suboptions instead of the major option and thus limit the trace to only those events included in the suboptions specified. A major option, such as DATA, and all of its suboptions (in this case DATA0, DATA1, and so forth) is referred to as an option group. In alphabetical order the values for the OPTIONS parameter are:

CONTROL
Traces the control within a module.

CONTROL0
Traces common allocation processing only.

CONTROL1
Traces allocation services 1 processing only.

CONTROL2
Traces unallocation processing only.

CONTROL3
Traces volume mount and verify processing only.

CONTROL4
Traces assign/unassign processing only.

CONTROL5
Traces allocation services 2 processing only.

CONTROL6
Traces allocation device management processing only.

CONTROL7
Traces Dynamic Allocation processing only.

CONTROL8
Reserved for IBM use.

CONTROL9
Traces JFCB Houskeeping processing only.

CONTROLA
Traces TCTIOT management processing only.

CONTROLB-CONTROLE
Reserved for IBM use.

CONTROLF
Traces unexpected or unusual control within a module.

DATA
Traces when data is processed or changed.

DATA0
Traces when an ATSP device type array is being processed only.

DATA1
Traces when an ATSP device array is being processed only.

DATA2
Traces when an IGDE is going through XCF messaging only.

DATA3
Traces when an IGDE goes through a state change only.

DATA4
Traces UCB changes only.

DATA5
Traces ENQ changes for DDR SWAP processing only.

DATA6
Traces device management data only.

DATA7
Traces Dynamic Allocation processing only.

DATA8
Traces when an allocation occurs only.

DATA9
Traces JFCB Housekeeping data only.

DATAA
Traces TCTIOT management data only.

DATAB-DATAE
Reserved for IBM use.

DATAF
Traces when data is unexpected or unusual.

FLOW
Traces the flow of control from one entry point to another.

FLOW0
Traces common allocation processing only.

FLOW1
Traces allocation services 1 processing only.

FLOW2
Traces unallocation processing only.

FLOW3
Traces volume mount and verify processing only.

FLOW4
Traces assign/unassign processing only.

FLOW5
Traces allocation services 2 processing only.

FLOW6
Traces device management data only.

FLOW7
Traces Dynamic Allocation processing only.

FLOW8
Reserved for IBM use.

FLOW9
Traces JFCB Housekeeping processing only.

FLOWA
Traces TCTIOT management processing only.

FLOWB-FLOWE
Reserved for IBM use.

FLOWF
Traces unexpected or unusual flow from one entry point to another.

SERIAL
Traces serialization events.

SERIAL0
Traces locking (SETLOCK) serialization events only.

SERIAL1
Traces ENQ/DEQ serialization events only.

SERIAL2
Traces latch manager serialization events only.

SERIAL3
Traces compare and swap serialization events only.

SERIAL4
Traces SYSDSN ENQ/DEQ serialization events only.

SERIAL5–SERIALE
Reserved for IBM use.

SERIALF
Traces unexpected or unusual serialization events.

Examples of requesting SYSIEFAL traces

- Example 1: CTIIEFxx member

  The member requests FLOW and DATA options and requests a buffer size of 8 megabytes.

```
TRACEOPTS
   ON
   OPTIONS('FLOW','DATA')
   BUFSIZE(8M)
```

- Example 2: TRACE command

  The example requests a trace of DATA1 and CONTROL1 trace events.

```
trace ct,on,comp=sysiefal
* 17 ITT006A ...
reply 17,options=(data1,control1),end
```

Formatting a SYSIEFAL trace

Format the trace with an IPCS CTRACE COMP(SYSIEFAL) subcommand. You can filter trace records for COMP (SYSIEFAL) by using the OPTIONS subcommand. Specify one or more of the following values on the OPTIONS subcommand:

- FLOW
- DATA
- CONTROL
- SERIAL
- ERROR
- MISC

Events in the ERROR and MISC trace groups are always recorded, and are always displayed regardless of what filters are requested.

Output from a SYSIEFAL trace

Figure 131 on page 422is an example of SYSIEFAL component trace records formatted with the CTRACE COMP(SYSIEFAL) SHORT subcommand.

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSIEFAL)
**** 09/13/2001

SYSNAME MNEMONIC ENTRY ID   TIME STAMP    DESCRIPTION
------- -------- -------- --------------- -------------
N67     FLOW0    00000100 15:40:34.104633 Common Allocation Flow
N67     CONTROL0 00000200 15:40:34.104639 Common Allocation Control
N67     CONTROL0 00000200 15:40:34.104693 Common Allocation Control
N67     CONTROL0 00000200 15:40:34.104852 Common Allocation Control
N67     FLOW0    00000100 15:40:34.104859 Common Allocation Flow
N67     FLOW0    00000100 15:40:34.106631 Common Allocation Flow
N67     FLOW0    00000100 15:40:34.125582 Common Allocation Flow
.
.
.
```

Figure 131. Example: SYSIEFAL component trace records formatted with CTRACE COMP(SYSIEFAL) SHORT

Figure 132 on page 422 is an example of SYSIEFAL component trace records formatted with the CTRACE COMP(SYSIEFAL) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSIEFAL)
**** 09/13/2001

SYSNAME MNEMONIC ENTRY ID   TIME STAMP    DESCRIPTION
------- -------- -------- --------------- -------------
N67     FLOW0    00000100 15:40:47.306228 Common Allocation Flow

   ASID..007A  TCB..008E1980  MODNAME..IEFAB492  JOBNAME..T015067

    EBCDIC Data...
    ENTR

N67     FLOW0    00000100 15:40:47.306231 Common Allocation Flow

   ASID..007A  TCB..008E1980  MODNAME..IEFAB492  JOBNAME..T015067

    EBCDIC Data...
    EXIT
    Hex Data...
    00000000                                | ....            |

.
.
.
```

Figure 132. Example: SYSIEFAL component trace records formatted with CTRACE COMP(SYSIEFAL) FULL

SYSIOS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

IOS component trace is described by the following attributes:

- Trace buffers reside in common ESQA Subpool 248. Size is controlled by the TRACE CT operator command. As the buffers become full they are copied to a private IOS data space. For information on specifying an IOS data space size, see "OPTIONS parameter" on page 425.
- Minimal and unexpected event tracing is activated during IOS NIP processing.
- Component trace buffers externalized through:
  - DUMP or SLIP operator command when the IOS address space is requested to be dumped.
  - SVC dumps issued by IOS, dynamic device reconfiguration (DDR), or execute channel program (EXCP) component recovery.
  - MVS component trace (CTRACE) external writer
- Trace options revert to minimal event and exception tracing when the operator turns the trace off.

The following summarizes information for requesting a SYSIOS component trace for IOS:

Information	For SYSIOS:
Parmlib member	CTnIOSxx specified in the IECIOSxx member through the CTRACE(CTnIOSxx) statement. Default member: None
Default tracing	Yes, activated during IOS NIP processing.
Trace request OPTIONS parameter	In CTnIOSxx or REPLY for TRACE command
Buffer	• Default: 324KB • Range: 324KB - 1.5M • Size set by: CTnIOSxx member or REPLY for TRACE command • Change size after IPL: Yes, when component trace (CTRACE) is active • Location: Common ESQA subpool 248 and SYSIOS private IOS data space.
Trace records location	Common ESQA subpool 248 and SYSIOS private IOS data space and trace data set. By DUMP or SLIP command when the IOS address space is requested to be dumped. In the REPLY for the DUMP command, specify the IOS address space to be dumped. By SLIP command. By the component during SVC dumps issued by IOS, DDR, or EXCP component recovery.
Trace formatting by IPCS	CTRACE COMP(SYSIOS)
Trace format OPTIONS parameter	No

The areas of IOS traced as part of minimal and unexpected event tracing include:

- Dynamic Configuration Changes
- Parallel Access Volume (PAV) Processing
- Dynamic Channel Path Management (DCM) Processing
- Unconditional Reserve (U/R) Recovery Processing
- Channel Subsystem Call (CHSC) Processing
- Channel Report Word (CRW) Processing
- Missing Interrupt Handler (MIH) Recovery Processing
- Control Unit Initiated Reconfiguration (C.U.I.R.) Request Processing
- Dynamic Pathing Support (DPS) Validation
- Dynamic Device Reconfiguration (DDR) Processing

- Self-Description Processing
- PCIE Initialization and Exceptional Conditions Processing

Note: Additional areas are traced when OPTIONS are set for IOS component trace. See "OPTIONS parameter" on page 425.

Requesting a SYSIOS trace

No actions are required to request a SYSIOS trace. Minimal and unexpected event tracing is activated during IOS NIP processing and is always active, even when TRACEOPS OFF is specified in CTnIOSxx.

Some functions CTRACE more data than others; for example, when z/OS Hyperswap is enabled, SYSIOS CTRACE entry usage will increase. In addition to this tracing, the user can request a SYSIOS trace using specific options by doing the following:

- Using a CTnIOSxx SYS1.PARMLIB member during NIP processing by specifying the CTRACE(CTnIOSxx) statement in the IECIOSxx SYS1.PARMLIB member.
- Using a CTnIOSxx SYS1.PARMLIB member after NIP by issuing the TRACE system command.
- Using the TRACE system command and specifying the options in response to the prompts that CTRACE provides.

CTnIOSxx parmlib member

The following table indicates the parameters you can specify in a CTnIOSxx parmlib member.

Parameters	Allowed on CTnIOSxx?
ON or OFF	One is required
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

If additional SYSIOS tracing options are turned on, additional buffer space might be required.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON or OFF	One is required
nnnnK, nnnnM	Yes
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

The WTR and WTRSTART parameters can be used in a parmlib member specified on the TRACE CT command. The parameters cannot be specified in a parmlib member that is read at IPL because the external writer is not available when the IOS component is defined.

OPTIONS parameter

The values for the OPTIONS parameter for the CTnIOSxx parmlib member and reply for a TRACE command are listed below.

DCM
Traces events relating to Dynamic Channel Path Management.

Note: When DCM is active and this option is set, large amounts of trace data will be recorded. Users may wish to consider using an external writer when this option is set.

Traces the results of the cancel subchannel (XSCH) instruction.

EXTEND
Traces all functions that are attached in the IOS address space.

Note: Some functions attached in the IOS address space are traced during minimum or unexpected event tracing.

Traces the results of the cancel subchannel (XSCH) instruction.

STORAGE
Traces events related to IOS or EXCP storage management. When the STORAGE option is specified, the NOFILTER option or ASID/JOBNAME keywords must also be set.

CAPTURE
Traces the capturing and uncapturing of UCBs. When the CAPTURE option is specified, the NOFILTER option or ASID/JOBNAME keywords must also be set.

NOFILTER
Allows the STORAGE option to be set without requiring ASID or JOBNAME filtering.

DS=*nnnn*
Allows the user to tailor the IOS Trace Data Space where *nnnn* is the data space size in megabytes.

Note:

1. *nnnn* must be a valid decimal digit within the range of 1-1024.
2. This option can only be specified once at IPL time and cannot be modified using the TRACE CT command.
3. The default size for the IOS Trace Data Space is 512M. This can require additional auxiliary storage on systems with a small amount of available auxiliary storage. Please refer to “Decide where to collect the trace records” on page 357 for information about auxiliary storage for CTRACE data space buffers. Since some options such as STORAGE and DCM will cause more CTRACE entries to be recorded, the IOS Trace Data Set may fill up more rapidly than if none of these options is specified. Users who do not have enough auxiliary storage capacity to handle a full data space may choose to use the DS=*nnnn* option to set up a smaller IOS Trace Data Space. Similarly, users who do have enough auxiliary storage capacity to handle a full data space may choose to use

the DS=*nnnn* option to set up a larger IOS Trace Data Space. Doing this will prevent potentially valuable debug information from being lost due to wrapping. Note that the number of records contained in a trace data set are highly variable and dependent upon trace settings and system usage.

4. If the DS=*nnnn* option is specified more than once, the request is rejected and the following message is issued:

```
IOS622I IOS COMPONENT TRACE OPTION xxxxxxxx IS NOT VALID -
        THE TRACE DATA SPACE SIZE HAS ALREADY BEEN SET
        FOR THIS IPL
```

5. If the size specified is not valid, then the request is rejected, the default IOS Trace Data Space size is used, and the following message is issued:

```
IOS622I IOS COMPONENT TRACE OPTION xxxxxxxx IS NOT VALID -
        THE REQUESTED SIZE FOR THE TRACE DATA SPACE IS
        INCORRECT
```

HPAV
Traces events relating to HyperPAV bind and unbind activity. This option may cause large amounts of trace data to be recorded. It should be utilized as a debug tool when needed and not enabled for normal operations.

PCIE
Traces events related to PCIE activity.

Examples of Requesting SYSIOS traces

- Example 1: CTnIOSxx SYS1.PARMLIB member

  This SYS1.PARMLIB member sets the STORAGE option using JOBNAME filtering for JOB001 and sets the buffer size to 600K.

```
TRACEOPTS
   ON
   BUFSIZE(600K)
   OPTIONS(STORAGE)
   JOBNAME(JOB001)
```

Formatting a SYSIOS trace

IPCS CTRACE formatting services can be used to format the contents of the CTRACE trace entries. Format the trace with the following IPCS subcommand:

```
IPCS CTRACE COMP(SYSIOS) SUMMARY|FULL|SHORT|TALLY
```

SUMMARY
Shows the trace entry header and the formatted data for each trace entry.

FULL
Shows the trace entry header and the unformatted (hex) data for each trace entry.

SHORT
Shows the trace entry header for each trace entry.

TALLY
Shows each trace entry and how many times they were traced.

The subcommand has no options.

CTRACE COMP(SYSIOS) subcommand output

The following is an example of SYSIOS component trace records formatted with the CTRACE COMP(SYSIOS) SUMMARY option.

Example: SYSIOS component trace records formatted with CTRACE COMP(SYSIOS) SUMMARY option

```
CTRACE COMP(SYSIOS) SUMMARY
**** 03/28/1996
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

S530      MIH       00080001  19:42:42.220726  MIH Recovery Halt/Clear Block

  Trace Record Function: MIH
   MIH condition detected: Start Pending

  Record ID: IOSDMHCB.MHCBSHIB              Length: 0034

    +0000  00F168E8  289B0180  F00000F0  0027FFF0  | .1.Y....0..0...0 |
    +0010  3868B8E8  FFFFFFFF  00000000  00804400  | ...Y............ |
    +0020  3885DA88  00000000  00000000  | .e.h............ |
    +0030  00000000                                | ....             |

  Record ID: IOSDMHCB.MHCBUCB               Length: 0080

    +0000  00000940  20200000  01E13480  00000000  | ... ............ |
    +0010  00000000  00FCC8B4  00F16890  00000000  | ......H..1...... |
    +0020  00040040  00FC3800  00FC3100  0001001F  | ... ............ |
    +0030  28980027  F00080F0  3868B8E8  FFFFFFFF  | .q..0..0...Y.... |
    +0040  01000040  00000001  00000041  00FC3800  | ... ............ |
    +0050  0088FF84  01800800  00F16968  00F1F8F0  | .h.d.....1...180 |
    +0060  80062024  00F168C0  00010100  F1F8F0D7  | .....1.{....180P |
    +0070  C1D21000  00000000  00000000  | AK.............. |

  Record ID: IOSDMHCB.MHCBTMJB              Length: 0018

    +0000  ACA29DF9  49B6E706  F0F0F0F0  F1F5F0F0  | .s.9..X.00001500 |
    +0010  5CD4C1E2  E3C5D95C                      | *MASTER*         |

  Record ID: IOSDMHCB.MHCBADDL              Length: 0008

    +0000  BCBC2010  01000040                      | .......          |

S530      U/R       000C0001  19:42:52.885939  Unconditional Reserve Sense

  Device: 0180   Channel Path: 38
   U/R Sense issued by: Missing Interrupt Handler
   U/R Sense completion code: 52

S530      U/R       000C0002  19:43:13.927140  Unconditional Reserve I/O

  Device: 0180
   Recovery I/O issued: Reset Allegiance
   Return code from IOSRRRSV: 04

  Record ID: IOSDRESV.RESS                  Length: 0044

    +0000  D9C5E2E2  05000052  00800000  00000000  | RESS............ |
    +0010  00000000  00000000  | ................ |
    +0020  00000000  00000000  | ................ |
    +0030  00000000  F0000000  80000500  00010760  | ....0..........- |
    +0040  4000C000                                |  .{.             |


S530      U/R       000C0002  17:40:50.336526  Unconditional Reserve I/O

  Device: 0160   Channel Path: 65
   Recovery I/O issued: Sense Path Group ID
   Return code from IOSRRRSV: 00

  Record ID: IOSDICCW.SNID                  Length: 000C

    +0000  C0000111  13214381  AC9EB2D4            | {......a...M     |
S530      U/R       000C0002  22:51:49.169701  Unconditional Reserve I/O

  Device: 0180   Channel Path: 0B
   Recovery I/O issued: Reset Allegiance
   Return code from IOSRRRSV: 00

  Record ID: IOSDICCW.RSTA                  Length: 0020

    +0000  80000000  00000000  00000000  | ................ |
    +0010  00000000  00000000  | ................ |
```

```
S530      U/R       000C0002  19:43:39.951626  Unconditional Reserve I/O

 Device: 0180   Channel Path: 38
  Recovery I/O issued: Unconditional Reserve with release
  Return code from IOSRRRSV: 00
```

SYSJES component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSJES component trace for the JES common coupling services component, also known as JES XCF.

Information	For SYSJES:
Parmlib member	CTnJESxx. Default members: CTIJES01, CTIJES02, CTIJES03, CTIJES04
Default tracing	Yes; detailed tracing for sublevel FLOW; minimal tracing of unexpected events for sublevels MSGTRC, USRXIT, XCFEVT
Trace request OPTIONS parameter	None
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: In the component address space
Trace records location	Address-space buffer, trace data set
Request of SVC dump	By the component
Trace formatting by IPCS	CTRACE COMP(SYSJES)
Trace format OPTIONS parameter	Yes

Note: To get a complete dump for JES XCF, request also the JES and JES XCF address spaces and data spaces, plus SDATA options RGN, SQA, and CSA.

SYSJES tracing is started during initialization. SYSJES contains 4 sublevel traces, which run concurrently. Each sublevel must be started individually. The sublevels are:

- **MSGTRC, message tracing**: MSGTRC records message data sent by the IXZXIXSM service. The default tracing for this sublevel is minimal tracing of unexpected events only. You can optionally start and stop detailed MSGTRC tracing. Use the data from this sublevel in conjunction with USRXIT trace data to get information about message data modified by installation exits IXZXIT01 or IXZXIT02.
- **USRXIT, installation exit tracing**: USRXIT records the exit parameter list (SPELL) passed to and returned from installation exits IXZXIT01, IXZXIT02, and IXZXIT03 processing. The default tracing for this sublevel is minimal tracing of unexpected events only. You can optionally start and stop detailed USRXIT tracing. Use the data from this sublevel in conjunction with MSGTRC trace data to get information about message processing through installation exits IXZXIT01, IXZXIT02, and IXZXIT03.
- **FLOW, module footprint tracing**: FLOW records messages and events as they flow through the JES common coupling services component. By default, FLOW is always active and produces detailed tracing.

  Note: IBM recommends that this trace always remain active to record diagnostic data such as errors, system state changes, and processing events.

- **XCFEVT, system event (SYSEVENT) tracing**: XCFEVT records SYSEVENT data processed by the JES common coupling services component. By default, XCFEVT always produces minimal tracing.

Tracing for SYSJES can run all 4 sublevels concurrently. USRXIT and MSGTRC trace only error events by default; you can turn on detailed tracing for these two sublevels.

Requesting a SYSJES trace

IBM recommends the following when requesting SYSJES TRACING:

- Start and stop the four sublevels for a system all at once in one parmlib member. Request SYSJES component tracing in a CTnJESxx parmlib member which you specify on a TRACE CT command.

  IBM provides two parmlib members, IXZCTION and IXZCTIOF, in SYS1.SAMPLIB as examples of how to start and stop SYSJES sublevels. Copy the members into parmlib, and rename them CTIJESON and CTIJESOF. The CTIJESON parmlib member starts all the sublevels and connects them to the external writer. The CTIJESOF parmlib member stops tracing in all sublevels and disconnects them from the external writer.
- Use the external writer for gathering trace records, because SYSJES tracing produces a large volume of data. Create source JCL for the external writer, using the following guidelines:
  - Code all TRCOUTnn DD statements with a SPACE parameter of at least 10 cylinders to accommodate the volume of SYSJES trace data.
  - For traces larger than 10 cylinders, specify a unique volser for each TRCOUTnn statements if you need to reduce I/O contention on one volume.
  - The data set name defined in the TRCOUT01 DD statement must be unique on each system.
  - Use the IPCS COPYTRC command to merge records from multiple TRCOUTnn DD statements into one data set. See *z/OS MVS IPCS Commands* for information.

  The following example shows an external writer procedure, IXZCTW, that sends SYSJES trace output to trace data sets.

```
//CTWDASD   PROC
//IEFPROC   EXEC  PGM=ITTTRCWR
//SYSPRINT  DD    SYSOUT=A
//TRCOUT01  DD    DSN=SYS1.JESXCF1,VOL=SER=TRACE6,UNIT=DASD,
//          SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02  DD    DSN=SYS1.JESXCF2,VOL=SER=TRACE7,UNIT=DASD,
//          SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
```

  Figure 133. Example: Cataloged procedure for SYSJES

- If you are tracing in a sysplex environment, the data set names on TRCOUTnn DD statements must be unique throughout the sysplex. An ENQUEUE error results if the data set names are not unique.

CTnJESxx parmlib member

The following table indicates the parameters you can specify on a CTnJESxx parmlib member.

Parameters	Allowed on CTnJESxx?
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	No
OPTIONS	No

Parameters	Allowed on CTnJESxx?
SUB	Yes
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON or OFF	One is required
nnnnK or nnnnM	No
COMP	Required
SUB	Yes
PARM	Yes

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	No
JOBNAME	No
OPTIONS	No
WTR	Yes

Examples of requesting SYSJES traces

- Example 1: Start SYSJES tracing with the CTIJESON member

  The following example shows the CTIJESON parmlib member supplied in SYS1.SAMPLIB to start tracing for one system:

  ```
  TRACEOPTS
  WTRSTART(IXZCTW) WRAP
  SUB(MSGTRC)
  ON
  WTR(IXZCTW)
  SUB(USRXIT)
  ON
  WTR(IXZCTW)
  SUB(FLOW)
  ON
  WTR(IXZCTW)
  SUB(XCFEVT)
  ON
  WTR(IXZCTW)
  ```

- Example 2: Stop SYSJES tracing with the CTIJESOF member

  The following example shows the CTIJESOF parmlib member supplied in SYS1.SAMPLIB to stop tracing for one system:

  ```
  TRACEOPTS
  SUB(MSGTRC)
  ```

```
OFF
SUB(USRXIT)
OFF
SUB(FLOW)
OFF
SUB(XCFEVT)
OFF
```

Stop tracing with the following command:

```
TRACE CT,OFF,COMP=SYSJES,PARM=CTIJESOF
```

Then specify the following command to stop the external writer (assuming IXZCTW is the membername of the source JCL for the external writer):

```
TRACE CT,WTRSTOP=IXZCTW
```

Requesting a SYSJES trace for problems during initialization

Use this procedure only when requested to by the IBM Support Center. The procedure requests SYSJES tracing for JES XCF problems occurring during JES subsystem initialization. The procedure consists of using the default parmlib members CTIJES01, CTIJES02, and CTIJES04 to request tracing of these SYSJES sublevels. Note that parmlib member CTIJES03 contains module footprint tracing that is active, by default, on your system. Therefore, you do not need to take any action to modify this trace.

Activating all four traces can negatively impact system performance because of the heavy volume of trace data produced. For that reason, you should only use this procedure when requested by IBM, and you should not leave this full tracing on. The identical parmlib members are supplied with tracing set off, since you should only run with full tracing at IBM's request. The default contents of parmlib members CTIJES01, CTIJES02, and CTIJES04 is:

```
TRACEOPTS
OFF
```

The default contents of parmlib members CTIJES03 is:

```
TRACEOPTS
ON
```

IBM recommends that you keep this tracing sublevel on at all times.

1. Modify the CTIJES01, CTIJES02, and CTIJES04 parmlib members to turn tracing on: In parmlib members CTIJES01, CTIJES02, and CTIJES04, alter the parmlib members to return sublevel tracing on and connect the sublevel to the external writer. The parmlib members are supplied with tracing off and no connection to the writer.

When you initialize the JES subsystem with the modified parmlib members, full tracing for JES XCF starts automatically.

Figure 134 on page 431 is an example of the CTIJESxx parmlib member after having been modified for gathering trace data during JES subsystem initialization at the direction of the IBM Support Center. The member turns tracing on for the sublevel and connects the sublevel to the external writer.

```
TRACEOPTS
WTRSTART(IXZCTW)
ON
WTR(IXZCTW)
```

Figure 134. Example: Turning on tracing in a CTIJESxx member

2. Create a CTIJESOF parmlib member to stop SYSJES tracing: Use the CTIJESOF parmlib member to stop the full SYSJES tracing turned on during initialization and to disconnect them from the external writer.

3. Stop SYSJES tracing after initialization tracing is complete: Enter a TRACE CT operator command referencing the CTIJESOF parmlib member on the console with master authority as follows:

```
TRACE CT,OFF,COMP=SYSJES,PARM=CTIJESOF
```

4. Remodify the CTIJES01, CTIJES02, and CTIJES04 parmlib members to return to default Tracing: In parmlib members CTIJES01, CTIJES02, and CTIJES04, alter the parmlib member to return sublevel tracing to off.

Figure 135 on page 432 shows the CTIJES01, CTIJES02,and CTIJES04 parmlib members after having been returned to their original contents, with tracing set off.

```
TRACEOPTS
OFF
```

Figure 135. Example: Return to default in a CTIJESxx member

Formatting a SYSJES trace

Format the trace with an IPCS CTRACE COMP(SYSJES) subcommand. To format SYSJES tracing, you must:

- Enter the CTRACE command for SYSJES once for each of the four sublevels you wish to format. See “Format SYSJES sublevel information” on page 432.
- Specify SYSJES options on the OPTIONS parameter. See “OPTIONS parameter for formatting a SYSJES trace” on page 432.
- Merge the output from the different sublevels requested. See “Merging SYSJES information from sublevels” on page 433.

For SYSJES traces, use the IPCS MERGE subcommand to display traces that are not likehead in timestamp order.

Format SYSJES sublevel information

You must enter the CTRACE command separately for each SYSJES sublevel you wish to format. For example, to request formatting of SYSJES trace data for sublevels MSGTRC and USRXIT, you would enter the following two commands:

```
CTRACE COMP(SYSJES) SUB((USRXIT)) FULL
CTRACE COMP(SYSJES) SUB((MSGTRC)) FULL
```

These examples would yield tracing without any options requested.

OPTIONS parameter for formatting a SYSJES trace

IBM might request that you enter options for SYSJES tracing. You can specify options for SYSJES tracing on the OPTIONS parameter of the CTRACE command. The options include:

- Options valid for all sublevels:
  - MSGTOKEN=*msgtoken*
  - REQTOKEN=*reqtoken*
  - MSGBUF=*msgbuf*
  - CTCR=*ctcr*
- Options valid for the FLOW sublevel only:
  - MODID=*id*
  - MODFLOW

  - MSGFLOW
- Options valid for the MSGTRC sublevel only:
  - SEND
  - RECEIVE
- Options valid for the USRXIT sublevel only:
  - EXIT1
  - EXIT3
  - EXIT2

Merging SYSJES information from sublevels

Because SYSJES can run four sublevel traces simultaneously, you will need to merge the data for a complete chronological picture of SYSJES trace data. For example, to merge JESXCF trace data, from all four sublevels, enter the following command:

```
MERGE
CTRACE COMP(SYSJES) SUB((USRXIT)) FULL
CTRACE COMP(SYSJES) SUB((MSGTRC)) FULL
CTRACE COMP(SYSJES) SUB((XCFEVT)) FULL
CTRACE COMP(SYSJES) SUB((FLOW)) FULL
MERGEEND
```

You can write an IPCS CLIST to issue the CTRACE command for the four sublevels and merge the output automatically. See *z/OS MVS IPCS Customization* for information on writing a CLIST.

Output from a SYSJES trace

The output from each SYS*jes2* subtrace is stored in a one-megabyte trace buffer in 64 bit JES2 private. The number of trace entries for each subtrace depends on the current size of an entry. The entries are presented in a wrapping (or rolling) trace format. That is, when the trace table is filled with the maximum number of entries (for example, 500 entries), the next entries (501, 502, 503,...) overwrite entries 1, 2, 3... in a continuous wrapping manner.

"Example: merged output from all of the SYSjes2 sublevel traces with the FULL parameter" on page 433 is an example of merged output from all of the SYS*jes2* sublevel traces with the FULL parameter specified.

Example: merged output from all of the SYSjes2 sublevel traces with the FULL parameter

```
********** MERGED TRACES ***********
01. CTRACE COMP(SYSJES2) SUB((JQE)) FULL
02. CTRACE COMP(SYSJES2) SUB((JOE)) FULL
03. CTRACE COMP(SYSJES2) SUB((DISP)) FULL
04. CTRACE COMP(SYSJES2) SUB((CKPT)) FULL
05. CTRACE COMP(SYSJES2) SUB((SAPI)) FULL
06. CTRACE COMP(SYSJES2) SUB((QGET)) FULL

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((JQE))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((JOE))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((DISP))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((CKPT))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((SAPI))
```

```
 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((QGET))

    SYSNAME MNEMONIC ENTRY ID TIME STAMP      DESCRIPTION
    ------- -------- -------- --------------- -------------
03. SY1     DISP     04000021 15:23:58.803778 Dispatch PCE
  PCE Address->0C615228 Exit->00 JOB#/offset->00000000 00000000
  Module/seq#->HASPHOPE 01520000 Wait time->00000000 00002D94
  $POST type-->0000

      PCE description:OUTPUT PROCESSOR
      $WAIT Events: POST
      $WAIT Resource: HOPE
      $WAIT Options: INHIBIT=NO
      $POST Reason: Resource post
-----------------------------------------------------------------------
01. SY1     JQE     02000009 15:23:58.803794  $QBUSY
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->00 Caller->0C38A2FE
      Artificial JQE

      PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
01. SY1     JQE     0200100A 15:23:58.803803   $DOGJQE,CKPT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->00 Caller->0C38B906

      PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
06. SY1     QGET     07000054 15:23:58.803824   Job Phase QGET
  QGET call for QUEUE=OUTPUT JOBT04
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  JQEs defined-->000001F4 JQEs in use--->00000075
  JQEs scanned-->00000001 $DOGJQE calls->00000001
  $QGET RC------>00000000 JQE selected-->00000001
  QGET CPU time->00000000 0000002C Run time->00000000 0000002C
  QGET $QSUSE--->00000000 0000002C Elapsed-->00000000 0000002C
  X14 CPU time-->00000000 00000002 Run time->00000000 00000002
  X14 $QSUSE---->00000000 00000002 Elapsed-->00000000 00000002
  X14 Ret Code-->00000000
  X49 CPU time-->00000000 00000003 Run time->00000000 00000003
  X49 $QSUSE---->00000000 00000003 Elapsed-->00000000 00000002
  X49 Skip cnt-->00000000 X49 Call cnt-->00000001
  Caller addr--->8C36E0B4 CB address---->00000000
      Exit 14 was entered
      Exit 49 was entered
      Optimization not allowed
01. SY1     JQE      02000006 15:23:58.803828   $GETJLOK
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->02 Caller->0C36E366
      Artificial JQE

PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
01. SY1     JQE      0200100A 15:23:58.803836   $DOGJQE,CKPT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->02 Caller->0C38B1AA

PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
03. SY1     DISP     04000024 15:23:58.803867   PCE $WAIT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Module/seq#->HASPNUC 17001900 Clr Mod/seq->HASPHOPE 04356000
  Run time->00000000 00000058 CPU time---->00000000 0000004B

      PCE description:OUTPUT PROCESSOR
      Artificial JQE
      $WAIT Events: IO
      $WAIT Options:
```

Figure 136 on page 435 is an example of merged output from all of the SYS*jes2* sublevel traces with the TALLY parameter specified.

```
FMTID    COUNT       Interval     MNEMONIC DESCRIBE
-------- ----------- ------------ -------- -------------------------------
02000000         570    2,796,165 JQE      $QADD
02000001           0              JQE      $QPUT
02000002         503    2,449,940 JQE      $QREM
02000003       2,653      600,459 JQE      $QMOD
02000004         571    2,791,312 JQE      $QJIX (ALLOC new number)
02000005           0              JQE      $QJIX (SWAP job numbers)
02000006       1,361    1,158,580 JQE      $GETJLOK
02000007 1,361 1,158,579 JQE $FREJLOK
02000008 0 JQE $QRBDCHK (add to queue)
02000009 2,192 726,801 JQE $QBUSY
0200000A 0 JQE $DOGJQE,FETCHNEXT
0200040A 0 JQE $DOGJQE,FETCH
0200080A 0 JQE $DOGJQE,MANAGELOCK
02000C0A 585 2,724,397 JQE $DOGJQE,RETURN
0200100A 12,048 133,092 JQE $DOGJQE,CKPT
0200140A 0 JQE $DOGJQE,REFRESH
0200180A 0 JQE $DOGJQE,FREE
02001C0A 0 JQE $DOGJQE,SETACCESS
0200240A 0 JQE $DOGJQE,CKPTFLD
03000010 1,842 704,973 JOE $#ADD
03000011 1 JOE $#PUT
03000012 996 1,314,369 JOE $#REM
03000013 0 JOE $#MOD
03000014 0 JOE $#RBDCHK (add to queue)
03000019 1,932 677,264 JOE $#BUSY
0300001A 258 4,813,951 JOE $#GET
0300001B 1 JOE $#CAN
0300001C 999 1,307,972 JOE $#REP
0300001D 0 JOE $DOGJOE,FETCHNEXT
0300041D 3,392 389,955 JOE $DOGJOE,FETCH
0300081D 3,172 417,010 JOE $DOGJOE,RETURN
03000C1D 8,197 161,492 JOE $DOGJOE,CKPT
0300101D 20 51,031,127 JOE $DOGJOE,CKPTFLD
0300141D 0 JOE $DOGJOE,REFRESH
0300181D 0 JOE $DOGJOE,FREE
0300201D 1,034 1,271,661 JOE $DOGJOE,SETACCESS
04000020 3,552 10,447 DISP PCE $WAIT
04000021 3,548 10,459 DISP Dispatch PCE
04000022 1,443 25,728 DISP MVS WAIT
04000024 768 30,845 DISP PCE $WAIT
04000025 771 30,729 DISP Dispatch PCE
05000031 9 50,462,372 SAPI Put/Get call
05000032 2 12,020,796 SAPI Count request
05000033 101 495,559 SAPI Bulk Modify
06000041 1,481 731,500 CKPT CKPT Read 1
06000042 1,481 731,500 CKPT CKPT Read 2
06000043 1,503 720,786 CKPT CKPT Primary Write
06000044 0 CKPT CKPT Skipped Primary Write
06000045 1,762 617,788 CKPT CKPT Intermediate Write
06000046 1,482 734,588 CKPT CKPT Final Write
06000047 0 CKPT CKPT Format
06000048 0 CKPT CKPT Reconfiguration
07000051 3 14 QGET Device QGET
07000052 1,055 23,696 QGET JES INIT QGET
07000053 330 104,030 QGET WLM INIT QGET
07000054 2,061 7,630 QGET Job Phase QGET
```

Figure 136. Example: merged output from all of the SYSjes2 sublevel traces with the TALLY parameter

SYSjes2 component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSjes2 component trace for the JES2 subsystem. For ease of explanation here, this component trace is referred to as ***SYSjes2*** although you might need to replace ***jes2*** with the name you assigned to your JES2 subsystem (primary or secondary). For example, to obtain trace information for JESA, a name you might have used to name your secondary JES2 in a poly-JES environment, use *SYSJESA* as the component name.

Information	For SYSjes2:
Parmlib member	n/a
Default tracing	Yes; full tracing for sublevels JQE, JOE, DISP, CKPT, SAPI, and QGET
Trace request OPTIONS parameter	None
Buffer	• Default: N/A • Range: N/A • Size set by: JES2 • Change size after IPL: No • Location: In the component address space
Trace records location	Address-space buffer
Request of SVC dump	By the component, or DUMP or SLIP
Trace formatting by IPCS	CTRACE COMP(SYSjes2)
Trace format OPTIONS parameter	Yes

SYSjes2 tracing is started automatically during initialization. SYSjes2 contains six sublevel traces, which run continuously and concurrently. The sublevels are:

- **JQE service tracing**: JQE records all job queue service calls (to include: $QADD, $QPUT, $QREM, $QMOD, $QJIX, $GETJLOK, $FREJLOK, $QRBDCHK, $QBUSY, and $DOGJQE).
- **JOE service tracing**: JOE records all job output element service calls (to include: $#ADD, $#PUT, $#REM, $#MOD, $#RBDCHK, $#BUSY, $#GET, $#CAN, $#REP, and $DOGJOE).
- **Dispatcher tracing**: DISP records processing done by the JES2 dispatcher (to include $WAIT, dispatch a PCE, and MVS wait JES2).
- **SYSOUT API tracing**: SAPI records request handled by the SYSOUT API PCE (to include PUT/GET, Count, and Bulk modify requests).
- **Checkpoint I/O tracing**: CKPT records I/O requests performed by the JES2 CKPT process (to include Read 1, Read 2, Primary write, skipped primary write, Intermediate write, Final write, Format write, and checkpoint reconfiguration processing).
- **$QGET service tracing**: $QGET records calls to the $QGET service to obtain a job for processing by a device, a JES mode initiator, a WLM mode initiator, or a job phase.

Requesting a SYSjes2 trace

You need not take any action to request a SYSjes2 trace. The trace is active whenever your JES2 subsystem is in control.

Formatting SYSjes2 sublevel trace Information

You must enter the CTRACE command separately for each SYSJES sublevel you wish to format. For example, to request formatting of SYSJES trace data for sublevels JQE and JOE, you would enter the following two commands:

```
CTRACE COMP(SYSjes2) SUB((JQE)) FULL
CTRACE COMP(SYSjes2) SUB((JOE)) FULL
```

Merging SYSjes2 information from sublevels

Because SYSjes2 runs six sublevel traces simultaneously, you might need to merge multiple sublevels for a complete chronological picture of SYSjes2 trace data. To merge multiple SYSjes2 trace data (JQE and JOE in this example), enter the following command string:

```
MERGE
CTRACE COMP(SYSjes2) SUB((JQE)) FULL
CTRACE COMP(SYSjes2) SUB((JOE)) FULL
MERGEEND
```

You can write an IPCS CLIST to issue the CTRACE command for multiple sublevels and merge the output automatically. See *z/OS MVS IPCS Customization* for information about writing a CLIST.

Output from a SYSjes2 trace

The output from each SYSjes2 subtrace in stored in a one megabyte trace buffer in 64 bit JES2 private. The number of trace entries for each subtrace depends on the current size of an entry. The entries are presented in a wrapping (or rolling) trace format. That is, once the trace table is filled with the max number of entries (say for example 500 entries), the next entries (501, 502, 503,...) overwrite entries 1, 2, 3... in a continuous wrapping manner.

Figure 137 on page 438 is an example of merged output from all the SYSjes2 sublevel traces with the FULL parameter specified.

```
********** MERGED TRACES ***********
01. CTRACE COMP(SYSJES2) SUB((JQE)) FULL
02. CTRACE COMP(SYSJES2) SUB((JOE)) FULL
03. CTRACE COMP(SYSJES2) SUB((DISP)) FULL
04. CTRACE COMP(SYSJES2) SUB((CKPT)) FULL
05. CTRACE COMP(SYSJES2) SUB((SAPI)) FULL
06. CTRACE COMP(SYSJES2) SUB((QGET)) FULL

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((JQE))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((JOE))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((DISP))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((CKPT))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((SAPI))

 COMPONENT TRACE FULL FORMAT
 COMP(SYSJES2) SUBNAME((QGET))

    SYSNAME MNEMONIC ENTRY ID TIME STAMP      DESCRIPTION
    ------- -------- -------- --------------- -------------
03. SY1     DISP     04000021 15:23:58.803778 Dispatch PCE
  PCE Address->0C615228 Exit->00 JOB#/offset->00000000 00000000
  Module/seq#->HASPHOPE 01520000 Wait time->00000000 00002D94
  $POST type-->0000

      PCE description:OUTPUT PROCESSOR
      $WAIT Events: POST
      $WAIT Resource: HOPE
      $WAIT Options: INHIBIT=NO
      $POST Reason: Resource post
-----------------------------------------------------------------------
01. SY1     JQE     02000009 15:23:58.803794  $QBUSY
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->00 Caller->0C38A2FE
      Artificial JQE

      PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
01. SY1     JQE     0200100A 15:23:58.803803   $DOGJQE,CKPT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->00 Caller->0C38B906

      PCE description:OUTPUT PROCESSOR
-----------------------------------------------------------------------
06. SY1     QGET     07000054 15:23:58.803824   Job Phase QGET
  QGET call for QUEUE=OUTPUT JOBT04
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  JQEs defined-->000001F4 JQEs in use--->00000075
  JQEs scanned-->00000001 $DOGJQE calls->00000001
  $QGET RC------>00000000 JQE selected-->00000001
  QGET CPU time->00000000 0000002C Run time->00000000 0000002C
  QGET $QSUSE--->00000000 0000002C Elapsed-->00000000 0000002C
  X14 CPU time-->00000000 00000002 Run time->00000000 00000002
  X14 $QSUSE---->00000000 00000002 Elapsed-->00000000 00000002
  X14 Ret Code-->00000000
  X49 CPU time-->00000000 00000003 Run time->00000000 00000003
  X49 $QSUSE---->00000000 00000003 Elapsed-->00000000 00000002
  X49 Skip cnt-->00000000 X49 Call cnt-->00000001
  Caller addr--->8C36E0B4 CB address---->00000000
      Exit 14 was entered
      Exit 49 was entered
      Optimization not allowed
```

Figure 137. Example: merged output from all of the SYSjes2 sublevel traces with the FULL parameter

```
01. SY1     JQE      02000006 15:23:58.803828   $GETJLOK
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->02 Caller->0C36E366
      Artificial JQE

PCE description:OUTPUT PROCESSOR
----------------------------------------------------------------------
01. SY1     JQE      0200100A 15:23:58.803836   $DOGJQE,CKPT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Original Queue->02 New Queue->02 Busy->02 Lock->02 Caller->0C38B1AA

PCE description:OUTPUT PROCESSOR
----------------------------------------------------------------------
03. SY1     DISP     04000024 15:23:58.803867   PCE $WAIT
  PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
  Module/seq#->HASPNUC 17001900 Clr Mod/seq->HASPHOPE 04356000
  Run time->00000000 00000058 CPU time---->00000000 0000004B

      PCE description:OUTPUT PROCESSOR
      Artificial JQE
      $WAIT Events: IO
      $WAIT Options:
```

Figure 138. Example: merged output from all of the SYSjes2 sublevel traces with the FULL parameter (Continued)

Figure 139 on page 440 is an example of merged output from all the SYSjes2 sublevel traces with the SHORT parameter specified:

```
********** MERGED TRACES ***********
01. CTRACE COMP(SYSJES2) SUB((JQE)) SHORT
02. CTRACE COMP(SYSJES2) SUB((JOE)) SHORT
03. CTRACE COMP(SYSJES2) SUB((DISP)) SHORT
04. CTRACE COMP(SYSJES2) SUB((CKPT)) SHORT
05. CTRACE COMP(SYSJES2) SUB((SAPI)) SHORT
06. CTRACE COMP(SYSJES2) SUB((QGET)) SHORT
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((JQE))
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((JOE))
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((DISP))
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((CKPT))
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((SAPI))
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSJES2) SUBNAME((QGET))

     SYSNAME   MNEMONIC  ENTRY ID     TIME STAMP       DESCRIPTION
     -------   --------  --------   ---------------    -------------
03.  SY1       DISP      04000021   15:23:58.803778    Dispatch PCE
01.  SY1       JQE       02000009   15:23:58.803794    $QBUSY
01.  SY1       JQE       0200100A   15:23:58.803803    $DOGJQE,CKPT
06.  SY1       QGET      07000054   15:23:58.803824    Job Phase QGET
01.  SY1       JQE       02000006   15:23:58.803828    $GETJLOK
01.  SY1       JQE       0200100A   15:23:58.803836    $DOGJQE,CKPT
03.  SY1       DISP      04000024   15:23:58.803867    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803867    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803871    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803873    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803874    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803878    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803879    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803880    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803884    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803885    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803885    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803889    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803891    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803891    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803896    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803897    PCE $WAIT
03.  SY1       DISP      04000021   15:23:58.803897    Dispatch PCE
06.  SY1       QGET      07000054   15:23:58.803901    Job Phase QGET
03.  SY1       DISP      04000020   15:23:58.803902    PCE $WAIT
```

Figure 139. Example: merged output from both SYSjes2 sublevel traces with SHORT parameter

Figure 140 on page 441 is an example of merged output from all of the SYS*jes2* sublevel traces with the SUMMARY parameter specified.

```
********** MERGED TRACES ***********
01. CTRACE COMP(SYSJES2) SUB((JOE)) SUMMARY
02. CTRACE COMP(SYSJES2) SUB((JOE)) SUMMARY
03. CTRACE COMP(SYSJES2) SUB((DISP)) SUMMARY
04. CTRACE COMP(SYSJES2) SUB((CKPT)) SUMMARY
05. CTRACE COMP(SYSJES2) SUB((SAPI)) SUMMARY
06. CTRACE COMP(SYSJES2) SUB((QGET)) SUMMARY

 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((JOE))
 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((JOE))
 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((DISP))
 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((CKPT))
 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((SAPI))
 COMPONENT TRACE SUMMARY FORMAT
 COMP(SYSJES2) SUBNAME((QGET))
 **** 04/30/2018

     SYSNAME    MNEMONIC    ENTRY ID     TIME STAMP      DESCRIPTION
     -------    --------    --------   ---------------   -------------
03.  SY1        DISP        04000021   15:23:58.803778   Dispatch PCE
    PCE Address->0C615228 Exit->00 Module/seq#->HASPHOPE 01520000
        $WAIT Events:    POST
        $WAIT Resource:  HOPE
        $WAIT Options:   INHIBIT=NO
-----------------------------------------------------------------------
06.  SY1        QGET        07000054   15:23:58.803824   Job Phase QGET
    QGET call for QUEUE=OUTPUT                     JOBT04
    PCE Address->0C615228 Exit->00 JOB#/offset->00000531 00002C88
    $QGET RC------>00000000 JQE selected-->00000001
03.  SY1        DISP        04000024   15:23:58.803867   PCE $WAIT
    PCE Address->0C615228 Exit->00 Module/seq#->HASPNUC 17001900
    $WAIT Events:       IO
    $WAIT Options:
-----------------------------------------------------------------------
03.  SY1        DISP        04000021   15:23:58.803867   Dispatch PCE
    PCE Address->0C620718 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
-----------------------------------------------------------------------
06.  SY1        QGET        07000054   15:23:58.803871   Job Phase QGET
    QGET call for QUEUE=OUTPUT
    PCE Address->0C620718 Exit->00 JOB#/offset->00000000 00000000
    $QGET RC------>00000004 JQE selected-->00000001
03.  SY1        DISP        04000020   15:23:58.803873   PCE $WAIT
    PCE Address->0C620718 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
-----------------------------------------------------------------------
03.  SY1        DISP        04000021   15:23:58.803874   Dispatch PCE
    PCE Address->0C620D38 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
-----------------------------------------------------------------------
06.  SY1        QGET        07000054   15:23:58.803878   Job Phase QGET
    QGET call for QUEUE=OUTPUT
    PCE Address->0C620D38 Exit->00 JOB#/offset->00000000 00000000
    $QGET RC------>00000004 JQE selected-->00000001
03.  SY1        DISP        04000020   15:23:58.803879   PCE $WAIT
    PCE Address->0C620D38 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
-----------------------------------------------------------------------
03.  SY1        DISP        04000021   15:23:58.803880   Dispatch PCE
    PCE Address->0C620A28 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
-----------------------------------------------------------------------
06.  SY1        QGET        07000054   15:23:58.803884   Job Phase QGET
    QGET call for QUEUE=OUTPUT
    PCE Address->0C620A28 Exit->00 JOB#/offset->00000000 00000000
    $QGET RC------>00000004 JQE selected-->00000001
```

Figure 140. Example: merged output from both SYSjes2 sublevel traces with SUMMARY parameter

```
03.  SY1        DISP        04000020   15:23:58.803885   PCE $WAIT
    PCE Address->0C620A28 Exit->00 Module/seq#->HASPHOPE 01520000
    $WAIT Events: POST
    $WAIT Resource: HOPE
    $WAIT Options: INHIBIT=NO
```

Figure 141. Example: merged output from both SYSjes2 sublevel traces with SUMMARY parameter (Continued)

Figure 142 on page 442 is an example of merged output from all of the SYS*jes2* sublevel traces with the TALLY parameter specified.

```
FMTID     COUNT      Interval      MNEMONIC  DESCRIBE
-------- ----------- ------------  --------  --------------------------------
02000000         570    2,796,165  JQE       $QADD
02000001           0               JQE       $QPUT
02000002         503    2,449,940  JQE       $QREM
02000003       2,653      600,459  JQE       $QMOD
02000004         571    2,791,312  JQE       $QJIX (ALLOC new number)
02000005           0               JQE       $QJIX (SWAP job numbers)
02000006       1,361    1,158,580  JQE       $GETJLOK
02000007       1,361    1,158,579  JQE       $FREJLOK
02000008           0               JQE       $QRBDCHK (add to queue)
02000009       2,192      726,801  JQE       $QBUSY
0200000A           0               JQE       $DOGJQE,FETCHNEXT
0200040A           0               JQE       $DOGJQE,FETCH
0200080A           0               JQE       $DOGJQE,MANAGELOCK
02000C0A         585    2,724,397  JQE       $DOGJQE,RETURN
0200100A      12,048      133,092  JQE       $DOGJQE,CKPT
0200140A           0               JQE       $DOGJQE,REFRESH
0200180A           0               JQE       $DOGJQE,FREE
02001C0A           0               JQE       $DOGJQE,SETACCESS
0200240A           0               JQE       $DOGJQE,CKPTFLD
03000010       1,842      704,973  JOE       $#ADD
03000011           1               JOE       $#PUT
03000012         996    1,314,369  JOE       $#REM
03000013           0               JOE       $#MOD
03000014           0               JOE       $#RBDCHK (add to queue)
03000019       1,932      677,264  JOE       $#BUSY
0300001A         258    4,813,951  JOE       $#GET
0300001B           1               JOE       $#CAN
0300001C         999    1,307,972  JOE       $#REP
0300001D           0               JOE       $DOGJOE,FETCHNEXT
0300041D       3,392      389,955  JOE       $DOGJOE,FETCH
0300081D       3,172      417,010  JOE       $DOGJOE,RETURN
03000C1D       8,197      161,492  JOE       $DOGJOE,CKPT
0300101D          20   51,031,127  JOE       $DOGJOE,CKPTFLD
0300141D           0               JOE       $DOGJOE,REFRESH
0300181D           0               JOE       $DOGJOE,FREE
0300201D       1,034    1,271,661  JOE       $DOGJOE,SETACCESS
04000020       3,552       10,447  DISP      PCE $WAIT
04000021       3,548       10,459  DISP      Dispatch PCE
04000022       1,443       25,728  DISP      MVS WAIT
04000024         768       30,845  DISP      PCE $WAIT
04000025         771       30,729  DISP      Dispatch PCE
05000031           9   50,462,372  SAPI      Put/Get call
05000032           2   12,020,796  SAPI      Count request
05000033         101      495,559  SAPI      Bulk Modify
06000041       1,481      731,500  CKPT      CKPT Read 1
06000042       1,481      731,500  CKPT      CKPT Read 2
06000043       1,503      720,786  CKPT      CKPT Primary Write
06000044           0               CKPT      CKPT Skipped Primary Write
06000045       1,762      617,788  CKPT      CKPT Intermediate Write
06000046       1,482      734,588  CKPT      CKPT Final Write
06000047           0               CKPT      CKPT Format
06000048           0               CKPT      CKPT Reconfiguration
07000051           3           14  QGET      Device QGET
07000052       1,055       23,696  QGET      JES INIT QGET
07000053         330      104,030  QGET      WLM INIT QGET
07000054       2,061        7,630  QGET      Job Phase QGET
```

Figure 142. Example: merged output from both SYSjes2 sublevel traces with TALLY parameter

SYSLLA component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSLLA component trace for library lookaside (LLA) of contents supervision.

Information	For SYSLLA:
Parmlib member	None
Default tracing	Yes; always on when LLA is running
Trace request OPTIONS parameter	None
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: In the component address space
Trace records location	Address-space buffer
Request of SVC dump	By the component
Trace formatting by IPCS	CTRACE COMP(SYSLLA)
Trace format OPTIONS parameter	None

Requesting a SYSLLA trace

The trace runs whenever LLA is in control. No actions are needed to request it.

Formatting a SYSLLA trace

Format the trace with an IPCS CTRACE COMP(SYSLLA) subcommand. The subcommand has no OPTIONS values.

SYSLOGR component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

Table 74 on page 444 summarizes information for requesting a SYSLOGR component trace for the system logger component. SYSLOGR tracing is started during initialization.

Table 74. Summary of SYSLOGR component trace request	
Information	**For SYSLOGR:**
Parmlib member	CTnLOGxx Default member: CTILOG00, is required and provided as a component trace option in GRSCNFxx parmlib member. 1. IXGCNFxx member CTRACE field specified at IPL or SET IXGCNF command 2. SETLOGR CTRACE command 3. TRACE command.
Parmlib default tracing options	CONNECT, LOGSTRM, DATASET, SERIAL, MISC, LOCBUFF, RECOVERY
Default tracing	Yes; activated during System Logger (IXGLOGR) address space initialization
Trace request OPTIONS parameter	In CTnLOGxx and REPLY for TRACE command
Buffer	• Parmlib default: 16MB • Default: 2MB • Range: 2MB - 2047MB • Size set by: CTnLOGxx parmlib member or REPLY for TRACE CT command. • Change size after IPL: Yes • Location: system logger trace data space
Trace records location	Address-space buffer; system logger trace data space, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSLOGR)
Trace format OPTIONS parameter	Yes

Obtaining a dump of system logger information

Use the following examples to obtain the appropriate diagnostic information for system logger. The amount of information requested in the dumps may be very large. You may need to set your MAXSPACE on the CHNGDUMP setting to 999 mb before obtaining the logger dumps.

```
CD SET,SDUMP,MAXSPACE=999M
```

For structure dumps, verify that the coupling facility has dump space defined by issuing the following command:

```
D CF,CFNAME=xxxx
```

Note: There are several sample Logger dump parmlib members that can be used as models for automating the procedures listed below. The samples parmlib members are shipped in 'SYS1.SAMPLIB(IEADMCLx)'. Refer to the IEADMCLx members for more information, or see Table 7 on page 18.

1. For all types of logstreams, always include the following:
   a. The IXGLOGR (Logger) asid and the data spaces associated with the IXGLOGR asid through the DSPNAME parm. These will be dumped using the JOBNAME= parm.

b. The trace data space SYSLOGR0, will be included in the dump if DSNAME=('IXGLOGR'.*) is specified in the reply for the dump command.

   Note: If you are running OS/390® Release 2 or lower, Logger will not have a SYSLOGR* data space for tracing, in which case the DSPNAME=('IXGLOGR'.*) option can be omitted.

```
DUMP  COMM=(dump system logger with component trace)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr,JOBNAME=(IXGLOGR),CONT
ss,DSPNAME=('IXGLOGR'.SYSLOGR*),CONT
tt,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,SUM,
          TRT,CSA,GRSQ,XESDATA),END
```

2. When using CF list structure based log streams, include the following

   a. The XCF asid and trace data spaces. These will be dumped using the JOBNAME= parm and DSPNAME parm.

   b. The XES STRUCTURE data. This is dumped using the STRLIST= parameter and by specifying the structure name. structure_name is the affected STRUCTURE name.

      Note: Be sure to allocate adequate DUMPSPACE() as defined in your CF definition in the CFRM policy. If you do not allocate adequate space, all or part of the STRUCTURE will NOT be dumped.

   c. In the case of “loss of data” or “block not found”, dumping the OFFLOAD data sets using IDCAMS is a good idea.

```
DUMP COMM=(dump system logger with xes and structure data)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr,JOBNAME=(IXGLOGR,XCFAS),CONT
ss,DSPNAME=('XCFAS'.*,'IXGLOGR'.SYSLOGR*),CONT
tt,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,SUM,
          TRT,CSA,GRSQ,XESDATA),CONT
uu,STRLIST=(STRNAME=structure_name,LOCKENTRIES,ACC=NOLIM,
    (LISTNUM=ALL,ADJUNCT=DIRECTIO,ENTRYDATA=UNSERIALIZE)),END
```

3. When system logger dumps are needed on multiple systems in the sysplex, include the REMOTE parameter.

```
DUMP COMM=(dumps for system logger around the sysplex)
* rr IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
rr,JOBNAME=(IXGLOGR,XCFAS),CONT
ss,DSPNAME=('XCFAS'.*,'IXGLOGR'.SYSLOGR*),CONT
tt,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,SUM,
           TRT,CSA,GRSQ,XESDATA),CONT
uu,STRLIST=(STRNAME=structure_name,LOCKENTRIES,ACC=NOLIM,
     (LISTNUM=ALL,ADJUNCT=DIRECTIO,ENTRYDATA=UNSERIALIZE)),CONT
vv,REMOTE=(SYSLIST=*)'XCFAS','IXGLOGR'),DSPNAME,SDATA),END
```

4. Other diagnostic considerations

   The CTILOG00 buffer size is 16MB. It is strongly recommended that this value not be lowered. See the CTnLOGxx member, described in Table 74 on page 444.

   Remember that much of the data that system logger uses is persistent across an IPL. That means that if this data is corrupted and adversely affects system logger, an IPL will not correct the problem. In the case of a persistent system logger failure, you can FORCE the IXGLOGR address space. Prior to doing this you should bring down all of the applications connected. Then issue the FORCE command (`FORCE IXGLOGR,ARM`) and restart system logger using the supplied procedure in SYS1.PROCLIB (IXGLOGRS). (`S IXGLOGRS`)

   If `FORCE IXGLOGR,ARM` does not resolve the situation, an IPL is not likely to either. This is the time to take a dump if one was not already taken by system logger.

   Note:

   a. A CICS dump will not dump the IXGLOGR address space. Connect to a new (non-corrupted) LOGSTREAM. This will result in a LOSS OF DATA for some applications such as CICS, forcing them to INITIAL START. However, this may be the only way to get the application restarted. Connecting to a new logstream (of a different name) will allow the corrupted data to remain in the structure for diagnostic review later.

b. In preparation for connecting to this new LOGSTREAM, you may want to define an unused LOGSTREAM to each STRUCTURE during setup.

If running CICS, always run with the following SLIP:

```
SLIP SET,COMP=1C5,REASON=804,
  STRLIST=(STRNAME=strname1,LOCKE,ACC=NOLIM,
  (LNUM=ALL,EDATA=SER,ADJ=CAP)),
  JL=(XCFAS,IXGLOGR),DN=("XCFAS".*,"IXGLOGR".*),
  SD=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,TRT,CSA,GRSQ,
    XESDATA,SUM),
  SUMLIST=(13R?-7FFF,13R?+7FFF),END
```

Note: You might add a JOBNAME=DFH* to limit SLIP to CICS. A RSN804 is a “block not found”, which is always bad for CICS but not necessarily so for other applications. Setting this SLIP will cause system logger to dump on all RSN804s in CICS.

5. Frequent stumbling blocks

   a. OFFLOAD data sets must be VSAM SHAREOPTIONS(3,3) unless you are in a MONOPLEX.

   b. After OW33261, system logger recommends for performance reasons using 24K CI size for OFFLOAD data sets. Staging data sets must remain at 4K CI size. Code your ACS routines appropriately.

   c. Size of XES structures. “Bigger is not always better.” Follow exploiting application recommendations.

   d. Allow for OFFLOAD directory extents. Reference “Format Utility for Couple Data Sets” in *z/OS MVS Setting Up a Sysplex*.

   e. System logger uses HSM services to recall (ARCHRCAL) and to delete (ARCHDEL) offload data sets. HSM contention or a wait for a WTOR such as ARC0055A can hang all of the log streams that require the system logger allocation task.

Requesting a SYSLOGR trace

Specify options for requesting a SYSLOGR component trace in a CTnLOGxx parmlib member or on the reply for a TRACE CT command.

Also, refer to the following as alternatives for specifying the CTnLOGxx parmlib member to be used:

- IXGCNFxx SYS1.PARMLIB member (see IXGCNFXX SYS1.PARMLIB member).
- SET IXGCNF=xx command.
- SETLOGR CTRACE command.

Specify options for requesting a SYSLOGR component trace in a CTnLOGxx parmlib member or on the reply for a TRACE CT command.

You can change the options and the trace data space buffer size for SYSLOG trace while the trace is running. However, if the SYSLOGR trace has not been connected to an external writer and you are reducing the size of the trace data space buffer, you **must** dump the contents of the buffer (see “Obtaining a dump of system logger information” on page 444) **before** reducing the buffer size if this data is important for debugging. Trace data in the trace data space is discarded when the buffer size is reduced.

Note that if the trace is being turned off (either through a TRACE CT,OFF command or a CTnLOGxx parmlib member) and if the SYSLOGR trace is not connected to an external writer, the trace data **must** be dumped **before** turning the trace off to avoid loss of data.

CTnLOGxx parmlib member

The following table indicates the parameters you can specify in a CTnLOGxx parmlib member.

Parameters	Allowed on CTnLOGxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM supplies the CTILOG00 parmlib member, which specifies the System Logger tracing activated at initialization. The contents of CTILOG00 as of V1.4 with OA07611 applied are:

```
TRACEOPTS ON
   BUFSIZE(16M)
    OPTIONS('CONNECT,LOGSTRM,DATASET,SERIAL,MISC,LOCBUFF,RECOVERY')
```

These parameters turn on the default system logger tracing to ensure that specific trace options are included and to establish a default trace buffer of 16MB. These trace options are activated at System Logger initialization.

If the PARMLIB trace options specify OFF or when the operator turns the trace off, logger uses minimal tracing which consists only of unexpected events (i.e. COMPERR trace entries).

Example of CTnLOGxx parmlib member

The statements in the following CTnLOGxx parmlib member example specify a 24MB trace buffer. All system logger trace records will be included in the trace output:

```
TRACEOPTS
  ON
  BUFSIZE(24M)
  OPTIONS('ALL')
```

The statements in the following CTxLOGxx example specify a 32MB trace buffer, with tracing of logstream functional request processing for logstreams SYSPLEX.OPERLOG in ASID 09. In addition, an external writer, EXTWTR, will be started, and SYSLOGR will be connected to the external writer:

```
TRACEOPTS
  WTRSTART(EXTWTR)
  ON
  BUFSIZE(32M)
  ASID(09)
  WTR(EXTWTR)
  OPTIONS('LOGSTRM','STRMNAME=(SYSPLEX.OPERLOG)')
```

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON, nnnnM, or OFF	Yes
COMP	Required
SUB	No

Parameters	Allowed on TRACE CT for trace?
PARM	Yes
OPTIONS	Yes, allowed only on REPLY

If you reduce the size of the trace data space buffer and the SYSLOG trace has not been connected to an external writer, you **must** dump the contents of the buffer (see "Obtaining a dump of system logger information" on page 444) **before** reducing the buffer size if this data is important for debugging. Trace data in the trace data space buffer is discarded when the buffer size is reduced.

If the trace is being turned off (either through a TRACE CT,OFF command or a CTnLOGxx parmlib member) and the trace is not connected to an external writer, the trace data **must** be dumped **before** turning the trace off to avoid loss of data.

Parameters	Allowed on REPLY for trace?
ASID	Yes
JOBNAME	No
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter for the CTnLOGxx parmlib member and reply for a TRACE command, in alphabetical order, are:

Option	Default	Subparameters
ALL	No	No
CONNECT	Yes	No
DATASET	Yes	No
INVENTRY	No	No
LOCBUFF	Yes	No
LOGSTRM	Yes	No
MISC	Yes	No
RECOVERY	Yes	No
SERIAL	Yes	No
STRMNAME	No	Logstream
STORAGE	No	No

ALL
Traces all system logger events.

ASID
Traces events for only the specified address space identifiers (ASID).

COMPERR
Traces internal system logger errors or unexpected events. This option is not specifiable and is always traced, as it is considered the minimal tracing.

CONNECT
Traces list structure connections, disconnections, rebuild and event exit processing.

DATASET
Traces log stream data set allocation and management.

INVENTRY
Traces log stream and structure definition and deletion processing as well as all LOGR CDS accesses. Do not specify this option unless requested by the IBM Support Center as this generate a large amount of records and may cause the buffer to wrap frequently.

LOCBUFF
Traces system logger local buffer management.

LOGSTRM
Traces log stream functional request processing.

MISC
Traces system logger internal miscellaneous services.

RECOVERY
Traces system logger component recovery, detecting abnormal conditions during processing.

SERIAL
Traces system logger serialization services.

STORAGE
Traces system logger storage management. Do not specify this option unless requested by the IBM Support Center as this will generate a large amount of records and may cause the buffer to wrap frequently.

STRMNAME
Traces events for only the specified log streams. If you specify STRMNAME, the specified log streams filter the CONNECT, LOGSTRM, INVENTRY, and DATA SET options. The STRMNAME parameter must be specified STRMNAME=(*strmname1*). If you specify more than one log stream, STRMNAME must be specified STRMNAME=(*strmname1*,*strmname2*). A maximum of eight log stream names can be specified.

Note that the system does not verify the log stream names specified.

Formatting a SYSLOGR trace

Format the trace with an IPCS CTRACE COMP(SYSLOGR) subcommand. IBM might request that you enter options for SYSLOGR formatting. You can specify options for SYSLOG0R tracing on the OPTIONS parameter of the CTRACE command. The options include:

COMPERR
Displays internal system logger component errors.

CONNECT
Displays list structure connections, disconnections, rebuild and event exit processing.

DATASET
Displays log stream data set allocation and management.

INVENTRY
Displays log stream and structure definition and deletion processing as well as LOGR policy processing.

LOCBUFF
Displays system logger local buffer management.

LOGSTRM
Displays log stream functional request processing.

MISC
Displays system logger internal miscellaneous services.

RECOVERY
Displays system logger component recovery, detecting abnormal conditions during processing.

RQE(*request_address*)
Specify an 8-byte hexadecimal RQE address. Displays the specified RQE control block.

SERIAL
Displays system logger serialization services.

STACK(*request_address*)
Specify an 8-byte hexadecimal stack address. Displays the stack at the specified address.

STORAGE
Displays system logger storage management.

Output from a SYSLOGR trace

Figure 143 on page 450is an example of system logger component trace records formatted with the CTRACE COMP(SYSLOGR) subcommand:

```
COMPONENT TRACE FULL FORMAT
COMP(SYSLOGR)
**** 09/12/1994

CONNECT   03190001  13:03:28.955894   System Logger Services
   C9E7C3E2  C9C7F0F3  40404040  40404040  | IXCSIG03          |
   E2D3C3E3  C5E2E3F1  0100                | SLCTEST1..        |
RECOVERY  07040001  13:03:58.055519   System Logger Services
   C9E7C7C3  F4D9C6C3  840C1000  00000001  | IXGC4RFCd.......  |
   03171D80  0000                          | ......            |
RECOVERY  07040001  13:09:55.907719   System Logger Services
   C9E7C7C3  F4D9C6C3  840C1000  00000001  | IXGC4RFCd.......  |
   031700A0  0000                          | ......            |
COMPERR   01070007  13:30:58.322696   System Logger Services
   E2E8E2F0  F0F0F0F1  C9E7C7D3  D6C7D94B  | SYS00001IXGLOGR.  |
   E2C3D6E3  E3F34BC1  F0F0F0F0  F0F0F140  | SCOTT3.A0000001   |
   40404040  40404040  40404040  40404040  |                   |
   40404040  0000                          |     ..            |
```

Figure 143. Example: system logger component trace records formatted with CTRACE COMP(SYSLOGR) subcommand

SYSOMVS component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

The following summarizes information for requesting a SYSOMVS component trace for z/OS UNIX.

Information	**For SYSOMVS:**
Parmlib member	CTnBPXxx. Default member: CTIBPX00 specified in BPXPRM00 member
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	In CTnBPXxx and REPLY for TRACE command
Buffer	• Default: 128 MB. • Range: 16 KB - 2047 MB. • Size set by: CTnBPXxx member and TR CTRACE commands and REPLY. • Location: Exception and XCF buffers are located in Data space. In the REPLY for the DUMP command, specify DSPNAME="*asid*.SYSZBPXX" where *asid* is the ASID for z/OS UNIX. The main Ctrace buffers are located in above the bar common storage.

Information	For SYSOMVS:
Trace records location	Data space buffer, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSOMVS)
Trace format OPTIONS parameter	Yes

Requesting a SYSOMVS trace

Specify options for requesting a SYSOMVS component trace on a CTnBPXxx parmlib member or on the reply for a TRACE CT command.

You can change options for SYSOMVS tracing while the trace is running.

CTnBPXxx parmlib member

The following table indicates the parameters you can specify on a CTnBPXxx parmlib member.

Parameters	Allowed on CTnBPXxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	Yes – see note
BUFSIZE	Yes – see note
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

Note:

- Specify the new buffer size in the BUFSIZE parameter on the CTnBPXxx member being used.
- The JOBNAME= parameter can be used for the SYSOMVS Ctrace to trace data just for jobs that run with the specific user ID(s) specified in the JOBNAME list. This filtering is based on the user ID of a job, not its jobname.
- The OMVS kernel is traced with jobname OMVS.

IBM supplies the CTIBPX00 parmlib member, which specifies the z/OS UNIX tracing begun at ipl. The contents of CTIBPX00 are:

```
TRACEOPTS
      ON
      BUFSIZE(128K)
```

The parameters turn on the minimal SYSOMVS tracing. These parameters request the unexpected or important z/OS UNIX System Services events. The trace buffer size is 128KB. This member activates the minimal trace at ipl. In the IBM-supplied BPXPRM00 parmlib member, the CTRACE parameter specifies CTIBPX00 as the default.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON or OFF	One is required
*nnnn*K or *nnnn*M	Yes
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

You can change options while a SYSOMVS trace is running.

OPTIONS parameter

The values for the OPTIONS parameter for the CTnBPXxx parmlib member and reply for a TRACE command, in alphabetical order, are:

ALL
Traces all events.

CBTR(*cbid*,*offset*,*length*)
Traces a field or fields of a control block to be traced. The contents of the trace will be included in the trace record for all SYSCALL trace records.

- *cbid* specifies the name of any of the supported z/OS UNIX control blocks in 1–4 characters.
- *offset* specifies the offset of the desired field in the control block in range X'0'-X'FFFF').
- *length* specifies the length of the data, in bytes, to be traced in the control block. *length* is an integer hexadecimal value with a range of X'1'-X'8'.

CHARS
Traces character special events.

DEVPTY
Traces pseudoterminal events.

DEVRTY
Traces outboard communication server (OCS) remote terminal events.

FILE
Traces file system events. In a shared file system configuration, selecting the FILE option also activates the XCF option.

IPC
Traces interprocess communication activity for shared memory, message queues and semaphores.

LOCK
Traces locking services events.

PIPE
Traces pipe events.

PROCESS
Traces process events.

PTRACE
Traces PTRACE events.

SIGNAL
Traces signaling events.

STORAGE
Traces storage management events.

SYSCALL
Traces callable service layer events.

XCF
Traces file sharing events when using a shared file system configuration.

Examples of requesting SYSOMVS traces

- Example 1: CTnBPXxx member

  The member requests DEVPTY, FILE, and SIGNAL options.

  ```
  TRACEOPTS
     ON
     OPTIONS('DEVPTY','FILE','SIGNAL')
  ```

- Example 2: TRACE command

  The example requests a trace of DEVPTY and FILE trace events.

  ```
  TRACE CT,ON,COMP=SYSOMVS
  * 18 ITT006A ...
  REPLY 18,OPTIONS=(DEVPTY,FILE),END
  ```

- Example 3: TRACE command

  The example requests a trace of four bytes at offset zero of control block PPRP.

  ```
  TRACE CT,ON,COMP=SYSOMVS
  * 18 ITT006A ...
  REPLY 18,OPTIONS=(CBTR(PPRP,0,4)),END
  ```

Formatting a SYSOMVS trace

Format the trace with an IPCS CTRACE COMP(SYSOMVS) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

ALL
Formats all events.

CHARS
Formats character special events.

DEVPTY
Formats pseudoterminal events.

DEVRTY
Formats OCS remote terminal events.

EXCEPTION
Formats exceptional events, such as recovery records or error records.

FILE
Formats file system events.

IPC
Formats events for shared memory, message queues and semaphores.

KERNINFO
Formats the output to include a header for each record that includes descriptive information regarding the system call, process ID, and the module that requests the trace.

LOCK
Formats locking services events.

PIPE
Formats pipe events.

PROCESS
Formats process events.

PTRACE
Formats PTRACE events.

SCCOUNTS
Counts the number of syscalls that occur in the trace. Also counts the number of function codes that occur in a trace. The outputs are displayed in tables. Formatting is suppressed. The function codes refer to the types of messages that are crossing between systems in a sysplexed environment. In a non-sysplex dump, the functions code table will be empty. You could run an application while collecting CTRACE data, and then use this option to determine the frequency of syscalls and function codes being made by the application.

SEARCH
Starting at the specified offset, searches trace entries for a specific value and displays the matches. A CLIST called BPXMSCER is provided to allow the search to be performed against specific entity ids that will identify syscall exits that have failed.

SIGNAL
Formats signaling events.

STORAGE
Formats storage management events.

SYSCALL
Formats callable service layer events.

SYSID(nnn)
Formats sysplex system events. When this is requested by the user, only those trace records that contain a sysplex system id will be formatted and displayed. (nnn) is the sysplex number or name of the system in the sysplex whose records will be displayed. See "Example of CTRACE DISPLAY PARAMETERS panel" on page 454 for an example of a CTRACE DISPLAY PARAMETERS panel and the SYSID option on that panel.

XCF
Formats XCF events.

Example of CTRACE DISPLAY PARAMETERS panel

The CTRACE DISPLAY PARAMETERS panel has the following format. When SYSID is specified, only those trace records that contain a sysplex system ID will be formatted and displayed. If SYSID is not specified, data from all the systems will be displayed.

```
--------- CTRACE DISPLAY PARAMETERS --------- Enter option
COMMAND ===>

System          ===>  (System name or blank
Component       ===>  SYSOMVS (Component name (required)
Subnames        ===>

GMT/LOCAL       ===>  G      (G or L, GMT is default)
Start/time      ===>         (mm/dd/yy,hh:mm:ss:dddddd or
Stop time       ===>          mm/dd/yy,hh.mm:ss.dddddd)
Limit           ===>  0       Exception ===>
Report type     ===>  SHORT  (SHort, SUmmary, Full, Tally)
User exit       ===>          (Exit program name)
Override source ===>
Options         ===>  SYSID(1)

To enter/verify required values, type any character
Entry IDs ===> Jobnames ===> ASIDs ===> OPTIONS ===> SUBS===>

CTRACE COMP(SYSOMVS) SHORT OPTIONS((SYSID(1)))

ENTER = update CTRACE definition.  END/PF3 = return to previous panel.
S = start CTRACE.  R = reset all fields.
```

Examples of subcommands to format a SYSOMVS trace

- Example 1: CTRACE subcommand requesting SEARCH option

  The example requests the SEARCH option to search every CTRACE entry, starting at the offset specified by *offset*, for the value specified by *value*.

  ```
  CTRACE COMP(SYSOMVS) FULL OPTIONS((SEARCH(x'offset',x'value'))
  ```

- Example 2: CTRACE subcommand requesting SCCOUNTS option

  The example requests the SCCOUNTS option to count the number of syscalls from within the trace.

  ```
  CTRACE COMP(SYSOMVS) FULL OPTIONS((SCCOUNTS))
  ```

Output from a SYSOMVS trace

"SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL" on page 455 is an example of SYSOMVS component trace records formatted with the CTRACE COMP(SYSOMVS) FULL subcommand.

SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL

```
COMPONENT TRACE FULL FORMAT
COMP(SYSOMVS)
**** 05/25/1999

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY1       XCF       0D890407  18:14:14.551107  XCF BUFFER I/O TRACE

    ASID..0025      USERID....WELLIE1   STACK@....2566DF18
    TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
   +0000  E2C5D5C4  80180101  02000001  000A0002  | SEND............ |
   +0010  B2DBC852  285F5AC7  7BA70500  403E3000  | ..H..¬!G#x.. ... |
   +0020  01FF0006  00008178  C6000000            | ......a.F...     |
SY1       XCF       0D6F0401  18:14:14.551325  NXMSO-->XCF MESSAGE OUT

    ASID..0025      USERID....WELLIE1   STACK@....2566DF18
    TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
   +0000  E2E8D5C3  80A001D3  0A010014  01170BD4  | SYNC...L.......M |
   +0010  0013C000  02000001  000A0002  00004C4B  | ..{...........<. |
   +0020  7BA70500  000080E0  00000000  00000000  | #x.....\........ |
   +0030  0D6F0000  00030025  009E04A0  0000000B  | .?.............. |
   +0040  00000000  2538E980  01000000            | ......Z.....     |
SY1       XCF       0D690402  18:14:14.554457  NXMSG-->XCF MESSAGE SRB EXIT

    ASID..000E      USERID....OMVS      STACK@....25385F28
    TCB...00000000  EUID......00000000  SYSCALL...00000000
```

```
    +0000  D9C5E2D7  B4600101  009D6C68  00000080  | RESP.-....%..... |
    +0010  00030000  0A010014  01170BD4  0013402C  | ...........M.. . |
    +0020  02000001  000A0002  00000118  01FF0006  | ................ |
    +0030  00300098  24C02910  00008000  00000000  | ...q.{.......... |
    +0040  00000000  00000000  00000000            | ............     |
 SY1       XCF       0D6F0401  18:14:14.554513  NXMSO-->XCF MESSAGE OUT

     ASID..0025      USERID....WELLIE1   STACK@....2566DF18
     TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
    +0000  C6D9C5C5  10000100  00000000  00000000  | FREE............ |
    +0010  00000000  00000000  00000000  00000000  | ................ |
    +0020  00000000  00000000  00000000  00000000  | ................ |
    +0030  0D6F0000  00000000  00000000  00000000  | .?.............. |
    +0040  00000000  2538E980  00000000            | ......Z.....     |
 SY2       XCF       0D690402  18:14:14.553698  NXMSG-->XCF MESSAGE SRB EXIT

     ASID..000E      USERID....OMVS      STACK@....25389F28
     TCB...00000000  EUID......00000000  SYSCALL...00000000
    +0000  D9C5C3E5  D4600201  009E04A0  002580E0  | RECVM-.........\ |
    +0010  00030000  0A010014  01170BD4  0013C000  | ...........M..{. |
    +0020  01000001  000A0001  00008178  01FF0006  | ..........a..... |
    +0030  40060098  80000009  00000000  00000000  |  ..q............ |
    +0040  00000000  00000000  4F4F4F4F            | ........||||     |
 SY2       XCF       0D6D0403  18:14:14.553715  NXWRK-->XCF WORKER TASK TR.

     ASID..0025      USERID....OMVS      STACK@....25CF0000
     TCB...009E04A0  EUID......0000000B
     FCODE.0003      SYSNAME...SY1
    +0000  E6D6D9D2  3000022C  009D6C68  0A010014  | WORK......%..... |
    +0010  01170BD4  0013C02C  01000001  000A0001  | ...M..{......... |
    +0020  01FF0006  40060098  000080E0  009E04A0  | .... ..q...\.... |
    +0030  00250003  00000000  00000000  00000000  | ................ |
    +0040  00000000                                | ....             |
 SY2       XCF       0D890407  18:14:14.553881  XCF BUFFER I/O TRACE

     ASID..0025      USERID....OMVS      STACK@....25CF0000
     TCB...009E04A0  EUID......0000000B
     FCODE.0003      SYSNAME...SY1
    +0000  E2C5D5C4  80180101  01000001  000A0001  | SEND............ |
    +0010  B2DBC852  29114142  7F636AD8  401FB000  | ..H....."..Q ... |
    +0020  01FF0006  00000118  C6000000            | ........F...     |
 SY2       XCF       0D6F0401  18:14:14.554039  NXMSO-->XCF MESSAGE OUT

     ASID..0025      USERID....OMVS      STACK@....25CF0000
     TCB...009E04A0  EUID......0000000B
     FCODE.0003      SYSNAME...SY1
    +0000  D9C5E2D7  202002D3  0A010014  01170BD4  | RESP...L.......M |
    +0010  0013402C  01000001  000A0001  00002BBC  | .. ............. |
    +0020  7F636AD8  00000080  00000000  00000000  | "..Q............ |
    +0030  0D6F0000  00030000  009D6C68  00000000  | .?........%..... |
    +0040  253A4088  00000000  00000000            | .. h........     |
```

SY1 trace flow

Figure 144 on page 457 and Figure 145 on page 457 contain the SY1 trace information found in “SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL” on page 455.

Figure 144 on page 457 describes the CTRACE entries generated by the BPXNXMSO processing on the client side. The ASID / TCB highlighted describe the client making the request.

The most important information is the Unique Request-ID (as noted with an asterisk (*)). This is used to track a request from the client to through the server, and back again.

Two separate trace entries are provided. One states that a message has been entered into a block of messages, and the other states that the block has been written. The buffer address (as noted with an @) is used to cross reference these two trace entries.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY1       XCF       0D890407  18:14:14.551107  NXFST-->WRITE XCF BUFFER

   #ASID..0025      USERID....WELLIE1   STACK@....2566DF18
   #TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
   +0000  E2C5D5C4  80180101  02000001  000A0002  | SEND............ |
   +0010  B2DBC852  285F5AC7 @7BA70500  403E3000  | ..H..¬!G#x.. ... |
   +0020  01FF0006  00008178  C6000000            | ......a.F...     |

SY1       XCF       0D6F0401  18:14:14.551325  NXMSO-->XCF MESSAGE OUT

   #ASID..0025      USERID....WELLIE1   STACK@....2566DF18
   #TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
   +0000  E2E8D5C3  80A001D3 !0A010014 *01170BD4  | SYNC...L.......M |
   +0010 $0013C000  02000001  000A0002  00004C4B  | ..{...........<. |
   +0020 @7BA70500  000080E0  00000000  00000000  | #x.....\........ |
   +0030  0D6F0000  00030025  009E04A0  0000000B  | .?.............. |
   +0040  00000000  2538E980  01000000            | ......Z.....     |

# - ASID / TCB of requester    @ - Buffer address containing request
$ - Block #, Index into NXRQ   ! - HFS function being requested
* - Unique Request-ID = System number w/ 3byte seq#
```

Figure 144. SY1 Trace Flow: Part 1

Figure 145 on page 457 describes the response arriving on the client system. First, the XCF SRB (BPXNXMSG) processes the incoming response to cause the client task to be activated. And then, the target task (no longer remapped) wakes up, and, in this example, explicitly frees the resources that were allocated to it as part of the XCF message processing.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY1       XCF       0D690402  18:14:14.554457  NXMSG-->XCF MESSAGE SRB EXIT

    ASID..000E      USERID....OMVS      STACK@....25385F28
    TCB...00000000  EUID......00000000  SYSCALL...00000000
   +0000  D9C5E2D7  B4600101  009D6C68  00000080  | RESP.-....%..... |
   +0010  00030000 !0A010014 *01170BD4 $0013402C  | ...........M.. . |
   +0020  02000001  000A0002  00000118  01FF0006  | ................ |
   +0030  00300098  24C02910  00008000  00000000  | ...q.{.......... |
   +0040  00000000  00000000  00000000            | ............     |

SY1       XCF       0D6F0401  18:14:14.554513  NXMSO-->XCF MESSAGE OUT

   #ASID..0025      USERID....WELLIE1   STACK@....2566DF18
   #TCB...009E04A0  EUID......0000000B  SYSCALL...00000036
   +0000  C6D9C5C5  10000100  00000000  00000000  | FREE............ |
   +0010  00000000  00000000  00000000  00000000  | ................ |
   +0020  00000000  00000000  00000000  00000000  | ................ |
   +0030  0D6F0000  00000000  00000000  00000000  | .?.............. |
   +0040  00000000  2538E980  00000000            | ......Z.....     |

# - ASID / TCB of requester
$ - Block #, Index into NXRQ   ! - HFS function being requested
* - Unique Request-ID
```

Figure 145. SY1 Trace Flow: Part 2

SY2 trace flow

Figure 146 on page 458 and Figure 147 on page 458 contain the SY2 trace information found in "SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) FULL" on page 455.

Figure 146 on page 458 describes the server side XCF SRB processing by first queuing the request (BPXNXMSG), and then having a worker task pick up that piece of work for subsequent processing (BPXNXWRK).

As noted with an *, the Request-ID is used to cross reference the individual trace entries.

When a SYSNAME field is included in a trace entry, the ASID / TCB information actually describes the client side requester information. The SYSNAME field describes the originating system. The highlighted field with an & is the TCB address of the worker task resident in the server system.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY2       XCF       0D690402  18:14:14.553698  NXMSG-->XCF MESSAGE SRB EXIT

    ASID..000E      USERID....OMVS      STACK@....25389F28
    TCB...00000000  EUID......00000000  SYSCALL...00000000
   +0000  D9C5C3E5  D4600201  009E04A0  002580E0  | RECVM-.........\ |
   +0010  00030000 !0A010014 *01170BD4 $0013C000  | ...........M..{. |
   +0020  01000001  000A0001  00008178  01FF0006  | ..........a..... |
   +0030  40060098  80000009  00000000  00000000  |  ..q............ |
   +0040  00000000  00000000  4F4F4F4F            | ........||||     |

SY2       XCF       0D6D0403  18:14:14.553715  NXWRK-->XCF WORKER TASK TRACE

   #ASID..0025      USERID....OMVS      STACK@....25CF0000
   #TCB...009E04A0  EUID......0000000B
    FCODE.0003      SYSNAME...SY1
   +0000  E6D6D9D2  3000022C &009D6C68 !0A010014  | WORK......%..... |
   +0010 *01170BD4 $0013C02C  01000001  000A0001  | ...M..{......... |
   +0020  01FF0006  40060098  000080E0  009E04A0  | .... ..q...\.... |
   +0030  00250003  00000000  00000000  00000000  | ................ |
   +0040  00000000                                | ....             |

# - ASID / TCB of requester    & - Real OMVS resident worker TCB
$ - Block #, Index into NXRQ   ! - HFS function being requested
* - Unique Request-ID          % - Indicates ASID/TCB remapped to requester
```

Figure 146. SY2 Trace Flow: Part 1

Figure 147 on page 458 describes the response arriving on the client system. First the XCF SRB (BPXNXMSG) processes the incoming response to cause the client task to be activated. And then the target task (no longer remapped) wakes up, and in this case explicitly frees the resources that were allocated to it as part of the XCF message processing.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY2       XCF       0D890407  18:14:14.553881  XCF BUFFER I/O TRACE

   #ASID..0025      USERID....OMVS      STACK@....25CF0000
   #TCB...009E04A0  EUID......0000000B
    FCODE.0003     %SYSNAME...SY1
   +0000  E2C5D5C4  80180101  01000001  000A0001  | SEND............ |
   +0010  B2DBC852  29114142 @7F636AD8  401FB000  | ..H....."..Q ... |
   +0020  01FF0006  00000118  C6000000            | ........F...     |

SY2       XCF       0D6F0401  18:14:14.554039  NXMSO-->XCF MESSAGE OUT

   #ASID..0025      USERID....OMVS      STACK@....25CF0000
   #TCB...009E04A0  EUID......0000000B
    FCODE.0003     %SYSNAME...SY1
   +0000  D9C5E2D7  202002D3 !0A010014 *01170BD4  | RESP...L.......M |
   +0010 $0013402C  01000001  000A0001  00002BBC  | .. ............. |
   +0020 @7F636AD8  00000080  00000000  00000000  | "..Q............ |
   +0030  0D6F0000  00030000  009D6C68  00000000  | .?........%..... |
   +0040  253A4088  00000000  00000000            | .. h........     |

# - ASID / TCB of requester    @ - Buffer address containing request
$ - Block #, Index into NXRQ   ! - HFS function being requested
* - Unique Request-ID          % - Indicates ASID/TCB remapped to requester
```

Figure 147. SY2 Trace Flow: Part 2

Control block trace

Figure 148 on page 459 is an example of SYSOMVS component trace records requested with OPTIONS(CBTR(PPRP,0,4)) to trace a four byte field at offset zero in the PPRP control block. The trace data was then formatted with the CTRACE COMP(SYSOMVS) subcommand:

```
SY1 SYSCALL   0F080001  20:06:58.662146  STANDARD SYSCALL ENTRY TRACE

 ASID..0020      USERID....IBMUSER   STACK@....25D58010
 TCB...008BF088  EUID......00000000  SYSCALL...00000019
 +0000  00000019  00000000  D1C3E2C5  8C0000F4  | ........JCSE...4 |
 +0010  0000000C  00000000  80D1AE06  25D596F4  | .........J...No4 |
 +0020  25D596E4  00000085  00000000  7F5FF0A8  | .NoU...e...."¬0y |
 +0030  00000006  2501BB50  00D1AFA4  7F5FF0E8  | .......&.J.u"¬0Y |
 +0040  7F5FF0CC  7F5FF0D0  FF5FF0D4            | "¬0."¬0}.¬0M     |
 +0050  D7D7D9D7  00000004  D7D7D9D7 00000000    PPRP.........PPRP

   SY1  SYSCALL   0F080002  20:06:58.662171  STANDARD SYSCALL EXIT TRACE

 ASID..0020      USERID....IBMUSER   STACK@....25D58010
 TCB...008BF088  EUID......00000000  SYSCALL...00000019
 +0000  00000019  00000000  D1C3E2C5  8C000000  | ........JCSE.... |
 +0010  0000000A  00000000  00000000  00000002  | ................ |
 +0020  00000002  D7D7D9D7  00000004  D7D7D9D7   PPRP..........PPRP
        00000000
```

Figure 148. Control block trace output

Figure 149 on page 459 is an example of SYSOMVS component trace records formatted with the CTRACE COMP(SYSOMVS) SHORT subcommand.

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSOMVS)
**** 06/17/92

MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
--------  --------  ---------------  -----------

SYSCALL   0F080002  22:02:42.314264  STANDARD SYSCALL EXIT TRACE
SYSCALL   0F080001  22:02:42.821156  STANDARD SYSCALL ENTRY TRACE
PROCESS   0B080007  22:02:42.821219  PROCESS LATCH OBTAIN
PROCESS   0B08000A  22:02:42.821256  PROCESS LATCH-ON THE WAY OUT
FILE      05700103  22:02:42.821324  TRACE CALL TO VN_RDWR
CHARS     05A90503  22:02:42.821398  TRACE CHARSPEC CALL TO DEVICE DR
PROCESS   0B080008  22:02:42.821452  PROCESS LATCH RELEASE REQUEST
PROCESS   0B08000A  22:02:42.821472  PROCESS LATCH-ON THE WAY OUT
DEVPTY    0223E005  22:02:42.821530  MASTER READ BEGIN
DEVPTY    0223E008  22:02:42.821566  MASTER READ END
PROCESS   0B080007  22:02:42.822182  PROCESS LATCH OBTAIN
PROCESS   0B08000A  22:02:42.822206  PROCESS LATCH-ON THE WAY OUT
CHARS     05A90603  22:02:42.822253  TRACE DEVICE DRIVER READ RETURN
FILE      05700203  22:02:42.822506  TRACE RETURN FROM VN_RDWR
```

Figure 149. SYSOMVS component trace formatted with CTRACE COMP(SYSOMVS) SHORT

The output from a SYSOMVS trace using the SCCOUNTS option has 2 formats, shown in Figure 150 on page 459 and Figure 151 on page 460.

```
SYSCALL#     SYSCALL NAME      COUNT      FREQUENCY/SEC
--------     ------------      -----      -------------
    F        BPX1CHO            5000           nnn
   2F        BPX1STA             150           nnn
```

Figure 150. SCCOUNT Function Displaying SYSCALL Frequency

Figure 150 on page 459 is sorted by frequency, with the highest values appearing at the top of the list. SYSCALL# is the hexadecimal number of the syscall. FREQUENCY/SEC is the number of times the syscall has been invoked within the interval.

```
FuncCode    FuncCode Name     Count      Functions/Sec
--------    ------------      -----      -------------
00001001    MntCatchup         76           0.0593
00001010    GetPathName        60           0.0468
00000012    UnQuiesce          38           0.0296
```

Figure 151. SCCOUNT Function Displaying Function Code Frequency

Figure 151 on page 460 is sorted by frequency, with the highest values appearing at the top of the list. FuncCode is the hexadecimal number of the function code. Functions/Sec is the number of times the function code has been invoked within the interval.

The output from a SYSOMVS trace using the KERNINFO option is shown in Figure 152 on page 460. The syscall function name (FCN) and the process ID (PID) are shown on the first line of the trace entry.

```
FCN...open          SYSCALL...BPX1OPN  PID...00010006  MODULE...BPXJCPC
SY1       SYSCALL   0F080001  17:34:03.106875  STANDARD SYSCALL ENTRY TRACE

       ASID..0020      USERID....MEGA      STACK@....26E09068
     TCB...008C1470  EUID......00000000  PID.......00010006
   +0000  00000026  00000000  D1C3E2C5  8C000048  | ........JCSE.... |
   +0010  8002000E  00000000  863CED16  02290200  | ........f....... |
   +0020  60000000  269D8CA0  00000000  25B36828  | -............... |
   +0030  00000007  00000010  61A39497  614BA288  | ......../tmp/.sh |
   +0040  6D8889A2  A39699A8  00000000  0000048B  | _history........ |
   +0050  00000180  FFFFFFFF  0000006F  5B4C0002  | ...........?$<.. |
   +0060  00000000  00000000  00000000  00000000  | ................ |
   +0070  00000000  00000000  00000000  00000000  | ................ |
   +0080  00000000  00000000  00000000            | ............     |
  FCN...open          SYSCALL...BPX1OPN  PID...00010006  MODULE...BPXJCPC
SY1       SYSCALL   0F080002  17:34:03.106876  STANDARD SYSCALL EXIT TRACE

            ASID..0020      USERID....MEGA      STACK@....26E09068
          TCB...008C1470  EUID......00000000  PID.......00010006
   +0000  00000026  00000000  D1C3E2C5  8C000000  | ........JCSE.... |
   +0010  8002000B  00000000  FFFFFFFF  0000006F  | ...............? |
   +0020  5B4C0002
```

Figure 152. CTRACE COMP(SYSOMVS) FULL OPTIONS((KERNINFO))

SYSOPS component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSOPS component trace for the operations services component (OPS).

Information	**For SYSOPS:**
Parmlib member	CTnOPSxx. Default member: CTIOPS00 specified in CONSOLxx member
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	In CTnOPSxx or REPLY for TRACE command

Information	For SYSOPS:
Buffer	• Default: 2MB • Range: 64KB - 16MB • Size set by: CTnOPSxx member or REPLY for TRACE CT command • Change size after IPL: Yes, when restarting a trace after stopping it • Location: Console address space (private)
Trace records location	Address-space buffer, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSOPS)
Trace format OPTIONS parameter	Yes

Note: To get a complete dump for OPS, request also the NUC, CSA, and SQA.

Requesting a SYSOPS trace

Specify options for requesting a SYSOPS component trace in a CTnOPSxx parmlib member or in the reply for a TRACE CT command.

CTnOPSxx parmlib member

The following table indicates the parameters you can specify on a CTnOPSxx parmlib member.

Parameters	Allowed on CTnOPSxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM supplies the CTIOPS00 parmlib member, which defines the component trace to the system and establishes a trace buffer of 2M. The contents of CTIOPS00 are:

```
TRACEOPTS
      ON
      BUFSIZE(2M)
```

IBM does not supply a sample CONSOL00 parmlib member. Create a CONSOLxx parmlib member and specify CON=xx in the IEASYSxx parmlib member. Specify CTIOPS00 as the default on the CTRACE parameter of the INIT statement of CONSOLxx.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, nnnnK, nnnnM, or OFF	One is required. The buffer size can be changed only when the trace is OFF or the trace is ON.
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter for the CTnOPSxx parmlib member and reply for a TRACE command, in alphabetical order, are:

COMMAND
Traces events related to command processing.

CONSDATA
Traces events related to console state changes.

LOGGING
Traces events related to Operlog and Syslog processing.

MCACHE
Traces events related to the message cache.

MESSAGES
This option includes the WTO, MSGDLVRY, MCACHE and LOGGING options.

MSG=*msgid*
Traces processing of a specific message. It REQUIRES either one of the following OPTIONS: MESSAGES, WTO, MSGDLVRY, MCACHE or LOGGING. *msgid* is 1-10 alphanumeric characters in length indicating the message id that will be traced.

MSGDLVRY
Traces events for WQE processing, MCS console message queueing, and extended MCS console message processing.

RECOVERY
Traces recovery events.

SERIALIZ
Traces latch serialization events.

SYSPLEX
Traces events for XCF signalling, sysplex serialization services, sysplex clean-up processing, and the manipulation of various queues.

WTO
Traces the effects of MPFLSTxx, the user exits, and the SSI on message content and attributes.

These additional options increase the number of trace records the system collects and can slow system performance. Each time you change the trace options, you must respecify any options you want to keep in effect from the last trace.

Note: Before you use the MESSAGES, WTO, MSGDLVRY, MCACHE or LOGGING options, you should do the following:

- Increase the buffer size
- Start and connect the external writer.

This is especially important if you are starting tracing at IPL This might not be necessary if you are tracing a message ID since you would only be cutting records for the particular message.

Examples of requesting SYSOPS traces

- Example 1: Activating trace options while system is running

  Create parmlib member CTIOPS01, specifying the following parameters. Assume that procedure OPSWTR is in SYS1.PROCLIB.

```
       TRACEOPTS
             WTRSTART(OPSWTR)
             ON
             WTR(OPSWTR)
             OPTIONS('MESSAGES','MSG=IEE136I')
              ASID(1,2,3)
             JOBNAME(PAYROLL)
             BUFSIZE(2M)
```

- Example 2: Specifying trace options on a REPLY command

  The example requests the same trace as Example 1, but specifies all options on the REPLY.

```
       trace ct,wtrstart=opswtr
       trace ct,2m,comp=sysops
```

  When the system prompts you for the trace options, enter the following command, replacing *id* with the reply identifier:

```
reply 27,wtr=opswtr,options=(messages,msg=IEE136I),asid=(1,2,3),
jobname=(payroll),end
```

- Example 3: CTnOPSxx member used at IPL

  The member requests the SYSPLEX option, doubles the default buffer size, and limits the tracing to ASID 1 and JOBNAME JOB1.

```
       TRACEOPTS
             ON
             OPTIONS('SYSPLEX')
             BUFSIZE(4M)
             ASID(1)
             JOBNAME(JOB1)
```

Formatting a SYSOPS trace

Format the trace with an IPCS CTRACE COMP(SYSOPS) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records. The options for formatting a SYSOPS trace are:

COMMAND
  Traces events related to command processing.

CONSDATA
  Traces events related to console state changes.

LOGGING
Traces events related to Operlog and Syslog processing.

MCACHE
Traces events related to the message cache.

MESSAGES
This option includes the WTO, MSGDLVRY, MCACHE and LOGGING options.

MSG=*msgid*
Traces processing of a specific message. It REQUIRES either one of the following OPTIONS: MESSAGES, WTO, MSGDLVRY. MCACHE or LOGGING. *msgid* is 1-10 alphanumeric characters in length indicating the message id that will be traced.

MSGDLVRY
Traces events for WQE processing, MCS console message queueing, and extended MCS console message processing.

RECOVERY
Traces recovery events.

SERIALIZ
Traces latch serialization events.

SYSPLEX
Traces events for XCF signalling, sysplex serialization services, sysplex clean-up processing, and the manipulation of various queues.

WTO
Traces the effects of MPFLSTxx, the user exits, and the SSI on message content and attributes.

Output from a SYSOPS trace

Figure 153 on page 464 is an example of OPS component trace records formatted with the CTRACE COMP(SYSOPS) SHORT subcommand.

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSOPS)
**** 05/20/93

MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
--------  --------  ---------------  -----------

INITMSG   00000001  14:41:06.918883  Message prior to MPF processing
POSTMPF   00000002  14:41:06.919198  Message after MPF processing
POSTEXIT  00000003  14:41:06.919595  Post Message Exit
POSTSSI   00000004  14:41:06.920118  Post Subsystem Interface
INITMSG   00000001  14:41:06.930088  Message prior to MPF processing
POSTMPF   00000002  14:41:06.930405  Message after MPF processing
POSTEXIT  00000003  14:41:06.930803  Post Message Exit
POSTSSI   00000004  14:41:06.931267  Post Subsystem Interface
INITMSG   00000001  14:41:06.931637  Message prior to MPF processing
POSTMPF   00000002  14:41:06.931934  Message after MPF processing
```

Figure 153. Example: OPS component trace records formatted with CTRACE COMP(SYSOPS) SHORT subcommand

Figure 154 on page 465 is an example of OPS component trace records formatted with the CTRACE COMP(SYSOPS) FULL subcommand.

```
COMPONENT TRACE FULL FORMAT
COMP(SYSOPS)
**** 05/20/93

MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
--------  --------  ---------------  -----------

INITMSG   00000001  14:41:06.918883  Message prior to MPF processing
     TR_GROUP WTO               Write to Operator Services
     HOMEASID 0001     HOMEJOBN *MASTER*
     REQSASID 0001     REQSJOBN UNKNOWN
     CPU_ADDR 0001
     KEY..... 0001
   +0000  LKP...... 00000000  TXTLN.... 005F      RTCT..... 0000
   +000A  USE...... 0000      PAD...... 40        TS....... 10.41.06
   +00D1  TS2...... .91       PAD1..... 40        JOBNM....
   +001E  PAD2..... 40        TXT......  ITT038I ALL OF THE TRANSACTIONS REQU
   +0044            ESTED VIA THE TRACE CT COMMAND WERE
   +009E  TXTL..... 40        PAD3..... 00        XA....... 01
   +00A1  ASID..... 0001      AVAIL.... 50        TCB...... 008D57C0
   +00A8  SYSID.... 03        SEQN..... 00049D    MCSF1.... C0
   +00AD  MCSF2.... 10        MSGT1.... 00        ROUT1.... 00
   +00B1  ROUT2.... 00        CHAR1.... 00        FLG3..... 00
   +00B4  UCMID.... 0A        FLG1..... 04        RPYID.... 0000
   +00B8  DC1...... 08        DC2...... 00        RSV26.... 0000
   +00BC  JSTCB.... 008D57C0  VRSN..... 05        MFLG1.... 00
   +00C2  MCSE1.... 42        MCSE2.... 00        SYSNM.... SYS2
   +00CC  DATE..... 93140     RFB1..... 00        RFB2..... 00
   +00D6  RFB3..... 00        SUPB..... 90        ML1...... 00
   +00D9  ML2...... 00        LENG..... 0160      DSQN..... 00000000
   +00E0  ERC1..... 00        ERC2..... 00        ERC3..... 00
   +00E3  ERC4..... 00        ERC5..... 00        ERC6..... 00
   +00E6  ERC7..... 00        ERC8..... 00        ERC9..... 00
   +00E9  ERC10.... 00        ERC11.... 00        ERC12.... 00
   +00EC  ERC13.... 00        ERC14.... 00        ERC15.... 00
   +00EF  ERC16.... 00        KEY...... 40404040  40404040
   +00F8  TOKN..... 00000000  CNID..... 0000000A  OJBID.... 40404040
   .
   .
   .
```

Figure 154. Example: OPS component trace records formatted with CTRACE COMP(SYSOPS) FULL subcommand

SYSRRS component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSRRS component trace for RRS.

Information	For SYSRRS:
Parmlib member	CTnRRSxx. No default member
Default tracing	Yes; minimal; unexpected events; general UR services
Trace request OPTIONS parameter	In CTnRRSxx and REPLY for TRACE command
Buffer	• Default: 1MB • Range: 1MB - 2045MB • Size set by: CTnRRSxx member or REPLY for TRACE CT command • Change size after IPL: Yes, when restarting a trace after stopping it • Location: Data space and component address space. In the REPLY for the DUMP command, specify DSPNAME=('RRS'.ATRTRACE) and SDATA=RGN.

Information	For SYSRRS:
Trace records location	Data space buffer, trace data set, trace buffers in the RRS address space
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSRRS)
Trace format OPTIONS parameter	Yes

Requesting a SYSRRS trace

Specify options for requesting a SYSRRS component trace on a CTnRRSxx parmlib member or on the reply for a TRACE CT command.

To change options or the buffer size, you have to stop the trace and restart it with the new options, buffer size, or both.

CTnRRSxx parmlib member

The following table indicates the parameters you can specify on a CTnRRSxx parmlib member.

Parameters	Allowed on CTnRRSxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	Yes
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

A CTnRRSxx parmlib member is optional. If not specified, the SYSRRS component trace runs a minimal trace beginning when the RRS component is started and ending when the component is stopped.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on a TRACE CT command and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, nnnnK, nnnnM, or OFF	One is required
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The OPTIONS parameter for the CTnRRSxx parmlib member or reply for a TRACE command contains keyword subparameters. These subparameters allow you to control the information that RRS component trace collects. The first subparameter, EVENTS, specifies the events to be traced; the other subparameters act as filters to screen the events with up to three checks. An event must pass all checks for component trace to generate a trace record. The order for the checks is:

1. That the event is specified.
2. That the event matches, in an OR check, one of the following filters:
   - The address space ID (ASID) in the REPLY command or CTnRRSxx TRACEOPTS statement
   - The job name (JOBNAME) in the REPLY command or CTnRRSxx TRACEOPTS statement
   - The user ID (USERID) in the OPTIONS parameter
   - The logical work unit ID (LUWID) in the OPTIONS parameter
3. That the event matches a resource manager name (RMNAME) in the OPTIONS parameter.

```
OPTIONS=([EVENTS(event[,event]...)]
         [USERID(userid[,userid]...)]
         [RMNAME(rmname[,rmname]...)]
         [LUWID((luwid)[,(luwid)]...)])
         [EID((eid)[,(eid)]...)
```

Note: In the REPLY to the TRACE CT command, separate the options by one or more blanks.

EVENTS(*event*[,*event*]...)
Indicates the events to be traced. **An EVENTS parameter is required if any other options are specified.** The events, in alphabetical order, are:

ALL
Traces all events. Component trace ignores other events, if specified.

CONTEXT
Traces calls to context services.

EXITS
Traces events related to running the RRS exit routines provided by the resource managers.

FLOW
Traces entry into and exit from RRS entry points.

LOGGING
Traces events related to logging data by RRS.

RESTART
Traces events related to RRS initialization and restart.

RRSAPI
Traces events related to the application programming interface, which consists of calls to the Application_Commit_UR service and the Application_Backout_UR service.

STATECHG
Traces events involving changes in the state of units of recovery (URs).

URSERVS
Traces general events related to services for a UR (traced by default).

USERID(*userid*[,*userid*]...)
Specifies 1 to 16 user IDs as filters for specified events. The system traces only events relating to the user IDs.

RMNAME(*rmname*[,*rmname*]...)
Specifies 1 to 16 resource manager names as filters for specified events. For trace events sensitive to the resource manager name, the system traces only events relating to the resource managers.

LUWID((*luwid*)[,(*luwid*)]...)
Specifies 1 to 16 logical unit of work identifiers (LUWIDs) as filters for specified events. The system traces only events relating to the LUWIDs. Each *luwid* consists of:

```
netid.luname[,instnum][,seqnum]
```

Component trace ignores leading and trailing blanks.

netid.luname
Specifies the network ID and the local logical unit name. These portions of the LUWID are required.

You can use an asterisk (*) as a wildcard character as:

- The last character in the *netid*, the *luname*, or both
- The only character in the either the *netid* or the *luname*, but not as the only character in both

instnum
Specifies the instance number as a 1 - 12 hexadecimal integer. You can omit leading zeros.

seqnum
Specifies the sequence number as a 1 - 4 hexadecimal integer. You can omit leading zeros.

Examples of LUWIDs are:

```
(A.B,5,1)
(A.B,5)
(A.B,,1)
(A.B)
(A.*,5,1)
(A*.B*)
```

EID((*eid*)[,(*eid*)]...)
Specifies 1 to 16 Enterprise identifiers (EIDs) as filters for specified events. The system traces only events relating to the EIDs. Each *eid* consists of:

```
[tid][,gtid]
```

You can omit leading zeros. Component trace ignores leading and trailing blanks.

tid
Specifies the 4-byte hexadecimal transaction identifier (TID).

gtid
Specifies the 8-40 byte hexadecimal global transaction identifier (GTID).

You can obtain the EID for a UR by using the RRS ISPF panels to browse the RRS log streams. The Retrieve_Work_Identifier service can also return an EID.

Examples of EIDs are:

```
(1,C)
(,C)
(1)
```

Examples of requesting SYSRRS traces

In Figure 155 on page 469, the member requests context services events filtered by the user ID JONES and requests a 1024KB buffer.

```
TRACEOPTS
   ON
   OPTIONS('EVENTS(CONTEXT) USERID(JONES)')
   BUFSIZE(1024K)
```

Figure 155. Example: CTnRRSxx member requests context services events

Figure 156 on page 469 is an example of using the TRACE command to request the same trace shown in Figure 155 on page 469.

```
trace ct,off,comp=sysrrs
trace ct,1024K,comp=sysrrs
* 17 ITT006A ...
reply 17,options=('events(context) userid(jones)'),end
```

Figure 156. Example: TRACE command requests context services events

Formatting a SYSRRS trace

Format the trace with an IPCS CTRACE COMP(SYSRRS) subcommand. Its OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the formatting options to narrow down the records displayed so that you can more easily locate any errors. If you specify no options on the CTRACE subcommand, IPCS displays all the trace records.

You can specify one or more OPTIONS subparameters. If you specify no OPTIONS subparameters, all trace records are formatted. A trace record must match all specified OPTIONS subparameters to be formatted.

```
OPTIONS=((option[,option]...))

option is one of the following:

         LUWID(luwid)
         EID(eid)
         RMNAME(rmname)
         URID(urid)
         USERID(userid)
```

LUWID(*luwid*)
: Specifies one of the logical unit of work identifiers (LUWIDs) specified when the trace was generated.

EID(*eid*)
: Specifies one of the Enterprise identifiers (EIDs) specified when the trace was generated. Specify *eid* as:

- `tid`
- `tid,gtid`

Or, if you omitted `tid` when you specified the identifier: `*,gtid`

RMNAME(*rmname*)
: Specifies one of the resource manager names specified when the trace was generated.

URID(*urid*)
: Specifies a UR identifier. The URID is a 16-byte character string returned to the resource manager by one of the following callable services: Change_Interest_Type, Express_UR_Interest, Retain_Interest,

Retrieve_UR_Interest, or Retrieve_UR_Data. The URID is saved in the resource manager log; you can obtain it through an RRS panel.

USERID(*userid*)
Specifies one of the user IDs specified when the trace was generated. Note that USERID does not filter out trace records in which the user ID is blank.

The following is an example of how to specify an IPCS CTRACE OPTIONS parameter:

```
OPTIONS=((RMNAME(datamgr),USERID(jjones)))
```

Output from a SYSRRS trace

Fields that do not contain printable characters are filled with asterisks (*). The value is shown in hexadecimal on a separate line.

Note: RRS provides the same report for the SUMMARY and FULL parameters on the CTRACE subcommand.

"CTRACE COMP(SYSRRS) SHORT subcommand output" on page 470 is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) SHORT subcommand.

CTRACE COMP(SYSRRS) SHORT subcommand output

```
 COMPONENT TRACE SHORT FORMAT
 COMP(SYSRRS)
 **** 01/20/1997

 SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
 -------   --------  --------  ---------------  -------------

 SY1       COMPERR   00000000  07:53:43.615114  Resource Recovery Services
 SY1       FLOW      0801FFFE  07:53:43.615114  ATRB1PCT EXIT
 SY1       FLOW      0801FFFF  07:53:43.619823  ATRB1PCT ENTRY
 SY1       FLOW      0201FFFF  07:53:43.619867  ATRU1EIN ENTRY
 SY1       CONTEXT   02010002  07:53:43.620027  CTXMEINT call
 SY1       URSERVS   02010001  07:53:43.620699  ATREINT invoked
 SY1       FLOW      0201FFFE  07:53:43.620887  ATRU1EIN EXIT
.
.
.
```

"CTRACE COMP(SYSRRS) SUMMARY or FULL output" on page 470 is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) SUMMARY or FULL subcommands.

CTRACE COMP(SYSRRS) SUMMARY or FULL output

```
 COMPONENT TRACE FULL FORMAT
 COMP(SYSRRS)
 **** 01/20/1997

 SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
 -------   --------  --------  ---------------  -------------

 SY1       COMPERR   00000000  07:53:43.615114  Resource Recovery Services
    FFF0003E  7EF6D000  0001FE6E  00000000  | .0..=6}....>.... |
    000A0000  00000000  00020000  | ................ |
    0101001F  00000000  7F04D000  01000000  | ........".}..... |
    00000000  00000000  0000      | ..............   |
 SY1       FLOW      0801FFFE  07:53:43.615114  ATRB1PCT EXIT
      HASID.... 00AA      HJOBNAME. APPL1AS
      SASID.... 00A3      SJOBNAME. RMAS1
      USERID... *         RMNAME...
      URID..... AE18AB4E  7ED2CF90  0000012B  01010000
      TID...... 000000000000                GTID..... 00000000
                00000000  00000000  00000000
      LUWID.... 00000000  00000000  00000000  0000
        NETNAME.. ********  LUNAME... ********
        INSTNUM.. ******    SEQNUM... **
 SY1       FLOW      0801FFFF  07:53:43.619823  ATRB1PCT ENTRY
      HASID.... 00AA      HJOBNAME. APPL1AS
```

```
       SASID.... 00A3      SJOBNAME. RMAS1
       USERID... *         RMNAME...
       URID..... AE18AB4E  7ED2CF90  0000012B  01010000
       TID...... 000000000000                  GTID..... 00000000
                 00000000  00000000  00000000
       LUWID.... 00000000  00000000  00000000  0000
         NETNAME.. ********  LUNAME... ********
         INSTNUM.. ******    SEQNUM... **
  SY1       FLOW      0201FFFF  07:53:43.619867  ATRU1EIN ENTRY
       HASID.... 00AA      HJOBNAME. APPL1AS
       SASID.... 00A3      SJOBNAME. RMAS1
       USERID... *         RMNAME...
       URID..... AE18AB4E  7ED2CF90  0000012B  01010000
       TID...... 000000000000                  GTID..... 00000000
                 00000000  00000000  00000000
       LUWID.... 00000000  00000000  00000000  0000
         NETNAME.. ********  LUNAME... ********
         INSTNUM.. ******    SEQNUM... **
 .
 .
 .
```

"CTRACE COMP(SYSRRS) TALLY output" on page 471 is an example of SYSRRS component trace records formatted with the CTRACE COMP(SYSRRS) TALLY subcommands.

CTRACE COMP(SYSRRS) TALLY output

```
  COMPONENT TRACE TALLY REPORT
  COMP(SYSRRS)
      TRACE ENTRY COUNTS AND AVERAGE INTERVALS (IN MICROSECONDS)

  FMTID    COUNT       INTERVAL     MNEMONIC DESCRIBE
  -------- ----------- ------------ -------- --------------------------------
  00000000           7   23,774,056 COMPERR  Resource Recovery Services
  02010001          46    2,398,672 URSERVS  ATREINT invoked
  02010002          46    2,398,670 CONTEXT  CTXMEINT call
  02018001           6   18,803,430 STATECHG UR State Change
  02018008           3   47,008,505 STATECHG UR being created
  0201FFFE          46    2,398,673 FLOW     ATRU1EIN EXIT
  0201FFFF          46    2,398,670 FLOW     ATRU1EIN ENTRY
  02030001           0              URSERVS  ATRDINT invoked
  02038009           0              STATECHG UR being destroyed
  0203FFFE           0              FLOW     ATRU1DIN EXIT
  0203FFFF           0              FLOW     ATRU1DIN ENTRY
  02050001           0              URSERVS  ATRPDUE invoked
  0205FFFE           0              FLOW     ATRU1PDU EXIT
  0205FFFF           0              FLOW     ATRU1PDU ENTRY
 .
 .
 .
```

SYSRSM component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSRSM component trace for the real storage manager (RSM).

Information	For SYSRSM:
Parmlib member	CTnRSMxx. No default member
Default tracing	No.
Trace request OPTIONS parameter	In CTnRSMxx or REPLY for TRACE command

Information	For SYSRSM:
Buffer	• Default: 3 buffers of 32 pages • 2 -7 page-fixed primary buffers, 4 - 262,144 pages per buffer 1 - 2047 MB for secondary buffers • Size set by: CTnRSMxx member or REPLY for TRACE CT command • Change size after IPL: Yes, when starting a trace • Common service area (LIKECSA) and, if specified in CTnRSMxx, high virtual private storage of the RASP address space.
Trace records location	In the trace data set, trace buffers in high common memory, and the RASP address space.
Request of SVC dump	• By DUMP or SLIP command • Through the DMPREC option on the CTnRSMxx parmlib member or on the REPLY for the TRACE CT command when RSM enters recovery processing (default) • Through the DMPOFF option of CTnRSMxx or the TRACE CT reply when SYSRSM tracing is turned off
Trace formatting by IPCS	CTRACE COMP(SYSRSM)
Trace format OPTIONS parameter	None

Requesting a SYSRSM trace

Specify options for requesting a SYSRSM component trace in a CTnRSMxx parmlib member or in the reply for a TRACE CT command.

CTnRSMxx parmlib member

The following table indicates the parameters you can specify on a CTnRSMxx parmlib member.

Parameters	Allowed on CTnRSMxx?
ON or OFF	Yes
ASID	Yes
JOBNAME	Yes. To trace all batch jobs, specify 'INIT' in the list of job names.
BUFSIZE	Yes
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

IBM provides two sample CTIRSMxx parmlib members in SYS1.PARMLIB. These are not default members.

- **CTIRSM01**: Shows how to request tracing of all RSM functions and events using the options DMPOFF, NOCOMASID, and NODMPREC.
- **CTIRSMSP**: Shows how to request address space, job name filtering, and trace request options to limit the tracing.

RSM trace data in high virtual private storage: RSM supports collecting trace data into high private memory. This storage is used as a secondary buffer data, which is first collected into buffers in fixed high common storage, and later moves out into high private memory. Having a secondary buffer is another way, besides trace data sets, to handle a large number of RSM trace records. Note that the paging activity for the secondary buffer can appear in the RSM trace records.

If you suspect that your system has a paging problem, collect the RSM trace records in page-fixed primary buffers to keep from losing records while paging. A record can be lost as the system reuses a full secondary buffer

For RSM, use BUFF in the CTnRSMxx OPTIONS parameter to specify the number of page-fixed buffers and their page sizes. In Figure 157 on page 473, the statements in CTWRSM05 specify four page-fixed buffers that are 64 pages and a secondary buffer in high private memory of 640 KB.

```
TRACEOPTS
   ON
   BUFSIZE(640K)
   OPTIONS('BUFF(4,64)')
```

Figure 157. Example: Using the CTWRSM05 parmlib member

For RSM, the BUFSIZE parameter in the CTnRSMxx parmlib member specifies the total size of the high virtual private buffers located in the RASP address space. The space in the buffer is divided evenly between the amount of installed processors.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, nnnnK, nnnnM, or OFF	One is required. *nnnnK* and *nnnnM* specify the size of the buffer iin the RASP address space (high virtual private).
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	Yes
JOBNAME	Yes. To trace all batch jobs, specify 'INIT' in the list of job names.
OPTIONS	Yes
WTR	Yes

Automatic Dump: The component requests an SVC dump when the operator stops the trace or when RSM enters recovery processing. To prevent these automatic dumps when the trace is written to a trace data set or when the operator is to request the dump, specify NODMPREC and NODMPOFF in the OPTIONS parameter in the TRACE CT command or the CTnRSMxx parmlib member.

OPTIONS parameter

The values for the OPTIONS parameter for the CTnRSMxx parmlib member and reply for a TRACE command follow.

If you turn on the SYSRSM trace without specifying any filters or options, the component trace records every RSM function and event in all address spaces and jobs. This trace collects an enormous amount of data and degrades system performance. Use the SYSRSM filters and options to limit the amount of data recorded by the component trace. Specify tracing of specific address spaces, jobs, RSM events, and RSM functions.

The RSM trace options are divided into three groups:

- Special trace options
- RSM function trace options
- RSM event trace options

Special trace options

These options set the size of the fixed RSM trace buffers, tell the trace to record common area activity, and tell the system when to dump the trace data. The options are:

BUFF=(*x*,*y*)
Specifies the number and size of the SYSRSM trace buffers, which reside in fixed high virtual common service area (LIKECSA) storage:

x
The number of buffers, from 2 to 7. The default is 3.

y
The number of pages per buffer, from 4 to 262,144. The default is 32.

The storage for each buffer is distributed between the installed processors in the system. For that reason, it is recommended to increase the buffer size on systems with a large amount of processors. The more buffers that are specified, the more often the SRB, which empties out the fixed buffers, is called (if there is a high virtual private buffer or an external writer). The larger the buffer, the greater the burst of events it can contain without losing any trace entries.

For example, if you specify BUFF=(5,10), the component trace uses five fixed trace buffers. Each buffer contains 10 pages. The total amount of fixed storage used is 200 kilobytes.

Note: When choosing the amount of fixed storage to use for trace buffers, consider the amount of central storage available.

COMASID
Traces activity in the common area page. This is the default.

NOCOMASID
Prevents tracing of activity in the common area page.

DMPREC
Includes trace data in the SVC dump requested when RSM enters recovery processing. The SYSRSM trace is suspended while the dump is in progress. The dump contains the most recent trace data recorded prior to the problem. With this dump option, which is a default:

- The trace tables are not dumped when RSM enters recovery processing.
- Tracing continues on other processors during recovery processing.

NODMPREC
Prevents trace data from being dumped if RSM enters recovery processing.

DMPOFF
Causes trace data to be dumped when tracing for RSM is turned off with a TRACE CT,OFF,COMP=SYSRSM command or with an OFF parameter in a CTnRSMxx parmlib member.

NODMPOFF
Prevents writing of a dump when the TRACE operator command is entered to stop the trace. This is the default.

Function trace options

Function trace options identify the RSM functions and services to be traced. The options are:

ALLOC2G
Traces events for allocating 2G pages.

ASPCREAT
Traces events for the address space create function.

BLOCKMGR
Traces RSM SCM block manager events.

COPYSRVG
Traces RSM copy services group. The group options, which can be specified separately, are:

COPYSERV
Traces copy services.

COPYSRVH
Traces high virtual copy service.

DFSTEAL
Traces events for the double frame steal function.

DIV
Traces all events in the data-in-virtual services group. The group options, which can be specified separately, are:

DIVACCUN
Trace the DIV ACCESS and DIV UNACCESS services.

DIVMAP
Traces the data-in-virtual MAP service.

DIVMAPLV
Traces the data-in-virtual MAP service (with LOCVIEW=MAP on previous ACCESS).

DIVRES
Traces the data-in-virtual RESET service.

DIVRESLV
Traces the data-in-virtual RESET service (with LOCVIEW=MAP on previous ACCESS).

DIVRTR
Traces the data-in-virtual services router.

DIVSAVE
Traces the data-in-virtual SAVE service.

DIVSLIST
Traces the data-in-virtual SAVELIST service.

DIVUNMAP
Traces the data-in-virtual UNMAP service.

DSPCONV
Traces events in the data space convert interface function.

DSPLIMIT
Traces events in the data space limit interface function.

DATASPAC
Traces all events in the data space and hiperspace group. The group options, which can be specified separately, are:

DSPSERV
Traces all events in the data space services group. The group options, which can be specified separately, are:

DSPCREAT
Traces events in the DSPSERV CREATE service.

DSPDELET
Traces events in the DSPSERV DELETE service.

DSPDRFOF
Traces events in DSPSERV define DREF off.

DSPDRFON
Traces events in DSPSERV define DREF on.

DSPEXTEN
Traces events in the DSPSERV EXTEND service.

DSPLOAD
Traces events in the DSPSERV LOAD service.

DSPIOOF
Traces events in the DSPSERV IOOFF service.

DSPIOON
Traces events in the DSPSERV IOON service.

DSPOUT
Traces events in the DSPSERV OUT service.

DSPREL
Traces events in the DSPSERV RELEASE service.

DSPSRTR
Traces events in the DSPSERV router service.

DSPSRTRD
Traces events in the DSPSERV disabled RTR service.

HSPSERV
Traces all events in the hiperspace services group. The group options, which can be specified separately, are:

HSPCACHE
Traces events in the HSPSERV cache services.

HSPSCROL
Traces events in the HSPSERV scroll services.

DUMPSERV
Traces the dumping function.

FAULTS
Traces all events in the fault services group. The group options, which can be specified separately, are:

FLTASP
Traces all events in the address space faults group. The group options, which can be specified separately, are:

FLTADPAG
Traces disabled address space page faults.

FLTAEPAG
Traces enabled address space page faults.

FLTAESEG
Traces enabled address space segment faults.

FLTAHPAG
Traces address space page faults for address above the 2 gigabytes bar.

FLTAHSEG
Traces address space segment faults for address above the 2 gigabytes bar.

FLTAREGN
Traces address space region faults.

FLTATYPE
Traces address space type faults.

FLTDSP
Traces all events on the data space faults group. The group options, which can be specified separately, are:

FLTDEN
Traces enabled data space faults.

FLTDDIS
Traces disabled data space faults.

FLTEPROT
Traces protection faults.

FREEFRAM
Traces the free frame function.

GEN
Traces all events in the general function group. The group options, which can be specified separately, are:

GENDEFER
Traces general defers.

GENIOCMP
Trace general I/O completion.

GENTERM
Traces general abends.

GLRUSTL
Traces the global LRU Steal function.

IARVSERV
Traces all IARVSERV requests. The Virtual Services group options, which can be specified separately, are:

VSCHGACC
Traces IARVSERV CHANGEACCESS requests.

VSROUTR
Traces IARVSERV service router.

VSSHARE
Traces IARVSERV SHARE requests.

VSSHSEG
Traces IARVSERV SHARESEG requests.

VSUNSHAR
Traces IARVSERV UNSHARE requests.

IARV64
Traces all IARV64 requests. The High Virtual services group options, which can be specified separately are:

V6CHACC
Traces IARV64 CHANGEACCESS requests.

V6CHGURD
Traces IARV64 CHANGEGUARD requests.

V6COUNT
IARV64 COUNTPAGES requests.

V6DETACH
Traces IARV64 DETACH requests.

V6DISCAR
Traces IARV64 DISCARDDATA requests.

V6GETCOM
Traces IARV64 GETCOMMON requests.

V6GETSHR
Traces IARV64 GETSHARED requests.

V6GETSTR
Traces IARV64 GETSTOR requests.

V6LIST
Traces IARV64 LIST requests.

V6PAGFIX
Traces IARV64 PAGEFIX requests.

V6PAGIN
Traces IARV64 PAGEIN requests.

V6PAGOUT
TracesIARV64 PAGEOUT requests.

V6PAGUNF
Traces IARV64 PAGEUNFIX requests.

V6PROTEC
Traces IARV64 PROTECT/UNPROTECT requests.

V6ROUTR
Traces IARV64 service router.

V6SHMOMB
Traces IARV64 SHAREMEMOBJ requests.

LPGALLOC
Traces the Large Page Frame Allocation function.

MACHCHK
Traces the machine check function.

PGSER
Traces all events in the paging services group. The group options, which can be specified separately, are:

PGANY
Traces events in the page anywhere service.

PGFIX
Traces events in the page fix service.

PGFREE
Traces events in the page free service.

PGLOAD
Traces events in the page load service.

PGOUT
Traces events in the page out service.

PGPROT
Traces events in the page protect service.

PGREL
Traces events in the page release service.

PGSRTR
Traces events in the paging service routers.

PGUNPROT
Traces events in the page unprotect service.

QFSTEAL
Traces events for the quad frame steal function.

RECONFIG
Traces the reconfiguration function.

RPBPMGMT
Traces the RSM cell pool management function.

RSMPIN
Traces the RSMPIN services.

SUBSPACE
Traces all events in the subspace group. The group options, which can be specified separately, are:

SSPCONV
Traces the subspace conversion services.

IARSUBSP
Traces the subspace services group. The group options, which can be specified separately, are:

SSPIDENT
Traces the IARSUBSP IDENTIFY service.

SSPCREAT
Traces the IARSUBSP CREATE service.

SSPASSIG
Traces the IARSUBSP ASSIGN service.

SSPUNAS
Traces the IARSUBSP UNASSIGN service.

SSPDELET
Traces the IARSUBSP DELETE service.

SSPSHARE
Traces the IARSUBSP SHARE service.

SSPUNID
Traces the IARSUBSP UNIDENTIFY service.

SSPSRTR
Traces the IARSUBSP router.

SWAP
Traces all events in the swap services group. The group options, which can be specified separately, are:

REALSWAP
Traces events during in-real swap processing.

SWAPIN
Traces events in the swap-in service.

SWAPOUT
Traces events in the swap-out service.

TRACE*
Traces the trace function. This function is always traced.

UIC
Traces the unreferenced interval count function.

UMCPU
Traces the free CPU related frames function.

VIO
Traces the virtual I/O function.

VR
Traces the V=R allocation function.

VSM
Traces all events in the VSM services group. The group options, which can be specified separately are:

VSMFRMN
Traces events in the FREEMAIN service.

VSMGTMN
Traces events in the GETMAIN service.

WAITSER
Traces RSM Wait function.

XCHUP
Traces the exchange up function.

XMPOST
Traces the cross memory posting function.

Event trace options

Event trace options identify the events for RSM to collect trace data. The options are:

ESTOR
Traces all events in the expanded storage management group. The group options, which can be specified separately, are:

ESGET
Traces get expanded storage.

ESENQ
Traces enqueue expanded storage.

ESDEQ
Traces dequeue expanded storage.

ESFREE
Traces free expanded storage.

FUNCREQ
Traces the function request event.

PAGEREQ
Traces all events in the page request group. The group options, which can be specified separately, are:

PAGEA2R
Traces requests to move a page from auxiliary to central storage.

PAGEDEF
Traces requests to move a page to central storage was deferred for lack of a frame

PAGEIPTE
Traces requests to invalidate a page table entry.

PAGEP2R
Traces requests to move a page from permanent to central storage.

PAGEREL
Traces requests related to I/O in-progress or related to a defer event.

PAGER2A
Traces requests to move a page from central storage to auxiliary storage.

PAGER2P
Traces requests to move a page from central storage to permanent storage.

PAGER2R
Traces requests to move a page from central storage to central storage.

PGEVENTS
Traces all events in the page fix/free group. The group options, which can be specified separately, are:

FIX
Traces a page being fixed.

FREE
Traces a page being freed.

REGIONGR
Traces all events in the region table group. The group options, which can be specified separately, are:

CREG1ST
Traces the creation of a region 1st table.

CREG2ND
Traces the creation of a region 2nd table.

CREG3RD
Traces the creation of a region 3rd table.

RSTOR
Traces all events in the frame management group. The group options, which can be specified separately, are:

HVFGRP
Traces events for frame management of high virtual frames. The group options, which can be specified separately, are:

HVFRDEQ
Traces when a frame is dequeued from the high virtual frame queue.

HVFRENQ
Traces when a frame is enqueued onto the high virtual frame queue.

HVPGTDEQ
Traces when a frame is dequeued from the high virtual page table frame queue.

HVPGTENQ
Traces when a frame is enqueued onto the high virtual page table frame queue.

RSDEQ
Traces all events in the dequeue frame group. The group options, which can be specified separately, are:

RSDDEFER
Traces when a frame is dequeued from the deferred FREEMAIN frame queue or the orphan frame queue.

RSDFIX
Traces when a frame is dequeued from the fixed frame queue or the local quad frame queue.

RSDGDFER
Traces when a frame is dequeued from the general defer frame queue.

RSDPAG
Traces when a frame is dequeued from the pageable frame queue.

RSDSBUF
Traces when a frame is dequeued from the central storage buffer frame queue.

RSDSQA
Traces when a frame is dequeued from the SQA frame queue.

RSDVRW
Traces when a frame is dequeued from the V=R waiting frame queue.

RSENQ
Traces all events in the enqueue frame group. The group options, which can be specified separately, are:

RSEDEFER
Traces when a frame is enqueued onto the deferred or orphan frame queue.

RSEFIX
Traces when a frame is enqueued onto to the fixed frame queue or the local quad frame queue.

RSEPAG
Traces when a frame is enqueued onto to the pageable frame queue.

RSESBUF
Traces when a frame is enqueued onto to the central storage buffer frame queue.

RSEGDFER
Traces when a frame is enqueued on the general defer frame queue.

RSESQA
Traces when a frame is enqueued onto to the SQA frame queue.

RSEVRW
Traces when a frame is enqueued onto to the V=R waiting frame queue.

RSFREE
Traces all events in the free frame group. The group options, which can be specified separately, are:

FREE2G
Traces when a 2G frame group is freed.

PLFREE
Traces when a pageable large (1 MB) frame is freed.

QFFREE
Traces when a quad group is freed.

QHFREE
Traces when a quad holding frame is freed.

QSFREE
Traces when a single quad frame is freed.

RSFDBL
Traces when a double frame is freed.

RSFSNG
Traces when a single frame is freed.

RSGET
Traces all events in the get frame group. The group options, which can be specified separately, are:

GET2G
Traces when a 2G frame group is gotten.

PLGET
Traces when a pageable large frame (1 MB) group is obtained.

QFGET
Traces when a quad group is gotten.

QHGET
Traces when a quad holding frame is gotten.

QSGET
Traces when a single quad frame is gotten.

RSGDBL
Traces when a double frame is gotten.

RSGSNG
Traces when a single frame is gotten.

RUCSAFLT
Traces restricted-use CSA (RUCSA) fault events.

SAF
Traces invocations of SAF from RSM:

SAFCHK
Traces SAF Verification events issued from RSM.

SAFEXTR
Traces SAF Environment Extract events issued from RSM.

SAFCREAT
Traces SAF Environment Create events issued from RSM.

SAFDELET
Traces SAF Environment Delete events issued from RSM.

SCMBLKMG
Traces SCM Block Manager events. The group options, which can be specified separately, are:

SCMEVACA
Traces SCM Evacuation Add to Table.

SCMFREE
Traces the SCM Block Manager Free Block.

SCMICHNG
Traces SCM Block Manager Increment Change.

SCMOBT
Traces the SCM Block Manager Obtain Block.

SCMOFLIN
Traces SCM Block Manager Offline Event.

SCMTRANS
Traces SCM Transfer Block ID Event.

SCMSTART
Start of SCM Evacuation.

SCMEND
End of SCM Evacuation.

SCMSTASP
Start SCM Evacuation Address Space.

SCMENDAS
End SCM Evacuation Address Space.

SCMSTDSP
Start SCM Evacuation Data Space.

SCMENDSP
End SCM Evacuation Data Space.

SCMPOOLG
Get a blockID from a pool.

SCMPOOLF
Free a blockID to a pool.

SCMEVACR
Evacuate SCM from stg range.

SCMDFGST
Start SCM defragmentation.

SCMADDIN
Add SCM increment.

SCMADDEX
Add SCM extent.

SHRDATA
Traces all events in the IARVSERV services group. The group options, which can be specified separately, are:

GRPCREAT
Traces the creation of new sharing groups.

GRPDEL
Traces the deletion of existing sharing groups.

GRPPART
Traces the partitioning of existing sharing groups.

VIEWADD
Traces the addition of views to sharing groups.

VIEWCHG
Traces the changing of storage attributes of the view.

VIEWDEL
Traces the deletion of views from sharing groups.

VIEWMOVE
Traces the move of existing views from one sharing group to another.

SHRINT
Traces High Virtual Shared Interest events. The group options, which can be specified separately, are:

SHRADD
Traces adding shared interest.

SHRDEL
Traces removing shared interest.

STORMOD
Traces all events in the storage state modification group. The group options, which can be specified separately, are:

CLONEPAG
Traces the page table entry copied to a subspace.

CLONESEG
Traces the segment table entry copied to a subspace.

TRACEB
Traces the trace buffer event. This event is always traced.

WORKUNIT
Traces all events in the net event trace group. The group options, which can be specified separately, are:

ENABLE
Traces requests to enable a unit of work.

SUSPEND
Traces requests to suspend a unit of work.

RESUME
Traces requests to resume a unit of work.

XEPLINK
Traces all events in the external entry point linkage group. The group options, which can be specified separately, are:

XEPENTRY
Traces entry to the entry point.

XEPEXIT
Traces exit from the entry point.

Examples of requesting SYSRSM traces

- Example 1: TRACE command
- Example 3: TRACE command

  The member requests tracing of the FAULTS services group, the PGANY service, and the VIO function, but only for address spaces X'11' and X'41' and for job PGM1.

```
TRACEOPTS
   ON
   ASID(11,41)
   JOBNAME(PGM1)
   OPTIONS('FAULTS','PGANY','VIO')
```

- Example 4: TRACE command

  The example specifies that options are to be obtained from the parmlib member CTWRSM17.

```
trace ct,on,comp=sysrsm,parm=ctwrsm17
```

- Example 5: TRACE command

  The example requests the same trace as Example 2, but specifies all options in the REPLY.

```
trace ct,on,comp=sysrsm
* 78 ITT006A ...
reply 78,options=(faults,pgany,vio),asid=(11,41),jobname=(pgm1),end
```

Formatting a SYSRSM trace

Format the trace with an IPCS CTRACE COMP(SYSRSM) subcommand. The subcommand has no OPTIONS values.

Output from a SYSRSM trace

"Example: RSM component trace records formatted with CTRACE COMP(SYSRSM) FULL subcommand" on page 485 is an example of RSM component trace records formatted with the CTRACE COMP(SYSRSM) FULL subcommand.

Example: RSM component trace records formatted with CTRACE COMP(SYSRSM) FULL subcommand

```
XEPENTRY  00000001  18:18:06.441411  External Entry Point Entry
      FUNC1... PGFIX             Page Fix
      JOBN1... ASNB     ASID1... 000D     PLOCKS.. 80000001 CPU..... 0000
      JOBN2... ASNB     ASID2... 000D     RLOCKS.. 80000000
      KEY..... 0036     ADDR.... 015FF008 ALET.... 00000000
      7D00
      KEY..... 002C     ADDR.... 005FEF80 ALET.... 00000000
      00000000 005EF000 005EFFFF 00F00200 00FF8DD8 00F8BD00 00F8BC98
      005EFFC3 005EF02C 00F8BD50 005EF02C 00F8BE00 00F8BC00 005FEF78
      00FF91DA 81082798
FIX       00000003  18:18:06.441921  Page Being Fixed
      FUNC1... PGFIX             Page Fix
      JOBN1... ASNB     ASID1... 000D     PLOCKS.. 88004001 CPU..... 0000
      JOBN2... ASNB     ASID2... 000D     RLOCKS.. 88004000
      KEY..... 0036     ADDR.... 015FF008 ALET.... 00000000
      7D003500
      KEY..... 005D     ADDR.... 005EF000 ALET.... 00000000
      KEY..... 0001     ADDR.... 011F8220 ALET.... 00000000
      0151182C 012D2E60 81C00000 03000001 0000000D 005EF000 00000000 00000000
XEPENTRY  00000001  18:18:06.461716  External Entry Point Entry
```

```
        FUNC1... PGFREE            Page Free
        JOBN1... ASNB     ASID1... 000D     PLOCKS.. 80000001 CPU..... 0000
        JOBN2... ASNB     ASID2... 000D     RLOCKS.. 80000000
        KEY..... 0036     ADDR.... 015FF008 ALET.... 00000000
        8100
        KEY..... 002C     ADDR.... 005FEF80 ALET.... 00000000
        005F5EC0 00F8BC08 00000000 00F00200 00FF8DD8 00FF6896 00F8DB00
        00000008 00F8BF2C 00FF7B60 00000C60 00F8BE00 00F8Bc00 005FEF78
        00FF96A4 810801B8
  FREE      00000004  18:18:06.461766  Page Being Freed
        FUNC1... PGFREE            Page Free
        JOBN1... ASNB     ASID1... 000D     PLOCKS.. 88004001 CPU..... 0000
        JOBN2... ASNB     ASID2... 000D     RLOCKS.. 88004000
        KEY..... 0036     ADDR.... 015FF008 ALET.... 00000000
        8100
        KEY..... 005D     ADDR.... 005F5000 ALET.... 00000000
        KEY..... 0001     ADDR.... 01210660 ALET.... 00000000
        011F9CA0 01242560 81C00000 03000000 0000000D 005F5000 00000000 00000000
  FREE      00000004  18:18:06.461805  Page Being Freed
        FUNC1... PGFREE            Page Free
        JOBN1... ASNB     ASID1... 000D     PLOCKS.. 88004001 CPU..... 0000
        JOBN2... ASNB     ASID2... 000D     RLOCKS.. 88004000
        KEY..... 0036     ADDR.... 015FF008 ALET.... 00000000
        8100
        KEY..... 005D     ADDR.... 005EF000 ALET.... 00000000
        KEY..... 0001     ADDR.... 011F8220 ALET.... 00000000
        0151182C 012D2E60 81C00000 03000000 0000000D 005EF000 00000000 00000000
```

The fields that you may need in the report are:

FUNC1
The function in control at the time the trace event was recorded.

JOBN1
The job name identifying the address space that contains the unit of work requesting the RSM service.

JOBN2
The job name that matched a name in the job name list provided with the TRACE operator command.

ASID1
The ASID identifying the address space that contains the unit of work requesting the RSM service.

ASID2
The ASID that matched an identifier in the ASID list provided with the TRACE operator command.

CPU
The central processor identifier for the processor the trace is running on.

SYSSPI component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSSPI component trace for the service processor interface (SPI).

Information	For SYSSPI:
Parmlib member	CTISPI00 If there is an CTISPI00 member in the parmlib concatenation, it will be used during IPL to control the initial state of the trace. If there is no CTISPI00 member in parmlib, the initial state will be OFF.
Default tracing	No
Trace request OPTIONS parameter	In CTISPI00 and REPLY for TRACE command

Information	For SYSSPI:
Buffer	• Default: 64KB • Range: 32KB to 1216KB • Size set by: `TRACE CT,nnnK,COMP=SYSSPI` command or the BUFSIZE specified in parmlib member CTISPIxx. • Change size after IPL: Yes, by turning the trace off, changing the size and then turning the trace back on. • Location: In the component area
Trace records location	ECSA
Request of SVC dump	By the component
Trace formatting by IPCS	CTRACE COMP(SYSSPI)
Trace format OPTIONS parameter	None

Requesting a SYSSPI trace

Request a SYSSPI trace at the direction of the IBM Support Center. Do the following:

1. Start the trace with the command:

```
TRACE CT,ON,COMP=SYSSPI
```

2. After the interval specified by IBM, stop the trace with the command:

```
TRACE CT,OFF,COMP=SYSSPI
```

When the buffer fills up, the component requests an SVC dump, which includes the contents of the buffer. Optionally, the operator could enter a DUMP command.

CTISPI00 parmlib member

The following table indicates the parameters you can specify on a CTISPI00 parmlib member.

Parameters	Allowed on CTISPI00?
ON or OFF	Yes
OPTIONS	No

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for trace?
ON or OFF	One is required
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on REPLY for trace?
OPTIONS	No

OPTIONS parameter

The values for the OPTIONS parameter for the CTISPI00 parmlib member and reply for a TRACE command, in an alphabetical order, are:

DMPOFF
Take a SVC dump when the trace is turned off. This is the default.

NODMPOFF
Do not take an SVC dump when the trace is turned off.

NOWRAP
When the buffer becomes full, take an SVC dump and discard all trace data. This is the default.

WRAP
When the buffer becomes full, do not take an SVC dump, and discard some of the oldest trace data to make room for new trace data.

Formatting a SYSSPI trace

1. On z/OS V2R3 or earlier systems, use SPZAP to change a module. IBM supplies the change.
2. Format the trace with an IPCS CTRACE COMP(SYSSPI) subcommand. The subcommand has no options values.

SYSTTRC transaction trace

Transaction trace does not participate in component trace-controlled processing. It is a standalone tracing facility. Do not use trace CT commands for transaction trace. Do not attempt to add a component trace parmlib member for transaction trace. For information on transaction trace, see Chapter 13, “Transaction trace,” on page 505.

SYSVLF component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

The following summarizes information for requesting a SYSVLF component trace for the virtual lookaside facility (VLF).

Information	For SYSVLF:
Parmlib member	None
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	None
Buffer	• Default: N/A • Range: N/A • Size set by: MVS system • Change size after IPL: No • Location: Data space. Enter DISPLAY J,VLF to identify the VLF data spaces. In the REPLY for the DUMP command, specify DSPNAME=('VLF'.Dclsname,'VLF'.Cclsname), where *clsname* is a VLF class name.
Trace records location	Address-space buffer, data-space buffer
Request of SVC dump	By DUMP or SLIP command or when SYSVLF full tracing is turned off

Information	For SYSVLF:
Trace formatting by IPCS	CTRACE COMP(SYSVLF)
Trace format OPTIONS parameter	None

Requesting a SYSVLF trace

A minimal trace runs whenever VLF is in control. No actions are needed to request the minimal trace.

To record more than the minimal trace, request full tracing with the TRACE operator command. Note that full tracing can slow system performance. The following table indicates the parameters you can specify on a TRACE CT command. In response to the command, the system does not prompt the operator for a reply.

Parameters	Allowed on TRACE CT for trace?
ON or OFF	One is required
nnnnK or nnnnM	No
COMP	Required
SUB	No
PARM	No

When you turn the full tracing off, the system writes a dump containing the trace records, then resumes minimal tracing.

Examples of requesting and stopping SYSVLF full traces

- Example 1: Requesting a SYSVLF full trace

  The command requests a full trace.

  ```
  TRACE CT,ON,COMP=SYSVLF
  ```

- Example 2: Stopping a SYSVLF full trace

  The command turns off full tracing. In response, the system writes a dump and resumes minimal SYSVLF tracing.

  ```
  TRACE CT,OFF,COMP=SYSVLF
  ```

- Example 3: Command for SYSVLF tracing in a sysplex

  The following command turns on tracing for a SYSVLF trace in the systems of a sysplex. Because SYSVLF has no parmlib member, the CTIITT00 member is used to prevent prompts.

  ```
  route *all,trace ct,on,comp=sysvlf,parm=ctiitt00
  ```

Formatting a SYSVLF trace

Format the trace with an IPCS CTRACE COMP(SYSVLF) subcommand. The subcommand has no OPTIONS values.

Output from a SYSVLF trace

Figure 158 on page 490 is an example of VLF component trace records formatted with the CTRACE COMP(SYSVLF) FULL subcommand. It shows formatted exception records from the trace buffers.

```
                   VLF  COMPONENT TRACE FULL FORMAT
 **** 01/27/90
 COFRCVRY  00000000  16:03:02.181262  VLF RECOVERY ENTRY
   HASID... 000E     SASID... 000E     CPUID... FF170284 30900000
   MODNAME. COFMPURG ABEND... 840C4000 REASON.. 00000011
   EPTABLE. PURG  ESTA  ....  ....  ....  ....  ....  ....  ....  ....
 COFRCVRY  00000001  16:03:02.181324  VLF RECOVERY EXIT
   HASID... 000E     SASID... 000E     CPUID... FF170284 30900000
   MODNAME. COFMPURG ABEND... 840C4000 REASON.. 00000011
   RETCODE. 00000000 RSNCODE. 00000000 FTPRTS.. 80300000 DATA.... 00000000
 .
 .
 .
```

Figure 158. Example: VLF component trace records formatted with CTRACE COMP(SYSVLF) FULL subcommand

The following explains fields in the report. Additional fields that are not shown in the example can be in a report. These additional fields are explained below in the Other Fields section.

COFRCVRY
The name or identifier of the trace record.

00000000
The identifier in hexadecimal

16:03:02.181262
The time stamp indicating when the record was placed in the trace table

HASID... 000E
The home address space identifier

SASID... 000E
The secondary address space identifier

CPUID... FF170284 30900000
The identifier of the processor that placed the record in the trace table

CALLER
The address of the routine that issued a VLF service request, such as DEFINE, CREATE, NOTIFY, PURGE, etc..

MODNAME. COFMPURG
The name of the module that was running

ABEND... 840C4000
The abend that occurred and caused VLF to enter recovery is 0C4

REASON.. 00000011
The reason code associated with the abend

EPTABLE. PURG ESTA
Information used for diagnosis by IBM

RETCODE. 00000000
The return code that was issued by the module that is exiting

RSNCODE. 00000000
The reason code that was issued by the module that is exiting

FTPRTS.. 80300000
Information used for diagnosis by IBM

DATA.... 00000000
Information used for diagnosis by IBM

Other Fields: Fields that are not shown in the example CTRACE output but that may appear in a report are:

CINDX
The concatenation index of the major name for which an object has been created or retrieved

CLASS... NPDS3
The name of a VLF class

DDNAME
The DDNAME of the concatenated data set list

FUNC=xxxx
Indication of the function for which a NOTIFY occurred

FUNCCODE
The hexadecimal value of the NOTIFY function code when it cannot be interpreted

MAJOR
The major name

MINADDR
Address of a field containing a minor name

MINALET
Access list entry token (ALET) associated with the address used to locate the minor name

MINOR
The minor name

OBJSIZE
The total size, in bytes, of the object returned by a COFRETRI macro

PARMS
Hexadecimal dump of the COFNOTIF macro parameter list

TLSTADDR
Address of a target list for a COFRETRI macro

TLSTALET
Access list entry token (ALET) of a target list for a COFRETRI macro

TLSTSIZE
The length, in bytes, of the target list

UTOKEN
User token returned by a COFIDENT macro and required as input for COFREMOV, COFCREAT, and COFRETRI macros

VOLSER
The volume serial

SYSWLM component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSWLM component trace for workload management (WLM).

Information	For SYSWLM:
Parmlib member	None
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	None

Information	For SYSWLM:
Buffer	• Default: 64KB • Range: 64KB - 16M • Size set by: MVS system • Change size after IPL: Yes, when starting a trace • Location: Extended common service area (ECSA)
Trace records location	Address-space buffer, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSWLM)
Trace format OPTIONS parameter	None

Requesting a SYSWLM trace

Request a SYSWLM component trace by a TRACE CT command.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON, nnnnK, OFF	One is required. nnnnK specifies the size of the buffer.
COMP	Required
SUB	No
PARM	No

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	No
JOBNAME	No
OPTIONS	No
WTR	Yes

Examples of requesting SYSWLM traces

Figure 159 on page 492 shows an example of how to request a SYSWLM component trace.

```
trace ct,on,comp=syswlm
* 17 ITT006A ...
reply 17,end
```

Figure 159. Example: requesting a SYSWLM component trace

Formatting a SYSWLM trace

Format the trace with an IPCS CTRACE COMP(SYSWLM) subcommand. The subcommand has no OPTIONS values.

Output from a SYSWLM trace

Figure 160 on page 493 is an example of SYSWLM component trace records formatted with the CTRACE COMP(SYSWLM) SHORT subcommand.

```
MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
--------  --------  ---------------  -----------

WLMEPEXT  00005004  14:07:42.807618  Entry Point Exited
WLMEPENT  00005003  14:07:46.335563  Entry Point Entered
WLMEPEXT  00005004  14:07:46.336376  Entry Point Exited
WLMEPENX  00005005  14:07:46.336540  Entry Point Entered Exception
WLMEPENT  00005003  14:07:46.336557  Entry Point Entered
WLMEPENT  00005003  14:07:46.337909  Entry Point Entered
SMSYNMEM  00000921  14:07:46.512018  SM Synch XCF Member
WLMEPEXT  00005004  14:07:46.512360  Entry Point Exited
WLMEPENT  00005003  14:07:46.512374  Entry Point Entered
SMSYNMEM  00000921  14:07:46.594486  SM Synch XCF Member
```

Figure 160. Example: SYSWLM component trace records formatted with CTRACE COMP(SYSWLM) SHORT subcommand

Figure 161 on page 493 shows an example of SYSWLM component trace records formatted with the CTRACE COMP(SYSWLM) FULL subcommand.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------
SY1       WLMEPENT  00005003  14:52:23.449339  Entry Point Entered
     FUNCID... 0409                CPU...... 0001
     HOMEASID. 000B                HJOBNAME. WLM
     REQASID.. 0000                RJOBNAME. UNKNOWN
     KEY...... 5018 RUCA_EPIDS IWMDMPRP
         04098000
     KEY...... 501E PARM1
         00000084
     KEY...... 501F PARM2
         00000040
     KEY...... 5020 PARM3
         00000000
```

Figure 161. Example: SYSWLM component trace records formatted with CTRACE COMP(SYSWLM) FULL subcommand

The following explains fields in the report.

FUNCID
The module table entry for the module that wrote the trace record.

CPU
The CPU that the module was running on.

HOMEASID
ASID from PSAAOLD.

REQASID
ASID that was explicitly coded on trace invocation.

HJOBNAME
JOBNAME of home address space.

RJOBNAME
JOBNAME that was explicitly coded on trace invocation.

KEY
Identifies the type of data that follows. The data is formatted in both HEX and EBCDIC.

SYSXCF component trace

Before using this component trace, ensure that you have read:

- "Planning for component tracing" on page 350
- "Obtaining a component trace" on page 358
- "Viewing the component trace data" on page 369

The following summarizes information for requesting a SYSXCF component trace for the cross-system coupling facility (XCF).

Information	For SYSXCF:
Parmlib member	CTnXCFxx. Default member: CTIXCF00 specified in COUPLE00 member
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	In CTnXCFxx or REPLY for TRACE command
Buffer	• Default: 4MB • Range: 16KB - 16MB (System rounds size up to a multiple of 72 bytes.) • Size set by: CTnXCFxx member • Change size after IPL: No • Location: Extended local system queue area (ELSQA) of XCFAS
Trace records location	Address-space buffer, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSXCF)
Trace format OPTIONS parameter	Yes

Requesting a SYSXCF trace

Specify options for requesting a SYSXCF component trace on a CTnXCFxx parmlib member or on the reply for a TRACE CT command.

If you specify additional tracing options while the system is running, place the trace records in a trace data set or sets, because the trace buffer size specified at initialization cannot be changed while the system is running. Specify NOWRAP to keep from losing trace records.

Note: NOWRAP prevents trace records written to the data set or sets from being overwritten. Once the data set or sets are filled, no more records are written to them. The system still writes trace records to the address-space buffers. The system wraps the address-space buffers, so that trace records may be lost. Be sure to allocate enough space on the data set or sets to hold all the records needed for diagnosis.

CTnXCFxx parmlib member

The following table indicates the parameters you can specify on a CTnXCFxx parmlib member.

Parameters	Allowed on CTnXCFxx?
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes, at IPL or when reinitializing XCF

Parameters	Allowed on CTnXCFxx?
OPTIONS	Yes
SUB	No
PRESET	No
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

Note: You can change the buffer size only at IPL or when reinitializing XCF. Specify the new buffer size in the BUFSIZE parameter on the CTnXCFxx member being used.

IBM supplies a CTIXCF00 parmlib member, which specifies that component tracing for XCF be initialized with the component default buffer size and minimal component tracing active. The content of CTIXCF00 is:

```
TRACEOPTS ON
```

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON or OFF	One is required
nnnnK or nnnnM	No
COMP	Required
SUB	No
PARM	Yes

Parameters	Allowed on TRACE CT for Write?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for Trace?
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter for the CTnXCFxx parmlib member and reply for a TRACE command, in alphabetical order, are:

ARM
Traces events for automatic restart management services.

CFRM
Traces events for coupling facility resource management services.

GROUP
Traces events for group services, such as XCF groups joining or disassociating from XCF services.

GRPNAME=(*groupname*[,*groupname*]...)
Reduces tracing to events for only the specified XCF groups. If GRPNAME is specified, the GROUP, SERIAL, SIGNAL, and STATUS options are filtered by the specified XCF group or groups; the STORAGE option is not filtered by GRPNAME. You can specify up to 8 XCF groups.

MODID=(modid [,modid]...)
Reduces tracing of all events to only the specified XCF module IDs. You can specify up to eight XCF Module IDs. Use XCF Module ID tracing only when requested by IBM Support.

NOTEPAD
Traces events for XCF note pad services processing.

NPNAME=(*npname*[,*npname*]...)
Reduces tracing of NOTEPAD events related to XCF note pad services processing to only the specified note pad names or note pad names that match the specified note pad name patterns. If NPNAME is specified, the NOTEPAD option trace events are filtered by the specified note pad names. You can specify up to four note pad names or note pad name patterns. To be valid, each *npname* must meet the following conditions:

- A note pad name can consist of two to four sections separated by periods.
- If a section is not specified, it is defaulted to all blanks.
- The first and the second sections must not be blank.
- Each section, if specified, must be left-justified with no trailing blanks.
- Each section can contain up to eight upper case alphanumeric (A-Z,0-9), national (@,#,$), or underscore (_) characters.
- Any section can contain the asterisk (*) wild card character, which is used to match zero (0) or more characters (for example, OWN*.*).

SERIAL
Traces events for serialization services.

SERVER
Traces events for client/server services processing.

SFM
Traces events for sysplex failure management services.

SIGNAL
Traces events for signalling services processing.

STATUS
Traces events for XCF monitoring services and sysplex partitioning services.

STORAGE
Traces events for storage management services.

SRVNAME=(servername [,servername]...)
Reduces tracing of SERVER events related to XCF client/server processing to only the specified server names or server names that match the specified server name patterns. If SRVNAME is specified, the SERVER option trace events are filtered by the specified server names. You can specify up to 4 server names/server name patterns. To be valid, the server name must meet the following conditions:

- Server names consist of one to four 8 byte sections separated by a period.
- If a section is not specified, it is defaulted to all blanks.
- Each section can contain any alphanumeric (A-Z,a-z,0-9), national (@,#,$), or underscore (_) character.
- Any section but the first can be entirely blank.
- Any section can contain the asterisk (*) wild card character which is used to match zero (0) or more characters (for example, SYS*.*)
- The server name can be enclosed within single quotes to preserve case sensitivity (for example, 'sysxcf.*')

VECTOR
Traces all events related to list notification or local cache vector processing. One of the other trace options might also trace these events.

Examples of requesting SYSXCF traces

In Figure 162 on page 497, the CTnXCFxx member requests STORAGE and SIGNAL options. To minimize lost trace data, the member also starts external writer WTRDASD1 with the NOWRAP option specified and connects the trace to the writer.

```
TRACEOPTS
   WTRSTART(WTDASD1) NOWRAP
   ON
   WTR(WTDASD1)
   OPTIONS('STORAGE','SIGNAL')
```

Figure 162. Example: CTnXESxx member requesting a SYSXCF trace

The example shown in Figure 163 on page 497 uses the TRACE command to request the same type of trace that is shown in Figure 162 on page 497.

```
trace ct,wrtstart=wtdasd1
trace ct,on,comp=sysxcf
* 62 ITT006A ...
r 62,wtr=wtdasd1,options=(storage,signal),end
```

Figure 163. Example: TRACE command for SYSXCF trace

Formatting a SYSXCF trace

Format the trace with an IPCS CTRACE COMP(SYSXCF) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

The options are:

ARM
Formats trace records for automatic restart management services.

CFRM
Formats trace records for coupling facility resource management services.

CLUSTER
Formats trace records for XCF z/OS cluster manageable resource services.

GROUP
Formats trace records for XCF group services, such as groups joining or disassociating from XCF services.

NOTEPAD
Formats trace records for XCF Note Pad Services.

SERIAL
Formats trace records for serialization services.

SERVER -
Formats trace records for XCF client/server services.

SFM
Formats trace records for sysplex failure management services.

SIGNAL
Formats trace records for signalling services processing.

STATUS
Formats trace records for XCF monitoring services and sysplex partitioning services.

STORAGE
Formats trace records for storage management services.

THREAD=(thread, [,thread]...)
Formats trace records filtered by one or more specific trace threads. The XCF component trace thread filter option provides the capability to limit the presentation of trace entries to specified trace threads. XCF uses trace threads to correlate processing of units of work between the subcomponents of the XCF component. You can enter trace threads entered in hexadecimal notation using x'tttttttt' or tttttttt notation (for example, x'01001768' or 01001768) with or without the leading x and surrounding quotes (').

Output from a SYSXCF trace

Figure 164 on page 498 shows an example of SYSXCF component trace records formatted with the CTRACE COMP(SYSXCF) FULL subcommand.

```
                  COMPONENT TRACE FULL FORMAT
      COMP(SYSXCF)
   **** 01/10/2012
 SYSNAME MNEMONIC ENTRY ID   TIME STAMP                DESCRIPTION
 ------- -------- -------- --------------- -------------
 SY1     STORAGE  0F030001 18:00:56.717138 Element class defined
     00FFF002 00000004 00000000 00000000 | ..0............. |
     00000000 00000000 00000000 00000000 | ................ |
     00000000 00000000 00000000 00000000 | ................ |
     00000000 0000                       |         ...... |
 SY1     SIGNAL   08560000 18:00:56.730065 Entry to Tranport Classes Group
     00FFF002 08018000 8B801F4C C9D5C9E3 | ..0........<INIT |
     40404040 00000000 00000000 00000000 |     ............ |
     40000000 7EBCDB00 00000000 00000000 |  ...=........... |
     00000000 0000                       |         ...... |
   .
   .
   .
```

Figure 164. Example: SYSXCF component trace records formatted with CTRACE COMP(SYSXCF) FULL subcommand

SYSXES component trace

Before using this component trace, ensure that you have read:

- “Planning for component tracing” on page 350
- “Obtaining a component trace” on page 358
- “Viewing the component trace data” on page 369

The following summarizes information for requesting a SYSXES component trace for cross-system extended services (XES).

Information	For SYSXES:
Parmlib member	CTnXESxx; default member: CTIXES00 specified in COUPLE00 member
Default tracing	Yes; minimal; unexpected events
Trace request OPTIONS parameter	In CTnXESxx or REPLY for TRACE command

Information	For SYSXES:
Buffer	• Default: – 336KB for connector-related SUB trace buffers. There are multiple instances of these trace buffers in use on the system – 32MB for the GLOBAL SUB trace buffer • Range: – 16KB - 16MB for connector-related SUB trace buffers – The size for the GLOBAL SUB trace buffer cannot be changed. • Size set by: CTnXESxx member or TRACE CT command • Change size after IPL: Yes • Location: 64-bit common storage above the 2GB bar. In the REPLY for the DUMP command, specify SDATA=XESDATA.
Trace records location	64-bit common storage above the 2GB bar, trace data set
Request of SVC dump	By DUMP or SLIP command
Trace formatting by IPCS	CTRACE COMP(SYSXES)
Trace format OPTIONS parameter	Yes

Ensure that the HVCOMMON system parameter which specifies the size of the 64-bit common area reflects a size adequate to allow for the allocation of 4GB of 64-bit common storage by XES for SYSXES trace buffers plus additional system requirements for 64-bit common storage. The system default for HVCOMMON is 64GB.

SYSXES supports sublevel tracing. Tracing options are inherited through a hierarchy of trace levels. If you set trace options without specifying a sublevel, the options apply at the highest level, or head, of the hierarchy. A sublevel inherits its trace options from the next higher level, unless options are specified explicitly for the sublevel. If you set trace options for a sublevel, the options are inherited by any sublevels lower in the hierarchy. Figure 165 on page 499 shows the hierarchical structure of SYSXES traces.

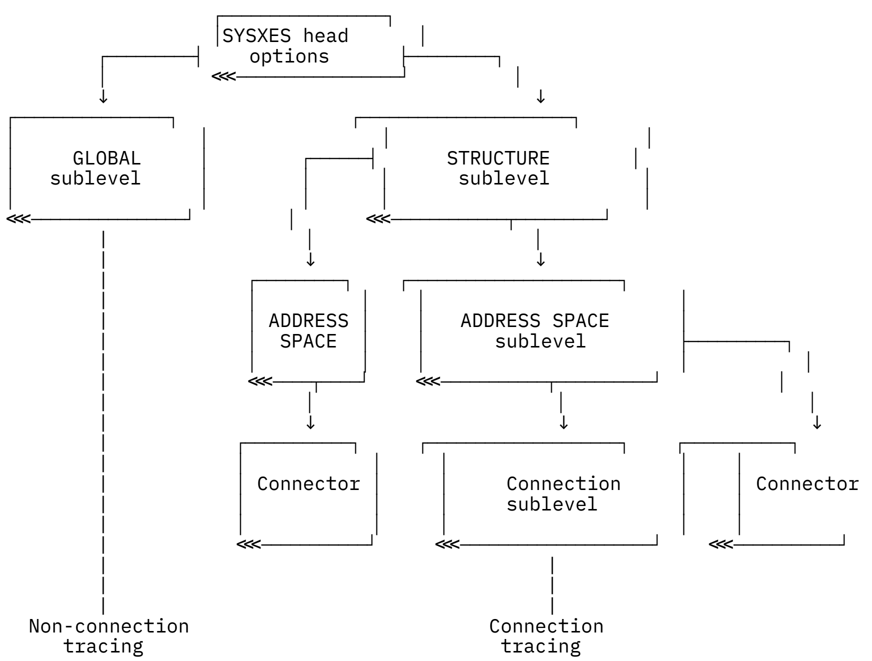


Figure 165. SYSXES SUB Trace Structure

Two classes of sublevel traces inherit the head trace options:

- The global sublevel trace has its own trace buffer and controls tracing that is not related to any particular connection. Request GLOBAL tracing by specifying SUB=(GLOBAL).
- Connection sublevels control tracing for a particular connection. Each connection has its own trace buffer. Connection sublevels are filtered hierarchically based on:
  1. Structure name (STRNAME) of the coupling facility structure to which the system is connected
  2. Address space identifier (ASID) of the address space from which the connection was made
  3. Connection name (CONNAME) of the particular connector for which tracing is requested

  Therefore, options specified for a particular structure name are inherited only by address spaces connected to that structure. Options specified for a particular address space are inherited only by connections that are connected through that address space.

  Specify SUB=(`strname`), SUB=(`strname.asid`), or SUB=(`strname.asid.conname`), depending on the degree of specificity you need. Do not specify `conname` without specifying `asid`. Also, do not specify `asid` without specifying `strname`. When specifying a structure name or connector name on the SUB option, if the name contains special characters, it must be in quotation marks. If it is in quotation marks, upper and lower case characters are not the same. Therefore the case information is important and must identically match the name used by the system. Once the name is enclosed in quotation marks it becomes case sensitive.

Requesting a SYSXES trace

Specify options for requesting a SYSXES component trace on a CTnXESxx parmlib member or on the reply for a TRACE CT command.

CTnXESxx parmlib member

The following table indicates the parameters you can specify on a CTnXESxx parmlib member.

Parameters	Allowed on CTnXESxx?
ON or OFF	Yes
ASID	No
JOBNAME	No
BUFSIZE	Yes
OPTIONS	Yes
SUB	Yes, but only for a sublevel trace
PRESET	Yes, but only for a sublevel trace
LIKEHEAD	No
WTR	Yes
WTRSTART or WTRSTOP	Yes

Setting buffer size: To select a size for your trace buffers, consider the following:

- The trace buffers can be smaller if you are using an external writer, because buffer wrapping is not a concern.
- When re-creating a problem, you might first want to make the buffer size larger.
- SYSXES has one trace buffer of the specified size **per connector**, plus one for the global trace. The amount of storage used can be significant if the system is going to have many connectors. The SYSXES trace buffers are allocated in 64-bit common storage. If the range of 64-bit common storage allocated by XES is used up (4GB), subsequent connections will not be traced because buffer space is not available.

- When the BUFSIZE parameter is not specified, SYSXES will allocate a buffer at the default size per connector.
- The SYSXES trace buffers are in disabled reference (DREF) 64-bit common storage, so storage constraints may limit buffer size.

Changing buffer size: To change the size of your trace buffers while a trace is running, either issue a TRACE CT command or activate a different CTnXESxx parmlib member. You can use these methods to change SUB levels in the hierarchy so that different SUB traces can have different sized buffers. The SYSXES GLOBAL sub buffer size is set by the system to 32MB. You cannot override this default value.

TRACE and REPLY commands

The following tables indicate the parameters you can specify on TRACE CT commands and a REPLY.

Parameters	Allowed on TRACE CT for Trace?
ON or OFF	One is required
nnnnK or nnnnM	Yes
COMP	Required
SUB	Yes
PARM	Yes

Parameters	Allowed on TRACE CT for Writer?
WTRSTART or WTRSTOP	One is required, if a writer is being used

Parameters	Allowed on REPLY for trace?
ASID	No
JOBNAME	No
OPTIONS	Yes
WTR	Yes

OPTIONS parameter

The values for the OPTIONS parameter for the CTnXESxx parmlib member and reply for a TRACE command, in alphabetical order, are:

ALL
Traces events listed for all of the options.

CONFIG
Traces changes in the state of connectivity to the coupling facility, such as addition or removal of paths.

CONNECT
Traces events for system and subsystem components that connect to or disconnect from XES resources and for exit processing.

HWLAYER
Traces events for the XES services that handle communications with the coupling facility.

LOCKMGR
Traces events related to global management of resources and to global management-related exits.

RECOVERY
Traces events within the modules that handle XES resource access failures, for both resource allocation and mainline command processing. This option provides more details than is provided by default.

REQUEST
Traces events related to requests to access data through XES mainline services.

SIGNAL
Traces events related to XES internal signalling.

STORAGE
Traces events related to management of XES control blocks.

VECTOR
Traces all events related to list notification or local cache vector processing. One of the other trace options might also trace these events. The VECTOR option also traces usage of the IXLVECTR API. This trace is distinct from component trace and is formatted by the IPCS XESDATA CONNECT report rather than through the IPCS CTRACE command. Activating the VECTOR option might impact performance, so enable it at the connector or structure subtrace level rather than for all SYSXES traces.

Examples of requesting SYSXES traces

In Figure 166 on page 502, the CTnXESxx member requests a trace of HWLAYER, LOCKMGR, CONNECT, and REQUEST trace events and a buffer size of 100KB.

```
TRACEOPTS
   ON
   OPTIONS('HWLAYER','LOCKMGR','CONNECT','REQUEST')
   BUFSIZE(100K)
```

Figure 166. Example: CTnXESxx member requesting a SYSXES trace

The example shown in Figure 167 on page 502 requests a trace of CONNECT, CONFIG, and STORAGE trace events for connection CON3 in ASID 5 for structure STR3.

```
trace ct,on,comp=sysxes,sub=(str3.asid(5).con3)
* 17 ITT006A ...
reply 17,options=(connect,config,storage),end
```

Figure 167. Example: TRACE command for SYSXES trace

Formatting a SYSXES trace

Format the trace with an IPCS CTRACE COMP(SYSXES) subcommand. The OPTIONS parameter specifies the options that select trace records to be formatted. Your formatting options depend to a great extent on the tracing options you requested. Use the options to narrow down the records displayed so that you can more easily locate any errors. If the CTRACE subcommand specifies no options, IPCS displays all the trace records.

ALL
Formats all trace records.

CONFIG
Formats changes in the state of connectivity to the coupling facility.

CONNECT
Formats events for system and subsystem components that connect to or disconnect from XES resources and for exit processing.

HWLAYER
Formats events for the XES services that handle communications with the coupling facility.

LOCKMGR
Formats events related to global management of resources and to global management-related exits.

RECOVERY
Formats events within the modules that handle XES resource access failures.

REQUEST
Formats events related to requests to access data through XES mainline services.

SIGNAL
Formats events related to XES internal signalling.

STORAGE
Formats events related to management of XES control blocks.

In the CTRACE subcommand, the SUB((subname.subname.subname)) parameter specifies the sublevel traces. A subname is:

- `GLOBAL` for an event not related to a particular connection.
- `strname.asid.conname` for an event related to the specified connector. The subname for a connection-related sublevel can contain up to three parts:
  - `strname` for the structure
  - `asid` for the address space identifier (ASID)
  - `conname` for the connection name, if `asid` is also specified

Any sublevel specification is valid for the QUERY option; for example:

- SUB(STR3)
- SUB(STR3.ASID(5))

Only the GLOBAL and fully qualified connection sublevel specifiers are valid with a COMP parameter; for example:

- SUB(GLOBAL)
- SUB(STR3.ASID(5).CON3)

Output from a SYSXES trace

Figure 168 on page 503 is an example of SYSXES component trace records formatted with the following subcommand:

```
CTRACE COMP(SYSXES) SUB((GLOBAL)) SHORT OPTIONS((CONNECT,HWLAYER))
```

```
COMPONENT TRACE SHORT FORMAT
COMP(SYSXES)  SUBNAME((GLOBAL))
**** 10/20/93

MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
--------  --------  ---------------  -----------

HWLAYER   090C0002  20:47:22.096016  EXIT FROM IXLMLTAM
HWLAYER   07140001  20:47:22.096296  ENTRY TO IXLERTRR
HWLAYER   07140002  20:47:22.096429  EXIT FROM IXLERTRR
CONNECT   08190001  20:47:30.171676  CONNECTOR DIE ROUTINE
HWLAYER   090C0001  20:47:30.171718  ENTRY TO IXLMLTAM
HWLAYER   09030001  20:47:30.171758  ENTRY TO IXLMLXRB
HWLAYER   09080001  20:47:30.171779  ENTRY TO IXLM2SR START IMMED RE
HWLAYER   09080003  20:47:30.171804  ISSUING A SMSG COMMAND
CONNECT   08110001  20:47:30.172316  MAINLINE TIMER EXIT ENTERED
CONNECT   08110004  20:47:30.172476  MAINLINE TIMER EXITED
HWLAYER   09080004  20:47:30.180754  COMPLETION OF A SMSG COMMAND
```

Figure 168. Example: formatted SYSXES component trace records

Chapter 13. Transaction trace

Transaction trace provides a consolidated trace of key events for the execution path of application or transaction type *work units* running in a multi-system application environment. By tracing the path of a work unit running in a single system, or (more importantly) across systems in a sysplex environment, that is being processed by multi-system transaction servers, subsystem interfaces, and resource managers, transaction trace enables a system programmer to debug problems in those environments.

The essential task of transaction trace is to aggregate data showing the flow of work between components in the sysplex that combine to service a transaction. Transaction trace traces events such as component entry, exit, exceptions and major events such as COMMIT, and ROLLBACK. Do not use transaction trace as a component tracing facility.

How transaction trace works

Transaction trace (TTrace) is attached as a daughter task in the system trace address space, after master scheduler initialization completes. Once initialization has completed, and the first transaction trace command is entered with a filter that specifies the attributes of the work unit(s) to be traced, transaction trace is activated. Additional information, such as the use of an external writer, is also allowable for transaction trace processing.

Once transaction trace is activated, WLM Classify invokes a filter exit to determine whether the current work unit is traced. The work unit's attributes are compared with the command filter attributes to determine if tracing should occur. If tracing is required, a non-zero token is built and returned to the Classify caller. If no tracing is performed for that work unit, set the transaction trace token to zero. The caller (CICS or IMS, for example) propagates the token in a manner similar to the propagation of the service class token.

Next, transaction trace macros:

- determine if tracing can be performed (ITZQUERY)
- initiate the writing of a transaction trace record (ITZEVENT).

Transaction trace writes trace data in a transaction trace data space in the trace address space. When an external writer is defined, the record is also written to the external writer. Use interactive problem control system (IPCS) to view the transaction trace records.

Transaction trace commands

Use the following commands with transaction trace. For information about using the TRACE or DISPLAY TRACE commands with transaction trace, see *z/OS MVS System Commands*.

- TRACE TT
- DISPLAY TRACE,TT

The TRACE TT command

Transaction trace uses the MVS TRACE command with the TT keyword to:

- Start transaction trace.
- Add additional trace filter sets.
- Remove an active trace filter set.
- Stop transaction trace.
- Start a CTRACE external writer.
- Stop a CTRACE external writer.

- Change the transaction trace buffer size.
- Specify a level indicator.
- Specify whether or not latent transactions is traced.

Starting transaction trace

Transaction trace is started when a TRACE TT command is issued with filter information. Following is an example of defining a transaction trace filter set with a user ID of TESTERP1 and transaction name of TRAN1.

```
trace tt,user=testerp1,tran=tran1
ITZ002I 'BUFSIZ' IS SET TO 0001M
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 01
```

When multiple filter keywords are specified, as in the preceding example, a 'logical AND' is used to determine if the transaction should be traced or not traced.

Adding additional trace filter sets

Up to five transaction trace filter sets can be concurrently active. They are activated when the TRACE TT command is issued with filter information. The command in the following example defines an additional transaction trace filter set with a user ID of DONNA*. The use of an asterisk (*) in the last character position indicates a wildcard is being defined. When determining if a transaction trace token is to be created, any user ID with a prefix of DONNA will result in a match.

```
trace tt,user=donna*
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 02
```

If multiple filter sets are specified a 'logical OR' is used among the filter sets to determine if the transaction should be traced or not traced.

Removing an active trace filter set

A transaction trace filter set is removed when the OFF=x keyword is used. For example, the following command indicates that the transaction trace filter set 02 has been turned off.

```
trace tt,off=2
ITZ016I TRANSACTION TRACE FILTER SET TURNED OFF
```

Stopping transaction trace

Use the OFF=ALL keyword to stop transaction trace. For example:

```
trace tt,off=all
ITZ007I TRANSACTION TRACE IS NO LONGER ACTIVE.
A DUMP COMMAND MAY BE ISSUED TO DUMP THE TRANSACTION TRACE
DATA SPACE.
```

Use the DUMP command to dump the transaction trace data space. For example:

```
DUMP COMM=(TTrace for TRAN=ATM1)
```

```
R x,DSPNAME='TRACE'.SYSTTRC
```

Starting a CTRACE external writer

Transaction trace supports the use of an external writer for processing transaction trace records. An external writer is specified on the initial command that activates transaction trace or is specified standalone while transaction trace is active. For example:

```
trace tt,wtr=abcdefg
```

Component trace messages are issued in response to this command.

Stopping a CTRACE external writer

Transaction trace external writer processing can be stopped with the use of the WTR=OFF keyword. For example:

```
trace tt,wtr=off
```

Component trace messages are issued in response to this command.

Changing the data space size

The transaction trace TTRACE TT command allows a change in the transaction trace data space size. The data space is from 16K to 999K or 1M to 32M. For example:

```
trace tt,bufsiz=2m
ITZ002I 'BUFSIZ' IS SET TO 0002M
```

Specifying a level indicator

The transaction trace TTRACE TT command allows definition of a level indicator for each filter set.

- 1 pertains to component entry, exit, exceptions, and major events.
- 2 pertains to detail, controlled by component external; the default is 2

```
trace tt,bufsiz=2m,user=testerp1,tran=tran1,lvl=01
ITZ002I 'BUFSIZ' IS SET TO 0002M
ITZ001I TRANSACTION TRACE IS NOW ACTIVE WITH FILTER SET 01
```

Tracing latent transactions

Use the transaction trace TTRACE TT command to specify whether or not latent transactions are traced. The default is to trace latent processing. Consider the following when deciding what to specify:

- The transaction is currently active in the system.
- The transaction is marked for tracing.
- The filter set used to mark the transaction eligible for tracing is no longer active.

```
trace tt,latent=no
ITZ002I 'LATENT' IS SET TO NO
```

DISPLAY TRACE,TT

Use the TT keyword on the DISPLAY TRACE command to determine the status of transaction trace. Do not use the component trace display command to inquire on the status of transaction trace. In addition to displaying information specified on the TRACE TT command, the DISPLAY TRACE,TT response also displays a list of the systems participating in transaction trace sysplex processing. Figure 169 on page 508 is an example of a DISPLAY TRACE,TT command response.

```
IEE843I 14.47.19  TRACE DISPLAY
SYSTEM STATUS INFORMATION
ST=(ON,0064K,00064K) AS=ON  BR=OFF EX=ON
MT=(ON,024K)
--------------------------------------------------------------
TRANSACTION TRACE STATUS: ON
   BUFSIZ= 0002M     WRITER= *NONE*      LATENT= YES
   01:  TRAN= TRAN1                  USER= TESTERP1
        LVL = 001
   02:  USER=DONNA*                LVL = 002
   SYSTEMS PARTICIPATING IN TT: SYS1     SYS2     SYS3
```

Figure 169. Example: DISPLAY TRACE,TT command response

Using IPCS to view transaction trace output

Use the IPCS subcommand CTRACE COMP(SYSTTRC) to view transaction trace records. To obtain a sysplex TTrace stream, use the IPCS MERGE subcommand to format TTrace records from multiple input data sets. Any generalized trace facility (GTF) records imbedded in the TTrace records are processed without having to specify additional keywords to the above command. Figure 170 on page 508 is an example of a short IPCS CTRACE COMP(SYSTTRC) SHORT command response.

```
ctrace comp(systtrc) short
COMPONENT TRACE SHORT FORMAT
COMP(SYSTTRC)
**** 09/23/1999
SYSNAME   MNEMONIC  ENTRY ID  TIME STAMP       DESCRIPTION
-------   --------  --------  ---------------  ----------

SY1       TTCMD     00000002  14:17:20.833847  TRACE TT Command

SY1       TTCMD     00000002  14:18:11.611755  TRACE TT Command

SY1       EVENT     00000003  14:31:55.813125  TRACE EVENT

SY1       EVENT     00000003  14:31:55.899216  TRACE EVENT

SY1       EVENTU    00000005  14:31:56.378480  TRACE EVENT with User
                                                   Data

SY1       EVENTG    00000004  14:31:56.818367  TRACE EVENT with GTF
                                                   Data
```

Figure 170. Example: IPCS CTRACE COMP(SYSTTRC) SHORT response

Figure 171 on page 509 is an example of a IPCS CTRACE COMP(SYSTTRC) LONG command response.

```
ctrace comp(systtrc) full
COMPONENT TRACE FULL FORMAT
COMP(SYSTTRC)
**** 09/23/1999
SYSNAME   MNEMONIC  ENTRY ID  TIME STAMP       DESCRIPTION
-------   --------  --------  ---------------  ----------
SY1       TTCMD     00000002  14:17:20.833847  TRACE TT Command
 CMDID.....0501
 COMMAND...TRACE TT,BUFSIZ=2M,USER=TESTERP1,TRAN=TRAN1,
 LVL=01

SY1       TTCMD     00000002  14:18:11.611755  TRACE TT Command
 CMDID.....0402
 COMMAND...TRACE TT,USER=DONNA*

SY1       EVENT     00000003  14:31:55.813125  TRACE EVENT

 COMPONENT..COMP      EVENTDESC..TTVAPIEA008       CMDID.....0501
 FUNCTION...TEST_ITZEVENT_WITH_FUNCTIONNAME.  TCB...007ED9C8
 ASID..0022
 TRACETOKEN..SY1       B2E447A3  F32E4048  05010100  00000000

SY1       EVENT     00000003  14:31:55.899216  TRACE EVENT

 COMPONENT..COMP      EVENTDESC..TTVAPIEA009       CMDID.....0000
 FUNCTION...................................  TCB...007ED7A8
 ASID..0022
 TRACETOKEN..          40404040  40404040  40404040  40404040
 LATENT workunit traced.

SY1       EVENTU    00000005  14:31:56.378480  TRACE EVENT with User
                                                       Data

 COMPONENT..COMP      EVENTDESC..TTVAPIEA003       CMDID.....0501
 FUNCTION...................................  TCB...007ED148
 ASID..0022
 TRACETOKEN..SY1       B2E447A3  F5D056C1  0105FF00  00000000
  +0000  E3C8C9E2  40C9E240  E3E340C4  C1E3C140 | THIS IS TT DATA  |
  +0010  C6D6D940  C140E3D9  C1D5E2C1  C3E3C9D6 | FOR A TRANSACTIO |
  +0020  D540E3D9  C1C3C540  D9C5C3D6  D9C44BE3 | N TRACE RECORD.T |
  +0030  C8C9E240  C9E240E3  E340C4C1  E3C140C6 | HIS IS TT DATA F |
  +0040  D6D940C1  40E3D9C1  D5E2C1C3  E3C9D6D5 | OR A TRANSACTION |
  +0050  40E3D9C1  C3C540D9  C5C3D6D9  C44B     |  TRACE RECORD.   |

SY1       EVENTG    00000004  14:31:56.818367  TRACE EVENT with GTF
                                                       Data

 COMPONENT..COMP      EVENTDESC..TTVAPIEA004       CMDID.....0402
 FUNCTION...................................  TCB...007ED368
 ASID..0022
 TRACETOKEN..SY1       B2E447A3  F566F584  03030300  00000000
 HEXFORMAT AID FF FID 00 EID  E000
  +0000  E3C8C9E2  40C9E240  C7E3C640  C4C1E3C1 | THIS IS GTF DATA |
  +0010  40C6D6D9  40C140E3  D9C1D5E2  C1C3E3C9 |  FOR A TRANSACTI |
  +0020  D6D540E3  D9C1C3C5  40D9C5C3  D6D9C44B | ON TRACE RECORD. |
  +0030  E3C8C9E2  40C9E240  C7E3C640  C4C1E3C1 | THIS IS GTF DATA |
  +0040  40C6D6D9  40C140E3  D9C1D5E2  C1C3E3C9 |  FOR A TRANSACTI |
  +0050  D6D540E3  D9C1C3C5  40D9C5C3  D6D9C44B | ON TRACE RECORD. |
```

Figure 171. Example: IPCS CTRACE COMP(SYSTTRC) LONG response

Chapter 14. GETMAIN, FREEMAIN, STORAGE (GFS) trace

GFS trace is a diagnostic tool that collects information about the use of the GETMAIN, FREEMAIN, or STORAGE macro. You can use GFS trace to analyze the allocation of virtual storage and identify users of large amounts of virtual storage. You must use the generalized trace facility (GTF) to get the GFS trace data output.

The following topics describe GFS trace:

Starting and stopping GFS trace

The following procedure explains how to request a GFS trace.

1. In the DIAGxx parmlib member, set the VSM TRACE GETFREE parameter to ON and define the GFS trace control data.

   Example: DIAGxx parmlib member for starting GFS tracing: The following DIAGxx parmlib member starts GFS trace and limits the trace output to requests to obtain or release virtual storage that is 24 bytes long and resides in address spaces 3, 5, 6, 7, 8 and 9:

   ```
        VSM TRACE GETFREE (ON)
                  ASID (3, 5-9)
                  LENGTH (24)
                  DATA (ALL)
   ```

   You will need another DIAGxx parmlib member defined to stop GFS tracing. See "5" on page 512.
2. Ask the operator to enter the SET DIAG=xx command to activate GFS trace using the definitions in the DIAGxx parmlib member.
3. Start a GTF trace (ask the operator to enter a START *membername* command on the operator console). *membername* is the name of the member that contains the source JCL (either a cataloged procedure or a job). Tell the operator to specify a user event identifier X'F65' to trace GTF user trace records.

   Example: Starting a GTF trace for GFS data: In the following example, the operator starts GTF tracing with cataloged procedure GTFPROC to get GFS data in the GTF trace output. The contents of cataloged procedure GTFPROC are as follows:

   ```
   //GTF     PROC MEMBER=GTFPROC
   //* Starts GTF
   //IEFPROC EXEC PGM=AHLGTF,REGION=32M,
   //  PARM='MODE=EXT,DEBUG=NO,TIME=YES,BLOK=40K,SD=0K,SA=40K'
   //IEFRDER DD DSN=D31POOL.PJREDGTF.TRACE,
   //           DISP=SHR,UNIT=3380,VOL=SER=CTDSD1
   ```

   The operator then replies to messages AHL100A with the USRP option. When message AHL101A prompts the operator for the keywords for option USRP, the operator replies with USR=(F65) to get the GFS user trace records in the GTF trace output.

   ```
   START GTFPROC

   00 AHL100A SPECIFY TRACE OPTIONS

   REPLY 00,TRACE=USRP

   01 AHL101A SPECIFY TRACE EVENT KEYWORDS--USR=
   ```

```
REPLY 01,USR=(F65)
02 AHL102A CONTINUE TRACE DEFINITION OR REPLY END
REPLY 02 END
AHL103I TRACE OPTIONS SELECTED--USR=(F65)
03 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 03,U
```

4. To stop the GTF trace, ask the operator to enter a STOP *procname* command on the operator console.
5. To stop GFS trace, create a DIAGxx parmlib member with VSM TRACE GETFREE(OFF) and have the operator enter a SET DIAG=xx command.

   Example: DIAGxx parmlib member for stopping GFS tracing: The following DIAGxx parmlib member stops GFS trace:

   ```
       VSM TRACE GETFREE (OFF)
   ```

For additional information, see the following references:

- See *z/OS MVS Initialization and Tuning Reference* for the syntax of the DIAGxx parmlib member.
- See *z/OS MVS System Commands* for the syntax of the SET and START commands.
- See Chapter 10, “The Generalized Trace Facility (GTF),” on page 211 for information about how to specify GTF EIDs.

Receiving GFS trace data

GTF places the GFS trace data in a user trace record with event identifier X'F65'. To obtain GFS trace data, do one of the following:

- When GTF writes trace data in a data set, format and print the trace data with the IPCS GTFTRACE subcommand.
- When GTF writes trace data only in the GTF address space, use a dump to see the data. Request the GTF trace data in the dump through the SDATA=TRT dump option.
- Issue the IPCS GTFTRACE subcommand to format and see the trace in an unformatted dump.

See *z/OS MVS IPCS Commands* for the GTFTRACE subcommand.

Formatted GFS trace output

Figure 172 on page 513 shows an example of formatted GFS trace output.

```
READY
   IPCS NOPARM
IPCS
   DROPD DA('D10JHM1.VSMNEW.GTF')
BLS18206I All records for 1 dump dropped
IPCS
   SETD NOCONFIRM
IPCS
   GTFTRACE DA('D10JHM1.VSMNEW.GTF') USR(F65)
IKJ56650I TIME-03:42:20 PM. CPU-00:00:01 SERVICE-52291 SESSION-00:00:20 JANUARY 22,1998
BLS18122I Initialization in progress for DSNAME('D10JHM1.VSMNEW.GTF')
IKJ56650I TIME-03:42:21 PM. CPU-00:00:01 SERVICE-54062 SESSION-00:00:20 JANUARY 22,1998
   **** GTFTRACE DISPLAY OPTIONS IN EFFECT ****
 USR=SEL

**** GTF DATA COLLECTION OPTIONS IN EFFECT: ****
 USRP option

                 **** GTF TRACING ENVIRONMENT ****
      Release: SP6.0.6   FMID: HBB6606   System name: CMN
      CPU Model: 9672  Version: FF  Serial no. 270067


USRDA F65 ASCB 00FA0800              JOBN GTFJM2
Getmain SVC(120)  Cond=Yes
Loc=(Below,Below)  Bndry=Dblwd
Return address=849CA064  Asid=001A  Jobname=GTFJM2
Subpool=229  Key=0  Asid=001A  Jobname=GTFJM2    TCB=008DCA70 Retcode=0
Storage address=008D6768 Length=10392 X'2898'
  GPR Values
      0-3  00002898  00000000  7FFFC918  0B601E88
      4-7  01FE3240  008FF830  849CA000  00FA0800
      8-11 00000000  00000DE8  049CBFFE  849CA000
     12-15 049CAFFF  0B601A9C  00FE9500  0000E510

               GMT-01/06/1998 21:15:43.111628  LOC-01/06/1998 21:15:43.111628
   .
   .
   .

USRDA F65 ASCB 00FA0800              JOBN GTFJM2
Freemain SVC(120)  Cond=No
Return address=8B2D608A  Asid=001A  Jobname=GTFJM2
Subpool=230  Key=0  Asid=001A  Jobname=GTFJM2    TCB=008DCA70  Retcode=0
Storage address=7F73DFF8  Length=8 X'8'
  GPR Values
      0-3  00000000  7F73DFF8  008D82D8  008D7BC0
      4-7  008D8958  008D6B08  008D85C8  0B335000
      8-11 00000002  00000000  7F73DFF8  008D862C
     12-15 8B2D6044  008D8C98  849D242A  0000E603

               GMT-01/06/1998 21:15:43.111984  LOC-01/06/1998 21:15:43.111984

IPCS
  SETD   CONFIRM
IPCS
  END
READY
END
```

Figure 172. Example of formatted GFS trace output

The GETMAIN / FREEMAIN / STORAGE trace produces a second type of record with a slightly different format. Following is an example of this record type:

```
USRDA F65 ASCB 00F4C280            JOBN IYCSCTS6
          Releasing Subpool=230  Key=1  Asid=003E  TCB=008B11E0
          Storage address=7F653E00  Length=512 X'200'
```

This type of record is unique because it does not trace a return address. It writes whenever an individual area of storage is FREEMAINed as part of a subpool FREEMAIN request. There may be many of these records in a row. The last record of the sequence is followed by a record that indicates a subpool FREEMAIN request. This record includes the return address of the issuer of the subpool FREEMAIN.

Unformatted GFS trace output

"Layout of the GFS trace output" on page 514 shows unformatted GFS trace output as it would appear in the trace data set where GTF puts the output. You can use this information to write your own formatting or analysis routines.

Layout of the GFS trace output

```
Unformatted GFS Trace Output

Part 1 - This part is in every GFS trace entry.

Offset  Length  Description
---------------------------------------------------------------------
    0      1    Flags
    X'80' - Common storage
    X'40' - Caller's registers are traced
    X'20' - This is a subpool release range entry
    X'10' - Copy of VSWKOWNINFO
    X'08' - Obtained storage is all zeros.
    X'04' - Caller was AMODE 64
    X'02' - Caller was AMODE 31
    X'01' - Obtain address was explicitly specified
---------------------------------------------------------------------
    1      1    Actual subpool after translation
---------------------------------------------------------------------
    2      2    ASID which owns the storage
---------------------------------------------------------------------
    4      4    Address of storage area
---------------------------------------------------------------------
    8      4    Actual length of storage area
---------------------------------------------------------------------
    C      4    Address of TCB
---------------------------------------------------------------------
   10      1    Copy of VSWKSKEY
---------------------------------------------------------------------
   11      1    Copy of VSWKRC
---------------------------------------------------------------------
   12      1    Modification level number  X'01' - HBB6606
                                           X'02' - HBB7703
                                           X'03' - HBB7730
---------------------------------------------------------------------
   13      1    Reserved
---------------------------------------------------------------------
   14      2    Offset of Part 2
---------------------------------------------------------------------
   16      2    Offset of Part 3
---------------------------------------------------------------------
Part 2 - This part is in every GRS trace entry except for subpool
         release range entries.

Offset  Length  Description
---------------------------------------------------------------------
    0      4    Caller's return address
---------------------------------------------------------------------
    4      4    Minimum length for a variable request
---------------------------------------------------------------------
    8      4    Maximum length for a variable request
---------------------------------------------------------------------
    C      8    Name of job which owns the storage
---------------------------------------------------------------------
   14      8    Name of job which contained the program which
                requested the storage
---------------------------------------------------------------------
   1C      2    ASID which contained the program which
                requested the storage
---------------------------------------------------------------------
   1E      1    Copy of VSWKESPL
---------------------------------------------------------------------
   1F      1    Copy of VSWKSVC
---------------------------------------------------------------------
   20      1    Copy of VSWKRFLG
---------------------------------------------------------------------
   21      1    Copy of VSWKPFLG
---------------------------------------------------------------------
   22      1    Copy of VSWKFLGS
---------------------------------------------------------------------
   23      1    Copy of VSWKRFLG2
```

```
 --------------------------------------------------------------------
    24      4    Copy of VswkRetAddrHigh
 --------------------------------------------------------------------
    28      4    Copy of VswkAR15Value
 --------------------------------------------------------------------
    2C      4    Copy of VswkAR1Value
 --------------------------------------------------------------------
Part 3 - This part is in the GFS trace record if the caller's registers are traced.
Offset  Length  Description
 --------------------------------------------------------------------
     0    X'40'  Caller's registers 0-15
--------------------------------------------------------------------
```

Note: The IGVVSMWK macro contains field names beginning with VSWK. For more information, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Chapter 15. Recording logrec error records

When an error occurs, the system records information about the error in the logrec data set or the logrec log stream. The information provides you with a history of all hardware failures, selected software errors, and selected system conditions. Use the Environmental Record, Editing, and Printing program (EREP):

- To print reports about the system records
- To determine the history of the system
- To learn about a particular error

Collection of software and hardware information

Use the records in the logrec data set or the logrec log stream as additional information when a dump is produced. The information in the records will point you in the right direction while supplying you with symptom data about the failure. Figure 173 on page 517 shows the error processing for a logrec data set, named SYS1.LOGREC, which is the default name for the logrec data set.

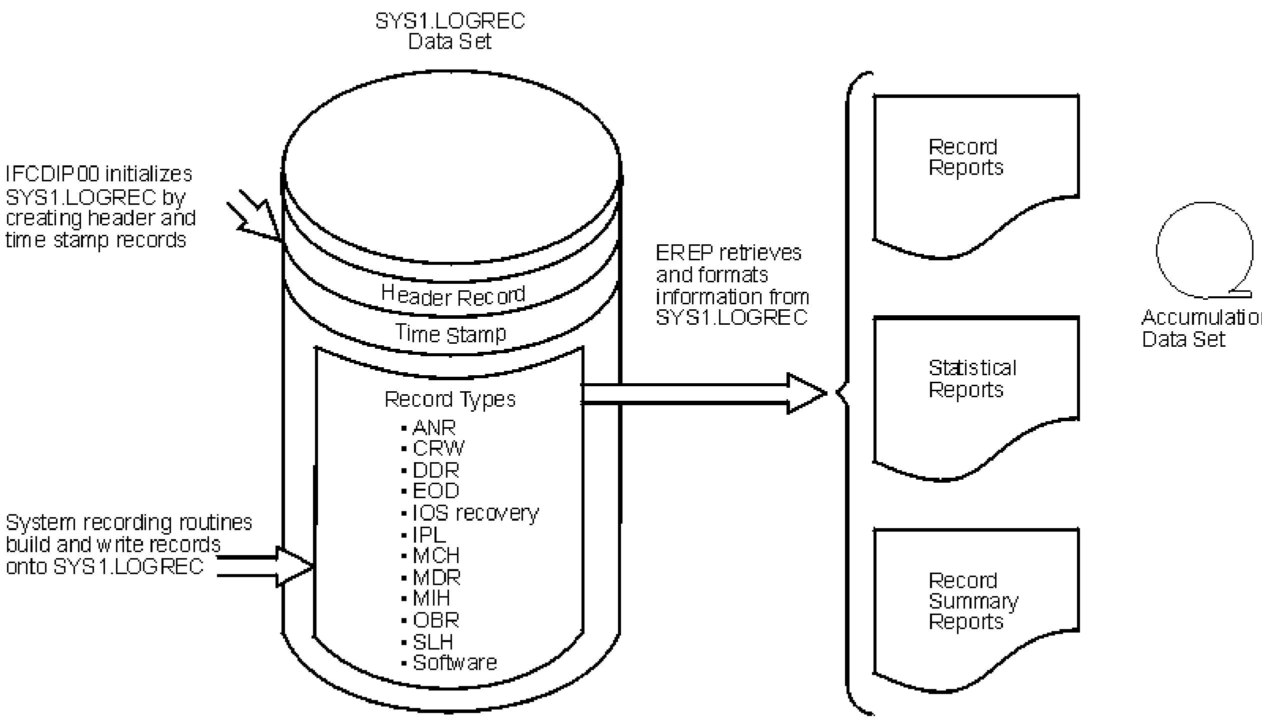


Figure 173. Logrec Error Recording Overview

You can set your system up to record errors on either a logrec data set or in a logrec log stream. This topic tells you what you need to know about each medium before deciding how to collect error records. To use the logrec data set or a logrec log stream, you need to know how to initialize each of them, how to record system events on each of them, how to collect the data when it is available, and how to interpret the output through EREP.

This topic describes each of these tasks:

- “Choosing the correct logrec recording medium” on page 518
- “Initializing and reinitializing the logrec data set” on page 518
- “Defining a logrec log stream” on page 519
- “Changing the logrec recording medium” on page 521
- “Error recording contents” on page 521

- “Obtaining information from the logrec data set” on page 524
- “Obtaining records from the logrec log stream” on page 526
- “Obtaining information from the logrec recording control buffer” on page 531
- “Interpreting software records” on page 532

Choosing the correct logrec recording medium

You can choose where the system will record logrec error records. When a system is not in a sysplex, an installation can use a logrec data set, associated with an individual system, to record error records. An installation can choose to continue this type of recording by initializing the logrec data set before IPLing the system that will use it.

In a sysplex, however, because each system requires its own logrec data set, you might need to look at each logrec data set when an error occurs.

To eliminate the problem of having to manage up to 32 logrec data sets, an installation can choose to define one coupling facility logrec log stream. Using a coupling facility logrec log stream eliminates the following:

- Running IFCDIP00 to initialize multiple logrec data sets
- Handling full or emergency data set conditions
- Scheduling the daily offload of logrec data sets
- Concatenating multiple history data sets
- Archiving logrec records

For more information, see the following references:

- See “Initializing and reinitializing the logrec data set” on page 518 if you want to initialize a logrec data set for your system.
- See “Defining a logrec log stream” on page 519 if you want to define a logrec log stream for your installation.
- See “Changing the logrec recording medium” on page 521 for more information about changing the logrec recording medium after you start the system.

Initializing and reinitializing the logrec data set

You must initialize the logrec data set before starting the system that will use it. You reinitialize the logrec data set when an uncorrectable error occurs. You clear the logrec data set when it is full or near full. You must also initialize a logrec data set before using the SETLOGRC command to switch to it.

To initialize or reinitialize the logrec data set, use the service aid program IFCDIP00. To clear a full logrec data set, use EREP. IFCDIP00 creates a header record and a time stamp record for the logrec data set.

Attention: The logrec data set is an unmovable data set. If you attempt to move it using a program, such as a defragmentation program, your system will experience difficulty both reading from and writing to the data set. Logrec records physical addresses into the logrec data set rather than to relative address, and this prevents the ability to move the data set. You can use the SETLOGRC command to switch to IGNORE, LOGSTREAM, or a different data set to free the data set and allow use of maintenance programs for the volume that contains the logrec data set. However, the data set itself remains unmovable.

Initializing the logrec data set

If the logrec data set does not exist, you must first allocate it and then initialize it.

Note: Whenever you allocate the logrec data set, it must be initialized before using the SETLOGRC command to change it.

Figure 174 on page 519 is an example of a JCL job that renames and uncatalogs an existing logrec data set and allocates, catalogs, and initializes a new one. (If you do not currently have a logrec data set, only use with the second step of the example JCL job.)

```
//KATHYLR JOB (9999),'CREATE NEW LOGREC DS',CLASS=A,MSGCLASS=X,
//        MSGLEVEL=(1,1),NOTIFY=KATHY
//*-----------------------------------------------------------------
//*  RENAME THE CURRENT LOGREC DATASET
//*  UNCATLG SYS1.LOGREC SO THE NEW LOGREC CAN BE ALLOCATED ON
//*  ANOTHER VOLUME, IF DESIRED
//*-----------------------------------------------------------------
//RENAME    EXEC PGM=IEHPROGM
//M43RES    DD VOL=SER=M43RES,UNIT=3390,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
         RENAME DSNAME=SYS1.LOGREC,VOL=3390=M43RES,                    X
               NEWNAME=SYS1.LOGREC.OLD
         UNCATLG DSNAME=SYS1.LOGREC
/*
//*-----------------------------------------------------------------
//*  CREATE THE NEW LOGREC DATASET AND INITIALIZE IT
//*-----------------------------------------------------------------
//IFCDIP00 EXEC    PGM=IFCDIP00,COND=(0,LT)
//SERERDS  DD  DSN=SYS1.LOGREC,DISP=(,CATLG),
//          VOL=SER=M43RES,UNIT=SYSDA,SPACE=(CYL,3,,CONTIG)
/*
                                                        //
```

Figure 174. Example: Changing the space allocation

Reinitializing the logrec data set

You need to reinitialize the logrec data set either when the data set is full or when an uncorrectable error occurs.

If the data set is full, use EREP to record the data in a history data set and reinitialize logrec.

In the case of an error, invoke IFCDIP00 with JCL statements to reinitialize your existing logrec data set. IFCDIP00 resets the logrec data set header record field to indicate that the entire data set can be used and clears the time stamp record to hexadecimal zeros.

For information on using EREP, see the *EREP User's Guide*.

Figure 175 on page 519 is an example of using the IFCDIP00 service aid to reinitialize the logrec data set. It use the following JCL statements:

- The JOB statement initiates the job; the job name INSERLOG has no significance.
- The EXEC statement specifies the program name (PGM=IFCDIP00).
- The SERERDS DD statement specifies the reinitialized logrec data set (in this case SYS1.LOGREC), which must be on a permanently mounted volume (VOL=SER=111111 in this example); the DDNAME must be SERERDS.

```
//INSERLOG  JOB
//STEP1     EXEC  PGM=IFCDIP00
//SERERDS   DD    DSNAME=SYS1.LOGREC,UNIT=3380,
//      VOL=SER=111111,DISP=SHR
```

Figure 175. Example: Reinitializing the logrec data set

Defining a logrec log stream

To use a logrec log stream, you must first prepare your installation to use system logger functions. IBM recommends that you use a coupling facility log stream for LOGREC so that you can merge data from multiple systems in a sysplex.

To obtain logrec records for a single system sysplex, you can also use a DASD-only log stream, which is single system in scope. Note that this is not recommended for a multi-system sysplex, because you can only have one system connect to a DASD-only log stream per sysplex. This means that if you make your logrec log stream DASD-only, only one system will be able to write and read from it. See the system logger chapter of *z/OS MVS Setting Up a Sysplex* for information on DASD-only log streams.

If you use the SETLOGRC command and specify a log stream name, you can additionally set up other DASD-only and coupling facility log streams to record logrec data.

See *z/OS MVS Setting Up a Sysplex* for more information about system logger setup.

The following steps describe how to use a coupling facility logrec log stream in place of a logrec data set:

1. Define a log stream using the system logger log stream definition utility, IXCMIAPU.

   IFBLSJCL (see Figure 176 on page 520) is available in SYS1.SAMPLIB as an example of using the administrative data utility, IXCMIAPU, to define the coupling facility logrec log stream to a sysplex.

   There are no restrictions on naming the log stream, however, when naming your log stream, consider making 'LOGREC' a part of the log stream name so as to differentiate it from other log streams.

```
//IFBLSJCL JOB
//* Member Name:  IFBLSJCL                                           *
//*  Descriptive Name:                                               *
//*   Sample JCL to provide an example of using the System Logger    *
//*   utility to define the Logrec log stream to a sysplex.          *
//* Function:                                                        *
//*   This JCL sample provides an example of running the System      *
//*   Logger utility (IXCMIAPU) to define the Logrec log stream      *
//*   in the logger inventory.                                       *
//*                                                                  *
//*   Note that the MAXBUFSIZE parameter must have at least 4068     *
//*   specified, or Logrec will not be able to write to the Log      *
//*   stream.                                                        *
//*                                                                  *
//* Suggested Modifications:                                         *
//*   Provide the specifications that are relevant for your          *
//*   installation on the SYSIN DATA TYPE(LOGR) definition.          *
//*   For example, the following parameters define the log stream    *
//*   data set attributes:                                           *
//*                                                                  *
//*    LS_DATACLAS(data class)        - Name of data class           *
//*    LS_MGMTCLAS(management class)  - Name of management class     *
//*    LS_STORCLAS(storage class)     - Name of storage class        *
//*                                                                  *
//* Distribution Library:  ASAMPLIB                                  *
//*                                                                  *
//DEFINE   EXEC PGM=IXCMIAPU
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
 DATA TYPE (LOGR)
    DEFINE STRUCTURE NAME(LOGRECSTRUCTURE)
           LOGSNUM(1)
           AVGBUFSIZE(4068)
           MAXBUFSIZE(4068)
    DEFINE LOGSTREAM NAME(SYSPLEX.LOGREC.ALLRECS)
           STRUCTNAME(LOGRECSTRUCTURE)
/*
```

   Figure 176. Example: Sample JCL of using IXCMIAPU

   Note: MAXBUFSIZE must be at least 4068 because logrec writes records in one page blocks. Specify SMS storage group, storage, data and management classes such that when one data set is full, another is allocated. Allocate as much space as is allocated for all the logrec data sets on the systems in the sysplex before migrating to a logrec log stream.

   The most effective way to manage all logrec records is to specify the automatic migration of log data sets to HSM. This automatic migration eliminates the need to create and maintain archival history data sets, with one exception. If the log stream data set directory is full, you can, using *SUBSYS-options2* of the LOGR subsystem, copy data from a log stream to a history data set and then delete the copied data from the log stream.

2. Either specify LOGREC=LOGSTREAM in the IEASYSxx parmlib member, or use the SETLOGRC command to change the logrec recording medium to a logrec log stream. In general, any records written to any logrec data sets before changing to a logrec log stream must be read by a separate EREP job. However, the MERGE option can be used to combine logrec output from the log stream with a logrec data set in a single EREP job.
3. Change the EREP job stream as follows:
   - Change the SERLOG DD DSN= statement pointing to a logrec data set to an ACCIN DD DSN= statement pointing to a logrec log stream name, with corresponding SUBSYS parameters, to associate EREP with the logrec log stream. The SUBSYS parameters are described in "Obtaining records from the logrec log stream" on page 526.
   - Identify the input as a history data set. Leave the output to a history data set as currently recommended, because all subsequent steps should already use the history data set as input.

     Note: Using a logrec log stream as input for multiple steps is not recommended because each subsequent step processes more records than the prior, causing numbers and data in successive reports not to match.
   - Subsequent EREP report steps that normally process history data sets no longer need to concatenate one history data set per system.

For more information, see the following documentation:

- See *z/OS MVS Setting Up a Sysplex* for information about preparing an installation to use system logger functions.
- See *EREP User's Guide* for more information about running an EREP job to obtain a history data set.
- See *z/OS MVS System Commands* for more information about the SETLOGRC command.
- See *z/OS MVS Initialization and Tuning Reference* for more information about the IEASYSxx parmlib member.

Changing the logrec recording medium

You can use the SETLOGRC command to dynamically change your logrec recording medium regardless of the setting with which you started the system. Additionally, you can change to different logrec data sets or log streams. For more information, see *z/OS MVS System Commands*.

Error recording contents

The system creates records for every hardware or software failure and system condition that occurs and stores these records in the logrec data set or the logrec log stream. The records can contain two types of data that document failures and system conditions:

- **Error statistics**, which include the number of times that channels, machine models, and I/O devices have failed
- **Environmental data**, which include time and circumstances for each failure or system condition

Note: A programmer can also build symptom records using the SYMRBLD macro and have those records written into the logrec data set or the logrec log stream using the SYMREC macro.

See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for information about the macros.

Each record is recorded in hexadecimal format as an undefined length record. Each record provides:

- Relevant system information at the time of the failure
- Device hardware status at the time of the failure
- Results of any device/control unit recovery attempt
- Results of any software system recovery attempt

- Statistical data

When taken as a whole, these records create a history of the system, which begins early in system initialization and ends when the system stops. These records contain:

- **Full Abend History**: The system writes a logrec record for every abend, regardless of whether the dump is requested or suppressed. The logrec data set or the logrec log stream contains a full record of abnormal ends.
- **System Initialization Errors**: The system writes errors during system initialization, before other diagnostic services are completely functioning.
- **Lost Record Counts**: The system writes a logrec record to summarize lost error records. Sometimes hardware-detected or software-detected errors occur close together. When errors are too close together, the system cannot write an individual record for each error; instead, the system counts the errors and writes a summary record.

These sections describe what is in the logrec data set:

- “Logrec data set header record” on page 522
- “Logrec data set time stamp record” on page 522

This section describes what is in the logrec data set or the logrec log stream:

- “Types of logrec error records” on page 523

See *z/OS MVS Diagnosis: Reference* for the format of the header record, time stamp record, and logrec error records.

Logrec data set header record

IFCDIP00 creates a header record on the logrec data set. The logrec data set header record includes:

- Information that the system recording routines can use to determine where to write new record entries onto the logrec data set
- Information that EREP can use to find existing record entries on the logrec data set. This information is valuable when you run an EREP report to find a particular error.
- Information that the system recording routines can use to issue a warning message when the logrec data set is 90% full.

Note: The logrec log stream does not have a header record generated.

Logrec data set time stamp record

IFCDIP00 creates a time stamp record on the logrec data set in the first record space following the header record. The time stamp record provides current date and time information for the IPL record. This allows you to measure the approximate time interval, recorded in the IPL records, between the ending and reinitialization of the operating system.

At preset time intervals, the system obtains the current date and time and writes this information on the time stamp record, overlaying the previous date and time.

During a subsequent initialization of the system, the system obtains the date and time from the time stamp record and adds it to the IPL record.

If IFCDIP00 is used to reinitialize the logrec data set, the information in the time stamp record is overlaid with hexadecimal zeros until the system writes the current date and time.

Note: The logrec log stream does not have a time stamp record generated.

Types of logrec error records

When the logrec data set or the logrec log stream is initialized, the system begins recording events. The system records the following types of error records, containing device-dependent or incident-dependent information:

- Asynchronous notification record (ANR) records for:
  - External timer reference (ETR) records for information related to Sysplex Timer incidents.
  - Direct access storage device (DASD)-service information message (SIM) records for information concerning servicing needs.
  - Link maintenance information (LMI) records for information for a particular link incident.
- Channel report word (CRW) records for:
  - Channel path error
  - Subchannel error
  - Configuration alert error
  - Monitoring facility error
- Dynamic device reconfiguration (DDR) records for:
  - Operator and system swaps between direct access and magnetic tape devices
  - Operator swaps on unit record devices
- End-of-day (EOD) records for information related to end-of-day and system ending conditions whenever the RDE option has been included in the system.
- Input/output supervisor (IOS) records for information related to IOS recovery actions.
  - Dynamic pathing services validation (DPSV) records for recovery actions.
- Initial program load (IPL) records for information related to system initializations whenever the RDE option has been included in the system.
- Machine check handler (MCH) records for:
  - Central processor failure
  - Storage failure
  - Storage key failure
  - Timer failure
- Miscellaneous data (MDR) records for:
  - Buffer overflow and device failures on buffered log devices
  - Demounts on DASD with buffered logs
  - Demounts by the DFSMSdss program between DASD having buffered logs and removable disk packs
  - Device failures on teleprocessing devices connected to an IBM communication controller
  - Statistical recording by EREP on DASD with buffered logs
- Missing interruption handler (MIH) records for:
  - Missing I/O interruptions
  - Specified time intervals
  - Recovery actions required
  - Recovery actions performed
- Outboard (OBR) records for:
  - Counter overflow statistics and device failures on devices supported by the teleprocessing access methods
  - End-of-day (EOD) requests

  - Paging I/O errors
  - Permanent channel and I/O device failures
  - Statistic counter overflow
  - Temporary or intermittent I/O device failures
  - Demounts on IBM magnetic tape drives
  - Devices that have their own diagnostic buffers
  - Statistical recording by EREP on DASD with buffered logs
- Subchannel logout handler (SLH) records for channel errors.
- Software records, including:
  - Machine checks (hardware-detected hardware errors, such as software recovery attempts for hard machine failures)
  - Program checks (hardware-detected software errors)
  - Restart errors (operator-detected errors)
  - Lost record errors (count of the records that did not fit in the buffer to be written to the logrec data set)
  - Software-detected errors, such as:
    - Abnormal ends, which are also called *abends*; reported in software records or erroneous supervisor call (SVC) instructions. These are known as SDWA-type software records.
    - Errors that are not abnormal ends; reported in symptom records.
    - Errors generated by application programs or system components; reported in symptom records.

As you can see, the system records a comprehensive list of error records that can help you when you need to diagnose a system failure.

Obtaining information from the logrec data set

You can obtain the information recorded in the logrec data set using EREP, which formats error records. EREP can perform the following functions:

- Create an accumulation data set from the logrec data set
- Clear the logrec data set
- Copy an input accumulation data set to an output accumulation data set
- Merge data from an accumulation data set and the logrec data set
- Print a detailed description of selected hardware and software error records
- Summarize and print statistics for device failures

EREP places the information from the logrec data set into reports. Using JCL, you determine the type of report you want EREP to produce.

Using EREP

EREP presents information from the logrec software error records in five reports.

Detail Edit Report for an Abend

The system obtains most of the information for an abend logrec error record from the system diagnostic work area (SDWA). The report contents are:

- Record header: report type (SOFTWARE RECORD), system, job name, error identifier (ERRORID), date, and time
- Search argument abstract
- Serviceability information

- Time of error information
- Status information from the request block
- Recovery environment
- Recovery routine action
- Hexadecimal dump of the SDWA, including the variable recording area (VRA)

Figure 177 on page 525 shows how to generate detail edits and summaries of all software and operational records:

```
//STEP7    EXEC  PGM=IFCEREP1,PARM='CARD'
//ACCIN     DD   DSN=EHISTORY,DISP=SHR
//DIRECTWK  DD   UNIT=SYSDA,
//               SPACE=(CYL,5,,CONTIG)
//EREPPT    DD   SYSOUT=A,DCB=BLKSIZE=133
//TOURIST   DD   SYSOUT=A,DCB=BLKSIZE=133
//SYSIN     DD   *
PRINT=PS
TYPE=SIE
HIST
ACC=N
ENDPARM
```

Figure 177. Example: Printing a detail edit report

Detail edit report for a symptom record

The system obtains most of the information for a non-abend logrec error record from the symptom record identified in the SYMREC macro. A programmer can build the symptom record using the SYMRBLD macro. The report contents are:

- Record reader: report type (SYMPTOM RECORD), system, date, and time
- Search argument abstract
- System environment
- Component information
- Primary and secondary symptom strings
- Free-format component information
- Hexadecimal dump of the symptom record

System summary report

The report summarizes errors for each of your installation's principle parts, or subsystems: processors, channels, subchannels, storage, operating system control programs, and I/O subsystems. The report contents are:

- Record header: report type (SYSTEM SUMMARY), system, date, time
- Total errors and errors for each processor for the following types of errors:
  - IPL
  - Machine check
  - Program error
  - End of day
- Identifications for processors in the report

Event history report

The report shows the error history: the frequency, order, and pattern of errors. The report contents are:

- Record header: report type (EVENT HISTORY)
- Abstracts for abend and non-abend logrec error records in chronological order
- Totals of the types of logrec error records for the system and for each processor

The JCL shown in Figure 178 on page 526 defines a two-step job. The first step prints an event history report for all logrec data set records. The second step formats each software, IPL, and EOD record individually. The event history report is printed as a result of the EVENT=Y parameter on the EXEC statement of the first step. It can be a very useful tool to the problem solver because it prints the records in the same sequence they were recorded and therefore shows an interaction between hardware error records and software error records.

```
//EREP JOB MSGLEVEL=1
//EREPA EXEC PGM=IFCEREP1,PARM='EVENT=Y,ACC=N',
//    REGION=256K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREPPT DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN  DD DUMMY
//EREPB EXEC PGM=IFCEREP1,PARM='TYPE=SIE,PRINT=PS,ACC=N',
//    REGION=256K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREPPT DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN  DD DUMMY
/*
```

Figure 178. Example: Printing an event history report

Detail summary report

The report summarizes information about data in logrec error records. The report contents are:

- Record header: report type being summarized
- Summary information and counts

The example in Figure 179 on page 526 shows how to generate detail summaries of all I/O errors.

```
//STEP6   EXEC  PGM=IFCEREP1,PARM='CARD'
//ACCIN    DD DSN=EHISTORY,DISP=(OLD,PASS)
//DIRECTWK DD UNIT=SYSDA,
//            SPACE=(CYL,5,,CONTIG)
//EREPPT   DD SYSOUT=A,DCB=BLKSIZE=133
//TOURIST  DD SYSOUT=A,DCB=BLKSIZE=133
//SYSIN     DD  DSN=EREP.PARMS(STEP6),
//              DISP=(OLD,PASS)
//          DD  DSN=EREP.CONTROLS,
//              DISP=(OLD,PASS)
PRINT=SU
TYPE=DOTH
DEV=(N34XX,N3704,N3705,N3720,N3725,N3745)
HIST
ACC=N
ENDPARM
```

Figure 179. Example: Printing a detail summary report

Obtaining records from the logrec log stream

You can access records in the logrec log stream by either:

- Writing a program using IXGCONN and IXGBRWSE services, see "Using System Logger services to obtain records from the logrec log stream" on page 526.
- Using EREP. see "Using EREP to obtain records from the logrec log stream" on page 527.

Using System Logger services to obtain records from the logrec log stream

You can obtain records from the logrec log stream by writing a program that uses the IXGCONN and IXGBRWSE system logger services to return log data. The data returned by the IXGBRWSE service for the logrec log stream is mapped by the IFBLOGLB data area. (See *z/OS MVS Programming: Assembler Services Guide* for information on using system logger services.)

Note that the logrec log stream output from the IXGBRWSE service contains an individual log stream record. However, the log stream record actually contains a group of records. The logrec log stream record is mapped by the IFBLOGLB mapping macro. Fr information on the IFBLOGLB mapping macro, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Using EREP to obtain records from the logrec log stream

You can use EREP to access the records in the logrec log stream for each system. The log stream subsystem allows existing programs to access error records from a log stream in the same way records were accessed from a logrec data set. See *z/OS MVS Programming: Assembler Services Guide* for information about using and starting the log stream subsystem.

JCL for the LOGR Subsystem

Use the SUBSYS parameter to call the log stream subsystem (LOGR) to access log stream data.

```
//ddname   DD  DSNAME=log.stream.name,
//           SUBSYS=(LOGR[,exit_routine_name][,'SUBSYS-options1'][,'SUBSYS-options2'])

where:SUBSYS-options1:
[FROM={({[yyyy/ddd][,hh:mm[:ss]]})  | OLDEST}]
[TO={({[yyyy/ddd][,hh:mm[:ss]]})  | YOUNGEST}]
[,DURATION=(nnnn,HOURS)]
[,VIEW={ACTIVE|ALL|INACTIVE}]
[,GMT|LOCAL] SUBSYS-options2:
defined by the log stream owner
```

Note: Quotation marks around keywords are required when parentheses, commas, equal signs, or blank characters are used within the SUBSYS keyword.

Other DD keywords are validated, if specified, but are ignored in the LOGR subsystem processing.

DSNAME=*log.stream.name*
: Specifies the name of the log stream to read. The name can be 1 to 26 characters in a data-set-name format.

SUBSYS=(LOGR[,*exit_routine_name*][,'*SUBSYS-options1*'][,'*SUBSYS-options2*'])
: Specifies that processing of this DD is to be handled by the LOGR subsystem. The *exit_routine_name* is the second positional parameter and specifies the name of the exit routine to receive control from the LOGR subsystem.

- Specify or use the default value to IXGSEXIT to use the log stream subsystem exit routine.
- Specify IFBSEXIT to access records from the logrec log stream. See SUBSYS-options2 for logrec-specific paramters.
- Specify IFASEXIT to access records from SMF log streams. See SUBSYS-options2 for SMF-specific parameters.

SUBSYS-options1
: Specifies options that are meaningful to all exit routines. See the documentation for a specific log stream exit for exceptions to these common options. The keywords are:

FROM=*starting_time*
: Indicates the starting time of the first log stream block to be processed based on the log stream view that the VIEW keyword specifies. The first block is the one with a time stamp later than or equal to the specified time.

OLDEST
: Indicates the first block read is the oldest block on the log stream. OLDEST is the default.

yyyy/ddd
: Specifies the start date. If the date is omitted, the current date is assumed. *yyyy* is a 4-digit year number and *ddd* is a 3-digit day number from 001 through 366 (366 is valid only on leap

years). For example, code February 20, 2000 as 2000/051, and code December 31, 1996 as 1996/366.

hh:mm[:ss]
Specifies the start time. If the time is omitted, the first block written after midnight is used. *hh* is a 2–digit hour number from 00 to 23, *mm* is a two digit minute number from 00 to 59, and *ss* is a 2–digit second number from 00 to 59. The seconds field and associated : delimiter can be omitted if it is not required by the log stream owner.

The FROM keyword is mutually exclusive with the DURATION keyword.

TO=*ending_time*
Indicates the ending time of the last log stream block to be processed based on the log stream view that the VIEW keyword specifies. The last block is the one with a time stamp earlier than or equal to the specified time.

YOUNGEST
Indicates the last block read will be the youngest block on the log stream at the time the allocation for the DD occurs. YOUNGEST is the default.

yyyy/ddd
Specifies the end date. If the date is omitted, the current date is assumed. *yyyy* is a 4-digit year number and *ddd* is a 3-digit day number from 001 through 366 (366 is valid only on leap years). For example, code March 7, 2001 as 2001/066, and code November 12, 2000 as 2000/317.

hh:mm[:ss]
Specifies the end time. If the time is omitted, the last block written before midnight will be used. If the end date is the same as the current day, then the youngest block on the log stream at the time the allocation for the DD occurs will be used. *hh* is a 2–digit hour number from 00 to 23, *mm* is a two digit minute number from 00 to 59, and *ss* is a 2–digit second number from 00 to 59. The seconds field and associated: delimiter can be omitted if it is not required by the log stream owner.

The TO keyword is mutually exclusive with the DURATION keyword.

Note: If the value specified for the FROM keyword is greater than the value specified for the TO keyword, the system ends the jobstep with a JCL error.

DURATION=(*nnnn*,HOURS)
Specifies which blocks are to be processed. Each *n* is a numeric from 0 to 9. Specifying (nnnn,HOURS) requests the blocks for the last *nnnn* hours up to the youngest block that is to be processed based on the log stream view that the VIEW keyword specifies. The last *nnnn* hours are calculated from the current time of the allocation for the DD.

The first block is the one with a time stamp greater than or equal to the calculated start time. The last block read is the youngest block on the log stream at the time the allocation for the DD occurs.

The DURATION keyword is mutually exclusive with the TO and the FROM keywords.

VIEW=ACTIVE|ALL|INACTIVE
Specifies the view or portion of log data to be used to obtain records from the log stream. System logger maintains two kinds of log stream data in a log stream: an active portion and an inactive portion. The active portion of the log stream is the log data that the log stream owner has not logically deleted through an IXGDELET request. The inactive portion of the log stream is the log data that the log stream owner has logically deleted but that has not yet been physically deleted from the log stream because the retention period (RETPD) specified for the log stream has not yet expired.

The VIEW option designates the portion(s) of the log stream to be used to obtain log data from the log stream, in addition to applying the other parameters.

Because the other parameters also apply, the combination of the FROM, TO, or DURATION parameters and the VIEW parameter might mean that the log stream subsystem exit returns no log data or only a portion of the intended log data. For example, if FROM=starting_time and

VIEW=INACTIVE are both specified, and the starting_time is later (younger) than the log data in the inactive portion of the log stream, then there is no log data to meet the access criteria. In the same way, if TO=ending_time and VIEW=ACTIVE are both specified, and the ending_time is earlier (older) than the log data in the active portion of the log stream, then there is no log data to meet the access criteria.

ACTIVE
The view of the log stream is to include only active log data, in addition to applying the other log stream access parameters. ACTIVE is the default.

ALL
The view of the log stream is to include both active and inactive log data, in addition to applying the other log stream access parameters.

INACTIVE
The view of the log stream is to include only the inactive log data, in addition to applying the other log stream access parameters.

GMT|LOCAL
Specifies whether the time is local time (based on the time zone offset at the time the log was written) or GMT time. GMT is the default.

Additional parameters for system logger

Along with the general parameters that can be specified for a log stream subsystem data set, system logger provides additional parameters in the *SUBSYS-options2* specifications. The following values can be coded for a logrec log stream:

SUBSYS-options2
Specifies unique exit routine options. Refer to information provided by the specific log stream owner concerning these parameters.

LASTRUN
Indicates that the starting point of the records to be read from the logrec log stream will be from the last record read by a previous use of an application that used LASTRUN. The end point of the records will be to the youngest block in the logrec log stream.

LASTRUN is mutually exclusive with the FROM, TO and DURATION keywords in *SUBSYS-options1* and with DELETE from *SUBSYS-options2*.

DELETE
Indicates that log stream records are to be deleted from the logrec log stream. The log stream itself is not deleted and remains available for use.

If the logrec log stream has been opened in the job step, all records up to but not including the last complete block read by the program will be deleted from the logrec log stream.

If the logrec log stream has not been opened in the job step, all records prior to the time indicated on the TO keyword will not be deleted from the logrec log stream.

DELETE is mutually exclusive with the FROM and DURATION keywords in *SUBSYS-options1* and the LASTRUN and SYSTEM keywords from *SUBSYS-options2*.

DEVICESTATS
Requests that the device statistics kept on the system where this job is running are to be recorded in the logrec log stream before any records are read.

SYSTEM=*system name*
Indicates that only records originating from the specified *system name* are to be returned to the application reading the logrec log stream. The *system name* value should match the name specified in the SYSNAME parameter of the IEASYSxx parmlib member. SYSTEM= is mutually exclusive with the DELETE keyword from SUBSYS-options2.

Time of day considerations for logrec

When using the SUBSYS DD statement for LOGR, handle the time of day filtering carefully. The SUBSYS parameter does not accept a stop time of 24:00, but the EREP parameters do accept 24:00 as a stop time. If it is necessary to write JCL and EREP control statements, you might have to request filtering through both the SUBSYS DD statement and the EREP parameters:

- SUBSYS parameters use blocks of records, and filtering of these blocks is done using time stamps assigned after each logical record enclosed in a block has been assigned its own time stamp.

  Figure 180 on page 530 shows how to select logrec log stream records that were produced between 05:00 on June 1st, 1997, and the end of that day.

```
//ACCIN      DD DSN=SYSPLEX.LOGREC.ALLRECS,DISP=SHR,
//              DCB=(RECFM=VB,BLKSIZE=4000),
//              SUBSYS=(LOGR,IFBSEXIT,
//              'FROM=(1997/152,05:00),TO=(1997/153,23:59),GMT')
```

Figure 180. Example: Using SUBSYS parameters

- EREP parameters use logrec logical records. When you use the TIME parameter with EREP, you are specifying a range of hours and minutes of interest on each day selected.

  Table 75 on page 530 shows how to correctly select logrec records that were produced between 05:00 on June 1st, 1997, and the end of that day.

Table 75. Example: Using EREP parameters

Correct coding example	Incorrect coding example
`DATE=(97152-97152),TIME=(0500-2400)`	`DATE=(97152-97153),TIME=(0500-0000)`

Creating a history data set for log data

Use the JCL in “Example: Creating a history data set” on page 530 to create a history data set from log data recorded on the logrec log stream. In this example, DEVICESTATS requests device statistics and the records are to be recorded in the log stream. Records are read from the last block that was processed on the previous submission of a "LASTRUN" EREP job up to the youngest block in the log stream. The first time a job with the "LASTRUN" option is run, the records are read from the oldest block in the log stream.

Example: Creating a history data set

```
//EREPDALY EXEC  PGM=IFCEREP1,PARM=('HIST,ACC=Y,SYSUM')
//ACCIN    DD DSN=SYSPLEX.LOGREC.ALLRECS,
//            SUBSYS=(LOGR,IFBSEXIT,,'DEVICESTATS,LASTRUN'),
//            DCB=(RECFM=VB,BLKSIZE=4000)
//ACCDEV   DD DSN=EREP.HISTORY,
//            DISP=(NEW,CATLG),
//            DCB=(RECFM=VB,BLKSIZE=4000),
//            UNIT=SYSDA,SPACE=(CYL,(25,5))
//SERLOG   DD DUMMY
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,15,,CONTIG)
//TOURIST  DD SYSOUT=A,DCB=BLKSIZE=133
//EREPPT   DD SYSOUT=A,DCB=BLKSIZE=133
//SYSABEND DD SYSOUT=A
//SYSIN    DD DUMMY
/*
```

Producing an event history from logrec

Use the JCL shown in "Example: Producing an event history" on page 531 to produce an event history report from records on the logrec log stream. By not specifying the FROM or TO keywords, the default is FROM=OLDEST and TO=YOUNGEST, indicating processing should include records from the beginning of the log stream to the end of the log stream. By specifying a print data set, EREPPT, the report can be browsed online for an overview of significant activity. When reading records by date and time, you can provide both EREP and SUBSYS parameters. EREP selects records from those passed to it from the SUBSYS parameters.

Example: Producing an event history

```
//EREPNOW  EXEC  PGM=IFCEREP1,REGION=4M,
//               PARM='CARD'
//ACCIN      DD  DSN=SYSPLEX.LOGREC.ALLRECS,
//               DISP=SHR,
//               SUBSYS=(LOGR,IFBSEXIT,,)
//DIRECTWK   DD  UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//EREPPT     DD DSN=EREP.EVENT,DISP=(NEW,CATLG),
//            DCB=BLKSIZE=133,
//            UNIT=SYSDA,SPACE=(CYL,(25,5))
//TOURIST    DD SYSOUT=A,DCB=BLKSIZE=133
//SYSABEND   DD SYSOUT=A
//SYSIN      DD *
             EVENT
             HIST
             ACC=N
             TYPE=ACDEHIMOSX
             ENDPARM
/*
```

Obtaining information from the logrec recording control buffer

When the system writes a dump, the dump includes the records in the logrec buffer in storage; the buffer records have been either written to the logrec data set or are queued to be written to the logrec data set.

When you begin to diagnose a dump for a system problem, you can use IPCS to view the system records in the logrec recording control buffer.

The logrec recording control buffer is one of the most important areas to be used when analyzing problems in MVS. This buffer serves as the interim storage location for hardware and software error records that are queued to be written to the logrec data set. The buffer is significant because of the error history it contains. Also, any records in the buffer that have *not* reached the logrec data set are almost certainly related to the problem you are trying to solve.

Formatting the logrec buffer

To format the logrec buffer, use the IPCS subcommand VERBEXIT LOGDATA. The entries that are still in the buffer will be formatted in the same way as entries that are printed in the EREP detail edit report.

Finding the logrec and WTO recording control buffers

There are two recording control buffers (RCB) in the SQA. The system uses one buffer for logrec messages, and the other for WTO messages. The CVT+X'16C' (CVTRBCB) points to the recording buffers control block (RBCB). The RBCB contains the following information about the two recording control blocks (which are also referred to as RCBs or buffers):

For the logrec RCB:

- RBCB+X'10' (RBCBLRCB) points to the logrec buffer.
- RBCB+X'14' (RBCBLLEN) contains the length of the logrec buffer.

For the WTO RCB:

- RBCB+X'18' (RBCBWRCB) points to the WTO buffer.
- RBCB+X'1C' (RBCBWLEN) contains the length of the WTO buffer.

The logrec and WTO recording control buffers reside in fetch-protected SQA. Entries in these buffers have time stamps (8-byte TOD clock values) that allow you to look at a dump and create a chronological list of the logrec events and WTO messages.

Reading the logrec recording control buffer

The logrec recording control buffer is a “wrap-table” similar to the system trace table. The entries are variable in size. The latest entries are the most significant especially if they have not yet been written to the logrec data set. Knowing the areas of the system that have encountered errors and the actions of their associated recovery routines, information obtained from the logrec data set and from the logrec recording control buffer helps provide an overall understanding of the environment you are about to investigate.

Note: The SDWA in the logrec buffer is a compressed SDWA in which the recordable extensions start directly after the used portion of the SDWAVRA. The SDWAURAL field contains the length of the SDWAVRA.

You can find the oldest entry in the buffer by locating the end of the unused or free area, obtained from RCBFREE+RCBFLNG. (If this sum brings you to a point beyond the end of the buffer, subtract RCBTLNG from the sum.) You can also read the buffer backwards by using the entry length at the end of each entry. The latest entry appears directly before the free or unused area of the buffer.

For details about the format of the RCB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Interpreting software records

There are two types of software records that are recorded in the logrec data set or the logrec log stream:

- **Software record**

  The system generates these records, providing information from the system diagnostic work area (SDWA) that describes problems detected because of an abend or a program check. See “Detail edit report for a software record” on page 533 for more information.
- **Symptom record**

  Either a user's application program or the system can issue the SYMREC macro to request the creation of a symptom record. Generally, the symptom record describes problems not accompanied by an abend, but there are exceptions. See “Detail edit report for a symptom record” on page 537 for more information.

Use the search argument you obtain from the detail edit reports for either a software record or a symptom record to search for a known problem. If you do not conduct the search yourself, contact the IBM Support Center. The result of the search will be one of the following:

- The PTF that corrects the problem.

  Apply the PTF that corrects the error.
- The APAR, and possibly the related APAR, that describes the problem. In some cases, a temporary fix (either ZAP or update) or a procedure might circumvent the problem.

  Apply the temporary fix if it is available; otherwise, follow the circumvention procedure.
- A description of why the problem might have occurred, which often describes a frequent misuse of a product that causes the error record. This type of problem is referred to as a user error.

  When an error occurs because of the misuse of a product other than MVS, use the procedures documented for that product to determine how best to debug the problem.

For any case other than the three listed above, including the case where the service link database does not contain a record matching the search criteria, contact the IBM Support Center to report the problem.

Detail edit report for a software record

The detail edit report for a software record shows the complete contents of an error record for an abnormal end, including the system diagnostic work area (SDWA). The report is produced by EREP and, through the VERBEXIT LOGDATA subcommand, under IPCS.

Use the detail edit report for a software record to determine the cause of an abend, and the recovery action that the system or application has either taken or not taken. This report enables you to locate where an error occurred, similar to the analysis of an SVC dump. Once you locate the error, you can develop a search argument to obtain a fix for the problem.

For more information, refer to the following documentation:

- See *EREP User's Guide* for information about producing a detail edit report for an SDWA-type record.
- See *z/OS MVS IPCS Commands* for information about the VERBEXIT LOGDATA subcommand.

The example output shown in "Example: One record with SDWARC4 and 64-bit information" on page 533 is from one record with SDWARC4 and 64-bit information. This record also has information in the VRA, which is formatted.

Example: One record with SDWARC4 and 64-bit information

```
TYPE:  SOFTWARE RECORD      REPORT:  SOFTWARE EDIT REPORT           DAY.YEAR
       (SVC 13)                                         REPORT DATE: 343.04
FORMATTED BY: IEAVTFDE  HBB7703                          ERROR DATE: 336.04
                            MODEL:   2084                            HH:MM:SS.TH
                            SERIAL:  11778D                     TIME: 17:43:44.72

JOBNAME: PIDA1028   SYSTEM NAME: J50
ERRORID: SEQ=00757  CPU=0056  ASID=0097  TIME=17:43:44.7

SEARCH ARGUMENT ABSTRACT

  PIDS/5752SCLDR RIDS/IEWLDR00#L RIDS/IEWLUNF0 AB/S0378 PRCS/00000014
  RIDS/IEWLRECV#R

  SYMPTOM             DESCRIPTION
  -------             -----------
  PIDS/5752SCLDR      PROGRAM ID: 5752SCLDR
  RIDS/IEWLDR00#L     LOAD MODULE NAME: IEWLDR00
  RIDS/IEWLUNF0       CSECT NAME: IEWLUNF0
  AB/S0378            SYSTEM ABEND CODE: 0378
  PRCS/00000014       ABEND REASON CODE: 00000014
  RIDS/IEWLRECV#R     RECOVERY ROUTINE CSECT NAME: IEWLRECV

OTHER SERVICEABILITY INFORMATION

  SUBFUNCTION:             LSLOADER

SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE

  RECOVERY ROUTINE LABEL
  DATE ASSEMBLED
  MODULE LEVEL

TIME OF ERROR INFORMATION

  PSW: 07041000 80000000 00000000 012F902E
  INSTRUCTION LENGTH: 02   INTERRUPT CODE: 000D
  FAILING INSTRUCTION TEXT: 00181610 0A0D18CE 18FB180C

  BREAKING EVENT ADDRESS: 00000000_00000000
  AR/GR 0-1    00000000/00000000_84000000  00000000/00000000_84378000
  AR/GR 2-3    00000000/00000000_00000020  00000000/00000000_0000FC03
  AR/GR 4-5    00000000/00000001_008FD098  00000000/00000000_00FD5750
  AR/GR 6-7    01FF000C/00000000_00000003  01FF000C/00000001_00F52C00
  AR/GR 8-9    00000000/00000001_7F33F4A8  00000000/00000001_00001748
  AR/GR 10-11  00000000/00000001_2C417000  01FF000C/00000001_7F36EC88
  AR/GR 12-13  00000000/00000000_8651F240  00000000/00000001_7F33F0E8
  AR/GR 14-15  00000000/00000000_8651FF54  00000000/00000000_00000014

  HOME ASID: 0097    PRIMARY ASID: 0097    SECONDARY ASID: 0097
  PKM: 00C0          AX: 0000              EAX: 0000
```

```
  RTM WAS ENTERED BECAUSE AN SVC WAS ISSUED IN AN IMPROPER MODE.
  THE ERROR OCCURRED WHILE: A TYPE 1 SVC WAS IN CONTROL
                            A LOCKED OR DISABLED ROUTINE WAS IN CONTROL
  LOCKS HELD: LOCAL/CML
  NO SUPER BITS WERE SET.

RECOVERY ENVIRONMENT

  RECOVERY ROUTINE TYPE: FUNCTIONAL RECOVERY ROUTINE (FRR)
  PSW AT ENTRY TO FRR: 070C0000 86502368
  FRR PARAMETER AREA ON ENTRY TO FRR:
  +00  7F33F4A8 00000000 00000000 00000000 00000000 00000000
RECOVERY ROUTINE ACTION

  THE RECOVERY ROUTINE RETRIED TO ADDRESS 0651FFA2.
  THE REQUESTED SVC DUMP WAS SUCCESSFULLY STARTED.
  NO LOCKS WERE REQUESTED TO BE FREED.
  THE SDWA WAS REQUESTED TO BE FREED BEFORE RETRY.
  THE REGISTER VALUES TO BE USED FOR RETRY:
  REGISTERS 0-7
  GR: 008FDD00 00000000 7FFFC100 00000000  008FD098 7FFFC150 7FFFBF48 00F52C00
  AR: 00000000 00000000 00000000 00000000  00000000 00000000 01FF000C 01FF000C
  REGISTERS 8-15
  GR: 7F33F4A8 06520100 00FD5750 00000001  8651F240 7F33F0E8 7F33F638 00000000
  AR: 00000000 00000000 00000000 01FF000C  00000000 00000000 00000000 00000000

HEXADECIMAL DUMP

  HEADER
  +000    40831820    00000000    0004336F    17434472    | C.........?....|
  +010    0011778D    20848000                            |.....D..        |

  JOBNAME
  +000    D7C9C4C1    F1F0F2F8                            |PIDA1028        |

  SDWA BASE
  +000    00000C00    04378000    00000000    00000000    |................|
  +010    00000000    00000000    84000000    84378000    |........D...D...|
  +020    00000020    0000FC03    008FD098    00FD5750    |..........}Q...&|
  +030    00000003    00F52C00    7F33F4A8    00001748    |.....5..".4Y....|
  +040    2C417000    7F36EC88    8651F240    7F33F0E8    |...."..HF.2 ".0Y|
                          .
                          .
                          .
  +180    00000000    00000000    00000000    0009BD27    |................|
  +190    00FFA0B0                                        |....            |

  VARIABLE RECORDING AREA (SDWAVRA)

  +000    KEY: 37     LENGTH: 06
  +002    C4C4D5C1    D4C5                                |DDNAME          |

  +008    KEY: 39     LENGTH: 08
  +00A    6060C8C6    E2606060                            |--HFS---        |
                          .
                          .
                          .
    +0AA    0003                                            |..              |

  +0AC    KEY: 53     LENGTH: 00

  +0AE    KEY: FF     LENGTH: 00

  SDWA FIRST RECORDABLE EXTENSION (SDWARC1)
  +000    E2C3D3C4    D9D3E2D3    D6C1C4C5    D9404040    |SCLDRLSLOADER   |
  +010    40404040    40404040    40404040    00000000    |            ....|
  +020    00000000    00000000    00000000    00000014    |................|
  +030    00000000    00000000    F5F7F5F2    01000001    |........5752....|
                          .
                          .
                          .
  +1B0    00000000    00000000    00000000    00000000    |................|
  +1C0    D1F5F040    40404040                            |J50             |

  SDWA SECOND RECORDABLE EXTENSION (SDWARC2)
  +000    00000000    00000000    00000000    00000000    |................|

  SDWA THIRD RECORDABLE EXTENSION (SDWARC3)
  +000    00000000    00000000    00000000    00000000    |................|
  +010    00000000    00000000    00000000    00000000    |................|

  SDWA FOURTH RECORDABLE EXTENSION (SDWARC4)
```

```
  +000    00000000    84000000    00000000    84378000    |....D.......D...|
  +010    00000000    00000020    00000000    0000FC03    |................|
  +020    00000001    008FD098    00000000    00FD5750    |......}Q.......&|
  +030    00000000    00000003    00000001    00F52C00    |.............5..|
                          .
                          .
                          .
  +150    00000000    00000000    07041000    80000000    |................|
  +160    00000000    012F902E                            |........        |

  SDWA FIFTH RECORDABLE EXTENSION (SDWARC5)
  +000    00000000    00000000    00000000    00000000    |................|
  +010    00000000    00000000    00000000    00000000    |................|
  +020    00000000    00000000    00000000    00000000    |................|
  +030    00000000    00000000    00000000    00000000    |................|
                         .
                         .
                         .
  +090    00000000    00000000    00000000    00000000    |................|

  ERRORID
  +000    02F50056    00970009    BD27                    |.5...P....      |
```

TYPE: SOFTWARE RECORD
Indicates that the detail edit report is for an SDWA-type record.

REPORT DATE
Indicates the date on which the EREP report was created.

ERROR DATE
Indicates the date on which the error occurred.

TIME
Indicates the time, as local, at which the error occurred.

JOBNAME
If the jobname is NONE-FRR, the error being recorded occurred in system or subsystem code covered by a functional recovery routine (FRR).

SYSTEM NAME
Indicates the name of the system where the SDWA-type record was created.

ERRORID
Allows you to coordinate diagnostic information from logrec, the console log (SYSLOG), and system dumps. The ERRORID is a concatenation of the following:

SEQ
A unique number assigned to each error. The sequence number indicates the order of the errors, but the records might not be listed in order. It is important to scan all entries and examine the sequence numbers to understand which error occurred first.

You might find the same sequence number used in more than one entry when several recovery routines, as a result of percolation, get control and request recording for the same error; however, the error time stamp will be different.

CPU
The internal identification number of the central processor that the failing process was running on at the time the error occurred. Use information from the system trace table about this CPU to learn more about the error.

ASID
The address space identifier (ASID) of the current, or home, address space at the time the error occurred.

TIME
Indicates the time of the error.

PIDS/... RIDS/... AB/... PRCS/...
Use this symptom string to do a structured search of any IBM database.

PROGRAM ID
The program ID (PID) indicates the product and the component where the error occurred. For IBM products, see the tables in *z/OS MVS Diagnosis: Reference* that list the products and components. For non-IBM products, see the appropriate vendor-supplied documentation.

LOAD MODULE NAME
Indicates the load module in control at the time of the error.

CSECT NAME
Supplied by the recovery routine that obtained control for the error. See the PSW for more information.

SYSTEM ABEND CODE
Indicates what system or user completion code was issued by the system, application, or component. See *z/OS MVS System Codes* for information about system abend codes. See the appropriate product documentation for user abend codes.

ABEND REASON CODE
Indicates the reason code, when available, associated with a system or user abend code.

RECOVERY ROUTINE CSECT NAME
Indicates the recovery routine that was given control to handle the error condition.

PSW
Indicates the program status word (PSW) at the time of the error.

If the software record is an SVC 13, the address in the second half of the PSW indicates the address of the module that detected the error. You need to find the caller of that module. The caller's address will reside either in register 14, or, if register 14 points to module IEAVEEXP, use the STATUS section of the software record to determine the caller. In the STATUS section, the interrupt code will indicate the last SVC that was issued.

If the software record is a program interrupt, the address in the second half of the PSW usually points to the failing module.

FAILING INSTRUCTION TEXT
Contains 12 bytes of the instruction stream at the time of the error, including the actual instruction that caused the abend. Starting at the end of the sixth byte, subtract the instruction length to indicate the failing instruction. In the preceding example, the failing instruction is X'0A0D'.

THE ERROR OCCURRED WHILE . . .
Provides information about the system environment at the time of error, indicating what type of routine was in control, whether locks were held, and whether supervisor FRRs were set at the time of the error.

STATUS
The PSW and registers that follow come from the request block (RB) associated with the ESTAE recovery routine that obtained control for the error. Using the information indicated will enable you to determine the program that was running at the time of the error. This information included in the STATUS section does not appear when an FRR handles recovery.

RECOVERY ROUTINE ACTION
Describes the recovery action performed or requested to be performed by the recovery routine. In the preceding example, an SVC dump was not requested. There are times, however, when the recovery routine will request an SVC dump. If SVC DUMP SUCCESSFULLY STARTED appears in this section, the error identifier (ERROR ID) appears in the SVC dump and in message IEA911E as it appears in the logrec error record.

HEXADECIMAL DUMP
Provides an unformatted hexadecimal dump of the SDWA control block. Depending on an indicator in the SDWA, which is set by the recovery routine generating the record, the SDWA is displayed in hexadecimal; EBCDIC text; or key, length, and data format.

VARIABLE RECORDING AREA (SDWAVRA)
Provides component-specific information. Using the information in the PROGRAM ID field, determine the component. For IBM products, see *z/OS MVS Diagnosis: Reference* for diagnostic information related to system components.

The SDWAVRA can optionally be mapped in a key-length-data format. Recovery routines use the SDWAVRA to construct messages and provide data that often contains valuable debugging information. Some MVS recovery routines use the key-length-data format to provide standardized diagnostic information for software incidents. This formatted information allows you to screen duplicate errors.

Constants for the key field have been defined to describe data, such as: return and/or reason codes, parameter lists, registers, and control block information. For example, a key of X'10' indicates a recovery routine parameter area. The SDWAVRAM bit (in the fixed portion of the SDWA) indicates that the SDWAVRA has been mapped in the key-length-data format as described by the IHAVRA mapping macro.

For the format of the SDWA, including a description of the keys, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

SDWA RECORDABLE EXTENSIONS

In addition to the SDWA standard area and the SDWAVRA, the SDWA recordable extensions also contain valuable debugging information, as follows:

- *SDWARC1* (recording extension 1) contains additional component service data (such as the component ID, the component name, the address of the TCB representing the task that incurred the failure, the control registers, original completion code and reason code, linkage stack pointer, and translation exception access register number).
- *SDWARC2* (recording extension 2) contains additional I/O machine check data (such as the machine check interruption code).
- *SDWARC3* (recording extension 3) contains locking information (such as the locks to be freed, and the addresses of lockwords).

Note: The SDWA that is in the logrec buffer is a compressed SDWA in which the recordable extensions start directly after the used portion of the SDWAVRA. The SDWAURAL field contains the length of the SDWAVRA.

Detail edit report for a symptom record

The SYMREC macro updates a symptom record with system environment information and then logs the symptom record in the logrec data set or logrec log stream. The system or application, using the SYMREC macro, creates a symptom record. The ADSR mapping macro maps the symptom record, and the symptom record contains diagnostic information determined by the application.

As an application or a system component detects errors during processing, it stores diagnostic information into the symptom record and issues the SYMREC macro to log the record. The diagnostic information consists of a description of a programming failure and a description of the environment in which the failure occurred.

See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for information about the SYMREC macro. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for information about the ADSR data area.

Report output

The example in "Example: Detail edit report for a symptom record" on page 537 contains output from one record created by the system. Following the example is a list of the fields that are most important for diagnosis; only the highlighted fields are discussed.

Example: Detail edit report for a symptom record

```
TYPE:  SYMPTOM RECORD       REPORT:  SOFTWARE EDIT REPORT             DAY.YEAR
                                                         REPORT DATE: 176.92
SCP:   VS 2 REL 3                                         ERROR DATE: 175.92
                            MODEL:   9021                             HH:MM:SS.TH
                            SERIAL:  031347                     TIME: 10:41:14.37

SEARCH ARGUMENT ABSTRACT:
```

```
    PIDS/5752SC1CM AB/S080A PRCS/00000010 RIDS/IEAVTRSR
    RIDS/IGC0101C#L FLDS/SR#ORIGIN VALU/CIEAVTRSR PCSS/FAILING

SYSTEM ENVIRONMENT:

    CPU MODEL:  9021               DATE:  175   92
    CPU SERIAL: 031347             TIME:  10:41:14.37
    SYSTEM:     CPUR               BCP:   MVS

    RELEASE LEVEL OF SERVICE ROUTINE:      JBB4422
    SYSTEM DATA AT ARCHITECTURE LEVEL:     10
    COMPONENT DATA AT ARCHITECTURE LEVEL:  10

    SYSTEM DATA:  00000000 00000000    |........|

COMPONENT INFORMATION:

    COMPONENT ID:             5752SC1CM
    COMPONENT RELEASE LEVEL:  D10
    DESCRIPTION OF FUNCTION:  RTM2 RECURSION ERROR RECORDING

PRIMARY SYMPTOM STRING:

    PIDS/5752SC1CM AB/S080A PRCS/00000010 RIDS/IEAVTRSR
    RIDS/IGC0101C#L FLDS/SR#ORIGIN VALU/CIEAVTRSR PCSS/FAILING
    PCSS/CSECT PCSS/UNKNOWN FLDS/RTM2SCTC FLDS/FROM#PRWA
    VALU/H00040000

    SYMPTOM            SYMPTOM DATA     EXPLANATION
    ---------------    -------------    -----------
    PIDS/5752SC1CM     5752SC1CM        COMPONENT IDENTIFIER
    AB/S080A           080A             ABEND CODE - SYSTEM
    PRCS/00000010      00000010         RETURN CODE
    RIDS/IEAVTRSR      IEAVTRSR         ROUTINE IDENTIFIER
    RIDS/IGC0101C#L    IGC0101C#L       ROUTINE IDENTIFIER
    FLDS/SR#ORIGIN     SR#ORIGIN        DATA FIELD NAME
    VALU/CIEAVTRSR     IEAVTRSR         ERROR RELATED CHARACTER VALUE
    PCSS/FAILING       FAILING          SOFTWARE STATEMENT
    PCSS/CSECT         CSECT            SOFTWARE STATEMENT
    PCSS/UNKNOWN       UNKNOWN          SOFTWARE STATEMENT
    FLDS/RTM2SCTC      RTM2SCTC         DATA FIELD NAME
    FLDS/FROM#PRWA     FROM#PRWA        DATA FIELD NAME
    VALU/H00040000     00040000         ERROR RELATED HEXADECIMAL VALUESECONDARY SYMPTOM STRING:

    FLDS/RTM2SCTR VALU/H00040000 FLDS/RTM2SCTX VALU/H00040000

    SYMPTOM            SYMPTOM DATA     EXPLANATION
    ---------------    -------------    -----------
    FLDS/RTM2SCTR      RTM2SCTR         DATA FIELD NAME
    VALU/H00040000     00040000         ERROR RELATED HEXADECIMAL VALUE
    FLDS/RTM2SCTX      RTM2SCTX         DATA FIELD NAME
    VALU/H00040000     00040000         ERROR RELATED HEXADECIMAL VALUE

FREE FORMAT COMPONENT INFORMATION:

    KEY = FF00     LENGTH = 00048 (0030)

    +000    C5D9D9D6   D940C4C5   E3C5C3E3   C5C440C2   |ERROR DETECTED B|
    +010    E840D9E3   D4F240D9   C5C3E4D9   E2C9E5C5   |Y RTM2 RECURSIVE|
                          .
                          .
                          .
HEX DUMP OF RECORD:

  HEADER

    +000    4C831800   00000000   0092175F   10411437   |<C.......K. ....|
    +010    A6031347   90210000                         |W.......        |

  SYMPTOM RECORD

    +000    E2D9F9F0   F2F1F0F3   F1F3F4F7   00000000   |SR9021031347....|
    +010    A5E2A254   A5ED4104   40404040   40404040   |VSS.V...        |
    +020    4040C3D7   E4D94040   4040F5F7   F5F2D1C2   |  CPUR    5752JB|
                           .
                           .
                           .
```

TYPE: SYMPTOM RECORD
Indicates that the detail edit report is for a symptom record.

SEARCH ARGUMENT ABSTRACT
Provides information you can use to create a search argument. If enough information exists in this field, you can search the IBM service link problem reporting database to determine if there is a PTF to correct the error.

The information that follows the search argument abstract in a symptom record depends on the options specified on the SYMREC macro either by a user program or by a system component. In the report output listed above, the system recorded a recursive error. The information contained in a symptom record is variable. To obtain an interpretation, contact the IBM Support Center for the product or for the component that built the record.

Customizing symptom record location

You can control the location of logrec symptom records from non-authorized programs. Use the ASREXIT installation exit just before writing the logrec record to control:

- If a program can write symptom records
- The location of the symptom record: the logrec data set, job log, both, or neither

See *z/OS MVS Installation Exits* for information about ASREXIT.

Chapter 16. AMBLIST: Map load modules and program objects

AMBLIST provides the following problem data:

- Formatted listing of an object module
- Map of the control sections (CSECTs) in a load module or program object
- List of modifications to the code in a CSECT
- Map of all modules in the link pack areas (LPA)
- Map of the contents of the DAT-on nucleus. The map no longer represents the IPL version and message AMB129I will be issued.

These formatted listings can help you diagnose problems related to modules as they currently exist on your system. AMBLIST is a batch job that runs in problem state.

The following topics describe AMBLIST:

- "Obtaining AMBLIST output" on page 541
- "Reading AMBLIST output" on page 552

Long name support: AMBLIST will process external names (labels and references) up to 32767 bytes long. Names exceeding 16 bytes in length will be abbreviated in the formatted part of the listings and an abbreviation-to-long name equivalence table will be printed at the end of the listing. AMBLIST functions that provide long names support are: LISTLOAD, LISTIDR, and LISTOBJ (XSD and GOFF only).

Note:

1. Any load module to be formatted and printed by AMBLIST must have the same format as those created by the linkage editor or by the program management binder.
2. Any program object to be formatted and printed by AMBLIST must have the same format as those created by the program management binder.
3. A program object format 2 or greater having the non-editable attribute cannot be listed by AMBLIST.

See "LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)" on page 584 for a comparison of the formatted output of a load module and a program object.

AMBLIST supports all data sets allocated in the extended addressing space (EAS) of an extended address volume (EAV).

AMBLIST supports the following dynamic allocation (DYNALLOC or SVC 99) options for all data sets: S99TIOEX(XTIOT), S99ACUCB (NOCAPTURE), and S99DSABA (DSAB above the line).

Obtaining AMBLIST output

To obtain AMBLIST output, you must code JCL or use the UNIX System Services amblist command. The amblist command is described in the *z/OS UNIX System Services Command Reference*.

This section describes these topics:

- "Specifying the JCL statements" on page 542
- "Controlling AMBLIST processing" on page 542
- "Examples of running AMBLIST" on page 546
- "Examples for z/OS UNIX System Services file support" on page 551

Specifying the JCL statements

Generally, the minimum partition or region for running AMBLIST is 64 kilobytes for all functions except LISTLPA, which requires 100 kilobytes. However, for large load modules or program objects, IBM recommends you do not specify REGION, or specify REGION=16M or higher.

AMBLIST requires the following JCL statements:

JOB
Initiates the job.

EXEC PGM=AMBLIST
Calls for the processing of AMBLIST.

SYSPRINT DD
Defines the message data set.

Anyname DD
Defines an input dataset or a z/OS UNIX System Services pathname.

Use the PATH parameter to specify a z/OS UNIX System Services pathname, and also use the PATHOPTS parameter to be able to access the file for reading. If pathname is a directory name, the filename must be specified on the MEMBER parameter of the AMBLIST control statement. If the pathname ends with the filename, then omit the MEMBER parameter.

SYSIN DD
Defines the data set (in the input stream) that contains AMBLIST control statements.

Controlling AMBLIST processing

You control AMBLIST processing by supplying one or more control statements in the input stream. Code the control statement that applies to the data you want to obtain according to the following rules:

- Leave column 1 blank, unless you want to supply an optional symbolic name. A symbolic name is analogous to the label name in a program. The maximum length of a symbolic name is eight characters. A symbolic name must end with one or more blanks.
- If a complete control statement will not fit on a single line, end the first line with a comma or a non-blank character in column 72 and continue on the next line. Begin all continuation statements in columns 2 - 16. Do not split parameters between two lines. The only exceptions are the MEMBER parameters, which you can split at any internal comma.

LISTLOAD control statement

Use the LISTLOAD control statement to obtain a listing of load module or program objects. The listed data can help you verify why certain link-edit errors might have occurred.

```
LISTLOAD
 [OUTPUT={MODLIST|XREF|BOTH|MAP}]
 [,TITLE=('title',position)]
 [,DDN=ddname]
 [,MEMBER={member|(member1,membern...)}]
 [,RELOC=hhhhhhhh]
 [,ADATA={YES|NO}]
 [,IMPEXP={DUMP|SYMBOLS}]
 [,SECTION1={YES|NO}]
```

OUTPUT={MODLIST|XREF|<u>BOTH</u>|MAP}
OUTPUT=MODLIST requests a formatted listing of the text and control information of a load module or program object.

OUTPUT=XREF requests a module map and cross-reference listing for the load module or program object.

OUTPUT=BOTH requests both a formatted listing of the load module or program object and its map and cross-references.

OUTPUT=MAP requests a numerical map for the load module or program object.

If this parameter is omitted, OUTPUT=BOTH will be assumed.

TITLE=('title',position)
Specifies a title, from one to 40 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 is 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 to leave blank before the title. If you specify a position greater than 80, the indentation from the margin defaults to 1.

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

MEMBER={member|(member1,membern...)}
Identifies the input load module or program object by member name or alias name. To specify more than one load module or program object, enclose the list of names in parentheses and separate the names with commas.

Note:

1. If you specify MEMBER=IEANUCxx, where xx is the suffix of the member used during the current IPL, AMBLIST will list the DAT-ON nucleus. If you do not include the MEMBER parameter, AMBLIST will print all modules in the data set.
2. AMBLIST will accept member names up to 63 bytes in length. For aliases longer than 63 bytes, their primary member names must be entered instead.
3. If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, the pathname must end with a file, or there must also be a MEMBER parameter specifying the file or files in that directory.

RELOC=hhhhhhhh
Specifies a relocation or base address in hexadecimal of up to eight characters. When the relocation address is added to each relative map and cross-reference address, it gives the absolute central storage address for each item on the output listing. If you omit the RELOC parameter, no relocation is performed.

ADATA={YES|NO}
ADATA=YES on LISTLOAD OUTPUT=MODLIST or OUTPUT=BOTH requests a formatted listing of all ADATA classes, if they exist, in the program object to be displayed in the traditional dump format, 32 bytes per line, with hexadecimal representation on the left and EBCDIC on the right, in addition to the output listing with the specified output parameter.

OUTPUT=NO on LISTLOAD OUTPUT=MODLIST or OUTPUT=BOTH requests a normal formatted listings with the specified output parameter, and ADATA suppressed.

If this parameter is omitted, ADATA=NO will be assumed.

IMPEXP={DUMP|SYMBOLS}
IMPEXP=SYMBOLS indicates that section IEWBCIE text is displayed as a symbolically formatted IMPORT/EXPORT section of the output.

IMPEXP=DUMP indicates that section IEWBCIE text is displayed in the traditional dump format (as described under the ADATA parameter).

SECTION1={YES| NO}
SECTION1=YES requests that the module level section information be displayed.

SECTION1=NO requests that the module level section information not be displayed.

LISTOBJ control statement

Use the LISTOBJ control statement to obtain listings of selected object modules. LISTOBJ supports traditional object modules as well as object modules in XOBJ or GOFF format.

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

TITLE=('title',position)
Specifies a title, from one to 40 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 is 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 to leave blank before the title. If you specify a position greater than 80, the indentation from the margin defaults to 1.

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

MEMBER={member|(member1[,membern]...)}
Identifies the input object module by member name or alias name. To specify more than one object module, enclose the list of names in parentheses and separate the names with commas.

Note:

1. You must include the MEMBER parameter if the input object modules exist as members in a partitioned data set (PDS or PDSE). If you do not include the MEMBER parameter, AMBLIST will assume that the input data set is organized sequentially and that it contains a single, continuous object module.
2. AMBLIST will accept member names up to 63 bytes in length. For aliases longer than 63 bytes, their primary member names must be entered instead.
3. If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, the pathname must end with a file, or there must also be a MEMBER parameter specifying the file or files in that directory.

   Example: Processing a pathname: In this example, AMBLIST processes the pathname `/path/to/dir/longmembername`.

   ```
   //SYSLIB DD PATH='/path/to/dir'
   //SYSIN DD *
    LISTOBJ MEMBER=longmembername
   /*
   ```

LISTIDR control statement

Use the LISTIDR control statement to obtain listings of selected CSECT identification records (IDR).

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

OUTPUT={IDENT|ALL}
Specifies whether AMBLIST must print all CSECT identification records or only those containing SPZAP data and user data. If you specify OUTPUT=ALL, all IDRs associated with the module will be printed. If you specify OUTPUT=IDENT, AMBLIST will print only those IDRs that contain SPZAP data or user-supplied data. If you omit this parameter, AMBLIST will assume a default of OUTPUT=ALL. Do not specify OUTPUT if you specify the MODLIB parameter.

TITLE=('title',position)
Specifies a title, from one to 40 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 is indented; if TITLE=('title',1) is specified, or if the position parameter is omitted, the title is 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 are left blank before the title. If a position greater than 80 is specified, the indentation from the margin defaults to 1.

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

Note: If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, there must also be a MEMBER parameter specifying the file or files in that directory.

MEMBER={member|(member1,membern...)}
Identifies the input load module or program object by member name or alias name. To specify more than one load module or program object, enclose the list of names in parentheses and separate the names with commas.

Note:

1. Do not specify MEMBER if you specify the MODLIB parameter. If you do not include the MEMBER parameter, AMBLIST will print all modules in the data set.
2. AMBLIST will accept member names up to 63 bytes in length. For aliases longer than 63 bytes, their primary member names must be entered instead.
3. If the DD name associated with this operation is allocated to a z/OS UNIX System Services directory, the pathname must end with a file, or there must also be a MEMBER parameter specifying the file or files in that directory.

MODLIB
Prevents AMBLIST from printing the module summary. AMBLIST prints the IDRs that contain SPZAP data or user-supplied data. No page ejects occur between modules. When you specify MODLIB, the OUTPUT or MEMBER parameters are not valid parameters.

LISTLPA control statement

Use the LISTLPA control statement to obtain listings of selected modules in the fixed link pack area (LPA).

```
LISTLPA [FLPA][,MLPA][,PLPA]
```

LISTLPA
Lists the modules in the fixed link pack area, the modified link pack area, and the pageable link pack area (PLPA). This listing is a map that includes modules residing in the extended sections of each link pack area (LPA). If you do not specify any parameters on the LISTLPA control statement, then AMBLIST maps modules from all three LPAs.

The LISTLPA control statement does not support dynamic LPA. If the dynamic LPA support is used to update LPA, those changes will not be reflected in the AMBLIST LISTLPA output. The LISTLPA control statement will not be enhanced to support new operating system functions. The recommended replacement is the LPAMAP subcommand of IPCS. See *z/OS MVS IPCS Commands* for details about this command.

FLPA
Requests mapping of the modules in the fixed link pack area.

MLPA
Requests mapping of the modules in the modified link pack area.

PLPA
Requests mapping of the modules in the pageable link pack area.

Examples of running AMBLIST

Using the control statements as input into the JCL for the job, you can invoke AMBLIST to provide output. The following examples of AMBLIST include the control statement needed to produce the output and sample JCL for each function.

List the contents of an object module

You can use AMBLIST to format three types of object module:

1. OBJ (traditional object module)
2. XOBJ (extended object module, based on OBJ)
3. GOFF (Generalized Object File Format).

You can list the following information from an object module:

- the head record (HDR) - which may contain information about the character set and expected operating environment (GOFF only)
- external symbol dictionary entries (ESD or XSD)
- relocation dictionary entries (RLD)
- the text of the program - the instructions and data, as output by the language translator (TXT)
- translator identification record (IDRL) - which contains the compiler ID and compile date
- ADATA records (GOFF only)
- LEN records (GOFF only)
- and the END record.

To list object module contents, invoke AMBLIST with the LISTOBJ control statement. For sample outputs, see "LISTOBJ outputs" on page 556.

In Figure 181 on page 547, AMBLIST is used to format and list an object module included in the input stream.

```
//LSTOBJDK     JOB      MSGLEVEL=(1,1)
//             EXEC     PGM=AMBLIST
//SYSPRINT     DD       SYSOUT=A
//OBJMOD       DD       *
    object module
/*
//SYSIN        DD       *
    LISTOBJ         DDN=OBJMOD,
        TITLE=('OBJECT MODULE LISTING FOR MYJOB',25)
/*
```

Figure 181. Example: Listing an object module

OBJMOD DD Statement
Defines the input data set, which follows immediately. In this case, the input data set is an object module.

SYSIN DD Statement
Defines the data set containing AMBLIST control statements, which follows immediately.

LISTOBJ Control Statement
Instructs AMBLIST to format the data set defined by the OBJMOD DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

In Figure 182 on page 547, AMBLIST is used to list all object modules contained in the data set named OBJMOD, and three specific object modules from another data set called OBJMODS.

Note: If you are using AMBLIST to list multiple object modules, IBM recommends that you do not specify REGION, or specify REGION=16M or higher.

```
//OBJLIST       JOB    MSGLEVEL=(1,1)
//LISTSTEP      EXEC   PGM=AMBLIST
//SYSPRINT      DD     SYSOUT=A
//OBJLIB        DD     DSN=OBJMODS,DISP=SHR
//OBJSDS        DD     DSN=OBJMOD,DISP=SHR
//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)
/*
```

Figure 182. Example: Listing several object modules

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 AMBLIST control statements.

LISTOBJ Control Statement #1
Instructs AMBLIST to format the data set defined by the OBJSDS DD statement, treating it as a single member. It also specifies a title for each page of output, to be indented 20 characters from the left margin.

LISTOBJ Control Statement #2
Instructs AMBLIST to format three members of the partitioned data set (PDS or PDSE) 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.

Map the CSECTs in a load module or program object

You can list the organization of CSECTs within the load module or program object, the overlay structure (if any), and the cross-references for each CSECT. To map CSECTs, invoke AMBLIST with the LISTLOAD control statement.

For sample output, see "LISTLOAD OUTPUT=MODLIST output" on page 564, "Alphabetical cross-reference" on page 583, and "LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)" on page 584.

In Figure 183 on page 548, AMBLIST is used to produce formatted listings of several load modules or program objects.

Note: If you are using AMBLIST to format program objects, IBM recommends that you do not specify REGION, or specify REGION=16M or higher.

```
//LOADLIST   JOB         MSGLEVEL=(1,1)
//LISTSTEP   EXEC        PGM=AMBLIST
//SYSPRINT   DD          SYSOUT=A
//SYSLIB     DD          DSNAME=SYS1.LINKLIB,DISP=SHR
//LOADLIB    DD          DSNAME=LOADMOD,DISP=SHR
//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)
    LISTLOAD TITLE=('XREF&LD MOD LSTNG-ALL MOD IN LINKLIB',20)
/*
```

Figure 183. Example: Listing several load modules or program objects

SYSLIB DD Statement
Defines an input data set, SYS1.LINKLIB, that contains load modules or program objects 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 AMBLIST control statements.

LISTLOAD Control Statement #1
Instructs AMBLIST to format the control and text records including the external symbol dictionary and relocation dictionary records of the load module or program object 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 AMBLIST to produce a module map and cross-reference listing of the load modules or program objects 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 AMBLIST to produce a formatted listing of the load module or program object 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.

Figure 184 on page 549 shows how to use AMBLIST to verify three modules. Assume that an unsuccessful attempt has been made to link-edit an object module with two load modules or program objects to produce one large load module or program object.

```
>
   //LSTLDOBJ      JOB     MSGLEVEL=(1,1)
   //              EXEC    PGM=AMBLIST
   //SYSPRINT      DD      SYSOUT=A
   //OBJMOD        DD      DSN=MYMOD,DISP=SHR
   //LOADMOD1      DD      DSN=YOURMOD,DISP=SHR
   //LOADMOD2      DD      DSN=HISMOD,DISP=SHR
   //SYSIN         DD      *
       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 HSMOD',25)
       LISTIDR         DDN=LOADMOD2,OUTPUT=ALL,
           TITLE=('IDRS FOR HISMOD',25)
```

Figure 184. Example: Listing several load modules or program objects

OBJMOD DD Statement
Defines an input load module or program object data set.

LOADMOD1 and LOADMOD2 DD Statements
Define input load module or program object data sets.

SYSIN DD Statement
Defines the data set containing AMBLIST control statements.

LISTOBJ Control Statement
Instructs AMBLIST 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 AMBLIST 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 AMBLIST 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 AMBLIST 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.

LISTIDR Control Statement #2
Instructs AMBLIST to list all CSECT identification 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.

Trace modifications to the executable code in a CSECT

You can list the information in a load module or program object's CSECT identification records (IDRs). An IDR provides the following information:

- The version and modification level of the language translator and the date that each CSECT was translated. (Translation data is available only for CSECTs that were produced by a translator that supports IDR generation.)
- The version and modification level of the linkage editor or binder that built the load module or program object and gives the date the load module or program object was created.
- Modifications to the load module or program object, by date, that might have been done using SPZAP.

An IDR might also contain optional user-supplied data.

To trace modifications, invoke AMBLIST with the LISTIDR control statement. For sample output, see “LISTIDR output” on page 591.

In Figure 185 on page 550, AMBLIST is used to list the CSECT identification records in several load modules or program objects.

```
//IDRLIST     JOB     MSGLEVEL=(1,1)
//LISTSTEP    EXEC    PGM=AMBLIST
//SYSPRINT    DD      SYSOUT=A
//SYSLIB      DD      DSN=SYS1.LINKLIB,DISP=SHR
//LOADLIB     DD      DSN=LOADMODS,DISP=SHR
//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)
    LISTIDR   DDN=LOADLIB,MODLIB
/*
```

Figure 185. Example: Listing IDR information for several load modules

SYSLIB DD Statement
Defines an input data set, SYS1.LINKLIB, that contains load modules or program objects 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 AMBLIST control statements.

LISTIDR Control Statement #1
Instructs AMBLIST 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 AMBLIST to list CSECT identification records that contain SPZAP or user-supplied data for the load module or program object named 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 AMBLIST to list all CSECT identification records for of the load modules or program objects 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.

LISTIDR control statement #4
Instructs AMBLIST to list CSECT identification records that contain SPZAP or user-supplied data for the LOADLIB data set. The module summary print out is suppressed.

List the modules in the link pack area and the contents of the DAT-on nucleus

You can list all modules in the fixed link pack area, the modified link pack area, and the pageable link pack area.

To map link pack area modules, invoke AMBLIST with the LISTLPA control statement. For sample output, see “LISTLPA output” on page 593.

You can also produce a map and cross-reference listing of a nucleus.

To map the contents of the DAT-on nucleus, invoke AMBLIST with the LISTLOAD MEMBER=IEANUCxx control statement.

Figure 186 on page 551 shows how to use the LISTLOAD and LISTPA control statements to list a system nucleus and map the fixed link pack area, the modified link pack area, and the pageable link pack area. Note that in this example the data set containing the nucleus is named SYS1.NUCLEUS, and the nucleus occupies the member named IEANUC01. The map no longer represents the IPL version of the nucleus and message AMB129I will be issued. Use IPCS to format the NUCMAP. For information on using IPCS, see *z/OS MVS IPCS User's Guide* and *z/OS MVS IPCS Commands*.

```
//LISTNUC      JOB     MSGLEVEL=(1,1)
//STEP         EXEC    PGM=AMBLIST
//SYSPRINT     DD      SYSOUT=A
//SYSLIB       DD      DSN=SYS1.NUCLEUS,DISP=SHR,UNIT=3330,
//    VOL=SER=nnnnn
//SYSIN        DD      *
    LISTLOAD           DDN=SYSLIB,MEMBER=IEANUC01,
       TITLE=('LISTING FOR NUCLEUS IEANUC01',25)
    LISTLPA
/*
```

Figure 186. Example: Listing a system nucleus and the link pack area

SYSLIB DD Statement
Defines the input data set, which in this case contains the nucleus.

SYSIN DD Statement
Defines the data set containing AMBLIST control statements, which follows immediately.

LISTLOAD control statement
Instructs AMBLIST to format the control and text records including the external symbol dictionary and relocation dictionary records of the load module IEANUC01 in the data set defined by the SYSLIB DD statement. It also specifies a title for each page of output, to be indented 25 characters from the left margin.

LISTLPA control statement
Instructs AMBLIST to map the fixed link pack area (FLPA), the modified link pack area (MLPA), and the pageable link pack area (PLPA).

Examples for z/OS UNIX System Services file support

AMBLIST will support formatted listings of program objects and object modules in z/OS UNIX System Services files.

To obtain a formatted listing of a program object in a z/OS UNIX System Services file, specify the complete pathname in a DD statement and code the control statement. Use the JCL example shown in Figure 187 on page 551 as a guide.

```
//LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/outmod'
//            PATHDISP=(KEEP,KEEP)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 LISTLOAD DDN=HFS1,OUTPUT=MODLIST
```

Figure 187. Example: z/OS UNIX System Services program object

To obtain a formatted listing of an object module in a z/OS UNIX System Services file, specify the complete pathname in a DD statement and code the control statement. Use the JCL example shown in Figure 188 on page 552 as a guide.

```
/LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/myobject.o',PATHDISP=(KEEP,KEEP),
//            PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  LISTOBJ DDN=HFS1
```

Figure 188. Example: z/OS UNIX System Services object module

AMBLIST will support the LISTIDR control statement for program objects in z/OS UNIX System Services files. Use the JCL example shown in Figure 189 on page 552 as a guide.

```
//LIST EXEC PGM=AMBLIST
//HFS1 DD PATH='/u/USER1/outmod'
//            PATHDISP=(KEEP,KEEP)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 LISTIDR DDN=HFS1
```

Figure 189. Example: z/OS UNIX System Services control statement

For z/OS UNIX System Services files, MEMBER NAME is the file name and LIBRARY is the directory name. If the pathname is too long to fit in the space reserved on the line, it will be truncated on the left and preceded by two periods and a space (".. "). In Figure 190 on page 552, the library name is truncated while the member name is not.

```
MEMBER NAME:  0123456789012345678901234567890123456789012345678901234567890123456789abc   MAIN ENTRY POINT:  00000000
LIBRARY:      .. 3456789012345678901234567890123456789012345678901234567890123456789abc/  AMODE OF MAIN ENTRY POINT: 31
```

Figure 190. Example: Differences in output

Reading AMBLIST output

AMBLIST produces a separate listing for each control statement that you specify.

- The first page of each listing always shows the control statement as it was entered.
- The second page of the listing is a module summary, unless you requested LISTOBJ, LISTLPA, or MODLIB with LISTIDR; 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 and its addressing mode, alias entry points and their addressing modes, the attributes assigned to the module by the linkage editor or program management binder, the system status index information (SSI), the APF code, an residence mode for the module being formatted. For program objects, the PMAR and PMARL are displayed in hexadecimal for diagnostic information. Figure 191 on page 553 and Figure 192 on page 553 show samples of module summary processed by the linkage editor and the binder.
- The third page of the listing (or, for LISTOBJ, LISTLPA, or MODLIB with LISTIDR the second page) is the beginning of the formatted output itself.

Module summary

Figure 191 on page 553 is sample of a module summary for a load module that was processed by the linkage editor.

```
 LISTLOAD DDN=DD1,MEMBER=TESTPR

A                       *****  M O D U L E   S U M M A R Y  *****
        MEMBER NAME:  TESTPR                                                         MAIN ENTRY POINT:    00000000
        LIBRARY:      DD1                                                           AMODE OF MAIN ENTRY POINT: ANY
        NO ALIASES **
-------------------------------------------------------------------------------------------------------------------------

B       ****                ATTRIBUTES OF MODULE   *
                **   BIT  STATUS         BIT  STATUS         BIT  STATUS         BIT  STATUS   **
                      0  NOT-RENT         1  NOT-REUS         2  NOT-OVLY         3  NOT-TEST
                      4  NOT-OL           5  BLOCK            6  EXEC             7  MULTI-RCD
                      8  NOT-DC           9  ZERO-ORG         10 EP-ZERO          11 RLD
                      12 EDIT             13 NO-SYMS          14 F-LEVEL          15 NOT-REFR
-------------------------------------------------------------------------------------------------------------------------

C                         MODULE SSI:  NONE
                                    APFCODE:     00000000
                                    RMODE:       24
                                    LONGPARM:    NO

D      *****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
```

Figure 191. Example: Module summary for a load module processed by the linkage editor

Figure 192 on page 553 is sample of a module summary for a program object processed by the binder.

```
 LISTLOAD DDN=DD1,MEMBER=#THIS#ISA#LONG#NAME#BPBF6190

A        *****  M O D U L E   S U M M A R Y  *****
    MEMBER NAME:  B#Z49$EA                                                            MAIN ENTRY POINT:    00000000
    LIBRARY:      MYLIB                                                               AMODE OF MAIN ENTRY POINT: 31
        ** ALIASES **      ENTRY POINT    AMODE
          A1                00000000       31
          A2                00000000       31
          A3                00000000       31
          A4                00000000       31
          THIS#ISA-BF6190   00000000       31   ALT PRIMARY
-------------------------------------------------------------------------------------------------------------------------
B         **** ATTRIBUTES OF MODULE ***
                **   BIT  STATUS         BIT  STATUS         BIT  STATUS         BIT  STATUS   **
                      0  NOT-RENT         1  NOT-REUS         2  NOT-OVLY         3  NOT-TEST
                      4  NOT-OL           5  BLOCK            6  EXEC             7  MULTI-RCD
                      8  NOT-DC           9  ZERO-ORG         10 RESERVED         11 RLD
                      12 EDIT             13 NO-SYMS          14 RESERVED         15 NOT-REFR
                      16 RESERVED         17 <16M             18 NOT-PL           19 NO-SSI
                      20 APF              21 PGM OBJ          22 NOT-SIGN         23 RESERVED
                      24 ALTP             25 RESERVED         26 RESERVED         27 RMODE24
                      28 RESERVED         29 RESERVED         30 RESERVED         31 RESERVED
                      32 NON-MIGR          33 NO-PRIME         34 NO-PACK          35 RESERVED
                      36 RESERVED         37 RESERVED         38 RESERVED         39 RESERVED
-------------------------------------------------------------------------------------------------------------------------

C
                                    MODULE SSI:      NONE
                                    APFCODE:         00000000
                                    RMODE:           ANY
                                    PO FORMAT:       3
                                    OS COMPAT LEVEL: z/OSV1R3
                                    XPLINK:          YES

D           *****PROGRAM OBJECT PROCESSED BY BINDER


E
***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
PMAR  001E0308 02C00412 00000000 02900000   00B80000 00B80000 00000000 0000
PMARL 00629040 00000000 00050000 02780000   10000000 0B540000 40F40000 00740000
      01400000 00240000 011C0000 00050000   01B40001 00000000 10000000 00000000
      00002001 072F0144 340FD7D4 F6E3C5E2   E3403000 00010000 00180000 20000000
      0178
```

Figure 192. Example: Module summary for a program object processed by the binder

The following describes Figure 191 on page 553 and Figure 192 on page 553.

A
: Entry Names. For the member, the library (ddname) and member name are displayed, along with the primary entry point offset and AMODE. The MEMBER NAME field contains the primary name.

For each alias or alternate entry point, AMBLIST shows the alias name, entry point offset and AMODE. If no aliases are present, AMBLIST prints NO ALIASES. If the input name is an alias, then its name is printed in the alias section preceded by two asterisks.

The constants ALT PRIMARY will be added to the right of the amode of the alias name which was a long primary name in which the binder had converted to an alias.

B

Attributes of the Module. The attributes of the module are represented by bits. Each bit is set either ON or OFF. In the listing, AMBLIST interprets the bit settings and shows a descriptive value in the STATUS column. For example, in Figure 191 on page 553 and Figure 192 on page 553, bit 0 is interpreted as NOT-RENT. This means the module is not reentrant. For a description of all the STATUS values, see Table 76 on page 554.

C

Other Attributes. The remaining module attributes are displayed following the table. This includes the system status index (SSI) field, the APF (authorized program facility) code, the RMODE (residence mode) for the entire module, the PO format (loader data level), the OS compat level (binder data format), and XPLINK. If attribute bit 19 is the OFF state, NONE will be displayed in place of SSI. SSI is usually set through the SETSSI control statement in the binder or SPZAP programs.

Note: For an OS Compat Level less than z/OS V1R3, no Compat level will be printed.

PO format and XPLINK are applicable only for program objects. PO format is the version of the program object. XPLINK indicates whether any routines use XPLINK linkage conventions. Compat level designates the lowest OS release at which the release's binder could rebind this object. Note that the level at which the module can be executed is determined by the PO format.

D

Linking Program. The last line in the module summary identifies the linking program (VS linkage editor or binder) that created the module. For example:

```
*****PROGRAM OBJECT PROCESSED BY BINDER
```

This is applicable to a program object.

```
****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
```

The load module is either created by the linkage editor and processed by the binder, or created by the binder and processed by the linkage editor.

E

PMAR and PMARL. For program objects, the PMAR and PMARL are displayed in hexadecimal for diagnostic purposes, preceded by:

```
*** THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS
(PMAR AND PMARL)
```

Table 76 on page 554 summarizes the attributes of program objects and load modules. The first column shows the bit position. Columns 2 and 3 show the displayed constant and its meaning for the OFF condition. Columns 4 and 5 show the displayed constant and meaning for the ON condition.

Table 76. Program object and load module attributes

Bit Position	OFF Value	Meaning	ON Value	Meaning
00	NOT-RENT	Module is not reentrant.	RENT	Module is reentrant.
01	NOT-REUS	Module is not reusable.	REUS	Module is reusable.
02	NOT-OVLY	Module is not in overlay format.	OVLY	Module is in overlay format.
03	NOT-TEST	Test option was not specified during binding.	TEST	Test option was specified during binding.

Table 76. Program object and load module attributes (continued)

Bit Position	OFF Value	Meaning	ON Value	Meaning
04	NOT-OL	Program can be invoked through all CSV macros.	ONLY-LOAD	Program can be loaded only through LOAD macro.
05	BLOCK	Module consists of a single, contiguous block of text. This bit is always off for program objects.	SCTR	Module can be scatter loaded (MVS nucleus only).
06	NON-EXEC	Module is marked not executable.	EXEC	Module is marked executable.
07	MULTI-RCD	Module contains multiple text records. This bit is always off for program objects.	1-TXT	Module contains no RLD items and only one block of text.
08	DC	Module is processable by all levels of linkage editor.	NOT-DC	Module is processable only by F-level linkage editor and above. This bit is always on for program objects.
09	NOT-ZERO	Origin of first text block greater than zero.	ZERO-ORG	Origin of first text block is zero. This bit is always on for program objects.
10	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
11	RLD	Module contains RLD items.	NO-RLD	Module contains no RLD items.
12	EDIT	Module can be reprocessed by binder.	NOT-EDIT	Module cannot be reprocessed by binder.
13	NO-SYMS	Module contains no SYM records.	SYMS	Module contains SYM records.
14	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
15	NOT-REFR	Module is not refreshable.	REFR	Module is refreshable.
16	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
Note: The following bits are shown only for program objects.				
17	<16M	Module text size is less than 16 megabytes.	>16M	Module text size is greater than or equal to 16 megabytes.
18	NOT-PL	Page alignment is not required for loaded text.	P-ALIGN	Page alignment is required for loaded text.
19	NO-SSI	System status index is not present.	SSI	System status index is present.
20	NOT-APF	There is not an APF section in the directory. (APFCODE is not present.)	APF	There is an APF section in the directory. (APFCODE is present.)
21	NOT-PO	This is a load module.	PGM OBJ	This is a program object. Always on for program object.
22	NOT-SIGN	Module is not digitally signed.	SIGN	Module is digitally signed.
23	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
24	ALTP	Alternate primary name.	NOT-ALTP	Not an alternate primary name.
25	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
26	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
27	RMODE24	Module must be loaded below 16 megabytes.	RMODEANY	Module can be loaded anywhere below 2 gigabytes.
28	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
29	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.

Table 76. Program object and load module attributes (continued)

Bit Position	OFF Value	Meaning	ON Value	Meaning
30	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
31	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
32	NON-MIGR	Program object cannot be converted directly to PDS load module format.	MIGRATE	Program object can be converted to PDS load module format.
33	PRIME	FETCHOPT PRIME option.	NO-PRIME	FETCHOPT NOPRIME option.
34	PACK	FETCHOPT PACK option.	NO-PACK	FETCHOPT NOPACK option.
35	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
36	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
37	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
38	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.
39	RESERVED	Reserved for IBM use.	RESERVED	Reserved for IBM use.

LISTOBJ outputs

Figure 193 on page 556 shows sample output for LISTOBJ with an object module.

```
OBJECT MODULE LISTING                                                                                          PAGE 0001

ESD RECORD:                                                                                                     00000001
 ESDID    TYPE      NAME     ADDR    R/R/A   ID/LTH
 00001   SD(00)   MODULE00  000000    02     0026E0
 00002   ER(02)   MODULE01  000000    40     404040
 00003   ER(02)   MODULE02  000000    40     404040

ESD RECORD:                                                                                                     00000002
 ESDID    TYPE      NAME     ADDR    R/R/A   ID/LTH
 00004   ER(02)   MODULE03  000000    40     404040
 00005   SD(00)   MODULE04  0026E0    06     0004B4
 00006   ER(02)   MODULE05  000000    40     404040

ESD RECORD:                                                                                                     00000003
 ESDID    TYPE      NAME     ADDR    R/R/A   ID/LTH
 00007   ER(02)   MODULE06  000000    40     404040
 00008   ER(02)   MODULE07  000000    40     404040
 00009   ER(02)   MODULE08  000000    40     404040

ESD RECORD:                                                                                                     00000004
 ESDID    TYPE      NAME     ADDR    R/R/A   ID/LTH
 00010   ER(02)   MODULE09  000000    40     404040
 00011   ER(02)   MODULE10  000000    40     404040

TXT:                                                                                                            00000005
 ADDR=000000 ESDID=00001 TEXT: A7F4000D 15D4D6C4 E4D3C5F0 F040F1F1 F2F1F3C8 C2C2F7F7 F9F090EC D00CC0C0 000010A5 0D8041F0
                               00005800 C00C1891 58F093E0 0DEF18FD

RLD RECORD:    R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR       00000000
               00001   00001    0D    002168  00001   00001    0D    0021D4  00001   00001    0D    0021D8
               00001   00001    0D    00228C  00001   00001    0D    002290  00001   00001    0D    0022A0
               00001   00001    0D    0022B0  00001   00001    0C    00268C  00005   00001    0C    0022AC
               00006   00001    0C    00229C
RLD RECORD:    R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR       00000000
               00007   00001    0C    002294  00008   00001    0C    0022A4  00009   00001    0C    0022A8
               00010   00001    0C    002298  00011   00001    0C    002288  00002   00005    0D    002A97
               00002   00005    0D    002B0A  00002   00005    0C    002B46

RLD RECORD:    R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR       00000000
               00003   00005    0C    002B69  00004   00005    0C    002B8F
END RECORD:                                                     2569623400 010611213
```

Figure 193. Example: Output for LISTOBJ with an object module

The record formats for OBJ and XOBJ records are identical except that XOBJ modules contain XSD records rather than ESD records. Except for XSD records, AMBLIST formats the records in the object module one record at a time. XSD records support names up to 32767 characters. These names may be continued onto multiple records, but such a continued record will appear as a single XSD record in the AMBLIST output. If the name is longer than 16 characters, a 16–character abbreviated name is printed with the XSD record. An abbreviation table which maps abbreviated names to be true names is printed at the end of the listing.

See the description of ESD data items in *z/OS MVS Program Management: Advanced Facilities* for a description of the format of OBJ and XOBJ record formats.

Figure 194 on page 557 shows sample output for LISTOBJ with an XSD record.

```
OBJECT MODULE LISTING                             MEMBER= CALLEDA                                           PAGE 0001
             LIST OF CALLEDA
XSD RECORD:                                                                                                 00000001
 ESDID    TYPE      NAME             ADDR    R/R/A   ID/LTH
  0001   SD(00)   CAN_BE_ABBRV_16B  000000    06     0000BC
TXT:                                                                                                        00000002
 ADDR=000000 ESDID= 0001 TEXT: 90ECD00C 0DC050D0 C07241E0 C06E50E0 D00818DE 1B115010 C0660700 4510C048 80000070 00000003
                               02250000 C3C1D3D3 C5C4C140 C1C2D6E4
TXT:                                                                                                        00000004
 ADDR=000038 ESDID= 0001 TEXT: E340E3D6 40D9C5E3 E4D9D540 E3D640C3 C1D3D3C5 D9
TXT:                                                                                                        00000005
 ADDR=00004E ESDID= 0001 TEXT: 0A234100 00014110 C0660A01 1BFF58D0 D00458E0 D00C980C D0140B0E 00000000 0000E7E7
RLD RECORD:    R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR   R PTR   P PTR   FLAGS   ADDR       00000006
               0001    0001     0D    000020  0001    0001     0C    000024
END RECORD:                                                         1566896201 020191248                    00000007
```

Figure 194. Example: LISTOBJ output with XSD Record

Figure 195 on page 557 shows sample output for LISTOBJ with a GOFF records.

```
                       ***** G E N E R A L I Z E D  O B J E C T  F I L E  F O R M A T *****             PAGE       1

OBJECT MODULE LISTING

RECORD TYPE: HDR      SEQUENCE:        1
           --- CHARACTER SET --    LANGUAGE          HDR       MODULE
           ID         NAME          PRODUCT          VERSION    PROPERTIES

>         00000                                      00000

RECORD TYPE: ESD      SEQUENCE:        2
        ESD   OWNER/    ITEM      ITEM    NAME   ---------------- ATTRIBUTES -----------------
 ESDID  TYPE  PARENT   OFFSET    LENGTH   SP/S    BA  AMD  RMD  REUS  AL  TXT  ORD  STR  BINDER            SIGNATURE

>000001  SD     N/A      N/A       N/A    N/A    N/A N/A  N/A   N/S  N/A N/A  N/A  N/A  N/A                 N/A
              NAME(CSECT)

>000002  ED   000001        0        1C  01-N/A C   N/A  N/S   N/A  03  B-U  N/A  N/A  L,A,C              N/A
              NAME(B_TEXT)

>000004  ED   000001        0         0  01-N/A C   N/A  N/S   N/A  00  F-U  N/A  N/A  C                   N/A
              NAME(B_IDRL)

>000003  LD   000002        0     N/A    01-N/S N/A ANY  N/A   N/A  N/A N-U  N/A   S   N/A              00000000
              NAME(CSECT)

RECORD TYPE: TEXT     SEQUENCE:        6
 -- RESIDENT --      TRUE      TEXT    ENCODED
 ESDID   OFFSET    LENGTH   ENCODING  LENGTH    ------------------------------ T E X T  ------------------------------

>000002 00000000  00000000    0000   0000000C   58C07010 58C07014 41C07018

>000002 00000010  00000000    0000   0000000C   00000001 00000004 0000001F

RECORD TYPE: RLD      SEQUENCE:        8
 R-PTR  P-PTR  OFFSET  TYPE LEN ATTRIB   R-PTR  P-PTR  OFFSET  TYPE LEN ATTRIB   R-PTR  P-PTR  OFFSET  TYPE LEN ATTRIB
>000002 000002 000010   00+ 004         000002 000002 000014   00+ 004          000002 000002 000018   00+ 004

RECORD TYPE: IDRL     SEQUENCE:        9
 ESDID            |---- IDR  DATA ----|    |---- IDR  DATA ----|    |---- IDR  DATA ----|    |---- IDR  DATA ----|

>000004           |569623400.010295104|

RECORD TYPE: END      SEQUENCE:       10
 RECORD       --ENTRY POINT--
 COUNT        ESDID    OFFSET

>000000        N/S       N/S
```

Figure 195. Example: LISTOBJ output with GOFF Records

Description of LISTOBJ output for GOFF

The GOFF object listing is similar in function and content to the LISTOBJ format for traditional object modules. The output is formatted one logical record at a time. A logical record represents the concatenation of the first physical record (which contains the record type) and all continuation records. If a name in a record is longer than 16 characters, a 16-character abbreviated name is printed. The true name can be found from the abbreviated name to long name table, which is printed at the end of the listing. The start of a logical record is highlighted by a dingbat (">") in the first position.

A record group consists of one or more records of the same type and is preceded by a two- or three-line record header. The first line of each record header consists of the record type and the sequence number of the first record in the group. Following a page break, the record group header will be repeated, even though the record type may not have changed.

Although the GOFF format currently defines only six record types, the TXT record type is subdivided into three different text types:

- TEXT, containing the instructions and data of the program
- IDRL, containing IDR information from the compiler or assembler
- ADATA, containing additional data associated with the object module

Altogether there are eight different display formats.

Report Description: The keyed sections of this description correspond to the equivalent keys highlighting the page header and the eight record formats in "Example: LISTOBJ format for GOFF" on page 558. Note that some of the flags and lengths in the GOFF format are of a structural nature and do not represent the data content of the module. To save space, those elements have been omitted from the listing. For the same reason, unsupported data elements are not shown. A list of omitted elements is provided for each record type and the reason for omission is coded in parens following the field name. Code values are S (structural or syntactic data) and U (unsupported element). PTV for all record types is not formatted.

Example: LISTOBJ format for GOFF

```
1  LISTOBJ   MEMBER=HELLOW,TITLE=('MY PROGRAM IN GOFF FORMAT')
1                       ***** G E N E R A L I Z E D  O B J E C T  F I L E  F O R M A T
*****              PAGE       1
 MY PROGRAM IN GOFF FORMAT
 OBJECT MODULE LISTING                             MEMBER= HELLOW

0RECORD TYPE: HDR      SEQUENCE:        1
0           --- CHARACTER SET --    LANGUAGE          HDR       MODULE
            ID         NAME          PRODUCT          VERSION    PROPERTIES

0>        00000                                        00000

0RECORD TYPE: ESD      SEQUENCE:        2
         ESD   OWNER/    ITEM      ITEM    NAME   --------------------------- ATTRIBUTES
---------------------------
  ESDID  TYPE  PARENT   OFFSET   LEN/ADA   SP/S   BA  AMD  RMD  REUS  AL  TXT  ORD  STR  BINDER_FLAGS
LNK SIGNATURE

0>000001  SD   000000     N/A       N/A    N/A    N/A N/S  N/A  RENT  N/A N/A  N/A  N/A  N/A
N/A   N/A
               NAME( )

 >000002  ED   000001         0        28  01-N/A C   ANY  31    N/A  03  B-D  N/A  N/A  L,A
N/A   N/A
               NAME(C_EXTNATTR)

 >000003  ED   000001         0        B8  01-N/A C   ANY  31    N/A  03  B-I  N/A  N/A  L,R,A
N/A   N/A
               NAME(C_CODE)

 >000004  LD   000003         0    000006  01-L   N/A ANY  N/A   N/A  N/A N-I  N/A   S   N
X   00000000
               NAME( )

 >000005  ED   000001         0         0  03-N/A M   ANY  31    N/A  03  B-D  N/A  N/A  L,A,D
N/A   N/A
               NAME(C_WSA)
```

```
 >000006  PR   000005    000000    000018  03-L   N/A N/S  N/A   N/A  03  U-D  N/A   S   N
S     N/A
0              SORT KEY: 00000000 (HEX)
               NAME( )

 >000007  LD   000003        50    000006  01-L   N/A ANY  N/A   N/A  N/A N-I  N/A   S   N
X   00000000
0              EXTENDED ATTRIBUTES: ESDID = 000002, OFFSET=        0
               NAME(main)

 >000008  ER   000001     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   S   N/S
S   00000000
               NAME(CEESG003)

 >000009  ER   000001     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   S   N/S
S   00000000
               NAME(CBCSG003)

 >000010  SD   000000     N/A       N/A    N/A    N/A N/S  N/A  RENT  N/A N/A  N/A  N/A  N/A
N/A   N/A
               NAME(CEESTART)

 >000011  ED   000010         0        7C  01-N/A C   ANY  31    N/A  03  B-I  N/A  N/A  L,R,A
N/A   N/A
               NAME(C_CODE)

 >000012  LD   000011         0    000000  01-L   N/A ANY  N/A   N/A  N/A N-I  N/A   S   N/S
S   00000000
               NAME(CEESTART)

 >000013  ER   000010     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   W   N/S
S   00000000
               NAME(CEEMAIN)

 >000014  ER   000010     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   W   N/S
S   00000000
               NAME(CEEFMAIN)

 >000015  ER   000010     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   S   N/S
S   00000000
1                       ***** G E N E R A L I Z E D  O B J E C T  F I L E  F O R M A T
*****               PAGE       2
 MY PROGRAM IN GOFF FORMAT
               NAME(CEEBETBL)

 >000016  ER   000010     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-D  N/A   S   N/S
S   00000000
               NAME(CEEBLLST)

 >000017  ER   000010     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-I  N/A   S   N/S
S   00000000
               NAME(CEEROOTD)

 >000018  ER   000001     N/A       N/A    01-L   N   N/S  N/A   N/A  N/A N-I  N/A   S   N/S
S   00000000
               NAME(CEESTART)

 >000019  ED   000001         0         0  03-N/A M   ANY  31    N/A  03  B-D  N/A  N/A  L,A
N/A   N/A
               NAME(C_@@PPA2)

 >000020  PR   000019    000000    000008  03-L   N/A N/S  N/A   N/A  00  U-D  N/A   S   N
S     N/A
0              SORT KEY: 00000000 (HEX)
               NAME( )

 >000021  ER   000001     N/A       N/A    01-X   N   N/S  N/A   N/A  N/A N-I  N/A   S   G,N
X   00000000
0              EXTENDED ATTRIBUTES: ESDID = 000002, OFFSET=       14
               NAME(__ls__7os-amFPCc)

 >000022  ED   000001         0         0  03-N/A M   ANY  31    N/A  03  B-D  N/A  N/A  L,A,D
N/A   N/A
               NAME(C_WSA)

 >000023  PR   000022    000000    000000  03-X   N/A N/S  N/A   N/A  03  U-D  N/A   S   N
S     N/A
0              SORT KEY: 00000000 (HEX)
               NAME(cout)

 >000024  SD   000000     N/A       N/A    N/A    N/A N/S  N/A   N/S  N/A N/A  N/A  N/A  N/A
```

```
N/A   N/A
               NAME(CEEMAIN)

 >000025  ED   000024         0        10  01-N/A C   ANY  31    N/A  03  B-D  N/A  N/A  L,A
N/A   N/A
               NAME(C_DATA)

 >000026  LD   000025         0    000000  01-L   N/A ANY  N/A   N/A  N/A N-D  N/A   S   N/S
S   00000000
               NAME(CEEMAIN)

 >000027  ER   000001     N/A       N/A    01-L   N    N/S  N/A   N/A  N/A N-I  N/A   S   N/S
S   00000000
               NAME(EDCINPL)

 >000028  ER   000001     N/A       N/A    01-L   N    N/S  N/A   N/A  N/A N-I  N/A   S   N
X   00000000
               NAME(main)

 >000029  ED   000001         0       1B0  01-N/A C   ANY  31    N/A  03  B-D  N/A  N/A  A
N/A   N/A
               NAME(C_COPTIONS)

 >000030  ED   000001         0        22  01-N/A C   ANY  31    N/A  03  F-U  N/A  N/A
N/A   N/A
               NAME(B_IDRL)
0RECORD TYPE: TEXT     SEQUENCE:       32
  -- RESIDENT --     TRUE      TEXT    ENCODED
  ESDID   OFFSET    LENGTH   ENCODING  LENGTH    ------------------------------ T E X T
------------------------------

0>000002 00000000  00000000    0000   00000028   00000014 00010001 00010010 00040000 01000000 00000014
00010001 00010010
                                                 00040000 01000000

 >000003 00000000  00000000    0000   000000A0   41F0F050 07FF0700 00000000 F2F0F0F0 F0F1F3F1 F0F8F4F6
F1F6F0F2 F0F9F0F0
1                        ***** G E N E R A L I Z E D  O B J E C T  F I L E  F O R M A T
*****               PAGE        3
 MY PROGRAM IN GOFF FORMAT
                                                 02CE07F8 00000080 00000201 00000502 00000038 01000000
00049481 89950000
                                                 00C300C5 00C500F1 FFFFFFE0 00000050 905C47B4 A74AFFB0
0D8047F0 80205860
                                                 48045810 600C1826 98566010 0D764700 00044130 000047F0
802447F0 8004987C
                                                 480C4140 405007F7 FFFFFFB8 01000000 00C300C5 00C500F3
FFFFFFF6 00000000
 >000003 000000A0  00000000    0000   00000018   03012204 FFFFFF60 00000000 FFFFFF6C FFFFFFB0 01000000

 >000006 00000000  00000000    0000   00000018   C8859393 9640E696 99938400 00000000 00000000 00000000

 >000011 00000000  00000000    0000   0000007C   47000000 47000002 90ECD00C 053047F0 30180014 CE030209
0000002C C3C5C5E2
                                                 E3C1D9E3 000058F0 306A050F 00000000 00000000 00000000
00000000 00000000
                                                 FFFE004C 00000000 00000000 00000000 00000000 00000000
00000000 00000000
                                                 00000012 00000000 00000000 00000000 00000000 00000000
00000000

 >000020 00000000  00000000    0000   00000008   00000000 000000A0

 >000025 00000000  00000000    0000   00000010   02000001 00000000 00000000 00000000

 >000029 00000000  00000000    0000   000001B0   C1C7C7D9 C3D6D7E8 4DD5D6D6 E5C5D9D3 C1D75D40 C1D5E2C9
C1D3C9C1 E240C1D9
                                                 C3C84DF2 5D40C1D9 C7D7C1D9 E2C540D5 D6C3D6D4 D7D9C5E2
E240D5D6 C3D6D5E5
                                                 D3C9E340 D5D6C3E2 C5C3E340 C3E5C6E3 40C4D3D3 4DD5D6C3
C1D3D3C2 C1C3D2C1
                                                 D5E85D40 C5E7C5C3 D6D7E240 D5D6C5E7 D7D6D9E3 C1D3D340
C6D3D6C1 E34DC8C5
                                                 E76B40C6 D6D3C46B 40D5D6C1 C6D75D40 C7D6C6C6 40D5D6C7
D6D5E4D4 C2C5D940
                                                 D5D6C9C7 D5C5D9D9 D5D640D5 D6C9D5C9 E3C1E4E3 D640D5D6
C9D7C140 D3C1D5C7
                                                 D3E5D34D C5E7E3C5 D5C4C5C4 5D40D5D6 D3C9C2C1 D5E2C940
D5D6D3D6 C3C1D3C5
                                                 40D3D6D5 C7D5C1D4 C540D5D6 D6D7E3C9 D4C9E9C5 40D7D3C9
E2E34DC8 D6E2E35D
                                                 40D9C5C4 C9D940D5 D6D9D6C3 D6D5E2E3 40D5D6D9 D6E2E3D9
```

```
C9D5C740 D9D6E4D5
                                                  C44DE95D 40D5D6E2 C5D9E5C9 C3C540D5 D6E2D6D4 40E2D6D4
C5C9D5C9 E340D5D6
                                                  E2D6D4C7 E240E2D7 C9D3D34D F1F2F85D 40E2E3C1 D9E340E2
E3D9C9C3 E340D5D6
                                                  E2E3D9C9 C3E36DC9 D5C4E4C3 E3C9D6D5 40E3C1D9 C7C5E34D
D3C56B40 D6E2E5F2
                                                  D9F95D40 D5D6E3C5 E2E34DC8 D6D6D25D 40E3E4D5 C54DF35D
40E7D7D3 C9D5D240
                                                  C3D6D4D7 C9D3C5C4 6DD6D56D D4E5E2FF
0RECORD TYPE: IDRL     SEQUENCE:       40
0 ESDID            |---------- IDR  DATA ---------|    |---------- IDR  DATA ---------|

0>000030           |5647A01...02092000031084616000

0RECORD TYPE: RLD      SEQUENCE:       41
  R-PTR  P-PTR  OFFSET  TYPE LEN ATTRIB   R-PTR  P-PTR  OFFSET  TYPE LEN ATTRIB   R-PTR  P-PTR  OFFSET
TYPE LEN ATTRIB

0>000012 000011 000018 0000  004          000012 000011 000060 0000  004          000015 000011 000074
0001  004

 >000013 000011 00002C 0001  004          000014 000011 000068 0001  004          000016 000011 00006C
0001  004

 >000017 000011 000078 0001  004          000018 000003 0000A4 0000  004          000004 000003 0000A4
0002  004

 >000004 000020 000004 0000  004          000021 000006 000014 0001  004          000021 000006 000010
7001  004

 >000023 000006 00000C 0000  004          000028 000025 000004 0001  004          000027 000025 000008
0001  004

 >000028 000025 00000C 7001  004

0RECORD TYPE: END      SEQUENCE:       42
  RECORD       --ENTRY POINT--
  COUNT        AMODE    ESDID     OFFSET
1                       ***** G E N E R A L I Z E D  O B J E C T  F I L E  F O R M A T
*****                PAGE        4
 MY PROGRAM IN GOFF FORMAT
0>000042       ANY      000011    00000000

---------------------------------------------------------------------------------------------------------
--------------
1                            **  LONG NAME TABLE LISTING OF MEMBERMHELLOW    **
**             PAGE      1
 MY PROGRAM IN GOFF FORMAT
0ABBREVIATION        LONG NAME
0
0__ls__7os-amFPCc := __ls__7ostreamFPCc
0                            **  END OF LONG NAME TABLE LISTING OF MEMBER HELLOW     **
```

Display elements in “Example: LISTOBJ format for GOFF” on page 558 are described as follows. The numbers enclosed in braces following the field heading are the location (byte.bit) in the GOFF record where the data element can be found.

- **1** Page Header
  - The page header is printed at the top of each page.
  - The second line contains an optional user title.
- **2** HDR Record

  This is the first record in each GOFF module. The only data elements printed are the character set identifier and name and the language product (compiler or assembler) which produced the module.

  Data elements formatted:

  - CCSID
  - Character Set Name
  - Language Product Identifier

  Data elements not formatted:

  - Target Hardware Environment (U)
  - Target Operating System Environment (U)
- **3** ESD Record

  The ESD describes each external name defined or referenced in the module. Unlike the traditional object module, which provides for up to three names per record, the GOFF format contains only one name per record.

  Data elements formatted:

  - Line 1
    - ESDID. The identifier for the name being defined or referenced.
    - ESD TYPE. Symbol Type (SD, ED, LD, PR, ER)
    - OWNER/PARENT. The ESDID of the owning or referenced record type in the ESD hierarchy.
    - ITEM OFFSET. The offset, in bytes, of the start of this named entity from the start of the higher level entity.
    - ITEM LENGTH/ADA. For ED- or PR-type ESD records, the length (in bytes) of the entity being defined. If the length field is -1, the true length will be in a LEN-type GOFF record. For LD records, the ESDID of the associated data.
    - NAME SP/S. Name space (00-99) and binding scope (S (local or section),M (module), L (library), or X (import/export)).
    - BA. Binding algorithm (C=Catenate, M=Merge)
    - AMD. AMODE (N/S, 24, 31, 64, ANY, MIN)
    - RMD. RMODE (N/S, 24, 31, 64)
    - REUS. Reusability or tasking behavior. (N/S, NONE, REUS, RENT, REFR)
    - AL. Alignment. Print as decimal value n, where alignment boundary is 2**n. Range: 0-31.
    - TXT. Text type. Displayed in format x-y where
      - x is text record style (B (Byte oriented = 0), F (Fixed = 1), or V (Variable = 2)).
      - y is Executable (U (Unspecified= 0), D (Data=1), I (Instructions=2), or digits 3-7).
    - STR. Binding strength. (S (Strong=0), W (Weak=1)).
    - BINDER_FLAGS. Binder attributes is a string consisting of zero or more of the following characters. The ESD types to which the attribute is applicable are listed in parenthesis.
      - L. Initial or deferred load (ED)
      - M. Movable (ED)
      - R. Read-only (ED)
      - A. Addressable. Text may contain adcons. (ED)
      - C. Common (ED
      - I. Symbol defines or references a descriptor. (LD, ER, PR)
      - G. Mangled name (LD, PR, ER)
      - N. Name may be renamed. (LD, PR, ER)
      - D. Deferred load (ED)
      - V. Removable (ED)
    - LNK. Linkage Type (S (standard, non-XPLINK), X (XPLINK))
    - SIGNATURE. Any eight-byte string, printed in hexadecimal.
  - Sort Key. (Priority) Optional Field. PR only.
  - Extended Attributes Optional Field. Defines text location containing additional attributes for this ESD.
  - Symbol name. The first line begins with NAME, followed immediately by the name (up to 16 bytes). Names longer than 16 bytes will be abbreviated and displayed here, and an abbreviation-to-long

name equivalence table will be listed at the end of the listing. A closing parenthesis will immediately follow the last byte. A name consisting of a single blank character will be displayed as "NAME( )".

Data elements not formatted:

- Extended Attribute ESDID (U)
- Extended Attribute Data Offset (U)
- Alias or Alternate Symbol ID (U)
- Name Length (S)

- **4** TEXT Record

  TEXT records are a subset of the TXT record type. They contain the instructions and data of the program. TEXT is displayed in hexadecimal format.

  Data elements formatted:

  - Line 1
    - ESDID. Identifies the *element* or *part* to which the text belongs.
    - OFFSET. The offset within the element or part where the text is to begin.
    - TRUE LENGTH. The expanded length of the text once the encoding rules (if any) have been applied.
    - TEXT ENCODING. The technique for encoding or decoding the text. Current® values are 0 and 1.
    - ENCODED LENGTH. The unexpanded length of the text appearing in this record.
    - TEXT. The text, displayed as it appears in the record. The length of the text to be displayed appears in the ENCODED LENGTH field. Text is displayed in hexadecimal format, 32 bytes per line.
  - Lines 2-n

    All text beyond byte 31 is displayed on continuation lines. All bytes beyond the last text byte must be set to blank characters.

  Data elements not formatted:

  - Data Length (S)

- **5** IDRL Record

  The IDRL provides identification information for the language translator which produced the GOFF. It is a subset of the TXT record type, identified as structured record data. In format 1, the IDRL records will be displayed in 19-byte segments, four per line. In format 3, IDRL records will be displayed in 30-byte segments to support four-digit year values and time stamps, two per line.

- **6** RLD Record

  The relocation dictionary is a directory of address constants and other data areas which must be modified during binding and loading. Multiple such data areas or adcons can be described in a single RLD record. Relocation directory items begin at {8.0} in the RLD record and vary in length according to the presence or absence of various pointers and offsets in the item.

  Directory items are formatted three per line. Each item consists of up to five fields. Flags in the first byte of each directory item indicate which fields are present in the item. As a result, except for the flag bytes in positions 1-8, offsets are not fixed within the directory item as it appears in the GOFF file.

  Data elements formatted:

  - R-PTR. The ESDID of the target element, the value which will be used in relocating the address constant.
  - P-PTR. The ESDID of the element containing the adcon or data area to be modified.
  - OFFSET. The offset within the element described by the P-PTR at which the adcon or data area is located.
  - TYPE. This describes the type of adcon and implies the operation to be performed on it. Bytes 1 and 2 must be printed in hexadecimal.

- LEN. Length of the adcon or data area. Range: 2-255.
- ATTRIB.
  - H - the high order bit of the target field should be set from the target AMODE.
  - S - RLD is part of a conditional sequential RLD chain and the following RLD is also part of the chain.

Data elements not formatted:

- Total data length (S)
- Flag bytes 0 (except for 0.7) and 5 (S)
- Extended Attributes ESDID and offset (U)

• **7** END Record

The END record is the last record in the module. It contains a count of the records in the module and an optional entry point nomination, the latter specified by name or by class and offset.

Data elements formatted:

- Line 1
  - RECORD COUNT. Count of the *logical* records in the module, including the HDR and END records.
  - AMODE. Amode to be used for the entry point specified on this END record.
  - ENTRY POINT ESDID. The identifier of the element containing the entry point.
  - ENTRY POINT OFFSET. The offset of the entry point within the element identified by ESDID.
- Lines 2 contain the symbol name, if specified. The display format is identical to that on the ESD record type.

LISTLOAD OUTPUT=MODLIST output

"Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object" on page 564 is an example of the output produced by LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object.

Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object

```
     LISTLOAD MEMBER=ADATA3,OUTPUT=MODLIST,ADATA=YES
                                *****  M O D U L E   S U M M A R Y  *****
      MEMBER NAME:  ADATA3                                                                       MAIN ENTRY POINT:    00000368
      LIBRARY:      SYSLIB                                                                       AMODE OF MAIN ENTRY POINT: ANY
          NO ALIASES **

 ------------------------------------------------------------------------------------------------------------------------
                        ****          ATTRIBUTES OF MODULE          ****
                 **   BIT  STATUS          BIT  STATUS          BIT  STATUS          BIT  STATUS   **
                       0  NOT-RENT          1  NOT-REUS          2  NOT-OVLY          3  NOT-TEST
                       4  NOT-OL            5  BLOCK             6  EXEC              7  MULTI-RCD
                       8  NOT-DC            9  ZERO-ORG         10 RESERVED          11 RLD
                      12 EDIT              13 NO-SYMS           14 RESERVED          15 NOT-REFR
                      16 RESERVED          17 <16M              18 NOT-PL            19 NO-SSI
                      20 APF               21 PGM OBJ           22 NOT-SIGN          23 RESERVED
                      24 NOT-ALTP          25 RESERVED          26 RESERVED          27 RMODE24
                      28 RESERVED          29 RESERVED          30 RESERVED          31 RESERVED
                      32 NON-MIGR          33 NO-PRIME          34 NO-PACK           35 RESERVED
                      36 RESERVED          37 RESERVED          38 RESERVED          39 RESERVED

 ------------------------------------------------------------------------------------------------------------------------
                                     MODULE SSI:        NONE
                                     APFCODE:           00000000
                                     RMODE:             24
                                     PO FORMAT:         3
                                     XPLINK:            NO
                     *****PROGRAM OBJECT PROCESSED BY BINDER
  ***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
  PMAR  001E0309 02C00C03 00000000 05A40000   03680000 03680000 00000000 0000
  PMARL 00628000 00000000 000A0000 10180000   10000000 27AC0000 74980000 00E00000
        01400000 00240000 011C0000 000A0000   02200002 00000000 05300000 00180000
        20002011 017F0131 240FC1C4 C1E3C1F3   40403000 00010000 005C0000 30000000
        0410
                                           LISTING OF PROGRAM OBJECT ADATA3                                PAGE      1

  THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.12 ON 01/17/2011 AT 13:12:40

 ------------------------------------------------------------------------------------------------------------------------
```

```
MODULE SECTION:   $SUMMARY
 USABILITY: UNSPECIFIED    AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== ESDs =====
 C_WSA(ED)
     CLASS:            C_WSA      LENGTH:         5C (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE      RMODE:       ANY
     TEXT               DATA      DEFER                          FILL:                   0 (HEX)
 C_@@DLLI(ED)
     CLASS:         C_@@DLLI      LENGTH:          8 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE      RMODE:       ANY
     TEXT               DATA      LOAD                           FILL:                   0 (HEX)
 C_@@PPA2(ED)
     CLASS:         C_@@PPA2      LENGTH:          8 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE      RMODE:       ANY
     TEXT               DATA      LOAD                           READ-ONLY               0 (HEX)
 B_PRV(ED)
     CLASS:            B_PRV      LENGTH:          4 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           2      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE      RMODE:        24
     TEXT                         NOLOAD                         FILL:                    UNSPEC
 $PRIV000010(PD)
     CLASS:         C_@@DLLI      LENGTH:          8 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           3      ALIGNMENT:           BYTE       PRIORITY:               0 (HEX)      SCOPE:   LIBRARY
     ATTRIBUTES:     GENERATED,STRONG
 $PRIV000011(PD)
     CLASS:         C_@@PPA2      LENGTH:          8 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           3      ALIGNMENT:           BYTE       PRIORITY:               0 (HEX)      SCOPE:   LIBRARY
     ATTRIBUTES:     GENERATED,STRONG
 $PRIV000012(PD)
     CLASS:            C_WSA      LENGTH:         10 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       PRIORITY:               0 (HEX)      SCOPE:   SECTION
     ATTRIBUTES:     GENERATED,STRONG
 __ls__7os-amFPCc(PD)
     CLASS:            C_WSA      LENGTH:         20 (HEX)       CLASS OFFSET:          10 (HEX)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       PRIORITY:               0 (HEX)      SCOPE:    MODULE
     ATTRIBUTES:     GENERATED,STRONG,INDIRECT,MANGLED
 endl__FR7-stream(PD)
     CLASS:            C_WSA      LENGTH:         20 (HEX)       CLASS OFFSET:          30 (HEX)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       PRIORITY:               0 (HEX)      SCOPE:    MODULE
     ATTRIBUTES:     GENERATED,STRONG,INDIRECT,MANGLED
 HELLOW1#S(PD)
     CLASS:            C_WSA      LENGTH:          C (HEX)       CLASS OFFSET:          50 (HEX)
     NAME SPACE:           3      ALIGNMENT:    DOUBLE WORD       PRIORITY:               0 (HEX)      SCOPE:   LIBRARY
     ATTRIBUTES:     GENERATED,STRONG
 Q1(PD)
     CLASS:            B_PRV      LENGTH:          4 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           2      ALIGNMENT:      FULL WORD       PRIORITY:               0 (HEX)      SCOPE:    MODULE
     ATTRIBUTES:     WEAK
                                              LISTING OF PROGRAM OBJECT ADATA3                                 PAGE      2

            MODULE SECTION:   $SUMMARY
 ===== RLDs =====            CLASS:            C_WSA
 ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES            XATTR NAME      XATTR OFF
 00000008 00000018   BR UNRES  0004               1     (+)__ls__7os-amFPCc __ls__7os-amFPCc
 0000000C 0000001C N-BR   RES  0004               3     (+)__ls__7os-amFPCc __ls__7os-amFPCc
 00000010 00000020   BR UNRES  0004               1     (+)CEETGTFN         __ls__7os-amFPCc
 00000018 00000028   BR   RES  0004               1     (+)CEESTART         __ls__7os-amFPCc
 0000001C 0000002C SEGM   RES  0004               0     (+)C_WSA            __ls__7os-amFPCc
 00000008 00000038   BR UNRES  0004               1     (+)endl__FR7-stream endl__FR7-stream
 0000000C 0000003C N-BR   RES  0004               3     (+)endl__FR7-stream endl__FR7-stream
 00000010 00000040   BR UNRES  0004               1     (+)CEETGTFN         endl__FR7-stream
 00000018 00000048   BR   RES  0004               1     (+)CEESTART         endl__FR7-stream
 0000001C 0000004C SEGM   RES  0004               0     (+)C_WSA            endl__FR7-stream
 ===== TEXT =====            CLASS:            C_WSA
 00000000  C36DE6E2 C1404040 40404040 40404040 180F58FF 001007FF 00000000 00000010   *C.WSA...........................*
 00000020  00000000 00000000 00000368 00000000 180F58FF 001007FF 00000000 00000030   *................................*
 00000040  00000000 00000000 00000368 00000000 00000000 00000000 00000000             ................................*
 ===== TEXT =====            CLASS:         C_@@DLLI
 00000000  00000000 00000250                                                          ................................*
 ===== TEXT =====            CLASS:         C_@@PPA2
 00000000  00000000 00000350                                                          ................................*

-------------------------------------------------------------------------------------------------------------------------
            CONTROL SECTION:  A
 USABILITY: UNSPECIFIED    AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== IDRL =====
          TRANSLATOR   VER   MOD     DATE        TIME
          569623400     01    06    01/17/2011
 ===== ESDs =====
B_PRV(ED)
     CLASS:            B_PRV      LENGTH:          0 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           2      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE      RMODE:        24
     TEXT                         NOLOAD                         FILL:                    UNSPEC
 B_TEXT(ED)
     CLASS:           B_TEXT      LENGTH:         18 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE      RMODE:        24
     TEXT                         LOAD                           FILL:                    UNSPEC
 C_ADATA0000(ED)
     CLASS:      C_ADATA0000      LENGTH:         AA (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE      RMODE:        24
     DESCRIPTIVE DATA   DATA      NOLOAD                         READ-ONLY                UNSPEC
 C_ADATA0001(ED)
     CLASS:      C_ADATA0001      LENGTH:         16 (HEX)       CLASS OFFSET:           0 (HEX)      FORMAT:  F(0001)
     NAME SPACE:           1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE      RMODE:        24
```

```
    DESCRIPTIVE DATA   DATA      NOLOAD                          READ-ONLY                UNSPEC
A(LD)
    CLASS:            B_TEXT      TEXT TYPE:         UNSPEC      CLASS OFFSET:            0 (HEX)
    NAME SPACE:            1      SCOPE:             MODULE      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    ATTRIBUTES:     STRONG
ENTA(ER)
                                  TEXT TYPE:         UNSPEC      CLASS OFFSET:           14 (HEX)
    TARGET SECTION: A                                            TARGET CLASS:            B_TEXT
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:         14 (HEX)      AMODE:       UNS
    RESOLVED                      AUTOCALL
    ATTRIBUTES:     STRONG
ENTB(ER)
                                  TEXT TYPE:         UNSPEC      CLASS OFFSET:            0 (HEX)
    TARGET SECTION:                                              TARGET CLASS:
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    UNRESOLVED                    AUTOCALL
    ATTRIBUTES:     STRONG
Q1(PR)
    CLASS:             B_PRV      LENGTH:           4 (HEX)      CLASS OFFSET:            0 (HEX)
    NAME SPACE:            2      ALIGNMENT:      FULL WORD      PRIORITY:                0 (HEX)      SCOPE:    MODULE
    ATTRIBUTES:     WEAK
ENTA(LD)
    CLASS:            B_TEXT      TEXT TYPE:         UNSPEC      CLASS OFFSET:           14 (HEX)
    NAME SPACE:            1      SCOPE:             MODULE      ELEMENT OFFSET:         14 (HEX)      AMODE:       UNS
    ATTRIBUTES:     STRONG
===== RLDs =====            CLASS:           B_TEXT
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES            XATTR NAME      XATTR OFF
00000006 00000006   BR   RES  0004              1     (+)ENTA
0000000C 0000000C C-OF   RES  0004              2     (+)Q1
00000010 00000010   BR UNRES  0004              1     (+)ENTB
===== TEXT =====            CLASS:           B_TEXT
00000000  07FEC1C1 C1C10000 00140000 00000000 00000000 C5D5E3C1                      ..AAAA..............ENTA........*
==== ADATA ====             CLASS:      C_ADATA0000
00000000  10000003 00010000 0000009E F2F0F1F1 F0F1F1F7 F1F3F1F2 F5F6F9F6 60F2F3F4   *............2011011713125696.234*
00000020  F14BF64B F0404040 0006E4D2 F6F2F8F3 F540A961 D6E240F0 F14BF1F2 4BF0F040   *1.6.0.....UK62835.z.OS.01.12.00.*
00000040  40404040 40404040 4040C1C4 C1E3C1F3 4040C1E2 E2C5D4C2 D3C54040 40404040   *..........ADATA3..ASSEMBLE......*
00000060  40400000 00010000 006A0000 00000000 00010000 008A0000 00200000 00000000   *................................*
00000080  00000000 00000000 0000D3C5 D6D5C14B C1C4C1E3 C1F34BD1 D6C2F1F7 F1F9F74B   *..........LEONA.ADATA3.JOB17197.*
000000A0  C4F0F0F0 F0F1F0F1 4B6F                                                     D0000101.........................*
==== ADATA ====             CLASS:      C_ADATA0001
00000000  10000103 00000000 0000000A C731F038 09628623 0025                          ............G.0...f.............*

------------------------------------------------------------------------------------------------------------------------
           CONTROL SECTION:  CEESTART
USABILITY: REENTRANT      AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
===== ESDs =====
C_CODE(ED)
    CLASS:            C_CODE      LENGTH:          7C (HEX)      CLASS OFFSET:          368 (HEX)      FORMAT:  F(0001)
    NAME SPACE:            1      ALIGNMENT:    DOUBLE WORD      BIND METHOD:           CATENATE       RMODE:       ANY
    TEXT                INSTR     LOAD                           READ-ONLY                0 (HEX)
CEESTART(LD)
    CLASS:            C_CODE      TEXT TYPE:          INSTR      CLASS OFFSET:          368 (HEX)
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       ANY
    ATTRIBUTES:     STRONG
CEEMAIN(ER)
                                  TEXT TYPE:           DATA      CLASS OFFSET:            0 (HEX)
    TARGET SECTION:                                              TARGET CLASS:            C_DATA
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    UNRESOLVED                    AUTOCALL
    ATTRIBUTES:     WEAK
CEEFMAIN(ER)
                                  TEXT TYPE:           DATA      CLASS OFFSET:            0 (HEX)
    TARGET SECTION:                                              TARGET CLASS:
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    UNRESOLVED                    AUTOCALL
    ATTRIBUTES:     WEAK
CEEBETBL(ER)
                                  TEXT TYPE:           DATA      CLASS OFFSET:            0 (HEX)
    TARGET SECTION:                                              TARGET CLASS:
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    UNRESOLVED                    AUTOCALL
    ATTRIBUTES:     STRONG
CEEROOTA(ER)
                                  TEXT TYPE:          INSTR      CLASS OFFSET:            0 (HEX)
    TARGET SECTION:                                              TARGET CLASS:
    NAME SPACE:            1      SCOPE:            LIBRARY      ELEMENT OFFSET:          0 (HEX)      AMODE:       UNS
    UNRESOLVED                    AUTOCALL
    ATTRIBUTES:     STRONG
===== RLDs =====            CLASS:           C_CODE
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES            XATTR NAME      XATTR OFF
00000018 00000380 N-BR   RES  0004              1     (+)CEESTART
0000002C 00000394   BR   RES  0004              1     (+)CEEMAIN
00000060 000003C8 N-BR   RES  0004              1     (+)CEESTART
00000068 000003D0   BR UNRES  0004              1     (+)CEEFMAIN
00000074 000003DC   BR UNRES  0004              1     (+)CEEBETBL
00000078 000003E0   BR UNRES  0004              1     (+)CEEROOTA
===== TEXT =====            CLASS:           C_CODE
00000368  47000000 47000002 90ECD00C 053047F0 30180014 CE030310 00000394 C3C5C5E2   *...............0...........mCEES*
00000388  E3C1D9E3 000058F0 306A050F 00000008 00000000 00000000 00000000 00000000   *TART...0........................*
000003A8  FFFF004C 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *................................*
000003C8  0000037A 00000000 00000000 00000000 00000000 00000000 00000000             ................................*

------------------------------------------------------------------------------------------------------------------------
```

```
                                   LISTING OF PROGRAM OBJECT ADATA3                                PAGE     11

           CONTROL SECTION:  HELLOW1#C
USABILITY: REENTRANT      AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
===== IDRL =====
          TRANSLATOR   VER   MOD     DATE        TIME
          5647A01       02    10    08/16/2000
===== ESDs =====
C_CODE(ED)
    CLASS:          C_CODE      LENGTH:        368 (HEX)      CLASS OFFSET:          0 (HEX)      FORMAT:  F(0001)
    NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD     BIND METHOD:          CATENATE      RMODE:       ANY
    TEXT                INSTR     LOAD                          READ-ONLY              0 (HEX)
HELLOW1#C(LD)
    CLASS:          C_CODE      TEXT TYPE:          INSTR     CLASS OFFSET:          0 (HEX)
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:        0 (HEX)      AMODE:       ANY
    ATTRIBUTES:     STRONG
C_WSA(ED)
    CLASS:           C_WSA      LENGTH:          0 (HEX)      CLASS OFFSET:          0 (HEX)      FORMAT:  F(0001)
    NAME SPACE:          3      ALIGNMENT:    DOUBLE WORD     BIND METHOD:             MERGE      RMODE:       ANY
    TEXT                DATA      DEFER                         FILL:                  0 (HEX)
HELLOW1#S(PR)
    CLASS:           C_WSA      LENGTH:          C (HEX)      CLASS OFFSET:         50 (HEX)
    NAME SPACE:          3      ALIGNMENT:    DOUBLE WORD     PRIORITY:              0 (HEX)      SCOPE:   LIBRARY
    ATTRIBUTES:     STRONG
C_@@DLLI(ED)
    CLASS:        C_@@DLLI      LENGTH:          0 (HEX)      CLASS OFFSET:          0 (HEX)      FORMAT:  F(0001)
    NAME SPACE:          3      ALIGNMENT:    DOUBLE WORD     BIND METHOD:             MERGE      RMODE:       ANY
    TEXT                DATA      LOAD                          FILL:                  0 (HEX)$PRIV000010(PR)
    CLASS:        C_@@DLLI      LENGTH:          8 (HEX)      CLASS OFFSET:          0 (HEX)
    NAME SPACE:          3      ALIGNMENT:           BYTE     PRIORITY:              0 (HEX) SCOPE:   LIBRARY
    ATTRIBUTES:     STRONG,MAPPED
main(LD)
    CLASS:          C_CODE      TEXT TYPE:          INSTR     CLASS OFFSET:         58 (HEX)
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:       58 (HEX)      AMODE:       ANY
    ATTRIBUTES:     STRONG,MAPPED
CEESG003(ER)
                                TEXT TYPE:           DATA     CLASS OFFSET:          0 (HEX)
    TARGET SECTION:                                           TARGET CLASS:
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:        0 (HEX)      AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG
CBCSG003(ER)
                                TEXT TYPE:           DATA     CLASS OFFSET:          0 (HEX)
    TARGET SECTION:                                           TARGET CLASS:
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:        0 (HEX)      AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG
C_DATA(ED)
    CLASS:          C_DATA      LENGTH:          4 (HEX)      CLASS OFFSET:          0 (HEX)      FORMAT:  F(0001)
    NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD     BIND METHOD:          CATENATE      RMODE:       ANY
    TEXT                DATA      LOAD                          FILL:                  0 (HEX)
HELLOW1#T(LD)
    CLASS:          C_DATA      TEXT TYPE:           DATA     CLASS OFFSET:          0 (HEX)
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:        0 (HEX)      AMODE:       ANY
    ATTRIBUTES:     STRONG
C_WSA(ED)
    CLASS:           C_WSA      LENGTH:          0 (HEX)      CLASS OFFSET:          0 (HEX)      FORMAT:  F(0001)
    NAME SPACE:          3      ALIGNMENT:    DOUBLE WORD     BIND METHOD:             MERGE      RMODE:       ANY
    TEXT                DATA      DEFER                         FILL:                  0 (HEX)
endl__FR7-stream(PR)
    CLASS:           C_WSA      LENGTH:          0 (HEX)      CLASS OFFSET:         30 (HEX)
    NAME SPACE:          3      ALIGNMENT:           BYTE     PRIORITY:              0 (HEX)      SCOPE:    MODULE
    ATTRIBUTES:     STRONG,MAPPED,INDIRECT,MANGLED
endl__FR7-stream(ER)
                                TEXT TYPE:          INSTR     CLASS OFFSET:          0 (HEX)
    TARGET SECTION:                                           TARGET CLASS:
    NAME SPACE:          1      SCOPE:            EXP/IMP     ELEMENT OFFSET:        0 (HEX)      AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG,MAPPED,MANGLED
__ls__7os-amFPCc(PR)
    CLASS:           C_WSA      LENGTH:          0 (HEX)      CLASS OFFSET:         10 (HEX)
    NAME SPACE:          3      ALIGNMENT:           BYTE     PRIORITY:              0 (HEX)      SCOPE:    MODULE
    ATTRIBUTES:     STRONG,MAPPED,INDIRECT,MANGLED
__ls__7os-amFPCc(ER)
                                TEXT TYPE:          INSTR     CLASS OFFSET:          0 (HEX)
    TARGET SECTION:                                           TARGET CLASS:
    NAME SPACE:          1      SCOPE:            EXP/IMP     ELEMENT OFFSET:        0 (HEX)
AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG,MAPPED,MANGLED
cout(PR)
    CLASS:           C_WSA      LENGTH:          0 (HEX)      CLASS OFFSET:          0 (HEX)
    NAME SPACE:          3      ALIGNMENT:      FULL WORD     PRIORITY:              0 (HEX)      SCOPE:   EXP/IMP
    ATTRIBUTES:     STRONG,MAPPED
@@TRGLOR(ER)
                                TEXT TYPE:          INSTR     CLASS OFFSET:          0 (HEX)
    TARGET SECTION:                                           TARGET CLASS:
    NAME SPACE:          1      SCOPE:            LIBRARY     ELEMENT OFFSET:        0 (HEX)      AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG
CEESTART(ER)
                                TEXT TYPE:          INSTR     CLASS OFFSET:        368 (HEX)
    TARGET SECTION: CEESTART                                  TARGET CLASS:           C_CODE
```

```
    NAME SPACE:          1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)       AMODE:       UNS
    RESOLVED                    AUTOCALL
    ATTRIBUTES:     STRONG
C_@@PPA2(ED)
    CLASS:        C_@@PPA2      LENGTH:           0 (HEX)       CLASS OFFSET:           0 (HEX)       FORMAT:  F(0001)
    NAME SPACE:          3      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE       RMODE:       ANY
    TEXT              DATA      LOAD                            READ-ONLY               0 (HEX)
$PRIV000011(PR)
    CLASS:        C_@@PPA2      LENGTH:           8 (HEX)       CLASS OFFSET:           0 (HEX)
    NAME SPACE:          3      ALIGNMENT:           BYTE       PRIORITY:               0 (HEX)       SCOPE:   LIBRARY
    ATTRIBUTES:     STRONG,MAPPED
EDCINPL(ER)
                                TEXT TYPE:          INSTR       CLASS OFFSET:           0 (HEX)
    TARGET SECTION:                                             TARGET CLASS:
    NAME SPACE:          1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)       AMODE:       UNS
    UNRESOLVED                  AUTOCALL
    ATTRIBUTES:     STRONG     main(ER)
                                TEXT TYPE:          INSTR       CLASS OFFSET:          58 (HEX)
    TARGET SECTION: HELLOW1#C                                   TARGET CLASS:            C_CODE
    NAME SPACE:          1      SCOPE:            LIBRARY       ELEMENT OFFSET:        58 (HEX)       AMODE:       UNS
    RESOLVED                    AUTOCALL
    ATTRIBUTES:     STRONG,MAPPED
C_COPTIONS(ED)
    CLASS:      C_COPTIONS      LENGTH:         1F2 (HEX)       CLASS OFFSET:           0 (HEX)       FORMAT:  F(0001)
    NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE       RMODE:       ANY
    TEXT              DATA      NOLOAD                          READ-ONLY               0 (HEX)
===== RLDs =====          CLASS:        C_@@PPA2
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES          XATTR NAME      XATTR OFF
00000004 00000004 N-BR   RES  0004              1     (+)HELLOW1#C       $PRIV000011
===== RLDs =====          CLASS:        C_@@DLLIC_COPTIONS(ED)
    CLASS:      C_COPTIONS      LENGTH:         1F2 (HEX)       CLASS OFFSET:           0 (HEX)       FORMAT:  F(0001)
    NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE       RMODE:       ANY
    TEXT              DATA      NOLOAD                          READ-ONLY               0 (HEX)
===== RLDs =====          CLASS:        C_@@PPA2
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES          XATTR NAME      XATTR OFF
00000004 00000004 N-BR   RES  0004              1     (+)HELLOW1#C       $PRIV000011

ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES          XATTR NAME      XATTR OFF
00000004 00000004 N-BR   RES  0004              1     (+)HELLOW1#C       $PRIV000010
===== RLDs =====          CLASS:          C_CODE
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES          XATTR NAME      XATTR OFF
00000108 00000108 C-OF   RES  0004              3     (+)HELLOW1#S
0000010C 0000010C N-BR   RES  0004              1     (+)HELLOW1#C
00000110 00000110 N-BR   RES  0004              1     (+)HELLOW1#C
0000032C 0000032C C-OF   RES  0004              3     (+)HELLOW1#S
00000330 00000330 C-OF   RES  0004              3     (+)endl__FR7-stream
00000334 00000334 C-OF   RES  0004              3     (+)__ls__7os-amFPCc
00000338 00000338 C-OF UNRES  0004              3     (+)cout
0000033C 0000033C   BR UNRES  0004              1     (+)@@TRGLOR
00000354 00000354 N-BR   RES  0004              1     (-)HELLOW1#C
00000354 00000354 N-BR   RES  0004              1     (+)CEESTART
===== TEXT =====          CLASS:          C_CODE
00000000  F2F0F0F0 F0F8F1F6 F1F0F0F5 F4F3F0F2 F0C1F0F0 00000000 1CCEA106 000002F8   *20000816100543020A00...........8*
00000020  00000000 00000000 FF800000 00000001 90000001 00400012 00000000 50000058   *................................*
00000040  00000040 38270000 00000000 00000000 00049481 89950000 47F0F028 01C3C5C5   *..................main...00..CEE*
00000060  000000A0 FFFFFFC0 47F0F001 58F0C31C 184E05EF 00000000 05404140 401E07F4   *.........00..0C................4*
00000080  90E6D00C 58E0D04C 4100E0A0 5500C314 4140F040 4720F014 5000E04C 9210E000   *.W...........C...0...0.....k...*
000000A0  50D0E004 18DE5800 C1F45000 D0985810 D0985820 40704152 10005860 40745810   *........A4...q...q.............*
000000C0  50005820 500458F0 20085800 200C1826 4DE0F010 47000008 58205008 5800D098   *.......0..........0............q*
000000E0  58F04078 4DE0F010 47000008 5800D098 5000C1F4 180D58D0 D0041BFF 58E0D00C   *.0....0........q..A4............*
00000100  9826D01C 051E0707 00000050 00000340 00000180 00000000 1CCE2109 000001D0   *q...............................*
00000120  00000000 00000000 FE000000 00000001 E0000000 02400012 00000000 5000003F   *................................*
00000140  00000068 18260000 00000000 00000000 002A96A2 A3998581 947A7A96 97859981   *..................ostream..opera*
00000160  A396994C 4C4D96A2 A3998581 94504D5C 5D4D96A2 A3998581 94505D5D 00000000   *tor...ostream.....ostream.......*
===== TEXT =====          CLASS:       C_COPTIONS
00000000  C1C7C7D9 C3D6D7E8 4DD5D6D6 E5C5D9D3 C1D75D40 C1D5E2C9 C1D3C9C1 E240C1D9   *AGGRCOPY.NOOVERLAP..ANSIALIAS.AR*
00000020  C3C84DF0 5D40C1D9 C7D7C1D9 E2C540D5 D6C3D6D4 D7C1C3E3 40D5D6C3 D6D4D7D9   *CH.0..ARGPARSE.NOCOMPACT.NOCOMPR*
00000040  C5E2E240 D5D6C3D6 D5E5D3C9 E340C3E2 C5C3E34D C3D6C4C5 6B40C8C5 D3D3D6E6   *ESS.NOCONVLIT.CSECT.CODE..HELLOW*

------------------------------------------------------------------------------------------------------------------------
                       CONTROL SECTION:  IEWBLIT
USABILITY: UNSPECIFIED    AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
===== ESDs =====
B_LIT(ED)
    CLASS:           B_LIT      LENGTH:         120 (HEX)       CLASS OFFSET:           0 (HEX)       FORMAT:  F(0001)
    NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE       RMODE:       ANY
    TEXT              DATA      LOAD                            FILL:                    UNSPEC
IEWBLIT(LD)
    CLASS:           B_LIT      TEXT TYPE:           DATA       CLASS OFFSET:           0 (HEX)
===== ESDs =====
    NAME SPACE:          1      SCOPE:             MODULE       ELEMENT OFFSET:         0 (HEX)       AMODE:       ANY
    ATTRIBUTES:     GENERATED,STRONG
===== RLDs =====          CLASS:           B_LIT
ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES          XATTR NAME      XATTR OFF
00000028 00000028 LTKN   RES  0008              0     (+)
00000050 00000050  CPR   RES  0004              0        C_CODE
00000054 00000054 SEGM   RES  0004              0     (+)C_CODE
00000070 00000070  CPR   RES  0004              0        C_@@DLLI
00000074 00000074 SEGM   RES  0004              0     (+)C_@@DLLI
00000090 00000090  CPR   RES  0004              0        C_DATA
00000094 00000094 SEGM   RES  0004              0     (+)C_DATA
000000B0 000000B0  CPR   RES  0004              0        C_@@PPA2
000000B4 000000B4 SEGM   RES  0004              0     (+)C_@@PPA2
```

```
 000000D0 000000D0  CPR   RES  0004              0        B_TEXT
 000000D4 000000D4 SEGM   RES  0004              0     (+)B_TEXT
 000000F0 000000F0  CPR   RES  0004              0        B_LIT
 000000F4 000000F4 SEGM   RES  0004              0     (+)B_LIT
 00000110 00000110  CPR   RES  0004              0        C_WSA
 ===== TEXT =====            CLASS:            B_LIT
 00000000  C9C5E6C2 D3C9E340 00000120 01000000 00000040 00000020 00000007 00000001   *IEWBLIT.........................*
 00000020  00000000 00000000 00000000 00000000 00000000 18000000 00000000 00000000   *................................*
 00000040  C36DC3D6 C4C54040 40404040 40404040 000003E4 00000000 03038000 00000000   *C.CODE............U............*
 00000060  C36D7C7C C4D3D3C9 40404040 40404040 00000008 00000000 03030000 00000000   *C...DLLI........................*
 00000080  C36DC4C1 E3C14040 40404040 40404040 00000014 00000000 03030000 00000000   *C.DATA..........................*
 000000A0  C36D7C7C D7D7C1F2 40404040 40404040 00000008 00000000 03038000 00000000   *C...PPA2........................*
 000000C0  C26DE3C5 E7E34040 40404040 40404040 00000018 00000000 01030000 00000000   *B.TEXT..........................*
 000000E0  C26DD3C9 E3404040 40404040 40404040 00000120 00000000 03030000 00000000   *B.LIT...........................*
 00000100  C36DE6E2 C1404040 40404040 40404040 0000005C 00000000 03032080 00000000   *C.WSA...........................*

-------------------------------------------------------------------------------------------------------------------------
==== MERGE CLASS PART INITIALIZERS ====
 CLASS            PART             OFFSET REPEAT --------------------  I N I T I A L  T E X T  ----------------------

 C_@@DLLI         $PRIV000010      000000 00001  00000000 00000250
 C_@@PPA2         $PRIV000011      000000 00001  00000000 00000350

-------------------------------------------------------------------------------------------------------------------------
                             **  LONG NAME TABLE LISTING OF PROGRAM OBJECT ADATA3            **             PAGE     16

 ABBREVIATION        LONG NAME

 __ls__7os-amFPCc := __ls__7ostreamFPCc
 endl__FR7-stream := endl__FR7ostream
                             **  END OF LONG NAME TABLE LISTING OF PROGRAM OBJECT ADATA3           **
 **   END OF PROGRAM OBJECT LISTING
```

"Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object" on page 564 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for an overlay structured load module.

Example: Output for LISTLOAD OUTPUT=MODLIST for an overlay structured load module

```
                                        LISTING OF LOAD MODULE PL1LOAD                  PAGE 0001


RECORD# 1     TYPE 20 - CESD     ESDID 1                     ESD SIZE 240
             CESD#    SYMBOL    TYPE     ADDRESS     SEGNUM   ID/LENGTH(DEC)   (HEX)
              1    PLITC0Z   00(SD)     000000       1          1206         486
                2    PLITC0ZA  00(SD)     000488       1           608         260
                     3    IHEQINV   06(PR)     000000          3             4               4
                     4    IHESADA   02(ER)     000000
                     5    INESADB   02(ER)     000000
                     6    IHEQERR   06(PR)     000004          3               4             4
                     7    IHEQTIC   06(PR)     000008          3               4             4
                     8    IHEMAIN   00(SD)     000718          1               4             4
                     9    IHENTRY   00(SD)     000720          1            12           C
                   10    IHESAPC   02(ER)     000000
                   11    IHEQLWF   06(PR)     00000C         3              4           4
                   12    IHEQSLA   06(PR)     000010         3              4           4
                   13    IHEQLW0   06(PR)     000014         3              4           4
                   14    PLITC0ZB  06(PR)     000018         3              4           4
                   15    PLITC0ZC  06(PR)     00001C         3              4           4

RECORD# 2     TYPE 20 - CESD     ESDID 16                    ESD SIZE 240
             CESD#    SYMBOL    TYPE     ADDRESS     SEGNUM   ID/LENGTH(DEC)   (HEX)
                   16    IHELDQA   02(ER)     000000
                   17    IHELDQB   02(ER)     000000
                   18    IHEIQBT   02(ER)     000000
                   19    IHEIQBC   02(ER)     000000
                   20    IHESAFA   02(ER)     000000
                   21    IHESAFB   02(ER)     000000
                   22    AA          02(ER)     000000
                   23    C            00(SD)     000730       1                4           4
                   24    B            00(SD)     000738       1                4           4
                   25    A            00(SD)     000740       1                4           4
                   26    IHESPRT   00(SD)     000748       1              56           38
                   27    IHEQSPR   06(PR)     000020       3               4            4
                   28    IHEDNC    02(ER)     000000
                   29    IHEVPF    02(ER)     000000
                   30    IHEDMA    02(ER)     000000
RECORD# 3     TYPE 20 - CESD     ESDID 31                    ESD SIZE 64
             CESD#    SYMBOL    TYPE     ADDRESS     SEGNUM   ID/LENGTH(DEC)   (HEX)
                   31    IHEVPB    02(ER)         000000
                   32    IHEVSC    02(ER)         000000
                   33    IHEUPA    02(ER)         000000
                   34    IHEVQC    02(ER)         000000


-------------------------------------------
                                         LISTING OF LOAD MODULE PL1LOAD                  PAGE 0002
RECORD# 4    TYPE 01 - CONTROL          CONTROL SIZE 32                     CCW 060000000  40000780
```

```
                CESD#   LENGTH
                    1       0488
                    2     0260
                    8        0008
                    9        0010
                  23       0008
                    24     0008
                    25       0008
                    26     0038
RECORD# 5                                         T E X T
          000000    47F0F914 07D7D3F1 E3C3F0F2 000000D8    000004B8 90EBD00C 58B0F010 5800F00C
          000020    58F0B0Z0 05EF05A0 4190D0B8 50DC0018    9200D062 92919963 92C0D000 9202D063
          000040    F811D090 B132F810 D092B080 FA11D092    B130F821 99A80009 F821D0AB D092D203
          000060    D0AEB134 F811D090 B13CF810 D092B080    FA11D092 B13AF821 D0B2D090 F821D0B5
          000080    D09241A0 A0600700 9203D063 4110B174    58F0B05C 95EF4119 B1144120 818358F0
          0000A0    B05405EF 9203D063 58F0B058 05EF9204    D0635880 8979F821 D0908000 F821D093
          0000C0    8002FA20 D093B111 5879B06C D2017000    D091D201 79920004 9205D063 F821D090
          0000E0    7000F821 D0937002 FA20D093 B10F5860    B068D201 60000001 D2016002 D0949206
          000100    D0634150 D0AE5050 D0944150 D0905050    D0989680 00084119 D09458F0 B06405EF
          000120    5880B070 D2038000 D0909207 D063F811    D090B10C F8100092 B080FA11 D092B10A
          000140    F9118000 D0904770 A0C8F911 8002D092    4780A0EE 92989963 4110B168 58F0B05C
          000160    05EF4110 B14058F0 B05005EF 9208D063    58F0B058 95EF9208 D0639210 D0634180
          000180    D0A85080 D0984180 D0825080 D09C4180    D0905080 99A99689 D0A04110 D09858F0
          0001A0    B04005EF D205D0B2 D0909211 D063D202    D090D0B2 F9210009 B0D19200 D0904780
          0001C0    A13E9280 D090D202 D091D0B5 F921D091    B0CF9200 00014789 A1569280 D091D200
          0001E0    D094D090 D600D094 D0919180 D0944780    A19E9212 99634119 B15C58F0 B05C05EF
          000200    4110B0A0 4120B183 58F0B054 05EF4110    D0B24120 818758F9 B05405EF 9212D063
          000220    58F0B058 05EF9213 D0634110 B15058F0    B05C05EF 41198984 4120B183 58F0B054
          000240    05EF9213 D06358F0 B05805EF 9214D063    58F0B030 95EF47F9 47F0F00C 03C1E7F1
          000260    000000D0 90EBD00C 18AF41E0 A0285830    80381B22 59293959 58F0B02C 47F0F062
          000280    9201D084 58E01000 50E0D088 4580A03A    07FA05A0 41000989 50DC001C 9200D062
          000200    9209D063 41A0A088 07F80700 47F0F00C    03C1C3F1 00099258 90EBD00C 58A0F008
          0002C0    45E0A016 9202D084 D207D0A0 10009200    D0A458E0 199859E9 D0884580 A03A47F0
          0002E0    A0000700 47F0F00C 03C1C3F2 00000258    90EBD00C 58A0F008 45E0A016 9203D084
          000300    D207D0A8 10009200 D0AC58E0 100850E0    D0884580 A93047F9 A0860700 920BD063
          000320    920CD063 5880D0A0 F821D090 80005870    D0A4FA21 00097000 F821D093 8002FA21
          000340    D0937002 9502D084 4780A062 9503D084    4780A076 58600088 F872D098 D0904FE0
          000360    D09810FE 54E0B078 90EFD098 964ED098    2B006A00 00087000 600047F0 A0805880
          000380    D088D201 8000D091 D2018002 D09447F0    A0805880 99889295 8000D090 58F0B060
          0003A0    05EF920D D063920E D0635880 D0A8F822    D0908000 5870D0AC FB22D090 7000F822
          0003C0    D0938003 FB22D093 70039502 D0844780    A0E89503 99844789 A0FC5860 D088F872
          0003E0    D098D090 4FE0D098 10FE54E0 B07890EF    D098964E 00082899 6A00D098 70006000
          000400    47F0A106 5880D088 D2018000 D091D201    8002D094 47F9A196 5880D088 D2058000
          000420    D09058F0 B06005EF 920FD063 58F0B92C    05EFF014 91800091 4780F03C 5820D050
          000440    12224770 F03C59DC 00104770 F03C58D0    D00450DC 99199189 D0004710 F03258D0
          000460    D00447F0 F0225020 D00898EB D00C07FE    58F0B030 97FF584C 00001244 47B0F056
          000480    587C0014 D2033050 70504140 4001504C    00005040 39549299 304C5030 D00818D3
          0004A0    583C0010 5030D004 50DC0010 5020D008    5020D060 07FEIC00 00001000 000014B8
          0004C0    000024B8 000034B8 000044B8 000054B8    000064B8 000074B8 00000000 00000000
          0004E0    00000434 00000434 00000000 89300008    00000648 41660001 000002E4 000002AC

-----------------                                        LISTING OF LOAD MODULE PL1LOAD                        PAGE 0003
       000500    00000258 00000000 00000000 00000000    00000000 00000000 00000000 00000000
          000520    00000730 00000738 00000740 00000748    80000000 00000001 0C020000 00000544
       000540    00140014 40D7D3F1 E3C3F0F2 6060C3D6    D4D7D3C5 E3C5C440 00000560 00270027
          000560    40C5D9D9 6D9D6BC5 E7D7C5C3 E3C5C440    C1C440C9 E240F4F0 4EF2F0C9 40C2E4E3
          000580    40C1C440 C9E24002 0C040C00 000005D4    002C002C 40C5D9D9 969968C5 E7D7C5C3
          0005A0    E3C5C440 C140C9E2 40F1F84E F4F1C940    C2E4E340 C140C9E2 49D9C5C1 D3D3E840
          0005C0    000C041C 018C0C2C 0CIC0000 000005D4    00120012 40D7D3F1 E3C3F9F2 6060C5D5
          0005E0    E3C5D9C5 C440000C 040C050C 000C006C    000C020C 010C001C 0009958C 0000063B
          000600    00000740 80000638 00000748 00000242    80000534 00000748 0009921C 80000534
          000620    00099748 0000016C 80000534 00000748    000000A4 80000534 8993892C 8A060089
          000640    04800620 41C90008 C08000D0 1C021AC1    95043008 47808200 D2AFC000 40009680
          000660    900647F0 8206D2AF 4000C000 1BFF50FD    00101817 41000038 0A0A98EC D00C07FE
          000680    00033BC8 00480A0A 05804860 B08050E7    00309180 90064780 89189295 701047F0
          0006A0    801C9206 70104150 A05818C6 41D00020    1CCC1AD5 50D70014 18499595 70104770
          0006C0    804048D0 900447F0 80581B22 8D200008    41100001 19128C20 00084789 809648D7
          0006E0    00224820 B07A4BD0 B0864740 807A1BCC    48I0B07E 1DC11AD2 80009908 41DCD001
          000700    47F0808A 4AD0B086 4AD0B084 06208920    00081AD2 410D0000 00000099 47F0809E
          000720    58F0F008 07FF0000 00000000 50070034    003C004C 001058F0 003C004C 58070034
          000740    003C004C D2071024 00201002 00000000    00000004 00000000 00000000 00000000
          000760    07E2E8E2 D7D9C9D5 E3000000 00000000    00000000 00000000 00000000 00000000

RECORD# 6     TYPE 02 - RLD                                  RLD SIZE 236
              R-PTR  P-PTR    FL  ADDR   FL  ADDR   FL  ADDR   FL  ADDR   FL  ADDR   FL  ADDR   FL  ADDR
                     2          1      0C  000010
                    14          1      24  00002E
                    15          1      24  00029A
                     1          1      0D  0002B4  0C 0002EC
                    12          1      25  000448  24 000454
                     3          1      24  000478
                  13          1      24  000482
                     3          1      24  000490
                  12          1      25  0004A2  24 0004AA
                     2          2      0D  0004BC  0D 0004C0  0D 0004C4  0D 0004C8  0D 0004CC  0D 0004D0
                                       0C  0004D4
                     4          2      8C  0004D8
                     5          2      8C  0004DC
                     1          2      0D  0004E0  PC 0004E4
                     2          2      0C  0004F0
                     1          2      0D  0004F8  0D 0004FC  0D 000500  0C 000504
                    16          2      9C  000508
                    17          2      9C  00050C
```

```
                 18          2      9C  000510
                 19          2      9C  000514
                 20          2      9C  0004E8
                 21          2      9C  000518
                 22          2      9C 00051C
                 23          2      0C 000520

-----------------
                                    LISTING OF LOAD MODULE PL1LOAD                       PAGE 0004
RECORD# 7     TYPE 0E - RLD                                     RLD SIZE 236
              R-PTR  P-PTR    FL   ADDR    FL   ADDR    FL   ADDR    FL   ADDR    FL   ADDR    FL   ADDR
                 24          2    0C 000524
                 25          2    0C 000528
                 26          2    0C 00052C
                    2           2    09 00053D  09 000559  09 00058D  09 0005CD  0D 0005F8  0C 0005FC
                 25          2    0C 000600
                    2           2    08 000605
                 26          2    0C 000608
                    1           2    0C 00060C
                    2           2    08 000611
                 26          2    0C 000614
                    1           2    0C 000618
                    2           2    08 00061D
           26          2    0C 000620
                    1           2    0C 000624
                    2       2    08 000629
                 26          2    0C 00062C
                    1           2    0C 000630
                    2           2    08 000635
                    1           8    0C 000718
                 10      9       8C 000728
                 27     26    24 000748

******END OF LOAD MODULE LISTING
```

"Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module" on page 571 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module.

Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module

```
****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
LISTING OF LOAD MODULE MYMOD                              PAGE 0001

RECORD# 1     TYPE 20 - CESD     ESDID 1           ESD SIZE 144
CESD#    SYMBOL  TYPE      ADDRESS  R/R/A  ID/LENGTH(DEC)    (HEX)
   1    A       00(SD)    000000   00          24           18
   2    ENTA    03(LR)    000014   00           1            1
   3    ENTB    03(LR)    000028   00           5            5
   4    Q1      06(PR)    000000   03           4            4
   5    B       00(SD)    000018   00          20           14
   6    UNRES   02(ER)    000000
   7    DANGLE  02(ER)    000000
   8    $NULL   07(NULL)  000000   00           0            0
   9    $NULL   07(NULL)  000000   00           0            0
```

"Example: Output for LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1)" on page 571 is an example of the output produced by LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1).

Example: Output for LISTLOAD OUTPUT=MODLIST for a PDSE (program object Version 1)

```
                                          LISTING OF PROGRAM OBJECT TESTPR                                  PAGE      1

THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695DF108 AT LEVEL 01.00 ON 09/16/92 AT 09:42:47
------------------------------------------------------------------------------------------------------------------------
           CONTROL SECTION:  A
AMODE: 24                 ALIGNMENT: DOUBLE WORD       LENGTH:         24 (DEC)        MODULE OFFSET:          0 (DEC)
RMODE: 24                 USABILITY: UNSPECIFIED       LENGTH:         18 (HEX)        MODULE OFFSET:          0 (HEX)
                          STORAGE:   ANY               OVERLAY SEGMENT:       0        OVERLAY REGION:       0
===== IDRL =====
          TRANSLATOR   VER   MOD     DATE      TIME
          566896201     02    01    09/16/92
===== ESDs =====
ALPHA(PR)
                ALIGNMENT: FULL WORD        LENGTH: 00000004
===== RLDs =====
SEC.OFF   MOD.OFF   TYPE     BDY    STATUS       REFERENCED SYMBOL
```

```
00000004  00000004   CPR     NONE    RES         $CUMULATIVE PSEUDO REGISTER LENGTH
00000008  00000008    PR     NONE    RES      (+)ALPHA
00000010  00000010   CPR     NONE    RES         $CUMULATIVE PSEUDO REGISTER LENGTH
00000014  00000014   CPR     NONE    RES         $CUMULATIVE PSEUDO REGISTER LENGTH
===== TEXT =====
00000000  C1C1C1C1 00000014 00000000 00000000    00000014 00000014
                                          LISTING OF PROGRAM OBJECT TESTPR                                 PAGE      2

                PSEUDO REGISTER
                 VECTOR LOC   LENGTH    NAME
                         0        4    ALPHA
                         0       10    BETA
LENGTH OF PSEUDO REGISTERS        14
**   END OF PROGRAM OBJECT LISTING
```

"Example: Output for LISTLOAD OUTPUT=MODLIST,SECTION1=YES for a program object" on page 572 is an example of the output produced by LISTLOAD OUTPUT=MODLIST,SECTION1=YES for a program object.

Example: Output for LISTLOAD OUTPUT=MODLIST,SECTION1=YES for a program object

```
          MODULE SECTION:   $MODULE
 USABILITY: UNSPECIFIED    OVERLAY SEGMENT:       0     OVERLAY REGION:       0
 ===== IDRU =====
         DATE         USER DATA
         09/28/2007   *** z/OS V1.9 ***
 ===== ESDs =====
 TESTASM(ER)
                                 TEXT TYPE:         UNSPEC       CLASS OFFSET:          10 (HEX)
     TARGET SECTION: TESTASM                                     TARGET CLASS:           B_TEXT
     NAME SPACE:           1     SCOPE:            LIBRARY       ELEMENT OFFSET:          0 (HEX)
     RESOLVED                    AUTOCALL
     ATTRIBUTES:     STRONG
 ===== MAP  =====             CLASS:            B_MAP
 00000000  00000220 D4000000 FFFFFFFF 00000000 0000001D 00000000 00000000 00000000
 00000020  000000C0 C3000006 00000039 00000000 0000001D 6A000001 00000000 00000000
 00000040  00000080 E2800008 00000031 00000000 0000000C 64000001 00000000 00000000
 00000060  00000000 D3000008 00000029 00000000 00000000 01000001 00000000 00000000
 00000080  00000000 E2800007 00000022 00000010 0000000D 64000001 00000000 00000000
 000000A0  00000000 D3000007 0000001B 00000000 00000000 01000001 00000000 00000000
 000000C0  00000140 C3600005 00000016 00000000 00000007 6A000048 00000000 00000000
 000000E0  00000100 E2800004 00000012 00000000 00000001 64000048 00000000 00000000
 00000100  00000120 E2800008 00000031 00000001 00000003 64000048 00000000 00000000
 00000120  00000000 E2800007 00000022 00000004 00000003 64000048 00000000 00000000
 00000140  000001A0 C3600006 0000000C 00000000 00000002 6A000015 00000000 00000000
 00000160  00000180 E2800008 00000031 00000000 00000001 64000015 00000000 00000000
 00000180  00000000 E2800007 00000022 00000001 00000001 64000015 00000000 00000000
 000001A0  000001E0 C3600006 00000006 00000000 00000001 6A00006A 00000000 00000000
 000001C0  00000000 E2800004 00000012 00000000 00000001 6400006A 00000000 00000000
 000001E0  00000000 C3600006 00000000 00000000 00000001 6A000056 00000000 00000000
 00000200  00000000 E2800004 00000012 00000000 00000001 64000056 00000000 00000000
 00000220  00000000 C5000000 FFFFFFFF 00000000 00000000 00000000 00000000 00000000
 ===== MAP  =====             CLASS:            B_MAP
 00000000  C26DC9C4 D9E4C26D C9C4D9C2 C26DC9C4 D9D30000 0001C26D C5E2C4E3 C5E2E3C1
 00000020  E2D4E3C5 E2E3C1E2 D4C5C4C3 D6C5E7E3 E2C5C4C3 D6C5E7E3 E2C26DE3 C5E7E3
```

"Example: Output for LISTLOAD OUTPUT=MODLIST,IMPEXP=YES for a program object" on page 572 is an example of the output produced by LISTLOAD OUTPUT=MODLIST,IMPEXP=YES for a program object.

Example: Output for LISTLOAD OUTPUT=MODLIST,IMPEXP=YES for a program object

```
==== IEWBCIET VERS 02 ====  CLASS:         B_IMPEXP
EXPORTED FUNCTIONS  CLASS           +OFF/ADDR ADA_CLASS       +OFF/ADDR LN REF_CLASS       +OFF/ADDR ATTRIBUTES
   Failure           C_CODE           000000A8 C_WSA           +00000010                              X
   Traverse          C_CODE           00000140 C_WSA           +00000010                              X
EXPORTED VARIABLES  CLASS           +OFF/ADDR
   blks              C_WSA           +0000007C
   bytes             C_WSA           +0000008C
   chrs              C_WSA           +00000084
   dirs              C_WSA           +00000078
   pipes             C_WSA           +00000080
   regs              C_WSA           +00000074
   syms              C_WSA           +00000088
IMPORTED FUNCTIONS  DLL CELHV003        INDX                      LN REF_CLASS        +OFF/ADDR ATTRIBUTES
  fprintf                               006D                      04 C_WSA            +00000038 XNU
   printf                               006F                      04 C_WSA            +00000010 XNU
  strerror                              00A8                      04 C_WSA            +00000030 XNU
   __errno                              0156                      04 C_WSA            +00000018 XNU
   _exit                                0174                      04 C_WSA            +00000020 XNU
   closedir                             017F                      04 C_WSA            +00000050 XNU
```

```
    lstat                                01A7                      04 C_WSA            +00000048 XNU
    opendir                              01AD                      04 C_WSA            +00000028 XNU
    readdir                              01B3                      04 C_WSA            +00000040 XNU
```

Description of MODLIST output for a program object

The listing produced by LISTLOAD OUTPUT=MODLIST consists of multiple parts (see "Example: Output for LISTLOAD OUTPUT=MODLIST,ADATA=YES for a program object" on page 564):

A page heading, displayed at the top of each page.

The page heading consists of one or two heading lines, in the following format:

```
      LISTING OF PROGRAM OBJECT xxxxxxxx
```

The heading lines are followed by the title line, entered in the TITLE parameter of the LISTLOAD control statement.

The binder-generated program object identification record (IDRB).

The IDRB record is displayed on a line by itself, in the format:

```
   THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.12
   ON mm/dd/yyyy AT hh:mm:ss
```

The binder program identifier, version and level, date and time of binding are presented here. The IDRB line is followed by a line of dashes.

An individual listing for each control section in the program object, separated by a dashed line.

For each control section in the module, the ESD Section Definition (SD) record is formatted, followed by all data classes in the following sequence:

1. IDRZ - SPZAP identification data
2. IDRL - Language translator identification data
3. IDRU - User-supplied identification data
4. SYM - internal symbol dictionary
5. ESD - External Symbol Dictionary
6. RLD - ReLocation Dictionary
7. TEXT - Instructions and data for the CSECT
8. ADATA - ADATA information

The SD record occupies two print lines:

- The first begins with one of the constants CONTROL SECTION, SEGMENT TABLE, ENTRY TABLE, or MODULE SECTION, and displays either the section or common name. If there is no user-defined name, then a binder-generated name will be displayed as follows:
  - $PRIV*xxxxx*, where *x* is a number for user sections which originally had blank names or were unnamed.
  - $BLANKCOM - unnamed common
  - $MODULE - binder-generated section containing module level information, and is only output when SECTION1=YES
  - $SUMMARY - binder-generated section containing merge classes for the module
- The second line displays the USABILITY, AMODE, the overlay segment and region. USABILITY must contain one of the values UNSPECIFIED, NON-REUSABLE, REUSABLE, REENTRANT or REFRESHABLE. For non-overlay modules, the latter two fields will contain zero.

Each of the eight class subsections begin with an identifier line of the format:

```
===== class name =====
```

IDR detail is in the same format as described in "Description of LISTIDR output" on page 593, except that it is displayed only for the single section. The remainder of the classes are described below:

- SYM data is displayed 40 bytes per line
- ESD data occupies three to four lines per ESD record. The first containing the external name (abbreviated name, if name longer than 16 bytes), followed by the ESD record type in parenthesis. The rest of the formatted fields vary depending on the ESD record type:
  - ED records define an element definition. Its length, and various attributes will be used to bind and load the class contained in the section. Each ED record occupies three print lines:
    1. The first line displays:
       - CLASS name - up to 16 bytes.
       - LENGTH of defined class element in hexadecimal.
       - CLASS OFFSET - in hexadecimal.
       - FORMAT - where the first field is the class record format with either F (fixed length record), or V (variable length record), follows by the hexadecimal value of the record format in parenthesis.
    2. The second line shows:
       - NAME SPACE - in hexadecimal.
       - ALIGNMENT - any of the following:
         - DOUBLE WORD
         - QUAD WORD
         - 32 BYTE
         - 64 BYTE
         - 128 BYTE
         - 256 BYTE
         - 512 BYTE
         - 1024 BYTE
         - 2048 BYTE
         - PAGE

         DOUBLE WORD, QUAD WORD, or PAGE.
       - BIND METHOD - CATENATE or MERGE.
       - RMODE - 24, 64, ANY, or UNSPECIFIED
    3. The third line displays the binder and loader attributes:
       - Binder attributes can be DESCRIPTIVE DATA, TEXT, or REMOVABLE
       - Loader attributes can be NOLOAD, LOAD, DEFER, READ-ONLY (or FILL: UNSPEC is printed, if there is no fill character).
  - ER records define external references from the named section. For the $MODULE section these are external references with no corresponding RLDs. Each ER record occupies four print lines, where:
    1. The first line displays:
       - TEXT TYPE - can be either UNSPEC (unspecified), INSTRUC (instructions or code), DATA, or TRANS.DEF (translator defined).
       - CLASS OFFSET - in hexadecimal.
    2. The second line shows:
       - TARGET SECTION - Target section name (abbreviated name, if name is longer than 16 bytes).

    - TARGET CLASS - Name of class containing target label.
3. The third line shows:
    - NAME SPACE - in hexadecimal.
    - SCOPE - Scope of name (SECTION, MODULE, LIBRARY, or IMP/EXP).
    - ELEMENT OFFSET - in hexadecimal.
    - AMODE - 24, 31, 64, ANY, MIN or UNSPEC
4. The fourth line displays the ER status and autocall, where:
    - ER status can either be RESOLVED or UNRESOLVED
    - AUTOCALL can either be AUTOCALL or NEVERCALL
5. The fifth line displays binder attributes. Possible attributes are XPL, STRONG or WEAK, MAPPED, INDIRECT, GENERATED, or MANGLED.
6. If extended attributes exist, a sixth line displays the location of these attributes as the class and offset within that class.

– LD records define a label or entry point in the named section. Each LD record occupies three print lines, where:

1. The first line displays:
    - CLASS NAME - up to 16 bytes.
    - TEXT TYPE - can be either UNSPEC, INSTR, DATA, or TRANS.DEF
    - CLASS OFFSET - in hexadecimal.
2. The second line shows:
    - NAME SPACE - in hexadecimal.
    - SCOPE - Scope of name (SECTION, MODULE, LIBRARY, or IMP/EXP).
    - ELEMENT OFFSET - in hexadecimal.
    - AMODE - can either be 24, 31, 64, MIN, ANY, or UNSPECIFIED
3. The third line displays the binder attributes. Possible attributes are XPL (xplink), STRONG or WEAK, MAPPED, INDIRECT, GENERATED (the LD record was generated by the binder), or MANGLED. In addition, if the record contains the name of the symbol which defines the environment or associated data (ADA), that symbol will be printed.
4. If extended attributes exist, a fourth line will contain the resident class and offset.

– PD (Part Definition) and PR (Part Reference) records define parts or pseudo registers. The PR record is a local definition of the part (within the section), whereas the PD record is a global definition for all of the associated PRs (PRs with the same name). PR and PD records contain the same formatted fields. Each record occupies three print lines, where:

1. The first line displays:
    - CLASS name - up to 16 bytes.
    - LENGTH - in hexadecimal.
    - CLASS OFFSET - in hexadecimal.
2. The second line shows:
    - NAME SPACE - in hexadecimal.
    - ALIGNMENT - any of: the following:
        - BYTE
        - HALF WORD
        - FULL WORD
        - DOUBLE WORD

        - QUAD WORD
        - 32 BYTE
        - 64 BYTE
        - 128 BYTE
        - 256 BYTE
        - 512 BYTE
        - 1024 BYTE
        - 2048 BYTE
        - PAGE
      - PRIORITY - Controls the order of the part within the element.
      - SCOPE - Scope of part (SECTION, MODULE, LIBRARY, or IMP/EXP).
   3. The third line displays binder attributes. Possible attributes are: XPL, STRONG or WEAK, MAPPED, INDIRECT, GENERATED, or MANGLED.

- RLD data is displayed one line per RLD record, by element offset of the associated address constant. Where multiple RLD records refer to the same adcon, the element offsets will be the same. RLD data shown consists of:
  - Element Offset - The offset, in hex, of the associated address constant within the element.
  - Class Offset - The offset, in hex, of the associated address constant within the class.
  - ADCON TYPE - The type of address constant associated with this RLD entry. The type may be:
    - BR - Branch or V-type
    - N_BR - Non-branch or A-type
    - SEGM - address of start of class
    - C OF - Q-type, offset from start of class
    - CPR - total length of the class
    - LTKN - loader token, for use of the system loader
    - R-IM - reference to an external symbol from a relative immediate instruction
    - DATA - data associated with the target symbol
    - LDIS - QY-type, reference to an external symbol from a long displacement instruction
    - L-PR - length of the individual PR
  - Status - This field identifies the status of the associated address constant. Valid status values are: RES (resolved), UNRES (unresolved), and N-REL (non-relocatable constant).
  - LENGTH - The adcon length in hexadecimal.
  - ATTRIBUTES, which may be:
    - HOBCHG - High order bit of V-type adcon was changed by the binder. Possible value can be either YES or NO.
    - X - XPLINK linkage
    - S - part of a conditional sequential RLD string, will be followed by another RLD describing an adcon at the same location.
  - NAME SPACE of reference in hexadecimal.
  - TARGET NAME - Name of the referenced symbol.
  - PARTRES - If the RLD describes an adcon on a part (PR), this will be the name of the resident part.
  - XATTR NAME - Symbol defining the location at which the extended attributes (if any) are stored.
  - XATTR OFF - Offset from symbol XATTR NAME at which the extended attributes are stored.
- TEXT data is displayed by class name. In addition to the hexadecimal representation, text data is in EBCDIC format.

For the $MODULE section, instead of TEXT, it is identified as MAP and shows the binder-generated class B_MAP information.

For the IEWBCIE section, when IMPEXP=SYMBOLS, instead of TEXT, it is identifed as IEWBCIET VERS nn (where nn is the import/export table version number) and shows information about the imported and exported symbols. Columns with the heading +OFF/ADDR distinguish offsets from addresses by preceding offsets with a plus sign (+). They are always preceded by their corresponding class. However, when an address is shown, it is relative to the module beginning at a zero origin, not relative to the beginning of the class. To determine the address within the class (as shown in the NUMERICAL MAP), subtract off the location of the class from the address (the location of the class is available in the SEGMENT MAP TABLE). The attributes shown as follows:

O or X
referred symbol is OS linkage or XPLINK linkage

N or P
reference is by name or reference is by pointer

R or U
reference is Resolved or Unresolved

- ADATA information, if requested through ADATA=YES on the LISTLOAD OUTPUT=MODLIST control card, like TEXT data is displayed by class name in both hexadecimal and EBCDIC presentation. In the EBCDIC presentation, non-printable characters are replaced with periods (.).

  The abbreviation-to-long name equivalence table is displayed prior to end of the listing, with all the abbreviated names (external names exceeding 16 bytes in length) in the formatted part of the listing with their long names.

A trailer record.

```
** END OF PROGRAM OBJECT LISTING
```

Description of MODLIST Output for a Load module/PDS

The listing produced by LISTLOAD OUTPUT=MODLIST consists of multiple parts (see “Example: Output for LISTLOAD OUTPUT=MODLIST for an overlay structured load module” on page 569 and “Example: Output for LISTLOAD OUTPUT=MODLIST for a normal (non-overlay) structured load module” on page 571).

A page heading, displayed at the top of each page.
The page heading consists of one or two heading lines, in the following format:

```
     LISTING OF LOAD MODULE PL1LOAD
```

An individual listing for each record in the load module. Each record on DASD is identified by a sequence number (RECORD#), and type. Often there is size information for the record as well. This information is followed by its contents formatted according to the record type.

For more information on the details of the following fields, see *z/OS MVS Program Management: Advanced Facilities*, Appendix B. Load Module formats.

TYPE 20 - CESD
This is a record containing definitions or uses of external symbols.

- CESD# - An internal number for the symbol assigned by the binder.
- SYMBOL - The name of the symbol as used in the program, or an abbreviation of it.
- TYPE - See chapter 2 of *z/OS MVS Program Management: User's Guide and Reference* for a description of the various types of external symbol dictionary entries.
  - ER - External Reference
  - WX -Weak External reference
  - LR - Label Reference

  - SD - Section Definition, a control section (CSECT)
  - PC - Private Code
  - CM - Common area
  - PR - Pseudoregister
- ADDRESS - Offset within the bound program where the symbol is defined.
- SEGNUM - Overlay structured modules only: the overlay segment number.
- R/R/A - Non-overlay modules only: A bit-coded byte, displayed in hex, indicating AMODE, RMODE, and read- only attributes. The values of the three attributes are ORed together in the value displayed.
  - 08 - Read-only
  - 04 - RMODE=ANY (RMODE=24 if off)
  - 01 - AMODE=24
  - 02 - AMODE=31
  - 03 - AMODE=ANY
  - 10 - AMODE=64
- ID/LENGTH - ESD number of referenced symbol for an LR, length of the section or field for SD, PC, CM, or PR
  - (DEC) - Length or ID displayed in decimal
  - (HEX) - Length or ID displayed in hexadecimal

TYPE 0 - RLD (May be 02, 06, or 0E)

In the RLD record type, AMBLIST outputs information in the following columns:

- R-PTR - The ESDID of the target element
- P-PTR - The ESDID of the element containing the adcon or data area to be modified
- FL - flags
- ADDR - address

TYPE 0 - CONTROL (May be 01, 05, or 0D)

The CONTROL record lists information on each of the control sections for the load module:

- CESD# - CESD number of the control section
- LENGTH - length of that control section

TEXT

This record type contains the program as loaded for execution. Text data is displayed by class name. In addition to the hexadecimal representation, text data is in EBCDIC format.

A trailer record.

`******END OF LOAD MODULE LISTING`

LISTLOAD OUTPUT=XREF output

This section includes examples of LISTLOAD OUTPUT=XREF output as well as cross-reference listings, such as:

- “Example: Output for LISTLOAD OUTPUT=XREF for a program object with class names: B_PRV and B_TEXT” on page 579 shows the output from a program object version 2 that contains a single initial load text class. See the descriptions following this figure for explanations.
- Figure 196 on page 584 shows a sample segment map table for a multiple-text class module.
- “Example: Output for LISTLOAD OUTPUT=XREF for a load module” on page 584 and “Example: Output for LISTLOAD OUTPUT=XREF for a program object” on page 585 allow you to compare the output for a load module with the output for a program object version 1.

The listing has the following parts:

- Segment map, which only appears for Program Objects and not for Load Modules.
- Numerical map, which presents information approximately in the order in which it appears in the module.
- Numerical cross-reference.
- Alphabetical map, which presents information alphabetically by symbol name.
- Alphabetical cross-reference.

In z/OS V1R10, the format of the LISTLOAD numerical and alphabetical cross-references for program objects was significantly changed. The information is now presented in a more concise format that more closely resembles the format produced by the binder.

In the listing shown in "Example: Output for LISTLOAD OUTPUT=XREF for a program object with class names: B_PRV and B_TEXT" on page 579, page 1 shows the numerical map of the module. Page 2 shows the numerical cross-reference list of the module. Page 4 shows the alphabetical map, and page 5 the alphabetical cross-reference list.

Note: The module shown in the "Example: Output for LISTLOAD OUTPUT=XREF for a program object with class names: B_PRV and B_TEXT" on page 579 is not in overlay format; for overlay modules, each segment is formatted separately.

As with the other output from AMBLIST, each page begins with a standard heading. The first line of each page contains a page number and begins with one of the following heading constants:

- SEGMENT MAP OF PROGRAM OBJECT ....
- NUMERICAL MAP OF PROGRAM OBJECT ....
- NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT ....
- ALPHABETICAL MAP OF PROGRAM OBJECT ....
- ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT ....

The member name will appear following the heading. If the name is more than sixteen characters, its formatted 16-bytes abbreviation name is printed instead. An optional second line will be used to print the title information, provided by the user on the LISTLOAD control statement. Each of the four parts has its own subheading line(s), to describe the detail that follows.

Example: Output for LISTLOAD OUTPUT=XREF for a program object with class names: B_PRV and B_TEXT

```
                              ** SEGMENT MAP TABLE **                                                PAGE      1

CLASS             SEGMENT    OFFSET   LENGTH       LOAD      TYPE    ALIGNMENT    RMODE
B_TEXT                  1         0     2030       INITIAL   CAT     PAGE         31
 A                           ** NUMERICAL MAP OF PROGRAM OBJECT LOADMOD1        **                   PAGE      2

------------------------------------------------------------------------------------------------------------------------
      CLAS LOC   ELEM LOC    LENGTH  TYPE  RMODE    ALIGNMENT            NAME
        0                       430   ED   21       PAGE                 SD1
      430                       238   ED   21       DOUBLE WORD          SD2
      668                         8   ED   21       DOUBLE WORD          SDX
           668          0             LD   21                              LD1
           66C          4             LD   31                              LD2
     1000                        30   ED   31       PAGE                 $BLANKCOM
     2000                        30   ED   31       PAGE                 CM1
      CLASS LENGTH             2030
------------------------------------------------------------------------------------------------------------------------
RESIDENT CLASS:             B_PRV
      CLAS LOC   ELEM LOC    LENGTH  TYPE    ALIGNMENT             NAME
        0                         0   ED     PAGE                  SD1
        0                         0   ED     DOUBLE WORD           SD2
        0                         0   ED     DOUBLE WORD           SDX
        0                         0   ED     PAGE                  $BLANKCOM
        0                         0   ED     PAGE                  CM1
        0                        18   ED     DOUBLE WORD           $PRIV000003
             0                    1   PD     BYTE                    A
             1                    1   PD     BYTE                    E
             2                    1   PD     BYTE                    C
             3                    1   PD     BYTE                    G
             4                    2   PD     HALF WORD               F
             8                    4   PD     FULL WORD               D
```

```
           10                     8   PD     DOUBLE WORD             B
      CLASS LENGTH               18
-----------------------------------------------------------------------------------------------------------------------
 B                              ** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT LOADMOD1         **       PAGE      3

CLAS LOC ELEM LOC FROM CLASS    FROM SECTION  1   REFERS TO SYMBOL CLAS LOC ELEM LOC IN CLASS  2     IN SECTION  3
      48       48 B_TEXT           SD1             SD1                      0        0 B_TEXT
      C6       C6 B_TEXT           SD1             A                $CLASS_OFFSET     B_PRV
     12E      12E B_TEXT           SD1             B                $CLASS_OFFSET     B_PRV
     196      196 B_TEXT           SD1             C                $CLASS_OFFSET     B_PRV
     1FE      1FE B_TEXT           SD1             D                $CLASS_OFFSET     B_PRV
     266      266 B_TEXT           SD1             E                $CLASS_OFFSET     B_PRV
     2CE      2CE B_TEXT           SD1             F                $CLASS_OFFSET     B_PRV
     336      336 B_TEXT           SD1             G                $CLASS_OFFSET     B_PRV
     39C      39C B_TEXT           SD1             $CLASS_LEN       $CLASS_LEN        B_PRV
     478       48 B_TEXT           SD2             SD2                    430        0 B_TEXT
     4F4       C4 B_TEXT           SD2             LD2                    66C        4 B_TEXT          SDX
     554      124 B_TEXT           SD2             CM1                   2000        0 B_TEXT
     604      1D4 B_TEXT           SD2             CM1                   2000        0 B_TEXT
      CLASS LENGTH             2030
-----------------------------------------------------------------------------------------------------------------------
                                ** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT LOADMOD1         **       PAGE      4

RESIDENT CLASS:             B_PRV
     ****  NO RLD DATA ****
      CLASS LENGTH               18
LENGTH OF PROGRAM OBJECT     2030
-----------------------------------------------------------------------------------------------------------------------
 C                              ** ALPHABETICAL MAP OF PROGRAM OBJECT LOADMOD1         **                     PAGE      5

ENTRY NAME  CLAS LOC  ELEM LEN/LOC             CLASS NAME       SECTION NAME OR ENTRY TYPE
$PRIV000003
                  0        18                      B_PRV       (ED)
$BLANKCOM
                  0         0                      B_PRV       (ED)
$BLANKCOM
               1000        30                     B_TEXT       (ED)
A
                  0         1                      B_PRV       (PD)
B
                 10         8                      B_PRV       (PD)
C
                  2         1                      B_PRV       (PD)
CM1
               2000        30                     B_TEXT       (ED)
CM1
                  0         0                      B_PRV       (ED)
D
                  8         4                      B_PRV       (PD)
E
                  1         1                      B_PRV       (PD)
F
                  4         2                      B_PRV       (PD)
G
                  3         1                      B_PRV       (PD)
LD1
                668             0                 B_TEXT       SDX
LD2
                66C             4                 B_TEXT       SDX
SDX
                668         8                     B_TEXT       (ED)
SDX
                  0         0                      B_PRV       (ED)
SD1
                  0       430                     B_TEXT       (ED)
SD1
                  0         0                      B_PRV       (ED)
SD2
                430       238                     B_TEXT       (ED)
SD2
                  0         0                      B_PRV       (ED)
-----------------------------------------------------------------------------------------------------------------------
 D                              ** ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT LOADMOD1         **    PAGE      6

SYMBOL REFERRED  CLAS LOC ELEM LOC IN CLASS         IN SECTION       CLAS LOC ELEM LOC FROM CLASS       FROM SECTION
$CLASS_LEN       $CLASS_LEN        B_PRV                                  39C      39C B_TEXT           SD1
A                $CLASS_OFFSET     B_PRV                                   C6       C6 B_TEXT           SD1
B                $CLASS_OFFSET     B_PRV                                  12E      12E B_TEXT           SD1
C                $CLASS_OFFSET     B_PRV                                  196      196 B_TEXT           SD1
CM1                  2000        0 B_TEXT                                 554      124 B_TEXT           SD2
CM1                  2000        0 B_TEXT                                 604      1D4 B_TEXT           SD2
D                $CLASS_OFFSET     B_PRV                                  1FE      1FE B_TEXT           SD1
E                $CLASS_OFFSET     B_PRV                                  266      266 B_TEXT           SD1
F                $CLASS_OFFSET     B_PRV                                  2CE      2CE B_TEXT           SD1
G                $CLASS_OFFSET     B_PRV                                  336      336 B_TEXT           SD1
LD2                   66C        4 B_TEXT           SDX                   4F4       C4 B_TEXT           SD2
SD1                     0        0 B_TEXT                                  48       48 B_TEXT           SD1
SD2                   430        0 B_TEXT                                 478       48 B_TEXT           SD2
LENGTH OF PROGRAM OBJECT     2030
-----------------------------------------------------------------------------------------------------------------------
```

Segment map

When the binder produces a program object executable, it is composed of one or more segments. The binder will always arrange the program so that segment 1 contains the entry point. To understand the importance of the information contained in the Segment Map Table, see Understanding binder programming concepts in *z/OS MVS Program Management: Advanced Facilities*, in particular Multiple-text class modules, load segments and ESD offsets. For an example of a Segment map for a multiple-text class module, see "Sample segment map table for LISTLOAD OUTPUT=XREF of mutiple-text class module" on page 584.

Each of the columns in a Segment map table is described as follows:

CLASS
The class name of this part of the segment. The segment will be composed of one or more classes. Note that a section may contribute to more than one class.

SEGMENT
The segment in which this class resides. A given class will reside in only one segment, and all classes in the same segment will have the same attributes.

OFFSET
The offset the beginning of the named class from the beginning of the segment. Note that every segment has it's own origin (offset 0), because each may be independently loaded.

LENGTH
The length of the named class in the segment.

LOAD
The loading time attribute of the class, which will be the same for all classes with the same segment number. The possible values are:

INITIAL
An initial load class which is loaded when the program entry point or alias is loaded.

DEFER
A deferred load class which is not loaded when the program entry point or alias is loaded.

TYPE
The binding type of class. The possible values are:

CAT
A concatenated class. This is the usual way in which a section contributes to a class (known as elements). The elements are arranged one after the other.

MERGE
A "merge" class. This type of class is used for parts, which are for data, where one or more part comprises an element.

ALIGNMENT
The alignment of the class or elements contained in the class. The possible values are:

- BYTE
- HALF WORD
- FULL WORD
- DOUBLE WORD
- QUAD WORD
- 32 BYTE
- 64 BYTE
- 128 BYTE
- 256 BYTE
- 512 BYTE
- 1024 BYTE

- 2048 BYTE
- PAGE

RMODE
The residency mode of the class and the segment. If only one value is listed, it is the residency mode of both the class and segment. If a second value is given, the first value is the residency mode of the class and the second value is the residency mode of the segment (for example, 24,31 would indicate that a class with the attribute RMODE=24 is part of a segment that has RMODE=31). The segment is loaded according to the residency mode of the segment. The possible values are: **24**, **31**, and **64**.

Numerical map

The A *Numerical Map* prints one line for each defined element definition, part, or control section in the composite ESD. The detail line contains the class offset (in hex), either a section offset for (labels/parts) or a length (for control sections element definitions), the ESD record type, alignment (for LD/PD record type), and the label/part, section or element definition name. Sections generated by the binder, or binder-generated names for unnamed user sections, will be displayed as:

- $PRIV*xxxxxx* - where *xxxxxx* is numeric.
- $BLANKCOM
- $SEGTAB
- $ENTAB

All other entries will contain a valid name, assigned by the user for a label/part, a control section/element definition or named common. For label (LD) or part (PD) type ESD entries, the class offset and label/part name will be indented to show that the label/part is contained within the previous section entry.

If the module is in overlay format, the map and cross-reference will alternate for each segment. In this case, the map will begin with a segment identifier line:

```
SEG. nnnnn -------------------------------------------------
```

and will end with a segment length line:

```
LENGTH OF SEGMENT  nnnnn
```

The numerical map is functionally equivalent to the load map produced by the linkage editor or binder.

Numerical cross-reference

The B numerical cross-reference listing contains one entry for each RLD record in the module, presented in sequence by the hexadecimal class offset of the related address constant. There is one RLD record for each A-, V-, Q-, and J-type address constant in the module, and one RLD record for each class reference (RLD type=21), class length (CXD), or loader token.

Each entry consists of one line that is described as follows:

1 The first part of the line describes the adcon itself, showing the class and element offsets, in hex, and the name of the section (FROM SECTION) containing the adcon. Because all adcons must reside within a section, there will always be either a user-defined section name or a representation of the binder-generated name, such as $PRIV000001 or $BLANKCOM.

2 The second part of the line describes the referenced, or target, symbol. It contains the name of the referenced symbol (REFERS TO SYMBOL), the class and element offsets of the referenced label/part or section/element definition, and the class name of the referenced class name, if the reference is resolved, or one of the following constants:

- $UNRESOLVED - A strong reference (ER) could not be resolved during binding.
- $UNRESOLVED(W) - A weak reference (WX) could not be resolved during binding.
- $NEVER CALL - The symbol was marked never call, and no attempt was made during binding to resolve the symbol from the library.

- $CLASS-OFFSET - The reference was to a class offset (Q-con).
- $CLASS-LEN - The reference was to a class length (RLD type=40).

If the RLD item is for a class offset or class length, the constant string $CLASS_OFFSET or $CLASS_LEN will appear in place of a name.

3 The third part of the line will be blank except for resolved A-type and V-type address constants and displays the containing section name (IN SECTION). If the target section does not have a name, then a representation of the binder-generated name ($PRIV000005, $BLANKCOM) will be printed. If the target name in the second part of the line matches the containing section name, the containing section name will not be printed, since it would provide no additional information.

The last, or only, segment cross-reference will be followed by the length of the program object:

```
LENGTH OF PROGRAM OBJECT nnnn
```

If no RLD available in a class, the following message will appear instead of the formatted detail:

```
**** NO ADCONS IN THIS CLASS ****
```

Alphabetical map

The **C** alphabetical map displays label definitions, part definitions, control sections and element definitions (except ER and PR) in alphabetical sequence, two print lines per ESD entry. It contains all of the same information as the Numerical Map, but in a different sequence. This part always begins on a new page, with a standard page heading of ALPHABETICAL MAP OF PROGRAM OBJECT .....

The first detail line contains the label, section or common name.

The second detail line consists of the class offset, the element offset (type LD/PD records) or element definition length (all other types), the class name (name of the containing class), and the name of the containing section/element (type LD/PD records) or the ESD entry type (all other types). Element lengths are indented, to distinguish them from element offsets. If the module is in overlay format, the segment number is printed to the right of the section length.

Note: There is no preset order in which entries with identical names are output.

Alphabetical cross-reference

The **D** alphabetical cross-reference listing provides the same information as the numerical cross-reference listing, but in a different sequence. This part of the report is in collating sequence by referenced name (the name of the symbol being referred to in the address constant).

The alphabetical cross-reference begins on a new page with a standard page heading ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT, and like the numerical cross-reference, contains nine columns:

1. Class offset. This is the hex offset of the named item within a class of the program object. Class offsets for the second and third detail lines have been indented.
2. Element offset. This is the hex offset of the named label or part within its section. Lines referring to an element, rather than a label, will always display zero for the element offset.
3. Overlay segment. This is displayed for overlay format modules only.
4. Symbol. This field varies between the three detail lines, as described in the following text. If the displayed name is a special section name, then one of the binder-generated names for example, $PRIV*xxxxxx*), described earlier, will replace the name.

If no RLD is available in a program object, the following message will appear instead of the formatted detail:

```
**** NO RLD DATA ***
```

The cross reference listing concludes with the line

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

Sample segment map table for LISTLOAD OUTPUT=XREF of mutiple-text class module

Figure 196 on page 584 is an example of a segment map for a multiple-text class module.

```
                      ** SEGMENT MAP TABLE **
      CLASS          SEGMENT  OFFSET   LENGTH     LOAD      TYPE    ALIGNMENT    RMODE
      C_CODE            1          0      160     INITIAL   CAT     DOUBLE WORD  31
      B_TEXT            1        160      99A     INITIAL   CAT     DOUBLE WORD  31
  C_EXTNATTR            1        B00       28     INITIAL   CAT     DOUBLE WORD  31
      C_DATA            1        B28       18     INITIAL   CAT     DOUBLE WORD  31
    C_@@PPA2            1        B40        8     INITIAL   MERGE   DOUBLE WORD  31
       B_LIT            1        B48      140     INITIAL   CAT     DOUBLE WORD  31
    B_IMPEXP            1        C88       98     INITIAL   CAT     DOUBLE WORD  31
       C_WSA            2          0       24     DEFER     MERGE   DOUBLE WORD  31
```

Figure 196. Example: Segment map table for LISTLOAD OUTPUT=XREF of multiple-text class module

LISTLOAD OUTPUT=MAP

This section produces output identical to the SEGMENT MAP TABLE and NUMERICAL MAP sections of the LISTLOAD OUTPUT=XREF Output. See "Segment map" on page 581 and "Numerical map" on page 582 for more information.

LISTLOAD OUTPUT=XREF output (comparison of load module and program object version 1)

"Example: Output for LISTLOAD OUTPUT=XREF for a load module" on page 584 shows an example of output produced by LISTLOAD OUTPUT=XREF for a load module.

Example: Output for LISTLOAD OUTPUT=XREF for a load module

```
    LISTLOAD OUTPUT=XREF,DDN=DD1,
        MEMBER=MAINRTN
                                            *****  ATTRIBUTES OF MODULE  *****
    MEMBER NAME:  MAINRTN                                                                  MAIN ENTRY
POINT:    00000000
    LIBRARY:      DD1                                                                      AMODE OF MAIN
ENTRY POINT: 31
        NO ALIASES **
---------------------------------------------------------------------------------------------------------
---------------
                     ****   LINKAGE EDITOR ATTRIBUTES OF MODULE   ****
                **   BIT  STATUS         BIT  STATUS         BIT  STATUS         BIT  STATUS   **
                      0  NOT-RENT          1  NOT-REUS         2  NOT-OVLY         3  NOT-TEST
                      4  NOT-OL            5  BLOCK            6  NOT-EXEC         7  MULTI-RCD
                      8  NOT-DC            9  ZERO-ORG        10  EP-ZERO         11  RLD
                     12  EDIT             13  NO-SYMS         14  F-LEVEL         15  NOT-REFR
---------------------------------------------------------------------------------------------------------
---------------
                                    MODULE SSI:  NONE
                                    APFCODE:     00000000
                                    RMODE:       ANY
                                    LONGPARM:    NO
                   *****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
                             NUMERICAL MAP AND CROSS-REFERENCE LIST OF LOAD MODULE
MAINRTN                     PAGE 0001

                 CONTROL SECTION                                          ENTRY
                  LMOD LOC     NAME      LENGTH  TYPE                      LMOD LOC  CSECT LOC      NAME
                      00    MAINRTN        166    SD
                     168   $PRIVATE         BC    PC
                                                                               1D4         6C      NONAME1M
                     228   $PRIVATE         BC    PC
                                                                               294         6C      NONAME2M
```

```
                      2E8    SUBRTN         102   SD
                      3F0    AAAAAAAA        54   CM
                      448   $BLANKCOM        54   CM
---------------------------------------------------------------------------------------------------------
---------------
      LMOD LOC  CSECT LOC    IN CSECT             REFERS TO SYMBOL  AT LMOD LOC  CSECT LOC    IN CSECT
           AC        AC      MAINRTN                  AAAAAAAA           3F0         00      AAAAAAAA
           B0        B0      MAINRTN                  NONAME1M           1D4         6C     $PRIVATE
           B4        B4      MAINRTN                  NONAME2M           294         6C     $PRIVATE
          160       160      MAINRTN                  SUBRTN             2E8         00      SUBRTN
          394        AC      SUBRTN                  $BLANKCOM           448         00     $BLANKCOM
LENGTH OF LOAD MODULE     4A0
                                        ALPHABETICAL MAP OF LOAD MODULE
MAINRTN                                PAGE 0002

           CONTROL SECTION                                       ENTRY
               NAME     LMOD LOC   LENGTH  TYPE                     NAME    LMOD LOC  CSECT LOC  CSECT
NAME
            $BLANKCOM       448        54   CM
            $PRIVATE        168        BC   PC
            $PRIVATE        228        BC   PC
             AAAAAAAA       3F0        54   CM
             MAINRTN         00       166   SD
                                                                   NONAME1M       1D4      6C
$PRIVATE
                                                                   NONAME2M       294      6C
$PRIVATE
             SUBRTN         2E8       102   SD
                               ALPHABETICAL CROSS-REFERENCE LIST OF LOAD MODULE
MAINRTN                        PAGE 0003

  SYMBOL   AT LMOD LOC  CSECT LOC    IN CSECT             IS REFERRED TO BY LMOD LOC  CSECT LOC    IN
CSECT
$BLANKCOM        448         00     $BLANKCOM                          394              AC
SUBRTN
 AAAAAAAA        3F0         00      AAAAAAAA                           AC              AC
MAINRTN
 NONAME1M        1D4         6C     $PRIVATE                            B0              B0
MAINRTN
 NONAME2M        294         6C     $PRIVATE                            B4              B4
MAINRTN
 SUBRTN          2E8         00      SUBRTN                            160             160
MAINRTN
**    END OF MAP AND CROSS-REFERENCE LISTING
```

"Example: Output for LISTLOAD OUTPUT=XREF for a program object" on page 585 shows an example of output produced by LISTLOAD OUTPUT=XREF for a program object.

Example: Output for LISTLOAD OUTPUT=XREF for a program object

```
     LISTLOAD OUTPUT=XREF,DDN=DD1,

MEMBER=(MAINRTN,

THISISALONGALIASNAMEYOUMAYCHANGETHENAMEIFYOULIKEANYNAMEWILLDOOOO),
         TITLE=('XREF LISTINGS OF A LONG ALIAS
NAME',10)
                                 *****  M O D U L E   S U M M A R Y  *****
    MEMBER NAME:  MAINRTN                                                             MAIN ENTRY POINT:    00000000
    LIBRARY:      DD1                                                                 AMODE OF MAIN ENTRY POINT: 31
        ** ALIASES **      ENTRY POINT    AMODE
           THISISALO-LDOOOO  00000000
31
-----------------------------------------------------------------------------------------------------------------
-
                       ****          ATTRIBUTES OF MODULE           ****
                **   BIT  STATUS        BIT  STATUS        BIT  STATUS        BIT  STATUS   **
                       0  NOT-RENT        1  NOT-REUS        2  NOT-OVLY        3  NOT-
TEST
                       4  NOT-OL          5  BLOCK           6  EXEC            7  MULTI-
RCD
                       8  NOT-DC          9  ZERO-ORG       10  RESERVED       11
RLD
                      12  EDIT           13  NO-SYMS        14  RESERVED       15  NOT-
REFR
                      16  RESERVED       17  <16M           18  NOT-PL         19  NO-
SSI
                      20  APF            21  PGM OBJ        22  NOT-SIGN       23
RESERVED
                      24  NOT-ALTP       25  RESERVED       26  RESERVED       27
RMODEANY
                      28  RESERVED       29  RESERVED       30  RESERVED       31
RESERVED
```

```
                       32 NON-MIGR         33 NO-PRIME         34 NO-PACK          35
RESERVED
                       36 RESERVED         37 RESERVED         38 RESERVED         39
RESERVED
--------------------------------------------------------------------------------------------------------------------------
-                                    MODULE SSI:
NONE
                                     APFCODE:
00000000
                                     RMODE:
ANY
                                     LONGPARM:          NO

                                     PO FORMAT:
1
                                     XPLINK:
NO
                    *****PROGRAM OBJECT PROCESSED BY BINDER
 ***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND
PMARL)
 PMAR  001E0109 02C00C12 00000000 04A00000   00000000 00000000 00000000
0000
 PMARL 00328080 00000000 00020000 049C0000   01E80000 00000000 06A00000
00440000
       018C0000 00200000 016C0000 00140000
01D0
THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.10 ON 01/08/2008 AT 23:14:53
--------------------------------------------------------------------------------------------------------------------------
                             ** SEGMENT MAP TABLE **                                                       PAGE      1
          XREF LISTINGS OF A LONG ALIAS
NAME
CLASS            SEGMENT    OFFSET   LENGTH       LOAD      TYPE    ALIGNMENT    RMODE
B_TEXT                1         0      4A0       INITIAL   CAT     DOUBLE WORD  31
                             ** NUMERICAL MAP OF PROGRAM OBJECT MAINRTN          **                        PAGE      2
          XREF LISTINGS OF A LONG ALIAS
NAME
--------------------------------------------------------------------------------------------------------------------------
0RESIDENT CLASS:
B_TEXT
       CLAS LOC   ELEM LOC    LENGTH  TYPE    ALIGNMENT
NAME
         0                      166   ED     DOUBLE WORD
MAINRTN
       168                       BC   ED     DOUBLE WORD
$PRIV000010
            1D4        6C            LD
NONAME1M
       228                       BC   ED     DOUBLE WORD
$PRIV000011
            294        6C            LD
NONAME2M
       2E8                      102   ED     DOUBLE WORD
SUBRTN
       3F0                       54   ED     DOUBLE WORD
AAAAAAAA
       448                       54   ED     DOUBLE WORD
$BLANKCOM
       CLASS LENGTH
4A0
                              ** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT MAINRTN          **       PAGE
3
          XREF LISTINGS OF A LONG ALIAS
NAME
--------------------------------------------------------------------------------------------------------------------------
RESIDENT CLASS:           B_TEXT

CLAS LOC ELEM LOC FROM CLASS       FROM SECTION     REFERS TO SYMBOL CLAS LOC ELEM LOC IN CLASS         IN SECTION
       AC       AC B_TEXT           MAINRTN          AAAAAAAA             3F0       0
B_TEXT
       B0       B0 B_TEXT           MAINRTN          NONAME1M             1D4       6C B_TEXT
$PRIV000010
       B4       B4 B_TEXT           MAINRTN          NONAME2M             294       6C B_TEXT
$PRIV000011
      160      160 B_TEXT           MAINRTN          SUBRTN               2E8       0
B_TEXT
      394       AC B_TEXT           SUBRTN           $BLANKCOM            448       0
B_TEXT
       CLASS LENGTH
4A0
LENGTH OF PROGRAM OBJECT         4A0
--------------------------------------------------------------------------------------------------------------------------
                               ** ALPHABETICAL MAP OF PROGRAM OBJECT MAINRTN          **                     PAGE
4
          XREF LISTINGS OF A LONG ALIAS
NAME
ENTRY NAME  CLAS LOC  ELEM LEN/LOC             CLASS NAME       SECTION NAME OR ENTRY TYPE
$PRIV000010
                 168        BC                     B_TEXT
(ED)
$PRIV000011
                 228        BC                     B_TEXT
(ED)
$BLANKCOM
```

```
                 448        54                    B_TEXT
(ED)
AAAAAAAA
                 3F0        54                    B_TEXT
(ED)
MAINRTN
                   0       166                    B_TEXT
(ED)
NONAME1M
                 1D4            6C                B_TEXT
$PRIV000010
NONAME2M
                 294            6C                B_TEXT
$PRIV000011
SUBRTN
                 2E8       102                    B_TEXT
(ED)
------------------------------------------------------------------------------------------------------------------------
                              ** ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT MAINRTN          **    PAGE
5
          XREF LISTINGS OF A LONG ALIAS
NAME
SYMBOL REFERRED  CLAS LOC ELEM LOC IN CLASS         IN SECTION       CLAS LOC ELEM LOC FROM CLASS       FROM SECTION
$BLANKCOM             448        0 B_TEXT                                 394       AC B_TEXT           SUBRTN
AAAAAAAA              3F0        0 B_TEXT                                  AC       AC B_TEXT           MAINRTN
NONAME1M              1D4       6C B_TEXT           $PRIV000010            B0       B0 B_TEXT           MAINRTN
NONAME2M              294       6C B_TEXT           $PRIV000011            B4       B4 B_TEXT           MAINRTN
SUBRTN                2E8        0 B_TEXT                                 160      160 B_TEXT           MAINRTN
LENGTH OF PROGRAM OBJECT      4A0
------------------------------------------------------------------------------------------------------------------------
                             **  LONG NAME TABLE LISTING OF PROGRAM OBJECT MAINRTN           **             PAGE
6
          XREF LISTINGS OF A LONG ALIAS
NAME
ABBREVIATION        LONG NAME

THISISALO-LDOOOO := THISISALONGALIASNAMEYOUMAYCHANGETHENAMEIFYOULIKEANYNAMEWILLDOOOO
                            **  END OF LONG NAME TABLE LISTING OF PROGRAM OBJECT MAINRTN          **
**   END OF MAP AND CROSS-REFERENCE LISTING
```

LISTLOAD OUTPUT=BOTH Output

"Example: Output for LISTLOAD OUTPUT=BOTH for a PDSE" on page 587 shows an example of output produced by LISTLOAD OUTPUT=BOTH for a PDSE.

Example: Output for LISTLOAD OUTPUT=BOTH for a PDSE

```
1  LISTLOAD  MEMBER=TESTLR5,OUTPUT=BOTH,SECTION1=NO
1                                  *****  M O D U L E   S U M M A R Y  *****
0    MEMBER NAME:  TESTLR5                                                                 MAIN ENTRY
POINT:    00000028
0    LIBRARY:      SYSLIB                                                                  AMODE OF MAIN
ENTRY POINT: 24
0        NO ALIASES **
------------------------------------------------------------------------------------------------------------------------
----------------
0                       ****          ATTRIBUTES OF MODULE          ****
0                **   BIT  STATUS          BIT  STATUS          BIT  STATUS          BIT  STATUS   **
0                       0  NOT-RENT          1  NOT-REUS          2  NOT-OVLY          3  NOT-TEST
                        4  NOT-OL            5  BLOCK             6  EXEC              7  MULTI-RCD
                        8  NOT-DC            9  ZERO-ORG         10 RESERVED          11 RLD
                       12 EDIT              13 NO-SYMS           14 RESERVED          15 NOT-REFR
                       16 RESERVED          17 <16M              18 NOT-PL            19 NO-SSI
                       20 NOT-APF           21 PGM OBJ           22 NOT-SIGN          23 RESERVED
                       24 NOT-ALTP          25 RESERVED          26 RESERVED          27 RMODE24
                       28 RESERVED          29 RESERVED          30 RESERVED          31 RESERVED
                       32 MIGRATE           33 NO-PRIME          34 NO-PACK           35 RESERVED
                       36 RESERVED          37 RESERVED          38 RESERVED          39 RESERVED
0-----------------------------------------------------------------------------------------------------------------------
----------------
-                                    MODULE SSI:        NONE
                                     APFCODE:           00000000
                                     RMODE:             24
                                     LONGPARM:          NO
                                     PO FORMAT:         2
                                     XPLINK:            NO
0                   *****PROGRAM OBJECT PROCESSED BY BINDER
 ***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
 PMAR  001E0208 02C00400 00000000 00380000   00280000 00280000 00000000 0000
 PMARL 00520080 00000000 00020000 00380000   02280000 07FC0000 09EC0000 00400000
       013C0000 00200000 011C0000 00020000   017C0001 00000000 00380000 00000000
       00002005 013F0193 505FC3D6 D7E84040   4040
```

```
1                                          LISTING OF PROGRAM OBJECT
TESTLR5                                PAGE      1

0THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695PMB01 AT LEVEL 01.06 ON 01/13/2005 AT 19:35:05
0-----------------------------------------------------------------------------------------------------
----------------
0           MODULE SECTION:   $SUMMARY
0USABILITY: UNSPECIFIED    AMODE: ANY   OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== ESDs =====
0B_PRV(ED)
     CLASS:           B_PRV      LENGTH:           0 (HEX)       CLASS OFFSET:           0 (HEX)
FORMAT:  F(0001)
     NAME SPACE:           2      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE
RMODE:      UNS
     TEXT                         NOLOAD                          FILL:                    UNSPEC
 Q1(PD)
     CLASS:           B_PRV      LENGTH:           4 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           2      ALIGNMENT:      FULL WORD       PRIORITY:               0 (HEX)
SCOPE:   MODULE
     ATTRIBUTES:     WEAK
0-----------------------------------------------------------------------------------------------------
----------------
0           CONTROL SECTION:  A
0USABILITY: UNSPECIFIED    AMODE: 24    OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== IDRL =====
0          TRANSLATOR   VER   MOD     DATE        TIME
0          566896201     02    01    01/13/2005
 ===== IDRU =====
0          DATE         USER DATA
0          01/13/2005   MYDATA
 ===== ESDs =====
0B_TEXT(ED)
     CLASS:          B_TEXT      LENGTH:          18 (HEX)       CLASS OFFSET:           0 (HEX)
FORMAT:  F(0001)
     NAME SPACE:           1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE
RMODE:      24
     TEXT                         LOAD                            FILL:                    UNSPEC
 A(LD)
     CLASS:          B_TEXT      TEXT TYPE:         UNSPEC       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           1      SCOPE:             MODULE       ELEMENT OFFSET:         0 (HEX)
AMODE:      24
     ATTRIBUTES:     GENERATED,STRONG
 ENTA(LD)
     CLASS:          B_TEXT      TEXT TYPE:         UNSPEC       CLASS OFFSET:          14 (HEX)
     NAME SPACE:           1      SCOPE:             MODULE       ELEMENT OFFSET:        14 (HEX)
AMODE:      24
     ATTRIBUTES:     STRONG
 ENTA(ER)
                                  TEXT TYPE:         UNSPEC       CLASS OFFSET:           0 (HEX)
     TARGET SECTION: A                                            TARGET CLASS:            B_TEXT
     NAME SPACE:           1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)
AMODE:      UNS
     RESOLVED                     AUTOCALL
     ATTRIBUTES:     STRONG
 B_PRV(ED)
     CLASS:           B_PRV      LENGTH:           0 (HEX)       CLASS OFFSET:           0 (HEX)
FORMAT:  F(0001)
1                                          LISTING OF PROGRAM OBJECT
TESTLR5                                PAGE      2

0           CONTROL SECTION:  A
 ===== ESDs =====
0    NAME SPACE:           2      ALIGNMENT:    DOUBLE WORD       BIND METHOD:              MERGE
RMODE:      UNS
     TEXT                         NOLOAD                          FILL:                    UNSPEC
 Q1(PR)
     CLASS:           B_PRV      LENGTH:           4 (HEX)       CLASS OFFSET:           0 (HEX)
     NAME SPACE:           2      ALIGNMENT:      FULL WORD       PRIORITY:               0 (HEX)
SCOPE:   MODULE
     ATTRIBUTES:     WEAK
 ENTB(ER)
                                  TEXT TYPE:         UNSPEC       CLASS OFFSET:           0 (HEX)
     TARGET SECTION: B                                            TARGET CLASS:            B_TEXT
     NAME SPACE:           1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)
AMODE:      UNS
     RESOLVED                     AUTOCALL
     ATTRIBUTES:     STRONG
 ===== RLDs =====           CLASS:           B_TEXT
0ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES            XATTR
NAME      XATTR OFF
000000006 00000006   BR   RES  0004               1     (+)ENTA
 0000000C 0000000C C-OF   RES  0004               2     (+)Q1
```

```
 00000010 00000010   BR   RES  0004              1     (+)ENTB
 ===== TEXT =====           CLASS:           B_TEXT
000000000  07FEC1C1 C1C10000 00140000 00000000 00000028
C5D5E3C1                     ..AAAA.............ENTA........*
0-------------------------------------------------------------------------------------------------------
----------------
0           CONTROL SECTION:  B
0USABILITY: UNSPECIFIED    AMODE: 24    OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== IDRL =====
0          TRANSLATOR   VER   MOD     DATE        TIME
0          566896201     02    01    01/13/2005
 ===== ESDs =====
0B_TEXT(ED)
     CLASS:          B_TEXT      LENGTH:          14 (HEX)       CLASS OFFSET:          18 (HEX)
FORMAT:  F(0001)
     NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE
RMODE:       24
     TEXT                        LOAD                            FILL:                    UNSPEC
 B(LD)
     CLASS:          B_TEXT      TEXT TYPE:         UNSPEC       CLASS OFFSET:          18 (HEX)
     NAME SPACE:          1      SCOPE:             MODULE       ELEMENT OFFSET:         0 (HEX)
AMODE:       24
     ATTRIBUTES:     GENERATED,STRONG
 ENTB(LD)
     CLASS:          B_TEXT      TEXT TYPE:         UNSPEC       CLASS OFFSET:          28 (HEX)
     NAME SPACE:          1      SCOPE:             MODULE       ELEMENT OFFSET:        10 (HEX)
AMODE:       24
     ATTRIBUTES:     STRONG
 ENTB(ER)
                                 TEXT TYPE:         UNSPEC       CLASS OFFSET:           0 (HEX)
     TARGET SECTION: B                                           TARGET CLASS:            B_TEXT
     NAME SPACE:          1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)
AMODE:      UNS
1                                         LISTING OF PROGRAM OBJECT
TESTLR5                               PAGE      3

0           CONTROL SECTION:  B
 ===== ESDs =====
0    RESOLVED                    AUTOCALL
     ATTRIBUTES:     STRONG
 UNRES(ER)
                                 TEXT TYPE:         UNSPEC       CLASS OFFSET:           0 (HEX)
     TARGET SECTION: UNRES                                       TARGET CLASS:            B_TEXT
     NAME SPACE:          1      SCOPE:            LIBRARY       ELEMENT OFFSET:         0 (HEX)
AMODE:      UNS
     RESOLVED                    AUTOCALL
     ATTRIBUTES:     STRONG
 ===== RLDs =====           CLASS:           B_TEXT
0ELEM.OFF CLS.OFF  TYPE STATUS LENG ATTR       NSPACE      TARGET NAME      PARTRES            XATTR
NAME      XATTR OFF
000000006 0000001E   BR   RES  0004              1     (+)ENTB
 0000000C 00000024   BR   RES  0004              1     (+)UNRES
 ===== TEXT =====           CLASS:           B_TEXT
000000018  07FEC2C2 C2C20000 00280000 00000030
C5D5E3C2                              ..BBBB..........ENTB............*
0-------------------------------------------------------------------------------------------------------
----------------
0           CONTROL SECTION:  UNRES
0USABILITY: UNSPECIFIED    AMODE: 24    OVERLAY SEGMENT:    0   OVERLAY REGION:   0
 ===== IDRL =====
0          TRANSLATOR   VER   MOD     DATE        TIME
0          566896201     02    01    07/23/2004
 ===== ESDs =====
0B_TEXT(ED)
     CLASS:          B_TEXT      LENGTH:           7 (HEX)       CLASS OFFSET:          30 (HEX)
FORMAT:  F(0001)
     NAME SPACE:          1      ALIGNMENT:    DOUBLE WORD       BIND METHOD:           CATENATE
RMODE:       24
     TEXT                        LOAD                            FILL:                    UNSPEC
 UNRES(LD)
     CLASS:          B_TEXT      TEXT TYPE:         UNSPEC       CLASS OFFSET:          30 (HEX)
     NAME SPACE:          1      SCOPE:             MODULE       ELEMENT OFFSET:         0 (HEX)
AMODE:       24
     ATTRIBUTES:     GENERATED,STRONG
 ===== TEXT =====           CLASS:           B_TEXT
000000030  07FEE4D5 D9C5E2
..UNRES.........................*
0**   END OF PROGRAM OBJECT LISTING
1                            ** SEGMENT MAP TABLE
**                                                        PAGE      4

0CLASS            SEGMENT    OFFSET   LENGTH       LOAD      TYPE    ALIGNMENT    RMODE
```

```
0B_TEXT                  1         0       38       INITIAL   CAT     DOUBLE WORD  24
1                             ** NUMERICAL MAP OF PROGRAM OBJECT TESTLR5
**                        PAGE      5

0---------------------------------------------------------------------------------------------------------
----------------
0RESIDENT CLASS:           B_TEXT
0      CLAS LOC   ELEM LOC    LENGTH  TYPE    ALIGNMENT             NAME
0        0                      18   ED     DOUBLE WORD           A
             14         14            LD                            ENTA
        18                      14   ED     DOUBLE WORD           B
             28         10            LD                            ENTB
        30                       7   ED     DOUBLE WORD           UNRES
0      CLASS LENGTH              38
0---------------------------------------------------------------------------------------------------------
----------------
0RESIDENT CLASS:            B_PRV
0      CLAS LOC   ELEM LOC    LENGTH  TYPE    ALIGNMENT             NAME
0        0                       0   ED     DOUBLE WORD           A
         0                       0   ED     DOUBLE WORD           $PRIV000003
              0                  4   PD     FULL WORD               Q1
0      CLASS LENGTH              0
0---------------------------------------------------------------------------------------------------------
----------------
1                             ** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT TESTLR5
**       PAGE      6

0RESIDENT CLASS:           B_TEXT
0CLAS LOC ELEM LOC FROM CLASS       FROM SECTION     REFERS TO SYMBOL CLAS LOC ELEM LOC IN CLASS
IN SECTION
0       6        6 B_TEXT           A                ENTA                   14       14 B_TEXT
A
        C        C B_TEXT           A                Q1               $CLASS_OFFSET     B_PRV
       10       10 B_TEXT           A                ENTB                   28       10 B_TEXT
B
       1E        6 B_TEXT           B                ENTB                   28       10 B_TEXT
B
       24        C B_TEXT           B                UNRES                  30        0 B_TEXT
0      CLASS LENGTH              38
0---------------------------------------------------------------------------------------------------------
----------------
1                             ** NUMERICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT TESTLR5
**       PAGE      7

0RESIDENT CLASS:            B_PRV
0     ****  NO RLD DATA ****
0      CLASS LENGTH                0
0LENGTH OF PROGRAM OBJECT       38
0---------------------------------------------------------------------------------------------------------
----------------
1                             ** ALPHABETICAL MAP OF PROGRAM OBJECT TESTLR5
**                     PAGE      8

0ENTRY NAME  CLAS LOC  ELEM LEN/LOC             CLASS NAME       SECTION NAME OR ENTRY TYPE
0$PRIV000003
                   0         0                       B_PRV       (ED)
 A
                   0         0                       B_PRV       (ED)
 A
                   0        18                      B_TEXT       (ED)
 B
                  18        14                      B_TEXT       (ED)
 ENTA
                  14            14                  B_TEXT       A
 ENTB
                  28            10                  B_TEXT       B
 Q1
                   0         4                       B_PRV       (PD)
 UNRES
                  30         7                      B_TEXT       (ED)
0---------------------------------------------------------------------------------------------------------
----------------
1                             ** ALPHABETICAL CROSS-REFERENCE LIST OF PROGRAM OBJECT TESTLR5
**    PAGE      9

0SYMBOL REFERRED  CLAS LOC ELEM LOC IN CLASS         IN SECTION       CLAS LOC ELEM LOC FROM CLASS
FROM SECTION
0ENTA                  14       14 B_TEXT           A                       6        6 B_TEXT
A
 ENTB                  28       10 B_TEXT           B                      10       10 B_TEXT
A
 ENTB                  28       10 B_TEXT           B                      1E        6 B_TEXT
```

```
B
 Q1               $CLASS_OFFSET     B_PRV                                    C         C B_TEXT
A
 UNRES                  30        0 B_TEXT                                  24         C B_TEXT
B
0LENGTH OF PROGRAM OBJECT       38
0**   END OF MAP AND CROSS-REFERENCE LISTING
```

LISTIDR output

Figure 197 on page 591 shows an example of LISTIDR output for a load module processed by the linkage editor or binder.

```
  LISTIDR  DDN=DD1,MEMBER=TLIST
                                   *****  M O D U L E   S U M M A R Y  *****
    MEMBER NAME:  TLIST                                                                    MAIN ENTRY POINT:    0000000E
    LIBRARY:      DD1                                                                      AMODE OF MAIN ENTRY POINT: 24
        NO ALIASES **
--------------------------------------------------------------------------------------------------------------------------
                       ****          ATTRIBUTES OF MODULE           ****
                **   BIT  STATUS         BIT  STATUS         BIT  STATUS         BIT  STATUS   **
                      0  NOT-RENT         1  NOT-REUS         2  NOT-OVLY         3  NOT-TEST
                      4  NOT-OL           5  BLOCK            6  EXEC             7  MULTI-RCD
                      8  NOT-DC           9  ZERO-ORG         10 EP > ZERO        11 RLD
                      12 EDIT             13 NO-SYMS          14 F-LEVEL          15 NOT-REFR
--------------------------------------------------------------------------------------------------------------------------
                                    MODULE SSI:  NONE
                                    APFCODE:     00000000
                                    RMODE:       24
                                    LONGPARM:    NO
                     *****LOAD MODULE PROCESSED EITHER BY VS LINKAGE EDITOR OR BINDER
                                           LISTIDR FOR LOAD MODULE TLIST                      PAGE 0001

B                   CSECT                       YR/DAY                    SPZAP    DATA
                    A                           1972/271                    92240
                    B                           1972/271                    NO IDENT
--------------------------------------------------------------------------------------------------------------------------

A          THIS LOAD MODULE WAS PRODUCED BY THE BINDER 5695DF108  AT LEVEL 21.01 ON DAY 271 OF YEAR 1992.
--------------------------------------------------------------------------------------------------------------------------

C        CSECT        TRANSLATOR                VR.MD                          YR/DY
         A            566896201                 02.01                          1972/271
         B            566896201                 02.01                          1972/271
         D1           566896201                 02.01                          1972/271
         UNRES        566896201                 02.01                          1992/034
--------------------------------------------------------------------------------------------------------------------------

D        CSECT                        YR/DAY                    USER
         A                            1972/271                    ANOTHERONE
         B                            1972/271                    myprogram
--------------------------------------------------------------------------------------------------------------------------
```

Figure 197. Example: LISTIDR output for a load module processed by linkage editor or binder

Figure 198 on page 592 shows an example of LISTIDR output for a program object processed by the binder.

```
 LISTIDR     MEMBER=(LOADMOD2)                                                    00020905
                                  ***** M O D U L E   S U M M A R Y *****
    MEMBER NAME:  LOADMOD2                                                                        MAIN ENTRY POINT:    00000000
    LIBRARY:      SYSLIB                                                                          AMODE OF MAIN ENTRY POINT: 31
        NO ALIASES **
------------------------------------------------------------------------------------------------------------------------
                       ****          ATTRIBUTES OF MODULE          ****
                **   BIT  STATUS        BIT  STATUS        BIT  STATUS        BIT  STATUS   **
                      0  NOT-RENT        1  NOT-REUS        2  NOT-OVLY        3  NOT-TEST
                      4  NOT-OL          5  BLOCK           6  EXEC            7  MULTI-RCD
                      8  NOT-DC          9  ZERO-ORG       10 RESERVED        11 RLD
                     12 EDIT            13 NO-SYMS         14 RESERVED        15 NOT-REFR
                     16 RESERVED        17 <16M            18 NOT-PL          19 NO-SSI
                     20 NOT-APF         21 PGM OBJ         22 NOT-SIGN        23 RESERVED
                     24 NOT-ALTP        25 RESERVED        26 RESERVED        27 RMODEANY
                     28 RESERVED        29 RESERVED        30 RESERVED        31 RESERVED
                     32 MIGRATE         33 NO-PRIME        34 NO-PACK         35 RESERVED
                     36 RESERVED        37 RESERVED        38 RESERVED        39 RESERVED
------------------------------------------------------------------------------------------------------------------------
                                    MODULE SSI:  NONE
                                    APFCODE:     00000000
                                    RMODE:       ANY
                                    LONGPARM:    NO
                                    PO FORMAT:   2
                                    XPLINK:      NO
 A            *****PROGRAM OBJECT PROCESSED BY BINDER
***THE FOLLOWING ARE THE UNFORMATTED PDSE DIRECTORY ENTRY SECTIONS (PMAR AND PMARL)
PMAR  001E0206 02C00412 00000000 04500000   00000000 00000000 00000000 0000
PMARL 005200C0 00000000 00020000 04500000   02380000 08180000 0EF40000 00500000
      01240000 00200000 01040000 00020000   01740001 00000000 04500000 00000000
      00001995 154F0205 408FC2D7 C2C6F6F4   F5F5
                                          LISTIDR FOR PROGRAM OBJECT LOADMOD2                              PAGE      1

------------------------------------------------------------------------------------------------------------------------
THIS PROGRAM OBJECT WAS ORIGINALLY PRODUCED BY 5695DF108 AT LEVEL 01.01 ON 06/03/95 AT 20:54:08
------------------------------------------------------------------------------------------------------------------------
 B DATE        PTF NUMBER
CSECT:    SDX
          04/16/2001   ZAPIDR01
          04/16/2001   ZAPIDR02
          04/16/2001   ZAPIDR03
          04/16/2001   ZAPIDR04
          04/16/2001   ZAPIDR05
          04/16/2001   ZAPIDR06
          04/16/2001   ZAPIDR07
          04/16/2001   ZAPIDR08
          04/16/2001   ZAPIDR09
------------------------------------------------------------------------------------------------------------------------
 C TRANSLATOR   VER   MOD     DATE
CSECT:    CM1
          566896201     02    01    04/16/2001
CSECT:    SDX
          566896201     02    01    04/16/2001
CSECT:    SD1
          566896201     02    01    04/16/2001
CSECT:    SD2
          566896201     02    01    04/16/2001
------------------------------------------------------------------------------------------------------------------------
 D DATE         USER DATA
CSECT:    $MODULE LEVEL DATA
          04/16/2001   THIS IS A TEST
CSECT:    SD1
          04/16/2001   USER IDR TEST 2
          04/16/2001   USER IDR TEST 3
 ------------------------------------------------------------------------------------------------------------------------
```

Figure 198. Example: LISTIDR output for a program object processed by binder

The IDR listing, as in Figure 197 on page 591 and Figure 198 on page 592, has four sections that are separated by dashed lines. The four sections contain:

A

The linkage editor identification or binder identification record (IDRB). The identification record is displayed in a single line. This line shows the binder or linkage editor program identifier, version and release numbers, and the data and time of binding.

Note: The time of binding is listed only for a program object.

B

A list of SPZAP IDR entries (IDRZ), if any. The IDRZ records, if any, are formatted two or more lines per section. The first contains the associated CSECT name, and the second, and subsequent lines, a modification date and up to eight bytes of PTF number or other data entered on the SPZAP IDRDATA control statement. There will be one detail line for each modification to the control section. For load module output, the IDRZ records are formatted one line per section.

C

A list of language translator IDR records (IDRL). These entries are formatted only if OUTPUT=ALL was specified, or defaulted, on the LISTIDR control statement. The IDRL records, if any, are also formatted two or more lines per CSECT. The section name appears on the first line, and the translator program id, version and release, and date of translation on the second and subsequent lines. There will be one line of translator data for each compiler, assembler or other language product involved in the production of the object code for that section. For load module output, the IDRL records are formatted one line per section. Blank CSECT names in program objects will be seen as $BLANKCOM. They will be be seen as $BLANKCM in load modules.

D

A list of user-supplied IDR data (IDRU), if any. The IDRU records normally appear two lines per CSECT. The first line shows the section name, and the second line an entry date and up to 80 bytes of data, entered by the user on the binder IDENTIFY control statement. If the section name is a module level section (identified as '00000001'x), the constants $MODULE LEVEL DATA are printed in place of the section name.

For program objects, if no data is available in a section, one of the following messages will appear instead of the formatted detail:

```
          NO SPZAP DATA EXISTS FOR THIS PROGRAM OBJECT
         NO BINDER DATA EXISTS FOR THIS PROGRAM OBJECT
    NO TRANSLATION DATA EXISTS FOR THIS PROGRAM OBJECT
  NO IDENTITY/USER DATA EXISTS FOR THIS PROGRAM OBJECT
```

For load modules, if no SPZAP data is available, the following message will appear instead of the formatted detail:

```
          THIS LOAD MODULE CONTAINS NO INFORMATION SUPPLIED BY SPZAP
```

Description of LISTIDR output

LISTLPA output

Figure 199 on page 594 shows an example of LISTLPA output.

```
MODIFIED  LINK  PACK  AREA  MAP  -  ALPHABETICALLY BY NAME
NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME    NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME
IGC00020   --------   ------     00B42000                        IGC0005E   --------   ------     00B37FA0
IGC0006I   --------   ------     00B419C0                        IGGC019BN  --------   ------     00B414D8
                                                                                                  200K  SYS   276K
                                                                              N  00.12SEC  VIRT
MODIFIED  LINK  PACK  AREA  MAP  -  NUMERICALLY BY ENTRY POINT
NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME    NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME
IGC0005E   --------   ------     00B37FA0                        IGGC019BN  --------   ------     00B414D8
IGC0006I   --------   ------     00B419C0                        IGC00020   --------   ------     00B42000
                                                                                                  200K  SYS   276K
                                                                              N  00.12SEC  VIRT
PAGEABLE  LINK  PACK  AREA  MAP  -  ALPHABETICALLY BY NAME
NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME    NAME       LOCATION   LENGTH     EP ADDR    MAJOR  LPDE  NAME
AHLACFV                          819B595E   AHLTVTAM             AHLDMPMD                         81926EBE   AHLSETD
AHLDSP                           81963962   AHLTXSYS             AHLEXT                           8198F660   AHLTSYSM
AHLFI0                           8193A926   AHLTSYFL             AHLFPI                           8193A9FC   AHLTSYFL
AHLFRR                           8198F7EA   AHLTSYSM             AHLFSSCH                         8193A946   AHLTSYFL
AHLFSVC                          8193A9D8   AHLTSYFL             AHLMCER                          81926450   AHLTSETD
AHLPINT                          8198F748   AHLTSYSM             AHLREADR   01977C08   000003F8   81977C08
AHLSBCU1                         81991F4A   AHLWSMOD             AHLSBL0K                         819916B0   AHLWSMOD
AHLSBUF                          81991A90   AHLWSMOD             AHLSETD    01926000   00001708   81926000
AHLSETEV   01928000   00001998   81928000                        AHLSFE0B                         819917EE   AHLWSMOD
AHLSRB                           819639EE   AHLTXSYS             AHLSRM                           81963A62   AHLTXSYS
AHLSTAE                          8198F8C6   AHLTSYSM             AHLSVC                           8198F61A   AHLTSYSM
AHLTACFV                         819B596A   AHLTVTAM             AHLTCCWG   0192A000   00002378   8191A000
AHLTDIR                          81926A58   AHLSETD              AHLTDSP                          81971658   AHLTPID
AHLTEXT    01956920   000006E0   81956920                        AHLTFCG    0192D000   000016D0   8192D000
AHLTF0R    01954570   00000A90   81954570                        AHLTFRR                          81954694   AHLTFOR
AHLTLSR                          819717D2   AHLTPID              AHLTPI                           8197147E   AHLTPID
AHLTPID    01971468   00000B98   81971468                        AHLTSLIP   0192F000   00001C50   8192F000
AHLTSRB                                     AHLTPID              AHLTSRM                                     AHLTFOR
AHLSTAE                                     AHLTFOR              AHLTSVC    01931000   00002768
AHLTSYFL   0193A908   000006F8                                   AHLTSYSM   0198F508   00000AF8
AHLTUSR    019299C0   00000640                                   AHLTVTAM   019B5940   000006C0
AHLTXSYS   01963850   000007B0   81971770                        AHLVCOFF   019B6F40   000000C0   8195458C
AHLVCON    01989EE8   00000118   819547B4                        AHLWSMOD   019916B0   00000950   81931000
AHLWTOMD                         8193A908   AHLSETD              AMDSYS00   01934000   00001208   8198F508
AMDSYS01   01936000   00002AD8   819299C0                        AMDSYS02   019BB648   00000548   819B5940
AMDSYS03   01939000   00001828   81963850                        AMDSYS04   0193B000   00002038   819B6F40
           01975178   99999358   81989EE8                                   01961C08   000003F8   819916B0
           00F28000   00001E60   81926E4C                                   00BF1008   000006C8   81934000
                                 81936000   IMDUSRFF                                              819BB648   IMDUSRF8
           00B8E230   000003F8   91039000                                                         8193B000   ISTAICIR
           00C4E730   000008D0   81975178                                                         81961C08   DCM3B3
                                 00F28000   DCM3B3                          00F26000   00001360   00BF1008
AMDSYS05   00CB8078   00000F88   00C4C000                        AMDSYS06   00C54020   00000FE0   00C08590
AMDUSRFD   00C26318   00000CE8   00B8E230                        AMDUSRFE                         00C48000
AMDUSRFF                         00C4E730   DCM270               AMDUSRF8                         00C56328
AMDUSRF9                         00C56328                        CCKRIUWT                         00F26000
CVAFGTF                          00CB8078                        DCMBE0                           00C54020
DCMBE1                                                           DCM180
DCM181                                                           DCM182                00000FE0
DCM183                                                           DCM270     00F24000   000014E0
DCM271                                                           DCM272     00E1E830   000007D0
                                 00C26318                                                         00F24000
                                 00F24000                                                         00E1E830
```

Figure 199. Sample LISTLPA output

Chapter 17. SPZAP: Modify data in programs and VTOCs

SPZAP is a service aid program that operates in problem state. SPZAP allows you to dynamically update and maintain programs and data sets. SPZAP can be used to apply fixes to modules or programs that need to be at current levels of the operating system. The functions of SPZAP provide many capabilities, including:

- Using the inspect and modify functions of SPZAP, you can fix programming errors that require only the replacement of instructions in a load module member of a PDS or a program object member of a PDSE without recompiling the program.
- Using the modify function of SPZAP, you can set traps in a program by inserting incorrect instructions. The incorrect instructions will force abnormal ending; the dump of storage provided as a result of the abnormal ending is a valuable diagnostic tool, because it shows the contents of storage at a predictable point during processing.
- Using SPZAP to replace data directly on a direct access device, you can reconstruct VTOCs or data records that may have been destroyed as the result of an I/O error or a programming error.
- On the advice of the IBM Support Center, start tracing in system components that do not use component trace. The IBM Support Center will tell you how to use the SPZAP service aid to start traces in these components.
- Update the system status index (SSI) in the directory entry for any load module in a PDS or program object in a PDSE. Update the CSECT identification record (IDR) in any load module in a PDS or program object in a PDSE.

Planning for SPZAP

SPZAP is an application that provides editing capabilities for data on a direct access storage device (DASD). Protect against SPZAP (and other applications that can update data sets) being used to damage data through use of the installation's security protection scheme:

- In *z/OS DFSMS Using Data Sets*, see the chapter, "Protecting Data Sets" for information pertaining to protecting data sets.
- In *z/OS DFSMSdfp Advanced Services*, see the chapter, "Protecting the VTOC and VTOC Index" for information pertaining to protecting VTOCs.

  Installations that use RACF should employ a combination of GDASDVOL and DASDVOL resource profiles to establish this protection. For more information about these profiles, see *z/OS Security Server RACF Security Administrator's Guide*.

  IBM recognizes the particular sensitivity of the VTOC. For a VTOC, the console operator must respond to message AMA117D before SPZAP processes an update request. This authorization must be supplied in addition to authorization through use of the installation's security protection scheme.

To assist with packaging AMASPZAP jobs for distribution, CHECKSUM processing was enhanced with the introduction of PARM=PRECHECK. When PARM=PRECHECK is specified on the EXEC statement, SPZAP behaves differently. SPZAP first checks the SYSIN input for many error conditions and CHECKSUM validation. If errors in the input are detected, or a CHECKSUM mismatch occurs, SPZAP processing stops to avoid making partial updates. When no error conditions are detected, SPZAP processes all of the updates. Because this effectively results in [SUPERZAP] requiring two passes at the SYSIN data, other ramifications occur. Interactive input results in inconsistent behavior and is not supported. Also, output from SPZAP contains messages and data from both the prechecking and processing phases, which is double the usual amount of data. For example, output from commands like DUMPT is produced every time that it is encountered, in the PRECHECK and post-precheck phases. For more information about the PRECHECK option and sample output data from PRECHECK, see the following topics:

- PARM=PRECHECK in “JCL statements” on page 607
- Figure 218 on page 625
- Figure 219 on page 625

Invoking SPZAP

SPZAP provides functions to inspect, modify, update the IDR, dump, and other functions. The control statements, such as VERIFY, REPLACE, IDRDATA, DUMP, provide these functions. For more information, see “SPZAP control statements” on page 611. These functions are entered through the user's stream (SYSIN) in the JCL or through the system console. See “Running SPZAP” on page 606 for a detailed description.

Inspecting and modifying data

The inspection function is controlled by the VERIFY statement. VERIFY allows you to check the contents of a specific location in a load module member of a PDS, a program object member of a PDSE or a z/OS UNIX file, a specific physical record of a direct-access data set, or a record of a member of a data PDSE before you replace the contents. If the contents at the specified location do not agree with the contents as specified in the VERIFY statement, subsequent REP operations are not performed.

Note: A PDSE containing data other than a program object will be referred to as a PDSE data library.

The SPZAP modification function is controlled by the REP (replace) control statement. The REP control statement allows you to replace instructions or data at a specific location in a load module member of a PDS, a program object in a PDSE or a z/OS UNIX file, a physical record in a direct-access data set or a record of a member of a PDSE data library.

To avoid possible errors in replacing data, you should always precede any REP operation with a VERIFY operation.

SPZAP is often used to inspect and modify the contents of executable programs to correct errors. Executable programs can be in one of two forms:

- A load module is stored in a PDS and created by the linkage editor, binder, or IEBIMAGE utility.
- A program object is stored in a PDSE or z/OS UNIX file and created by the program management binder.

Note: All subsequent references in this topic to a program object in a PDSE also apply to a program object in a z/OS UNIX file.

In addition, SPZAP can be used to inspect and modify data other than executable programs. Examples of such types of data are:

- A sequential (QSAM/BSAM and EXCP) data set.
- A direct organization (BDAM) data set.
- A VSAM data set in a conventional DASD volume.

There are several types of data sets that are not supported by SPZAP:

- An extended format sequential data set.
- A VSAM data set in an extended address volume after the first 65520 cylinders.

See the following topics for more information:

- “Inspecting and modifying a load module or program object” on page 596
- “Inspecting and modifying a data record” on page 602

Inspecting and modifying a load module or program object

To inspect or modify data in a load module or program object, you need a NAME statement to supply SPZAP the name of the appropriate member of the file. The load module must be a member of the PDS, identified by the SYSLIB DD statement included in the JCL. The program object must be a member of the PDSE or a file in the z/OS UNIX directory identified by the SYSLIB DD statement included in the JCL.

To inspect or modify a program object that is in a z/OS UNIX file system, use the PATH parameter on the SYSLIB DD statement instead of the DSNAME parameter. Use PATH to identify the directory that contains the file that is the program object. Use the NAME statement to identify the file.

If the load module member of a PDS or program object member of a PDSE contains more than one control section (CSECT), you must also supply SPZAP with the name of the CSECT that is to be inspected or modified. If no CSECT name is given in the NAME statement, SPZAP assumes that the control section to be processed is the first one encountered in searching the load module.

Whenever SPZAP updates a CSECT in a load module member of a PDS or program object member of a PDSE in response to your NAME and REP control statements, it also puts descriptive maintenance data in a CSECT identification record (IDR) associated with the load module or program object. 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 “SPZAP control statements” on page 611 for an explanation of the IDRDATA statement.

Figure 200 on page 597 shows how to inspect and modify a load module containing a single CSECT.

```
//ZAPCSECT     JOB         MSGLEVEL=(1,1)
//STEP         EXEC        PGM=AMASPZAP
//SYSPRINT     DD          SYSOUT=A
//SYSLIB       DD          DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN        DD          *
 NAME        IEEVLNKT
 VERIFY      0018      C9C8,D2D9,D1C2,C7D5
 REP         0018      E5C6,D3D6,E6F0,4040
 SETSSI      01211234
 IDRDATA     71144
 DUMP        IEEVLNKT
/*
```

Figure 200. Example: Inspecting and modifying a single CSECT load module

SYSLIB DD Statement
Defines the system library SYS1.LINKLIB containing the module IEEVLNKT that SPZAP is to process.

NAME Control Statement
Instructs SPZAP that the operations defined by the control statements that follow are to be performed on the module IEEVLNKT.

VERIFY Control Statement
Requests that SPZAP check the hexadecimal data at offset X'0018' in 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, SPZAP continues processing the subsequent statements sequentially. If the data is not identical, SPZAP will not perform the REP and SETSSI operations requested for the module. It will, however, perform the requested DUMP operation before discontinuing the processing. It will also dump a hexadecimal image of the module IEEVLNKT to the SYSPRINT data set.

REP Control Statement
Causes SPZAP to replace the data at offset X'0018' in module IEEVLNKT with the data given in this control statement, provided the VERIFY statement was successful.

SETSSI Control Statement
Instructs SPZAP to replace the system status information in the directory entry for module IEEVLNKT with the SSI data given in the statement, if the VERIFY statement was successful. The new SSI is to contain:

- A change level of 01
- A flag byte of 21
- A serial number of 1234

IDRDATA Control Statement
Causes SPZAP 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 SPZAP if the VERIFY operation was successful.

Figure 201 on page 598 shows how to apply an IBM-supplied PTF in the form of an SPZAP fix, rather than a module replacement PTF.

```
//PTF40228   JOB      MSGLEVEL=(1,1)
//STEP       EXEC     PGM=AMASPZAP
//SYSPRINT   DD       SYSOUT=A
//SYSLIB     DD       DSNAME=SYS1.NUCLEUS,DISP=OLD
//SYSIN      DD       *
 NAME      IEANUC01  IEWFETCH
 IDRDATA   LOCFIX01
 VERIFY    01F0 47F0C018
 VERIFY    0210 5830C8F4
 REP       01F0 4780C072
 REP       0210 4130C8F4
 SETSSI    02114228
 DUMPT     IEANUC01  IEWFETCH
/*
```

Figure 201. Example: Modifying a CSECT in a load module

SYSLIB DD Statement
Defines the library (SYS1.NUCLEUS) that contains input module IEANUC01.

SYSIN DD Statement
Defines the input stream.

NAME Control Statement
Instructs SPZAP that the operations defined by the control statements that immediately follow this statement are to be performed on the CSECT IEWFETCH contained in the load module IEANUC01.

IDRDATA Control Statement
Causes SPZAP to update the IDR in module IEANUC01 for CSECT IEWFETCH with the data LOCFIX01, if either of the REP operations is successful.

VERIFY control statements
Requests that SPZAP 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, SPZAP continues processing subsequent control statements sequentially. However, if the data at the locations does not compare identically to the data given in the VERIFY control statements, SPZAP dumps a hexadecimal image of CSECT IEWFETCH to the SYSPRINT data set; the subsequent REP and SETSSI statements are ignored. The DUMPT function specified will be performed before SPZAP ends processing.

REP control statements
Causes SPZAP to replace the data at offsets X'01F0' and X'0210' from the start of CSECT IEWFETCH with the hexadecimal data specified on the corresponding REP statements.

SETSSI Control Statement
Causes SPZAP to 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 a change level of 02, a flag byte of 11, and a serial number of 4228.

DUMPT Control Statement
Causes SPZAP to produce a translated dump for CSECT IEWFETCH of load module IEANUC01.

Use the JCL in Figure 202 on page 599 to inspect and modify two CSECTs in the same load module.

```
//CHANGIT    JOB         MSGLEVEL=(1,1)
//STEP       EXEC        PGM=AMASPZAP
//SYSPRINT   DD          SYSOUT=A
//SYSLIB     DD          DSNAME=SYS1.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       0154 4770,C042
 IDRDATA   PTF01483
 SETSSI    01212448
 DUMPT     IEFX5000  IEFQMRAW
/*
```

Figure 202. Example: Inspecting and modifying two CSECTs

SYSLIB DD Statement
Defines the system library SYS1.LINKLIB containing the load module IEFX5000 that is to be changed by SPZAP.

NAME Control Statement #1
Instructs SPZAP that the operations requested through the control statements immediately following it are to be performed on CSECT IEFQMSSS in load module IEFX5000.

VERIFY Control Statement #1
Requests that SPZAP check the hexadecimal data at offset X'0284' in CSECT IEFQMSSS to make sure it is the same as the data specified in this control statement. If the data is identical, SPZAP continues processing the control statements. If the data is not identical, SPZAP does not perform the REP or SETSSI for CSECT IEFQMSSS, but it does perform the DUMPT operation. It also provides a hexadecimal dump of CSECT IEFQMSSS.

REP Control Statement #1
Causes SPZAP to replace the data at offset X'0284' in CSECT IEFQMSSS with the hexadecimal data given in this control statement.

IDRDATA Control Statement #1
Causes SPZAP 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 SPZAP to replace the system status information in the directory entry for module IEFX5000 with the SSI data given. The new SSI will contain a change level of 01, a flag byte of 21, and a serial number of 2448.

DUMPT Control Statement #1
Provides a translated dump of CSECT IEFQMSSS.

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 SPZAP perform the VERIFY function at offset X'0154' from the start of CSECT IEFQMRAW. If the VERIFY operation is successful, SPZAP continues processing the subsequent control statements sequentially. If the VERIFY is rejected, however, SPZAP does not perform the following REP or SETSSI operations, but it does dump a hexadecimal image of CSECT IEFQMRAW to the SYSPRINT data set and performs the DUMPT operation as requested.

REP Control Statement #2
Causes SPZAP 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 SPZAP 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 SPZAP to perform the same function as the previous SETSSI, but only if the second VERIFY is not rejected.

DUMPT Control Statement #2
Causes SPZAP to perform the DUMPT function on control section IEFQMRAW.

Use the JCL shown in Figure 203 on page 600 to inspect and modify control section PRINTF in z/OS UNIX System Services.

```
//ZAPUNIX  EXEC PGM=AMASPZAP
//SYSPRINT DD  SYSOUT=A
//SYSLIB   DD  PATH='/sj/sjpl/binder/unixzap/',
// PATHDISP=(KEEP,KEEP)
//SYSIN    DD  *
  NAME     LOADMOD1  PRINTF
  VERIFY   0000 58F0C210
  REP      0000 68F0D210
  DUMP     LOADMOD1 PRINTF
/*
```

Figure 203. Example: Inspecting and Modifying a CSECT in z/OS UNIX System Services

SYSLIB DD Statement
Defines the directory '/sj/sjpl/binder/unixzap/' containing the program object LOADMOD1 that SPZAP is to process.

SYSIN DD Statement
Defines the input stream.

NAME control statement
Instructs SPZAP that the operations defined by the control statements that follow are to be performed on the control section PRINTF of the program object LOADMOD1.

VERIFY control statement
Requests that SPZAP compare the contents of the location X'0000' in the control section PRINTF with the data given on the VERIFY control statement. If the comparisons are equal, SPZAP continues processing subsequent control statements sequentially. If the data does not compare, SPZAP dumps a hexadecimal image of CSECT PRINTF to the SYSPRINT data set; the subsequent REP control statement is ignored.

REP control statement
Causes SPZAP to replace the data at offset X'0000' from the start of the CSECT PRINTF with the hexadecimal data provided.

DUMP control statement
Requests that a hexadecimal image of program object LOADMOD1, control section PRINTF be dumped to the SYSPRINT data set. Because the dump statement follows the REP statement, the image will reflect the changes made by SPZAP if the VERIFY operation was successful.

Use the JCL in Figure 204 on page 601 to inspect and modify a CSECT within a program object.

```
//UPDATE  JOB   MSGLEVEL=(1,1)
//ZAPSTEP EXEC  PGM=AMASPZAP
//SYSPRINT  DD  SYSOUT=A
//SYSLIB    DD  DSN=SYS1.USERLIB,DISP=OLD
//SYSIN     DD  *
  NAME                                                        LONG#
              ALIASNAME       PDSPROCR
  VERIFY    000070   58E0,9118
  REP       000074   50E0,9434,9140,9058,47E0,C0A8,45E0,C476,94BF,9058,#
                     181D,58D0,D004,1FFF,43F0,A046,1F00,BF07,A047
  REP       00009A   1861,1870,1F55,0E64,98EC,D00C,07FE
```

Figure 204. Example: Using SPZAP to modify a CSECT

SYSLIB DD statement
Defines the library SYS1.USERLIB containing a program object with an alias of LONGALIASNAME. (Note the continuation character (#) following LONG.) One CSECT in this program object is being changed.

SYSIN DD statement
Defines the input stream.

NAME control statement
This control statement contains a ‘#’ in column 72 and is continued to a second control statement. The first 18 columns of the continued statement are blanks and are ignored. The string ALIASNAME on this continued statement is concatenated with the string LONG to form member name LONGALIASNAME. Note that this statement could have been contained in one record as NAME LONGALIASNAME PDSPROCR. Either way, the NAME statement indicates that SPZAP is to use the VERIFY and two REP statements to one CSECT PDSPROCR in the program object member whose alias is LONGALIASNAME.

Note: Leading blanks on the continued statement are ignored. No characters on the first card are skipped. Therefore, in order to split an operand, part on the first card and the rest on the second, it is important that the part of the operand on the first card extends to column 71. A blank in column 71 indicates that the non-blank string in the second card begins a new operand.

VERIFY control statement
Requests that SPZAP check the data at hexadecimal displacement X'000070' 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, SPZAP continues processing the following control statements sequentially. If the data is not identical, SPZAP does not perform the REP function but does perform the ABSDUMPT operation; it also dumps a formatted hexadecimal image of the data record defined by the CCHHR statement to the SYSPRINT data set.

REP Control Statement #1
Causes SPZAP to replace the data at offset X'000074' in CSECT PDSPROCR with the hexadecimal data given in this control statement. Notice that this statement contains a non-blank (#) in column 72 indicating that it is continued to a second control statement.

REP Control Statement #2
Causes SPZAP to replace the data at offset X'00009A' in CSECT PDSPROCR with the hexadecimal data given in this control statement.

Accessing data in a CSECT

For a complete description of the control statements mentioned in the following discussion, see “SPZAP control statements” on page 611.

Once the CSECT has been found, the use of offset parameters in the VERIFY and REP statements allow SPZAP to locate the data that is to be verified and replaced. The offset 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, SPZAP 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. You must compute the proper displacements by subtracting the assembly listing address delimiting the start of the CSECT from the assembly listing address of the data to be referenced.

You can, however, use the BASE control statement to eliminate the need for such calculations and allow you to use the assembly listing locations. The BASE control statement should be included in the input to SPZAP 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. SPZAP then subtracts this value from the offset specified on any VERIFY or REP statement that follows the BASE statement, and uses the difference as the displacement of the data.

Figure 205 on page 602 is a sample assembly listing showing more than one control section. To refer to the second CSECT (IEFCVOL2), you could include in the input to SPZAP a BASE statement with a location of 0398. Then, to refer to the subsequent LOAD instruction `(L R2,CTJCTAD)` you could use an offset of X'039A' in the VERIFY or REP statements that follow in the SPZAP input stream.

```
      LISTING TITLE

 LOC   OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT

000000                              1 IEFCVOL1 CSECT                 10000017

                                    .
                                    .
                                    .

000384 00000000                   378 VCNQMSSS DC   V(IEFQMSSS)       55800017
                                  379 *                               56000017
000388 00000000                   380 VCMSG15  DC   V(IEFVMG15)       56100017
00038C D200 1001 8000 00000 00000  381 MVCMSG   MVC  0(1,R1),0(R8)     56200017
                                  382 *                               56300017
000392 D200 1001 1000 00001 00000  383 MVCBLNKS MVC  1(1,R1),0(R1)     56400017
                                  384 *                               56500017



000398                            386          CSECT                  56600017
000398 0590                       387          BALR  R9.0             56700017
00039A                            388          USING *,R9             56800017
00039A 5820 C010            00010  389          L      R2,LCTJCTAD     56900017

                                    .
                                    .
                                    .
```

Figure 205. Sample Assembly Listing Showing Multiple Control Sections

Inspecting and modifying a data record

You will inspect and modify a data record differently depending on whether the data record is in a PDSE or some other type of data set, such as a VTOC or sequential data set.

Note: The following information does NOT apply to a PDS. SPZAP only supports a PDS that contains load modules.

Record not in a PDSE: To inspect or modify a specific data record that is not in a PDSE you must use a CCHHR control statement to specify its direct access address. This CCHHR address must be within the limits of the direct access data set defined in the SYSLIB DD control statement.

When you use the CCHHR control statement, SPZAP reads the physical record you want to inspect or modify. The offset parameters specified in subsequent VERIFY and REP statements are then used to locate the data that will be verified or replaced within the record. These hexadecimal offsets must define the displacement of data relative to the beginning of the record and include the length of any key field.

If you request a REP operation for a record identified by a CCHHR control statement, SPZAP issues message AMA112I to provide a record of your request.

In z/OS V1R7 and later DSNTYPE=LARGE data sets are supported when using V1R7 or a later release of SPZAP.

Record in a PDSE: To inspect or modify a specific data record in a PDSE data library, you must use the RECORD control statement preceded by a NAME control statement to specify its direct access address. This combination of RECORD and NAME serves as a pointer to a specific location in a PDSE data library member.

The CCHHR control statement does not apply to a PDSE. Any attempt to access data in a PDSE with a CCHHR control statement will cause an error message. Any VER|VERIFY, REP, IDRDATA, or SETSSI control statements immediately following a CCHHR statement will be flagged in error and ignored.

To determine the relative record number for a specific record, invoke SPZAP, specifying:

```
NAME membernam
ABSDUMP(T)  1 99999999
```

The results show a display of all records in the member, record length, relative record number, and other pertinent information.

In Figure 206 on page 603, the data set to be modified is a volume table of contents.

```
//ZAPIT         JOB         MSGLEVEL=(1,1)
//STEP          EXEC        PGM=AMASPZAP
//SYSPRINT      DD          SYSOUT=A
//SYSLIB        DD          DSNAME=FORMAT4.DSCB,DISP=OLD,
//      UNIT=3390,VOLUME=SER=111111,DCB=(KEYLEN=44)
//SYSIN         DD          *
 CCHHR             0005000001
 VERIFY            2C   0504
 REP               2C   0A08
 REP               2E   0001,03000102
 ABSDUMPT          ALL
/*
```

Figure 206. Example: Inspecting and modifying a data record

SYSPRINT DD Statement
: Defines the message data set.

SYSLIB DD Statement
: Defines the data set to be accessed by SPZAP in performing the operations specified by the control statements. In this example, it defines the VTOC (a Format 4 DSCB) on a 3390 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 SPZAP 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 SPZAP 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, SPZAP continues processing the following control statements sequentially. If the data is not identical, SPZAP does not perform the REP function but does perform the ABSDUMPT operation; it also dumps a formatted hexadecimal image of the data record defined by the CCHHR statement to the SYSPRINT data set.

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 SPZAP 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 is also dumped.

Note: If the VTOC is to be modified, message AMA117D is issued to the operator, requesting permission for the modification.

Figure 207 on page 604 shows how to inspect and modify a record within a PDSE data library.

```
//UPDDATA JOB   MSGLEVEL=(1,1)
//ZAPSTEP EXEC  PGM=AMASPZAP
//SYSPRINT  DD  SYSOUT=A
//SYSLIB    DD  DSN=IBMUSER.LMD.PDSE,DISP=OLD
//SYSIN     DD  *
  NAME    USERDATA
  RECORD    0003
  VER       000010   04B3,9017
  REP       000014   10C7,C5E3,C4E2
  ABSDUMP   3        3
```

Figure 207. Example: Using SPZAP to modify a data record

SYSLIB DD statement
: Defines the data set that SPZAP is to access to perform the operations specified by the control statements. In this example, it defines a private PDSE data library. The NAME statement identifies the member as USERDATA, which is shown in Figure 217 on page 624.

SYSIN DD statement
: Defines the input stream containing the SPZAP control statements.

NAME control statement
: Instructs SPZAP that the control statements that immediately follow this statement are to be performed on the member whose name is USERDATA.

RECORD control statement
: Indicates that SPZAP is to access relative record 3, the third record in the member USERDATA. Record 3 is the object of the VERIFY and REP operations that follow.

VERIFY control statement
: Requests that SPZAP check the data at hexadecimal displacement X'0010' to compare it to the string specified. If there is a difference, this VERIFY is flagged with an error message, the contents of record 3 are displayed, and the following REP statement is flagged and ignored.

REP control statement
: Causes SPZAP to replace the data at offset X'000014' in record 3 of member USERDATA with the data X'10C7C5E3C4E2' if the preceding VERIFY statement completed successfully. If the preceding VERIFY statement was flagged in error, then this statement is also flagged in error, and no data is replaced.

ABSDUMP control statement
: auses SPZAP to display record 3 of member USERDATA. Record 3 is displayed whether the VERIFY succeeded or failed.

Updating the System Status Index (SSI)

You can use the SETSSI control statement to overlay the existing data in the SSI with your own data. For a complete description of the SETSSI control statement, see "SPZAP control statements" on page 611.

The SSI is a 4-byte field created by the linkage editor in the directory entry of a load module. It is useful for keeping track of any modifications that are performed on a load module. SPZAP updates the system status index automatically whenever it replaces data in the associated module.

Not all load modules have system status information. In those that do, the SSI is located in the last four bytes of the user data field in the directory entry. Figure 208 on page 605 shows the position of the SSI in load module directory entries.

```
* Member Name   *   TTR     *   C  *  User Data Field   *  SSI       *
* 1           8 *  9    11  *  12  *  13 to 70 maximum  *  variable  *
```

Figure 208. SSI bytes in a load module directory entry

Figure 209 on page 605 shows the composition of the SSI field and the flag bits used to indicate the types of changes made to the corresponding load module program.

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 increased by one for each release that includes a new version of that program. If you make a change to the SSI for any of the IBM-released programs, take care not to destroy this maintenance level indicator unless you purposely mean to do so. To keep the change level byte at its original value, find out what information is contained in the SSI before using the SETSSI function. The LISTLOAD control statement of the LIST service aid can give you the information you need.

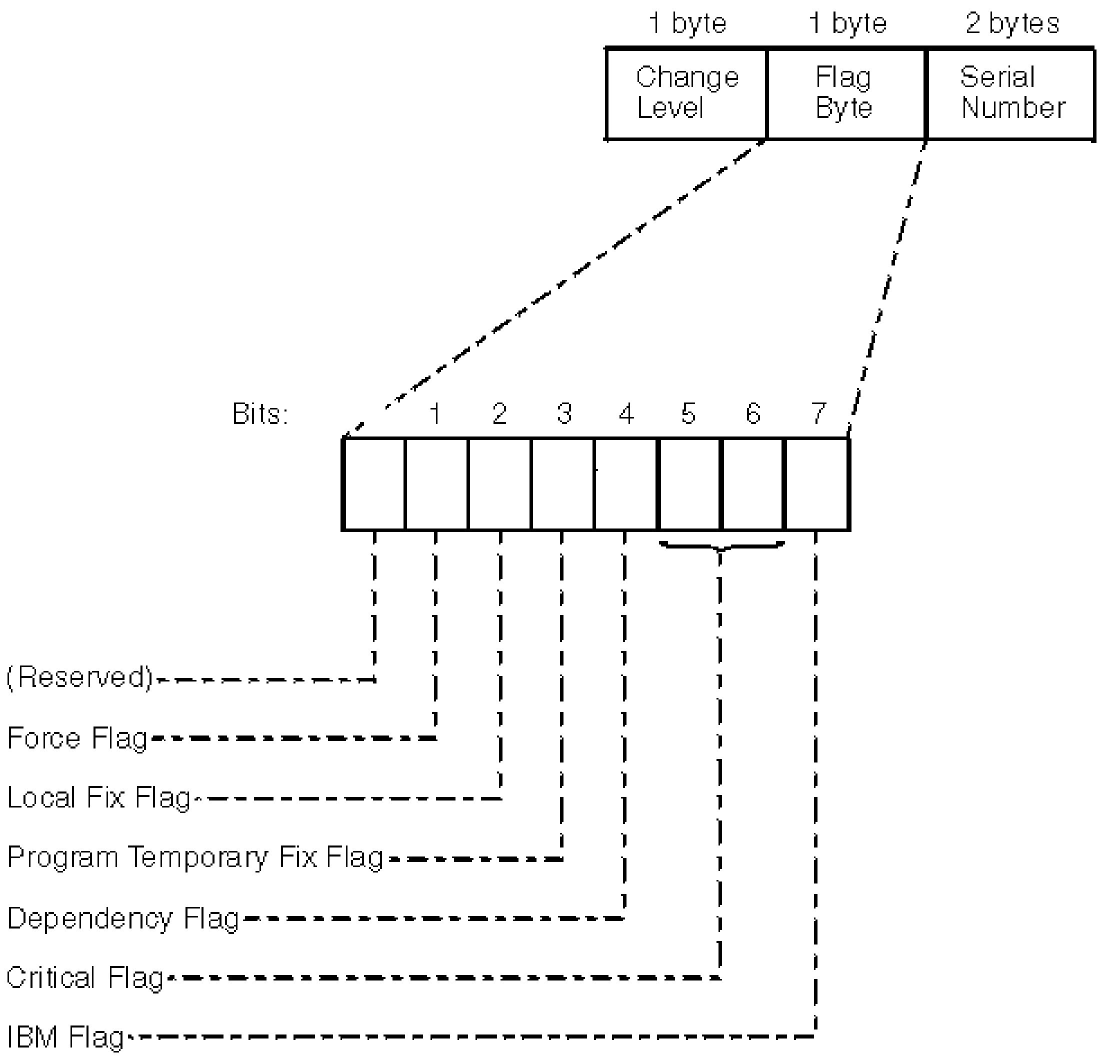


Figure 209. Flag bytes in the System Status Index field

The second byte of the SSI is called the *flag byte*. Bits within the flag byte contain information reflecting the member's maintenance status. You need only be concerned with two of the eight bits when you are using SPZAP:

- Bit 2, the local fix flag, indicates that the user has modified a particular member. (It is not used to reflect modifications made by IBM-supplied program temporary fix or a PTF.) SPZAP sets this local fix flag bit to one after successfully modifying a load module.

- Bit 3, the program temporary fix flag, is set to one when an IBM-authorized PTF is applied to a system library to correct an error in an IBM module.

All other bits in the flag byte should be retained in the SSI as they appeared before the SETSSI operation took place, 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. SPZAP will not change these bytes unless you request a change by using the SETSSI control statement.

Running SPZAP

You can run SPZAP using control statements as input into the job stream or dynamically as part of selected macros:

- "Using JCL and control statements to run SPZAP" on page 607
- "Invoking SPZAP dynamically" on page 609

Consider the following points when you run SPZAP:

- SPZAP uses the system OPEN macro. Therefore, SPZAP cannot modify or inspect RACF-protected data sets when SPZAP cannot successfully complete the access checks that occur during OPEN processing.
- A module can be a load module in a PDS or a program object in a PDSE. SPZAP replaces a program object in a PDSE rather than updating the program object in place. Users who have used the BLDL macro to establish a connection to a particular copy of a program object must invoke BLDL again to gain access to the new copy.

  If you are using LLA to manage a program object that has been changed through the use of SPZAP, then, to make the modified object available, the operator must refresh LLA for the directory entries for that program object. Otherwise, LLA continues to load the unmodified version of the program object.

  SPZAP itself cannot identify when a load module or a program object is in use by another user or is in the process of being loaded through LLA.

  See *z/OS DFSMS Using Data Sets* for more information about PDSEs and their data structure.
- Unexpired data sets such as system libraries cannot be modified unless the operator replies r xx,'U' to the expiration message that occurs during OPEN.
- If you use SPZAP to modify an operating system module that is made resident in virtual storage only at IPL time, you must IPL the system again to invoke the new version of the module you have modified. (Note that this requirement applies to all modules in SYS1.LPALIB, all data sets named in the LPALSTxx member of SYS1.PARMLIB, and all modules in SYS1.NUCLEUS.

  SPZAP itself cannot determine when a module is loaded only at IPL time.
- The SYSLIB DD statement cannot define a concatenated or a multi-volume data set.
- SPZAP supports only direct access storage devices (DASD) for the SYSLIB device.
- When modifying a system data set, such as SYS1.LINKLIB, specify DISP=OLD on the SYSLIB DD statement.
- If you use SPZAP for a digitally signed module, message AMA165I is issued. The control statement is to be processed, but the digital signature is no longer valid.
- SPZAP supports placement of SYSIN and SYSPRINT data sets in cylinder-managed space.
- SPZAP (AMASPZAP or IGWSPZAP) supports all data sets allocated in the extended addressing space (EAS) of an extended address volume (EAV).
- SPZAP (AMASPZAP or IGWSPZAP) supports the following dynamic allocation (DYNALLOC or SVC 99) options for all data sets: S99TIOEX(XTIOT), S99ACUCB (NOCAPTURE), and S99DSABA (DSAB above the line).

Using JCL and control statements to run SPZAP

One way to invoke SPZAP is through the job stream. The JCL statements you need to use when running SPZAP are:

- JOB statement
- EXEC statement
- SYSPRINT DD statement
- SYSLIB DD statement
- SYSABEND DD statement
- SYSIN DD statement

These JCL statements, when used with the control statements in "SPZAP control statements" on page 611, allow greater function for SPZAP.

Also, when running SPZAP, you must consider the region size available to your program. The minimum region size needed to run AMASPZAP is 200 kilobytes.

Usually, no REGION parameter is required on the EXEC statement but REGION=120K (or any other value less than 200K) will cause SPZAP to issue message AMA154I and stop processing with a return code of 16. In addition, SPZAP will issue message AMA154I if the program management binder has too small a region size. This problem might occur if the SYSLIB member is extremely large, when REGION=4M or REGION=6M might be needed.

JCL statements

JOB Statement
Marks the beginning of the job.

EXEC Statement
Invokes SPZAP. You identify AMASPZAP as the program to be run by specifying either PGM=AMASPZAP or PGM=IMASPZAP, which is an alias name for AMASPZAP.

Note: You must ensure that the region size is at least 200K for SPZAP to complete processing normally. The valid parameters that you can specify on the PARM parameter are:

1. IGNIDRFULL
2. PRECHECK

These options can be specified in any order, and are independent of each other. Each option can be the only value on the PARM parameter. The specification of multiple parameters requires a comma-separated list, where each parameter is individually enclosed within quotes and surrounded by parentheses. Duplicate or incorrect values cause SPZAP to fail with error message AMA129I being issued.

PARM=IGNIDRFULL enables SPZAP to override the standard restrictions that are placed on CSECT updates (through NAME and REP) when IDR space for the module is found to be full.

PRECHECK support provides a means for suppliers of SPZAP statements to ensure that their exact program, when combined with a non-blank CHECKSUM statement, was received unaltered by a consumer of their program. The PRECHECK parameter is supported to prevent most partial module updates from occurring if detectable errors are encountered while it is processing the input SYSIN data stream. Its sole purpose is to support the more intuitive CHECKSUM processing. So, when PARM=PRECHECK is specified on the EXEC statement, SPZAP behavior is radically different from the default behavior that occurs when PRECHECK is not specified. Therefore, any syntax or verification error or warning that is encountered during the PRECHECK processing phase, anywhere within the SYSIN input, prevents any updates from occurring against any of the target modules. When the SYSIN control statements are processed, the SYSPRINT output for PRECHECK contains messages and data from both the prechecking and modification processing phases. Therefore, PRECHECK might double the usual amount of output. For example, output from commands like DUMPT is produced every time that it is encountered. When the PRECHECK option is not specified, SPZAP behavior is unchanged.

Similarly, when PRECHECK is specified, modifications to the target data set, if requested, are done only when the SYSIN input has no errors. Syntax errors for CHECKSUM, VERIFY, or SETSSI in the input SYSIN cause PRECHECK to fail SPZAP, and no updates are made in the target data set. Error scenarios, like those scenarios that lead to AMA102I, AMA104I, AMA105I, AMA109I, AMA110I, AMA111I, AMA112I, AMA133I and AMA134I, or VER failures in any NAME would now fail the entire SPZAP step. Thus, users might find that their SPZAP jobs behave differently. For example, VERIFY statements that confirm previous REP statements cannot be supported, as shown in the following example:

```
//SYSIN DD *
  NAME IEECB866 IEAVTS0L
  VER 20 F024 034E
  REP 20 F0FF 035F
  VER 20 F0FF 035F
  …,
/*
```

Without PRECHECK, the second VER statement would work, if the preceding REP was successful.

With PRECHECK, the REP would not run during prechecking, so the subsequent VER statement would fail, thus failing the entire SPZAP.

Notes:

1. Do not use PARM=IGNIDRFULL with IBM-maintained modules.
2. PARM=IGNIDRFULL has no meaning if SYSLIB is a program object library. There is no restriction on the number of IDRZ records associated with a program object library member.
3. Use PARM=PRECHECK with SYSIN input in batch mode, and combine it with a non-blank CHECKSUM statement. If SPZAP detects that SYSIN data is entered interactively, it issues message AMA166I and fails SPZAP.

SYSPRINT DD Statement

Defines a sequential output data set for messages that can be written on a system printer, a magnetic tape volume, or a direct access volume. This statement is required for each run of SPZAP.

SYSLIB DD Statement

Defines the direct access data set that will be accessed by SPZAP when performing the operations specified on the control statements. The DSNAME parameter and DISP=OLD or DISP=SHR are required. The VOLUME and UNIT parameters are necessary only if the data set is not cataloged. This statement cannot define a concatenated or multi-volume data set. It is required to run SPZAP.

Notes:

1. When this data set is the VTOC, you must specify DSNAME=FORMAT4.DSCB. When you access a record in the VTOC (that is, a DSCB) for modification, SPZAP issues message AMA117D to the console. No message is issued, however, when an ABSDUMPT operation is performed on the VTOC.
2. Standard VSAM processing requires the use of an ACB to access the data set. However, because SPZAP only supports open with DCB, it does not obtain the correct information that is needed to operate upon a VSAM data set. Where there is a blocksize mismatch reported by SPZAP, explicit specification of the blocksize on the SYSLIB DD statement will override SPZAP's normal size processing.

SYSABEND DD Statement

Defines a sequential output data set to be used in case SPZAP ends abnormally. The data set can be written to a printer, a magnetic tape volume, or a direct access volume. This statement is optional.

SYSIN DD Statement

Defines the input stream data set that contains SPZAP control statements.

Return codes

When SPZAP ends, one of the following return codes is placed in general purpose register 15:

Code
: **Meaning**

00
: Successful completion.

04
: Warning of a condition. This can result in future errors if an action is not taken to correct the warning now.

08
: An SPZAP input statement contains an error or was overridden by operator intervention. Check the syntax of the statements to determine the cause of the error.

12
: A requested JCL statement is absent or specifies a data set that was not successfully opened. SPZAP ends immediately.

16
: A permanent I/O error occurred, perhaps caused by a JCL error, such as incorrect blocksize. SPZAP ends immediately. The region size might be too small. REGION=200K is the smallest region size permitted. However, the program management binder might require as much as 4M or 6M if the program object is very large.

20
: Using DUMP, DUMPT, VER, or REP processing, SPZAP found a control record for a specific control section that was larger than the specified BLOCKSIZE. SPZAP ends immediately.

Note: Return codes greater than 8 can result in the end of SPZAP processing at the point where the problem was encountered. Therefore, there might be no other indication that processing completed, other than the job ending.

Invoking SPZAP dynamically

You can run SPZAP from selected macros. SPZAP can be invoked by an application program at run time through the use of the CALL, LINK, XCTL, or ATTACH macro. The program must supply a list of alternate DDNAMEs of data sets to be used by SPZAP if the standard DDNAMEs are not used.

A program must be running APF authorized in order to update a VTOC through SPZAP. Other SPZAP functions do not require the calling program to be authorized.

The following diagram shows the general form of these macros when used to invoke SPZAP.

```
(anyname)   CALL AMASPZAP,(oplist,ddnamlst),VL
(anyname)   XCTL EP=AMASPZAP
(anyname)   LINK EP=AMASPZAP,PARAM=(oplist,ddnamlst),VL=1
(anyname)   ATTACH EP=AMASPZAP,PARAM=(oplist,ddnamlst),VL=1
```

anyname
: Indicates an optional statement label on the macro statement.

EP
: The entry point for the SPZAP program.

PARAM
: Specifies, as a sublist, parameters to be passed from the program to SPZAP.

oplist
: Specifies the name of either a halfword of zeros (indicating no options) or a non-zero halfword followed by a character string whose length is given in bytes. For the possible parameter value, see the information about the EXEC statement in "JCL statements" on page 607.

ddnamlst
: Specifies the name of a variable-length list containing alternate ddnames for data sets to be used during SPZAP processing. If all the standard ddnames (SYSPRINT, SYSLIB, and SYSIN) are used, then you can omit this parameter.

The DDNAME list must begin on a halfword boundary. The first two bytes contain a count of the number of bytes in the rest of the list. The format of the list is fixed, with each entry having eight bytes. Any name of less than eight bytes must be left justified and padded with blanks. If a name is left out in the list, the entry must contain binary zeros; the standard name is then assumed. Names can be omitted from the end of the ddname list by shortening the list.

The sequence of 8-byte entries in the list is as follows:

Entry
: **Standard name**

0-7
: not applicable

8-15
: not applicable

16-23
: not applicable

24-31
: SYSLIB

32-39
: SYSIN

40-47
: SYSPRINT

VL | VL=1
: Indicates that the high-order bit is to be set to 1 in the last word of the address parameter list.

Note: If you do not supply the name of a DDNAME list, you must ensure that the high-order bit of the oplist address is set on. Coding VL|VL=1 sets the bit correctly.

Figure 210 on page 610 is an example of two functionally-equivalent dynamic invocations of SPZAP.

```
         EXSPZAP  CSECT
         USING *,15            ASSUME REG15 IS BASE
         MODID                 MODULE ID AND DATE IN PROLOG
         SAVE  (14,12)         SAVE REGISTERS
         BALR  12,0            ESTABLISH BASE REGISTER
         USING *,12
         ST    13,SAVEAREA+4   CHAIN NEW SAVEAREA TO PREVIOUS
         LR    2,13            TEMPORARILY SAVE ADDRESS OF OLD SAVEAREA
         LA    13,SAVEAREA     INIT REG13 WITH ADDRESS OF NEW SAVEAREA
         ST    13,8(0,2)       CHAIN PREVIOUS SAVEAREA TO NEW
*************************************************************************
*                                                                       *
*   THIS EXAMPLE SHOWS TWO FUNCTIONALLY EQUIVALENT DYNAMIC              *
*   INVOCATIONS OF SPZAP.                                            *
*           NO OPTIONS ARE PASSED.                                      *
*           THE DDNAME FOR THE SYSLIB FILE IS CHANGED TO TESTLIBR.      *
*           THE DDNAME FOR THE SYSIN FILE IS NOT CHANGED.               *
*           THE DDNAME FOR THE SYSPRINT FILE IS CHANGED TO PRINTOUT.    *
*                                                                       *
*************************************************************************
LINKZAP1 LINK  EP=AMASPZAP,PARAM=(OPTLIST,DDLIST),VL=1
LINKZAP2 LINK  EP=AMASPZAP,PARAM=(0,DDLIST),VL=1
         L     13,SAVEAREA+4   LOAD ADDRESS OF PREVIOUS SAVEAREA
         RETURN (14,12),T,RC=0 RETURN TO CALLER
OPTLIST  DC    H'0'            NO OPTIONS ARE PASSED TO AMASPZAP
DDLIST   DS    0H              ALIGN DDNAMES TO HALFWORD BOUNDARY
         DC    H'48'           LENGTH OF THE CHARACTER STRING
*                               CONTAINING DDNAME OVERRIDES
         DC    24XL1'00'       FIRST 24 CHARACTERS ARE IGNORED
         DC    CL8'TESTLIBR'   CHANGE SYSLIB FILE TO DDNAME OF TESTLIBR
         DC    8XL1'00'        USE SYSIN FILE FOR INPUT OF CONTROL
*                               STATEMENTS
         DC    CL8'PRINTOUT'   CHANGE SYSPRINT FILE TO DDNAME OF
*                               PRINTOUT
SAVEAREA DC    18F'0'          REGISTER SAVEAREA
         END
```

Figure 210. Sample assembler code for dynamic invocation of SPZAP

SPZAP control statements

SPZAP control statements (entered either through the user's input stream in the JCL or through the system console) define the processing functions to be performed during a particular run of SPZAP. To enter other SPZAP control statements through the system console, you can use the CONSOLE control statement. The control statements that define the running of SPZAP are:

- ABSDUMP or ABSDUMPT
- BASE
- CCHHR
- CHECKSUM
- Comment (*)
- CONSOLE
- DUMP or DUMPT
- IDRDATA
- NAME
- RECORD
- REP
- SETSSI
- VERIFY

Coding rules for SPZAP control statements

Follow these rules when coding the control statements for SPZAP:

- The size of a SPZAP control card is 80 bytes; it can contain 71 bytes of control information.
- Statements can begin in any column up to column 71.
- The operation name of the statement must precede the parameters and must be complete on the first statement; you cannot continue the operation name.
- 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 blanks within data fields are not permitted.
- Data and offset values must be specified as a multiple of two hexadecimal digits.
- Following the last required parameter and its blank delimiter, the rest of the space on most control statements can be used for comments. Exceptions to this are the NAME and DUMP control statements: if you omit the CSECT parameter from either of these statements, do not use the space following the load module parameter for comments.
- A record beginning with an asterisk is considered to be a comment statement.
- A comment statement (one that begins with a single asterisk) cannot be continued.
- Member names and CSECT names for program objects can be as long as 1024 characters.
- When SYSLIB refers to a PDSE or a z/OS UNIX file, you can continue any non-comment statement as follows:
  - Column 72 of the control card to be continued must contain a non-blank character.
  - The string of characters on the immediately following card (starting with the first non-blank character) is concatenated with column 71 of the preceding card. AMASPZAP ignores leading blanks in a continuation card, but it displays the cards on SYSPRINT unchanged.
  - You can continue statements as necessary. You cannot, however, continue a comment field that follows the last parameter.

- Even though some parameters allow you to use a single asterisk (*) to indicate an omitted parameter, the first non-blank character on a continuation card cannot be an asterisk. Select the break point carefully to avoid starting a continuation statement with a single asterisk.

- In other words, for continuation:
  - When the SYSLIB is a PDSE or a z/OS UNIX file, IGWSPZAP is invoked, which supports continuation.
  - When the SYSLIB is a PDS, AMASPZAP is invoked, which does not support continuation.

Following are detailed descriptions of the SPZAP control statements, in alphabetical order.

{ABSDUMP|ABSDUMPT} {*startaddr stopaddr* | *startrec stoprec* | *membername* | ALL}

This statement can be used to dump the following, as defined in the SYSLIB DD statement:

- A group of physical records
- A group of records belonging to a member of a PDSE data library
- A load module member or all load module members of a PDS
- All members of a PDSE
- The directory of a PDSE that contains program objects

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 “Reading SPZAP output” on page 620. The variables are:

startaddr

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). This parameter must be exactly 10 digits long.

stopaddr

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.

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. If a record number of 0 is specified, SPZAP will change it to 1 since the READ routine skips over such records. 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.

Note: When the SYSLIB DD statement describes a data set placed in an extended address volume (EAV), the *startaddr* and *stopaddr* values must be specified in hexadecimal in the form CCCCcccHRR, where CCCCccc is referred to as a 28-bit cylinder address. The meanings of the codes are as follows:

- CCCC is the 16 low order bits of the cylinder number.
- ccc is the 12 high order bits of the cylinder number.
- H is the track number.
- RR is the record number.

startrec

The value of the first relative record of a member of a PDSE data library to display. This parameter can be 1 to 8 digits long. The first record of a member has a *startrec* value of 1.

Note: *ABSDUMP|ABSDUMPT startrec stoprec* is valid only following a *NAME member* statement where SYSLIB is a PDSE data library and *member* is a valid member of that library.

stoprec

The value of the last relative record of a member of a PDSE data library to display. This parameter can be 1 to 8 digits long. If the value of *stoprec* specifies a relative record value greater than that of the last physical record, printing stops after the last record of the member is printed. If the value of *stoprec* is less than the value of *startrec*, no records are displayed. One can display all the records of a member of a PDSE data library by using the following two statements:

```
NAME  member
ABSDUMP|ABSDUMPT   1 99999999
```

membername

The name of a member of a PDS or a PDSE, as specified by the SYSLIB DD statement. The name can refer to a load module member of a PDS or a member of a PDSE data library. In each case, the entire member is dumped when this variable is specified. (Use DUMP/DUMPT for program object members of a PDSE.)

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 a sequential data set, SPZAP dumps until it reaches end of file.
- For an indexed sequential and direct access data set, SPZAP dumps all extents.
- For a PDS, SPZAP dumps all extents, including all linkage editor control records, if any exist.
- For a PDSE data library, SPZAP displays a directory plus a listing of all members of the library. If the data set is a PDSE that contains program objects, SPZAP displays only the directory.

BASE *xxxxxx*

Used by SPZAP 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 SPZAP input stream. When SPZAP 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 variable is:

xxxxxx

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.

Note: The BASE statement should be included in the SPZAP input stream immediately following the NAME statement that identifies the control section that is to be involved in the SPZAP operations. The specified base value remains in effect until all VERIFY, REP, and SETSSI operations for the CSECT have been processed.

Figure 211 on page 614 shows how to use the BASE control statement 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=AMASPZAP
//SYSPRINT   DD           SYSOUT=A
//SYSLIB     DD           DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN      DD           *
 NAME             IEFMCVOL  IEFCVOL2
 BASE            0398
 IDRDATA         MOD04
 VERIFY          039A 5820C010
 REP             039A 47000000
 DUMP            IEFMCVOL   IEFCVOL2
```

Figure 211. Example: Using the BASE control statement

SYSLIB DD Statement
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 SPZAP control statements.

NAME Control Statement
Instructs SPZAP 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 SPZAP 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 SPZAP to update the IDR in module IEFMCVOL for CSECT IEFCVOL2 with the data MOD04, if the REP operation is successful.

VERIFY Control Statement
Requests that SPZAP 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, SPZAP 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, SPZAP continues processing the following statements in the order in which they are encountered. If the data is not identical, SPZAP does not perform the REP, SETSSI, or IDRDATA functions, but it does perform the DUMPs operation; it also dumps a hexadecimal image of CSECT IEFCVOL2 to the SYSPRINT data set.

REP Control Statement
Causes SPZAP 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 SPZAP 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 SPZAP (assuming no verification has been rejected).

CCHHR *record address*
Identifies 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 variable is:

record address
The actual direct access address of the record containing data to be replaced or verified. It must be specified as a 10-digit hexadecimal number in the form *ccccchhhhrr*, where *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. A zero record number is incorrect and defaults to 1.

Notes:

1. You can define more than one CCHHR statement in your input to SPZAP. However, the VERIFY and REP statements associated with each CCHHR statement must immediately follow the specific CCHHR statement to which they apply.
2. When the SYSLIB DD statement describes a data set placed in an extended address volume (EAV), the *record address* value must be specified in hexadecimal in the form CCCCcccHRR, where CCCCccc is referred to as a 28-bit cylinder address. The meanings of the codes are as follows:
   - CCCC is the 16 low order bits of the cylinder number.
   - ccc is the 12 high order bits of the cylinder number.
   - H is the track number.
   - RR is the record number.

CHECKSUM [*hhhhhhhh*]

Used to print or verify a fullword checksum (parity-check). All of the valid hexadecimal operands since the preceding CHECKSUM statement or SPZAP initialization are logically concatenated into a single string divided into fullwords, the sum of which is the checksum. For example, the string 12345678FACE produces the checksum 0D025678. Each CHECKSUM statement resets the accumulated checksum value to zeros.

The CHECKSUM statement is effective in detecting clerical errors that may occur when transcribing an SPZAP type of fix. CHECKSUM does not prevent errors; it only causes a message to be issued. By the time the CHECKSUM statement is processed, all prior replaces have been done. See the information about PARM=PRECHECK in "JCL statements" on page 607 for details about CHECKSUM errors that can lead to SPZAP failures without making updates to the target data set.

hhhhhhhh

8 hexadecimal characters that are compared with the checksum. If the two values are equal, a message is written indicating that the checksum was correct and has been reset.

If the operand field is blank, a message is written giving the actual value of the checksum, and indicating that the checksum has been reset.

When the CHECKSUM control statement is provided with an incorrect operand, the REP and SETSSI statements processed already are not affected.

If the operand is not valid or is not equal to the checksum, a message is written indicating incorrect operand or checksum error. All subsequent REP and SETSSI statements are ignored until the next NAME or CCHHR statement is encountered. The results of previously processed statements are not affected.

*** (Comment)**

When the first non-blank character in a statement is an asterisk, SPZAP recognizes the statement as a comment, used to annotate the SPZAP input stream and output listing. You can specify the asterisk in any position, but at least one blank space must follow the asterisk. You can include any number of comment statements in the input stream, but you cannot continue a comment statement. When SPZAP recognizes a comment, it writes the entire statement to the data set specified for SYSPRINT.

CONSOLE

Indicates that SPZAP 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:

```
AMA116A  ENTER AMASPZAP CONTROL STATEMENT OR END
```

The operator may then enter in any valid SPZAP control statement conforming to the specifications described in 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. SPZAP will then continue processing control statements from the input stream until an end-of-file condition is detected.

Notes:

1. You can enter control statements through the console in either uppercase or lowercase letters, but AMASPZAP does not fold lowercase input to uppercase.
2. You cannot continue a control statement entered through the console.

Figure 212 on page 616 shows how to enable SPZAP control statements to be entered through the console.

```
//CONSOLIN   JOB      MSGLEVEL=(1,1)
//STEP       EXEC     PGM=AMASPZAP
//SYSPRINT   DD       SYSOUT=A
//SYSLIB     DD       DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSIN      DD       *
        CONSOLE
/*
```

Figure 212. Example: Entering SPZAP control statements through console

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 SPZAP control statements are to be entered through the console.

{DUMP|DUMPT} member [csect | ALL | *] [class-name]
Dumps a specific control section or all control sections in a load module in a PDS, a program object in either a PDSE or a z/OS UNIX file. DUMP produces a hexadecimal printout only, while DUMPT prints the hexadecimal data, the EBCDIC translation, and the mnemonic equivalent of the data (see “Reading SPZAP output” on page 620). The variables are:

member
The member name of the load module in a PDS or program object in a PDSE that contains the control section(s) to be dumped. (Note: This variable, ‘member’, must correspond to the name of a member of the PDS or PDSE that is defined by the SYSLIB DD statement.

csect | ALL | *
Defines the name of the particular control section that is to be dumped. To dump all the CSECTs of a load module in a PDS or a program object in a PDSE, specify “ALL” instead of the CSECT name. If you omit the variable entirely, or, for program objects only, code “*”, SPZAP assumes that you mean to dump only the first CSECT in the load module or program object.

If you specify a CSECT name that SPZAP does not find in the member, SPZAP dumps all of the CSECTs in the member.

Note: DUMP or DUMPT applied to a CSECT consisting only of space allocations (DS statements) will produce no output between the statement printback and the dump-completed message.

class-name
Indicates, for program objects only, the class of text that you want to dump. The default is B_TEXT. Specifying B_*, C_*, or D_* causes SPZAP to dump all text classes beginning with the string that precedes the asterisk. If you want to omit the CSECT name and supply a class-name, code a single asterisk for the CSECT name followed by the class-name.

For information about the values you can specify for class name, see *z/OS MVS Program Management: User's Guide and Reference*.

Note: SPZAP does not fold lowercase input to uppercase; be sure to enter class-name in the correct case.

Figure 213 on page 617 shows how to use the DUMPT and DUMP control statements to inspect CSECTs in a program object with multiple text classes in a z/OS UNIX file:

```
//ZAPDUMP    EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD PATH='/u/mydir',PATHDISP=(KEEP,KEEP)
//SYSIN DD *
DUMPT hwz CEEMAIN C_CODE
DUMP  hwz CEESTART B_*
DUMP  hwz ALL C_*
/*
```

Figure 213. Example: Using the DUMP control statement with a class name

SYSLIB DD Statement
Defines the z/OS UNIX directory that contains the module hwz in which the CSECTs to be inspected reside.

SYSIN DD Statement
Defines the input stream that contains the SPZAP control statements.

DUMPT hwz CEEMAIN C_CODE
This control statement requests that the contents of class C_CODE in csect CEEMAIN in module hwz be dumped.

DUMP hwz CEESTART B_*
This control statement requests that the contents of all classes in csect CEESTART in module hwz whose class name begins with B_ be dumped.

DUMP hwz ALL C_*
This control statement requests that the contents of all classes whose class name begins with C_ in any csect in module hwz be dumped.

IDRDATA *xxxxxxxx*
Causes SPZAP to place up to eight bytes of user data into the SPZAP 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 was processed by the linkage editor to include CSECT identification records. The variable is:

xxxxxxxx
Eight (or fewer) bytes of user data (with no embedded blanks) that are to be placed in the user data field of the SPZAP IDR of the named load module. If more than eight characters are in the variable field, only the first eight characters will be used.

Note: The IDRDATA statement is valid only when used in conjunction with the NAME statement. It must follow its associated NAME statement, or the BASE statement associated with a NAME statement, and precede any DUMP, DUMPT, ABSDUMP or ABSDUMPT statement. IDRDATA statements associated with CCHHR statements will be ignored.

NAME member [csect | *] [class-name]
Identifies a CSECT in a load module member of a PDS, a program object member of a PDSE, or a z/OS UNIX file that is to be the object of subsequent VERIFY, REP, SETSSI, or IDRDATA operations. The variables are:

member
The member name of the load module belonging to a PDS, the program object belonging to a PDSE, or a z/OS UNIX file that includes the CSECT that contains the data to be inspected or modified. The load module or the program object must be a member of the 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. If you omit this variable, or, for program objects only, code "*", SPZAP assumes that the first CSECT in the load module contained in a PDS, the program object contained in a PDSE, or a z/OS UNIX file is the one to be used. If there is only one CSECT in the load module or program object, this variable is not necessary.

If you specify a CSECT name that SPZAP does not find in the member you name, then SPZAP does not perform any requested processing. Instead, it produces hexadecimal dumps of all CSECTs in the member. (The class of text dumped is specified on the class-name variable, and the default is B_text.)

class-name
Indicates, for program objects only, the class of text that you want to modify. The default is B_text. If you want to omit the CSECT name and supply a class-name, code a single asterisk for the CSECT name, followed by the class-name.

For information about the values you can specify for class name, see *z/OS MVS Program Management: User's Guide and Reference*.

Note: SPZAP does not fold lowercase input to uppercase; be sure to enter class-name in the correct case.

Note that you can define more than one NAME statement in your input to SPZAP. However, the VERIFY, REP and SETSSI statements associated with each NAME statement must immediately follow the NAME statement to which they apply.

NAME member
Identifies the member of a data library that is to be the object of subsequent VERIFY, REP, ABSDUMP, ABSDUMPT or RECORD operations. The variable is:

member
The member name of a data library whose contents are to be displayed, verified and/or replaced.

RECORD *nnnnnnnn*
This statement identifies a particular record in a member of a PDSE data library and must follow a *NAME member* statement where *member* specifies the name of the member. The combination of NAME and RECORD defines the record for which VER|VERIFY and possible REPs are to be performed. *nnnnnnnn* consists of 1 to 8 decimal digits and specifies the relative record of interest. Leading zeroes are ignored. For example, the first record of a member may be specified as 1 or 01 or 00000001.

REP offset data
Modifies data at a specified location in a CSECT or physical record that was previously defined by the NAME, NAME/RECORD combination, or CCHHR statement. The data specified on the REP statement will replace the data at the record or CSECT location stipulated in the offset variable field.

SPZAP issues message AMA122I to record the contents of the specified location as they were before the change was made.

Note: IBM recommends that, before you replace any data, you always use VER/VERIFY to make sure that the contents you are going to change with the REP function are what you expect. The offset and length that you specify on the VER/VERIFY statement, however, do not need to match any following REP statement exactly; a single successful VERIFY can validate multiple following REP statements.

offset
Provides the hexadecimal displacement of 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 (0D, 02C8, 001C52).

If the offset value is outside the limits of 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; that is, offset zero is the first byte of the key, not of the data.

data
Defines the bytes of data to be inserted at the location. As with the offset variable, the number of bytes of data defined must be specified as a multiple of two hexadecimal digits. If desired, the data within the variable 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 variable may look like this:

```
4160B820 (without commas)
    or like this:
4160,B820 (with commas).
```

If all the data to be modified does not fit into one REP statement (72 bytes), you can code another REP statement.

Notes:

1. Remember that SPZAP automatically updates the system status index (SSI) when it successfully modifies the CSECT named or implied on the previous NAME statement.
2. If you are performing multiple VERIFY and REP operations on a CSECT, make sure that all the VERIFY statements precede all the REP statements. This procedure ensures that all REP operations are ignored if one VERIFY reject occurs.
3. You are not required to supply a VERIFY statement before the first REP statement; however, when SPZAP encounters a VERIFY statement, it must be satisfied before SPZAP processes any following REP requests.
4. When you access a record in the VTOC (for example, the data set control block (DSCB)) for modification, SPZAP issues the message AMA117D to the console. No message is issued, however, when an ABSDUMPT operation is performed on the VTOC.
5. The REP statement associated with a NAME or CCHHR must precede any DUMP, DUMPT, ABSDUMP, and ABSDUMPT statements.

SETSSI xxyynnnn

Places user-supplied system status information in the directory entry for the load module member in a PDS or program object member in a PDSE. The SSI entry must have been created when the load module or program object member was link edited. The variable is:

xxyynnnn

Four 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 indicating the following:

```
xx   - change level
yy   - flag byte
nnnn - modification serial number
```

If SPZAP detects an error in any previous VERIFY or REP operation, the SETSSI function is not performed.

Note: Because all bits in the SSI entry are set (reset) by the SETSSI statement, be very careful when using it to avoid altering the vital maintenance-status information. SPZAP issues message AMA122I to record the SSI as it was before the SETSSI operation was performed. See "Updating the System Status Index (SSI)" on page 604.

{VERIFY|VER} offset expected-content

Causes the data at a specified location within a CSECT or physical record to be compared with the data supplied in the statement.

offset

The hexadecimal displacement of 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, such as 0D, 021C, 014682. If this offset value is outside the limits of the CSECT or data record defined by the preceding NAME, NAME/RECORD, or CCHHR statement, the VERIFY statement is rejected. If this offset value plus the length of the expected-content string is outside the limits of the CSECT or record defined by the preceding NAME, NAME/RECORD combination, or CCHHR statement, the VERIFY statement is rejected and flagged in error. When inspecting a record with a key, the length of the key should also be considered in the calculation of the displacement; that is, 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 variable, 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, expected content might look like this:

```
5840C032 (without commas),
or like this:
5840,C032 (with commas)
```

If all the data does not fit into one VERIFY statement (80-byte logical record), then another VERIFY statement must be defined.

Note: If the two fields being compared are not in agreement, that is, if the VERIFY operation is rejected, no succeeding REP or SETSSI operations are performed until the next NAME or CCHHR control statement is encountered. SPZAP provides a formatted dump of each CSECT or record for which a VERIFY operation failed. Also, note that a VER statement associated with a NAME or CCHHR must precede any DUMP, DUMPT, ABSDUMP and ABSDUMPT statements.

Reading SPZAP output

SPZAP provides two different dump formats for the purpose of checking the data that has been verified 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.

Formatted hexadecimal dump

When DUMP or ABSDUMP is the control statement used, the resulting printout is a hexadecimal representation of the requested data. Figure 214 on page 621 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 (ABSDUMP only), the length of the record, the class of text (program objects only), and the names of the member and the CSECT that contain the data being printed (if the dump is for 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 is printed under the last line of the dump printout:

```
AMA113I  COMPLETED DUMP REQUIREMENTS
```

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. Not all characters are translated to EBCDIC, only upper case and a few special characters are translated. Others, such as lowercase letters are not translated and their translations are substituted by periods.

```
DUMP    IEHMVESN   ALL

**CCHHR-  0022001108   RECORD  LENGTH-  0850            MEMBER NAME  IEHMVESN  CSECT NAME  IEHMVSSN
  000000    47F0F014    0EC5E2D5    60E6D9C1    D760E4D7    60606000    90ECD00C    189F5010    D0484110
  000020    D04850D0    10045010    D00818D1    5810D000    9200D00C    92FFD008    9140C20A    4780904A
  000040    9200C2F4    D20EC2F5    C2F49108    C20C4710    90E69500    C2FC4780    9064D203    C3009664
  000060    9200C2FC    D203C320    C31C95FF    C32A4770    908A4180    C00141F0    001450E0    964845E0
  000080    951858EO    96484520    95705820    C2640700    45109098    00000000    50210000    92801000
  0000A0    0A1495FF    C3274780    910A9108    C20C4710    91685820    C2749581    20114770    90D09102
  0000C0    C2084710    90F89110    C2084710    90F80700    451090D8    00000000    50210000    92801000
  0000E0    0A1447F0    910A9180    C1FC4780    9168947F    C1FC47F0    908A0700    45109100    00000000
  000100    50210000    92B01000    0A1495FF    C3344780    96DC41A0    C0089200    C2F49200    C2F89200
  000120    C2FC9200    C30094F7    A0429101    C2094780    91689102    C2094710    91685810    C27458F0
  000140    10149601    101748E0    F0044CE0    F0069101    10204710    915E4100    E00847F0    91624100
  000160    E0104110    F0000A0A    1B444340    C2245810    C2245830    C27C4833    000E95FF    30024780
  000180    918CD505    30041004    47F09192    D505301C    10044780    91E84111    000C4640    917A4140
  0001A0    000C1B14    41400001    D2031000    301095FF    30024780    91C0D205    10043004    47F091C6
  0001C0    D2051004    301C1B33    403096FC    D201100A    96FC4130    00019580    10024780    91E24030
  0001E0    96FCD201    100A96FC    5010C224    4240C224    9110C208    47109204    9102C208    47109204
  000200    47F09236    5810C224    95801002    47709236    D20196FC    100A4820    96FC4122    00014130
  000220    00011932    4770922C    41220001    402096FC    D201100A    96FC9140    C2094710    92B85820
  000240    00105822    00284832    00005930    92B44780    92B81233    47809268    91203012    47809268
  000260    91023003    47109270    41220002    47F09246    D203C228    C2005820    C200D203    200030
  000280    D2052004        C4122   000C5020    C2009640    C20947F0                C2004143    00

  000600    41F0C014    D205F000    100441FF    0006D201    41FF0005    41FF0001    4111000C    46009604
  000620    F0004EE0    C080F337    F001C080    96F0F004    58FF0000    D219C014    F0019200    C33C07FE
  000640    58E09648    07FE1BDD    7FFF0000    58F09660    D503C31C    97004780    96D89500    C3284780
  000660    00000708    04000668    41800668    1BF8189F    4770969A    96FFC334    07FE58B0    C32058F0
  000680    96C25881    00001288    478096D8    95801008    1BBB43B0    C32806B0    42B0C328    41F00008
  0006A0    1000D24F    F000B000    41BB0050    50B0C320    0A0AD707    C31CC31C    1BFF07FE    9600C334
  0006C0    07FE58B0    C31C4100    0280181B    41110000    58E09648    45209570    47F09112    8CA00000
  0006E0    4180C001    41F00018    50E09648    45E09518
  000700    00000000    43A0400B

**CCHHR-  0022001108   RECORD  LENGTH-  0850            MEMBER NAME  IEHMVESN  CSECT NAME  IEHMVSSN
  000000    00000724    0000073F    00000750    00000761    00000775    00000793    000007E6    19E4D5C9
  000020    E340D9C5    C340D6D9    40E4D5D3    C1C2C5D3    C5C440E3    C1D7C50F    C9C5C8F3    F6F1C940
  000040    C4C1E3C1    40E2C5E3    0F404040    40404040    40C4C1E3    C140E2C5    E312C3D6    D7C9C5C4
  000060    40E3D640    E5D6D3E4    D4C54DE2    5D1CD5D6    E340D4D6    E5C5C460    C3D6D7C9    C5C440E3
  000080    D640E5D6    D3E4D4C5    4DE25D51    C9C5C8F3    F3F1C940    E4E2C5D9    40D3C1C2    C5D3E240
  0000A0    C1D9C540    D5D6E340    D4D6E5C5    C461C3D6    D7C9C5C4    4B40D5D6    40E4E2C5    D940D3C1
  0000C0    C2C5D340    E3D9C1C3    D240C1D3    D3D6C3C1    E3C5C440    C6D6D940    C9D5D7E4    E34B66C9
  0000E0    C5C8F3F3    F5C940D7    C5D9D4C1    D5C5D5E3    40C961D6    40C5D9D9    D6D940E6    C8C9D3C5
  000100    40E6D9C9    E3C9D5C7    40E4E2C5    D940D6E4    E3D7E4E3    40E3D9C1    C9D3C5D9    40D3C1C2
  000120    C5D3E24E    40D5D640    D4D6D9C5    40D3C1C2    C5D3E240    E6C9D3D3    40C2C540    D7D9D6C3
  000140    C5E2E2C5    C44B58B0
HMA1131  COMPLETED DUMPB REQUIREMENTS
```

Figure 214. Sample formatted hexadecimal dump

Figure 215 on page 622 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 not 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 have been set not to be translated to valid printable characters.

```
DUMPT IEANUC05 SUTFPL59

**CCHHR- 01AB000416   RECORD LENGTH- 000068         MEMBER NAME  IEANUC05  CSECT NAME  SUTFPL59

 000000   47F0 F01C  16E2 E4E3  C6D7 D3F5  F940 F9F8     F1F0 F540  C8C2  C2F6 F6F0 F600  90EC D00C   *.00..SUTFPL50 98*
          BC   SRP   OR                    MVZ  CP   CP       MVO                             STM         *105 HBB6606.....*
 000020   18BF 41C0  BEFF 41F0  0000 5800  B064 18A1     50D0 A004  50A0  D008 98F1 D010  18DA B365   *.......0........*
          LR   LA    ICM  LA         L          LR       ST         ST         LM         LR   LXR    *&...&....1......*
 000040   0014 B362  0048 ED22  A00C 0006  ED32 B00C     C00E 58D0  D004  98EC D00C 07FE  0000 0000   *................*
               LTXR       ED--                 ED--                L             LM        BCR                     *................*
 000060   0000 0048  0000 0048                                                                    *........        *

AMA113I COMPLETED DUMP REQUIREMENTS

AMA100I AMASPZAP PROCESSING COMPLETED
```

Figure 215. Sample translated dump

Figure 216 on page 623 shows CSECT output (obtained through DUMP/DUMPT) for a program object module.

Notes:

1. There are no **CCHHR** values. The program management binder manages its own DASD storage and returns no physical location.
2. **RECORD LENGTH: indicates length of the CSECT or module, not the length of the physical record containing the CSECT or module.
3. Program management binder returns no text for named or unnamed common areas. The length of the common section will be indicated. Message AMA152I indicates that no text has been returned.
4. SPZAP displays MEMBER NAME and CSECT NAME on as many lines as necessary. The names can be as long as 1024 characters.
5. SPZAP labels common storage in a program object with the tag COMMON NAME instead of CSECT NAME. Named common displays that name. Unnamed common is flagged as $BLANK COMMON. Private code is displayed with the subheading CSECT NAME: $PRIVATE CODE.
6. **UNINITIALIZED DATA SKIPPED may appear.
7. IGWSPZAP is the part of SPZAP that receives control when accessing or updating program objects in a PDS/E or z/OS UNIX file, or data in a PDS/E. Listings and messages refer to AMASPZAP when processing a PDS or IGWSPZAP when processing a PDSE or a z/OS UNIX file.

```
IGWSPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
   DUMP     MAINRTN   ALL                                              01770000

 **RECORD LENGTH: 000000C0 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                            CSECT NAME: $PRIVATE CODE
 00000000  90ECD00C   05C050D0   C02241E0   C01E50E0     D00818DE   58D0D004   58E0D00C   980CD014
 00000020  07FE0000   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 00000040  00000000   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 00000060  00000000   00000000   00000000   E3C8C9E2     40C9E240   C140D4C5   E2E2C1C7   C540C4C5
 00000080  C6C9D5C5   C440C9D5   40C140E4   D5D5C1D4     C5C440C3   E2C5C3E3   40404040   40404040
 000000A0  40404040   40404040   40404040   40404040     40404040   40404040   40404040

 **RECORD LENGTH: 00000058 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                           COMMON NAME: AAAAAAAA
 AMA152I NO TEXT DATA FOR THIS SECTION

 **RECORD LENGTH: 00000108 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                            CSECT NAME: SUBRTN
 00000000  90ECD00C   05C050D0   C06241F0   C05E50FD     000818DF   4510C034   001E8000   D5D6E640
 00000020  C9D540E3   C8C540C3   C1D3D3C5   C440D9D6     E4E3C9D5   C5400000   FF800A23   58B0C0A6
 00000040  D24FB004   C0AA9200   B002D201   B000C0FA     184B1814   0A231BFF   58D0C062   98ECD00C
 00000060  07FE0000   00000001   00000001   00000001     00000001   00000001   00000001   00000001
 00000080  00000001   00000001   00000001   00000001     00000001   00000001   00000001   00000001
 000000A0  00000001   00000001   00000001   00000448     C7D9C5C5   E3C9D5C7   E240C6D9   D6D440E3
 000000C0  C8C540E4   D5C3D6D4   D4D6D540   40404040     40404040   40404040   40404040   40404040
 000000E0  40404040   40404040   40404040   40404040     40404040   40404040   40404040   40404040
 00000100  00500000   00000000

 **RECORD LENGTH: 00000058 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                           COMMON NAME: $BLANK COMMON
 AMA152I NO TEXT DATA FOR THIS SECTION

 **RECORD LENGTH: 00000168 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                            CSECT NAME: MAINRTN
 00000000  90ECD00C   05C050D0   C06241F0   C05E50F0     D00818DF   4140C0B2   18140A23   5850C0AA
 00000020  D24FC0B6   50001814   0A235850   C0AED24F     C0B65000   18140A23   58B0C0A6   D24FB004
 00000040  C10A9200   B002D201   B000C15E   184B1814     0A2358F0   C15A05EF   58D0C062   98ECD00C
 00000060  1BFF07FE   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 00000080  00000000   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 000000A0  00000000   00000000   00000000   00000168     8000022C   800002EC   00500000   C7D9C5C5
 000000C0  E3C9D5C7   E240C6D9   D6D440E3   C8C540C3     C1D3D3C9   D5C740D9   D6E4E3C9   D5C5E240
 000000E0  D4C1C9D5   40E2C5C3   E3C9D6D5   40404040     40404040   40404040   40404040   40404040
 00000100  40404040   40404040   40404040   00000000     C8C940C6   D9D6D440   E3C8C540   C3C1D3D3
 00000120  C5D9E240   C3D6D4D4   D6D540E2   C5C3E3C9     D6D54040   40404040   40404040   40404040
 00000140  40404040   40404040   40404040   40404040     40404040   40404040   40404040   40404040
 00000160  00000340   00500000

 **RECORD LENGTH: 000000C0 CLASS: B_TEXT   MEMBER NAME: MAINRTN
                                            CSECT NAME: $PRIVATE CODE
 00000000  90ECD00C   05C050D0   C02241E0   C01E50E0     D00818DE   58D0D004   58E0D00C   980CD014
 00000020  07FE0000   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 00000040  00000000   00000000   00000000   00000000     00000000   00000000   00000000   00000000
 00000060  00000000   00000000   00000000   E3C8C9E2     40C9E240   C4C1E3C1   40C4C5C6   C9D5C5C4
 00000080  40C9D540   C1D5D6E3   C8C5D940   C3E2C5C3     E340E6C9   E3C840D5   D640D5C1   D4C54040
 000000A0  40404040   40404040   40404040   40404040     40404040   40404040   40404040

 AMA113I COMPLETED DUMP REQUIREMENTS
 AMA100I IGWSPZAP   PROCESSING COMPLETED
```

Figure 216. Sample formatted hexadecimal dump for PDSE program object module

Figure 217 on page 624 shows output for a member of a PDSE data library. *ABSDUMPT 0001 0500* would have been preceded by a *NAME membername* statement (not shown).

Note: There are no **CCHHR** values. RECORD NUMBER: shows the 8 digit value of the relative record number of the member being printed. RECORD LENGTH: shows the length of the record, while MEMBER NAME: shows the member name as it appears on the *NAME membername* statement.

```
IGWSPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
  NAME     USERDATA                                                         03520001
  ABSDUMPT 0001  0050                                                       03530001

**RECORD NUMBER: 0000000001, RECORD LENGTH: 000050        MEMBER NAME: USERDATA
 000000   02C5 E2C4  4040 4040  4040 0030  4040 0001     C7C5 E3C3  E2C5 C3E3  0000 0000  0200 0530   *.ESD............*
                     STH  STH   STH        STH                                                 BALR   *USERDATA........*
 000020   E7E3 D5C4  E2E3 D240  0200 0000  4040 4040     E2E3 C1D9  E3C4 4040  0200 0000  4040 4040   *XTNDSTK.........*
               CLC        MVC              STH  STH                      STH              STH  STH    *STARTD..........*
 000040   4040 4040  4040 4040  F0F2 F4F8  F0F0 F0F1                                                  *........02480001*
          STH  STH   STH  STH   SRP        SRP  SRP                                                   *................*

**RECORD NUMBER: 0000000002, RECORD LENGTH: 000050        MEMBER NAME: USERDATA
 000000   02C5 E2C4  4040 4040  4040 0020  4040 0004     D7D9 D5E3  D3C9 D5C5  0200 0000  4040 4040   *.ESD............*
                     STH  STH   STH        STH           XC   CLC   MVZ  CLC              STH  STH    *PRNTLINE........*
 000020   D7C1 D4C1  F1F4 F0C9  0200 0000  4040 4040     4040 4040  4040 4040  4040 4040  4040 4040   *PAMA140I........*
          XC   NC    MVO  SRP              STH  STH      STH  STH   STH  STH   STH  STH   STH  STH    *................*
 000040   4040 4040  4040 4040  F0F2 F4F9  F0F0 F0F1                                                  *........02490001*
          STH  STH   STH  STH   SRP        SRP  SRP                                                   *................*

**RECORD NUMBER: 0000000003, RECORD LENGTH: 000050        MEMBER NAME: USERDATA
 000000   02E3 E7E3  4000 0000  4040 0038  4040 0001     47F0 F016  10C7 C5E3  C3E2 C5C3  E340 40F9   *.TXT............*
                     STH        STH        STH           BC   SRP   LPR                        STH    *.00..USERDATA..9*
 000020   F24B F2F4  F400 90EC  D00C 18CF  1FFF 43F0     C52C 1F00  BF07 C52D  581D 0008  1A01 58FD   *2.244..........0*
          PACK PACK       STM        LR    SLR  IC            SLR   ICM        L          AR   L      *E.....E.........*
 000040   0000 190F  47D0 C050  F0F2 F5F0  F0F0 F0F1                                                  *.......&02500001*;
               CR    BC         SRP        SRP  SRP                                                   *................*

         ....

**RECORD NUMBER: 0000000028, RECORD LENGTH: 000050        MEMBER NAME: USERDATA
 000000   02D9 D3C4  4040 4040  4040 0020  4040 4040     0002 0001  1C00 003C  0003 0001  0C00 0468   *.RLD............*
               MVZ   STH  STH   STH        STH  STH                 MR                   BASSM SPM    *................*
 000020   0004 0001  0C00 046C  0005 0001  0C00 0470     4040 4040  4040 4040  4040 4040  4040 4040   *.......%........*
                    BASSM SPM             BASSM SPM      STH  STH   STH  STH   STH  STH   STH  STH    *................*
 000040   4040 4040  4040 4040  F0F2 F7F5  F0F0 F0F1                                                  *........02750001*
          STH  STH   STH  STH   SRP        SRP  SRP                                                   *................*

**RECORD NUMBER: 0000000029, RECORD LENGTH: 000050        MEMBER NAME: USERDATA
 000000   02C5 D5C4  4040 4040  4040 4040  4040 4040     4040 4040  4040 4040  4040 4040  4040 4040   *.END............*
               CLC   STH  STH   STH  STH   STH  STH      STH  STH   STH  STH   STH  STH   STH  STH    *................*
 000020   F2F5 F6F6  F8F9 F6F2  F0F1 40F0  F2F0 F1F9     F2F2 F4F4  C37D D7D3  61C1 E27D  4040 F0F1   *2566896201.02019*
          PACK       ZAP        SRP  STH   PACK MVO      PACK             XC              STH  SRP    *2244C'PL/AS'..01*
 000040   F0F4 F9F2  F2F4 F440  F0F2 F7F6  F0F0 F0F1                                                  *0492244.02760001*
          SRP  CP    PACK       SRP        SRP  SRP                                                   *................*
AMA113I COMPLETED DUMP REQUIREMENTS
AMA100I IGWSPZAP   PROCESSING COMPLETED
```

Figure 217. Sample translated dump for PDSE data library

Figure 218 on page 625 shows sample output from a successful SPZAP when PARM=PRECHECK is specified.

```
AMASPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
AMA167I SYSIN PRECHECKING STARTED
  NAME EXSPZAP EXSPZAP                                                      00140000
  VER 0020 C2E8                                                             00150000
  REP 0020 9999                                                             00160000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  REP 0020 C2E8                                                             00170000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  IDRDATA CHKSUMOK                                                          00180000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  CHECKSUM 00621F69                                                         00190000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
AMA132I CHECKSUM WAS CORRECT , IS NOW 0.
 /*                                                                         00210000
AMA167I SYSIN PRECHECKING COMPLETED
========================================
AMA168I SYSIN PROCESSING STARTED
  NAME EXSPZAP EXSPZAP                                                      00140000
  VER 0020 C2E8                                                             00150000
  REP 0020 9999                                                             00160000
AMA122I OLD DATA WAS C2E8
  REP 0020 C2E8                                                             00170000
AMA122I OLD DATA WAS 9999
  IDRDATA CHKSUMOK                                                          00180000
  CHECKSUM 00621F69                                                         00190000
AMA132I CHECKSUM WAS CORRECT , IS NOW 0.
AMA125I EXSPZAP  IDR COUNT = 0002 (MAX=0019)
 /*                                                                         00210000
AMA168I SYSIN PROCESSING COMPLETED
AMA100I AMASPZAP PROCESSING COMPLETED
```

Figure 218. Sample report from a successful SPZAP with PARM=PRECHECK

Figure 219 on page 625 shows sample output from SPZAP when errors are found with PARM=PRECHECK specified.

```
AMASPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
AMA167I SYSIN PRECHECKING STARTED
  NAME EXSPZAP EXSPZAP                                                      00140000
  VER 0020 C2E8                                                             00150000
  REP 0020 9999                                                             00160000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  REP 0020 C2E8                                                             00170000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  IDRDATA CHKSUMOK                                                          00180000
AMA167I SYSIN PRECHECKING SKIPS UPDATES
  CHECKSUM 00621F68                                                         00190000
AMA133I CHECKSUM ERROR. NO-GO SWITCH SET
AMA132I CHECKSUM WAS 00621F69, IS NOW 0.
 /*                                                                         00200000
AMA167I SYSIN PRECHECKING FOUND ERRORS
AMA100I AMASPZAP PROCESSING COMPLETED
```

Figure 219. Sample report from SPZAP when errors are found with PARM=PRECHECK

Chapter 18. IEWSIGN: Sign, unsign, and report load modules

IEWSIGN, the signing utility on z/OS, provides the following functions:

- Sign a load module in a partitioned data set (PDS). The utility adds a signature to the load module and marks it as signed.
- Unsign a load module in a PDS. The utility removes the signature from the load module and marks it as un-signed.
- Report on a load module in a PDS. The utility reports signing-related information, such as the time the load module was signed, the signing algorithm, and the certificate fingerprint.

IEWSIGN must be invoked by either JCL that uses EXEC PGM=xxx, call, or LINK.

It resides in SYS1.SIEAMIGE.

Usage notes

The following list contains usage notes for the signed load module:

- To copy a signed load module from one PDS to another PDS, IEBCOPY with the control statement COPY should be used. Do not use IEBCOPY with the control statement COPYMOD, which causes the signature in the destination load module to become invalid. Do not use the z/OS UNIX command **cp**, which causes the destination load module to become unsigned.
- Copying a signed load module from a PDS to a PDSE causes the destination program to become unsigned.
- Relinking a signed load module causes the resultant program to become unsigned.
- Reprocessing a signed load module with the binder API SAVEW causes the resultant program to become unsigned.
- Renaming a signed load module causes its signature to become invalid.

Parameters of IEWSIGN

Parameters of IEWSIGN have the following rules:

- For JCL, parameters can be provided by PARM='..' or by PARMDD=xx with the data within the xx DD.
- For call or LINK, parameters consist of a halfword length followed by the parameter string, as the first parameter pointed to by the parameter list located by the PARAM keyword (and thus located by register 1 on entry to the target routine).
- Parameters are of the form keyword=value, separated from one another by a single comma.
- Parameters can be in any order.
- Optional parameters need not be provided.

The following table lists the parameters of IEWSIGN along with their descriptions:

Table 77. Parameters of IEWSIGN

Parameter name	Description
Action	Specifies the action to be run. There is no default value. It must be specified explicitly. Its eligible values are as follows: • Sign • Unsign • Report
State	For Action=Sign and Action=Report, it determines which name-matched members will be processed. For Action=Unsign, this parameter is ignored, since only signed load modules are processed. Its eligible values are as follows: **Unsigned** Only unsigned name-matched members will be processed. **Signed** Only signed name-matched members will be processed. **All** All name-matched members will be processed. This is the default value. When Action=Sign and State is Signed or All, existing signed modules will be re-signed.
RC4LIM/ RC8LIM	This integer specifies the limits for return code 4 / return code 8. Valid range is from 1 to 2147483647. If any limit is reached, this utility will terminate. The default value of RC4LIM is 2147483647. The default value of RC8LIM is 1 for ACTION=SIGN and ACTION=UNSIGN, and 2147483647 for ACTION=REPORT. This utility will terminate immediately if a return code 12 or higher occurs.
Verbose	Specifies the content that is provided in SYSPRINT. **Yes** Specifying this option provides more detailed content. **No** Specifying this option provides less detailed content. .

Table 77. Parameters of IEWSIGN (continued)	
Parameter name	**Description**
ReportLevel	Specifies the level of checking/printed information when ACTION=REPORT. **1** Only check whether a load module is signed. This is the default value. **2** Includes all reports for ReportLevel=1, and the text size, link time, signing time, binder version, hash algorithm, signing algorithm, certificate fingerprint. **3** Includes all reports for ReportLevel=2, and whether the hash in signature is valid.

The utility's processing of all parameters is not case-sensitive.

Parameters in the PARM string are separated by commas.

Data definitions (DD) of IEWSIGN

The following table lists the IEWSIGN data definitions and their descriptions:

Table 78. IEWSIGN data definitions	
DD name	**Description**

SYSPRINT	This is a required DD. It is valid for all Action values.
	This data set contains the IEWSIGN processing messages or reports. The specification for the data set is as follows:
	`//SYSPRINT DD SYSOUT=*`
	If DCB attributes are expected, the following recommendation is provided:
	`LRECL   RECFM`
	`121     FBA`

INFILE	This is a required DD. It is valid for all Action values. This DD specifies the source PDS library where load modules are read from. Concatenation of multiple data sets is not allowed. IEWSIGN will fail if the data set is not a PDS.

Table 78. IEWSIGN data definitions (continued)

DD name	Description
OUTFILE	When Action=Sign or Action=Unsign, this DD is required. When Action=Report, this DD is not required. This DD specifies the destination PDS library where load modules are written to. Concatenation of multiple data sets is not allowed. The signing utility will fail if the data set is not a PDS. In-place signing is supported by specifying the same DSN in the INFILE and OUTFILE. If the blocksize of OUTFILE is not 0, it must be equal to or larger than the blocksize of INFILE. Additionally, it must be equal to or larger than 1024. The maximum value can be 32760. If the blocksize of OUTFILE is 0, the IEWSIGN utility will set it to the blocksize of the INFILE data set.
INCLUDE	This is an optional DD. It is valid for all Action values. This DD specifies members to be included from INFILE. If this DD is unspecified, all members in INFILE will be included. The standard specification for this data set is as follows: `//INCLUDE DD *` If a regular data set is provided, it needs to be RECFM=FB and LRECL=80.
EXCLUDE	This is an optional DD. It is valid for all Action values. This DD specifies members to be excluded from INFILE. If this DD is unspecified, no members in INFILE will be excluded. The standard specification for this data set is as follows: `//EXCLUDE DD *` If a regular data set is provided, it needs to be RECFM=FB and LRECL=80.

Both INFILE and OUTFILE must be a PDS whose record format is U. In addition, the IEWSIGN utility will not process a member in INFILE if any of the following are true:

- It is not a load module.
- It is a load module with an overlay attribute.
- It is a load module without TEXT.

Rules for INCLUDE and EXCLUDE

Both INCLUDE and EXCLUDE specify one or multiple names, which are used to match primary members and aliases in INFILE. The following rules apply:

- A name must obey PDS member naming rules.
- Multiple names are not allowed on a single line.
- A name must not be continued to the next line.
- Blank characters are allowed before and after a name.
- Comments are supported with the following conditions:

- A line will be treated as a comment if the first nonblank character is “#”.
- A blank line, where all characters are space (its hex value is 0x40), is acceptable.

- Wildcards are supported. Both “*” and “?” can be used in a name. . “*” matches “0 or more characters” and “?” matches “exactly one character”. The following are some examples:
  - `*`
  - `ABC*`
  - `ABC??DEF`
  - `X*YYY`
- Only the first 72 characters in a line are parsed.

IEWSIGN uses the following steps to determine which members should be processed (for example, signed, unsigned or reported).

Table 79. Steps for IEWSIGN member processing

Step number	Step instructions
Step 1	Retrieve a list of names of all primary members of INFILE whose signing state is the one required by parameter STATE.
Step 2	If INCLUDE is not specified, skip this step. If INCLUDE is specified, remove a primary member from this list if both of the following are true: 1. The name of the primary member does not match any name specified in INCLUDE, and 2. None of its alias names matches any name specified in INCLUDE.
Step 3	If EXCLUDE is not specified, skip this step. If EXCLUDE is specified, remove a primary member from this list if either of the following are true: 1. The name of the primary member matches any name specified in EXCLUDE, or 2. One of its alias names matches any name specified in EXCLUDE.
Step 4	All primary members remaining in the list are processed.

Note: For Action=Sign, when a primary member is being signed, in addition to adding load module signing records, IEWSIGN will update the directory entries of the primary member and all its aliases. Therefore, when a primary member becomes signed, all its aliases also become signed. For Action=Unsign, analogous processing is performed.

A detailed example of INCLUDE/EXCLUDE is provided in next section as Example 1.

Examples

1. The following provides an example of INCLUDE/EXCLUDE processing. The conditions for this example are as such:
   - A PDS has 4 primary members: M1, M2, M3 and M4.
     - M1 has one alias A11;
     - M2 has two aliases A21 and A22;
     - M3 and M4 have no alias.
     - M1, M2 and M3 are unsigned.
     - M4 is signed.

- IEWSIGN is called with parameter Action=Sign,State=Unsigned.
- INCLUDE has two lines

```
M1
A21
```

- EXCLUDE has one line:

```
A11
```

Results of the processing are as follows:

Step 1 results
IEWSIGN receives a name list of all unsigned primary members of INFILE. At the end of step 1, M1, M2, M3 are in the list.

Step 2 results
M1 is kept, as its name matches a filter specified at the first line of INCLUDE. M2 is kept, as its alias A21 matches a filter specified on the second line of INCLUDE. M3 is removed, as it and all its aliases don't match any filters in INCLUDE. At the end of step 2, M1,M2 are in the list.

Step 3 results
M1 is removed, as its alias A11 matches a filter specified at the first line of EXCLUDE. M2 is kept, as it and all its aliases do not match any filters in EXCLUDE. At the end of step 3, M2 is in the list.

Step 4 results
IEWSIGN begins to sign M2. In this step, a signature is added to the load module records of M2. In addition, the directory entries of M2, A21 and A22 are all updated together. Therefore, M2, A21 and A22 all become signed.

2. The following is a JCL example that signs all load modules in place. In this example, both INFILE and OUTFILE use the same DSN. As a result, signed modules will be saved into its original PDS.

```
//SIGN EXEC PGM=IEWSIGN,PARM='Action=Sign'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=SYS1.LPALIB,DISP=SHR
//OUTFILE  DD DSN=SYS1.LPALIB,DISP=SHR
```

Note: In-place signing requires the OUTFILE data set to have room for both the original modules and for the updated versions of those modules, since the storage space of the original modules cannot be reused during the processing.

3. The following is a JCL example that signs all unsigned load modules specified by INCLUDE/EXCLUDE:

```
//SIGN EXEC PGM=IEWSIGN,PARM='Action=Sign,State=Unsigned'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE    DD DSN=SYS1.LINKLIB,DISP=SHR
//OUTFILE   DD DSN=SYS1.SIGN.LIB,DISP=(NEW,PASS),DSNTYPE=PDS,
//             SPACE=(CYL,(500,500,5)),UNIT=3390
//INCLUDE   DD *
# this is a comment line
AMBLIST
IEHMVE*
//EXCLUDE    DD *
IEHMVE2
```

For example, if INFILE has the following six unsigned members: AMBLIST, AMBLIST2, IEHMVE1, IEHMVE2, IEHMVE3, IEHMVE4, then the matched members are as follows: AMBLIST, IEHMVE1, IEHMVE3, IEHMVE4.

4. The following is a JCL example that unsigns all signed load modules in-place:

```
//UNSIGN EXEC PGM=IEWSIGN,PARM='Action=Unsign'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
```

```
//INFILE    DD DSN=SYS1.LINKLIB,DISP=SHR
//OUTFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
```

5. The following is a JCL example that reports all load modules:

```
//REPORT EXEC PGM=IEWSIGN,PARM='Action=Report’
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
```

6. The following is a JCL example that reports load modules specified by INCLUDE or EXCLUDE:

```
//REPORT EXEC PGM=IEWSIGN,PARM='Action=Report,Verbose=YES'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
//INCLUDE    DD *
AMBLIST
IEHMVE*
//EXCLUDE    DD *
IEHMVE2
```

7. The following two examples show the use of SYSPRINT with Action=Sign. To better understand these examples, know that the contents of SYSPRINT consists of two parts:
   - A selection part, which indicates which members will be selected by INCLUDE or EXCLUDE from INFILE. This part is the same for all ACTION values.
   - A processing part, which displays the results of actions such as signing, unsigning, and reporting. This part is different for each ACTION value.

   a. The following is an example of the selection part for Action=Sign:

      Note: In the selection part of the example, report lines in italic type are printed only when Verbose=Yes. (in the example, the italicized section starts with the line "`Member/Alias(es) in INFILE with STATE=UNSIGNED`" and goes to the end of the example)

```
Invocation parameters: ACTION=SIGN,STATE=UNSIGNED,VERBOSE=YES
Execution  Parameters:
ACTION=SIGN,STATE=UNSIGNED,VERBOSE=YES,RC4LIM=2147483647,RC8LIM=1,REPORTLEVEL=1

DD        Data Set Name                                Volume       Block Size
INFILE    SYS1.LPALIB                                  BPX111       32760
OUTFILE   SYS1.LPALIB                                  BPX222       32760

INFILE
summary:

          Unsigned primary members
1
          Unsigned aliases
2
          Signed   primary members
0
          Signed   aliases
0
          Non-LM   members
1
          Overlay       LM
0
          Zero-TEXT     LM              0

Member/Alias(es) in INFILE with STATE=UNSIGNED
Member      Alias(es)
ASM         AL1      AL2

Including members specified in INCLUDE ...
<NONE>

Member/Alias(es) selected after INCLUDing
Member      Alias(es)
ASM         AL1      AL2

Excluding members specified in EXCLUDE ...
<NONE>

Member/Alias(es) selected after EXCLUDing
```

```
Member      Alias(es)
ASM         AL1      AL2
```

Each line in SYSPRINT is one of the two types:

Report line
This line has no message ID. The text is provided for informational purposes.

Message line
This line has a message ID. These lines are only for warning and error messages. These messages are documented in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

b. The following is an example of the processing part for Action=Sign:

```
Signing results:
ASM      Successful

OUTFILE
summary:

          Unsigned primary members
0
          Unsigned aliases
0
          Signed   primary members
1
          Signed   aliases
2
          Non-LM   members
0
          Overlay       LM
0
          Zero-TEXT     LM              0

Processing summary of selected primary members:
          Selected                     1
          Processed                    1
          Processed successfully       1
          Processed with error         0

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.

Task completed with RC=4.
```

8. The following example shows the result of SYSPRINT for Action=Report,Verbose=No,ReporLevel=1:

Note: The selection part is not provided here, as it is the same as in Example 7a.

```
Name      Signed
BPXMIDMX  No
M1        Yes
M2        Yes
M3        Yes
M41ST     Yes
M4111     Yes
M4112     Yes
YM1       Yes
YM2       Yes
ZM1       Yes

Processing summary of selected primary members:
          Selected                     10
          Processed                    10
          Processed successfully       10
          Processed with error         0

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.

IEWSIGN exits with return code 4.
```

9. The following example shows the result of SYSPRINT for Action=Report,Verbose=No,ReporLevel=3:

Note: The selection part is not provided here, as it is the same as in Example 7a. Also, since the report formats for ReporLevel=2 and ReporLevel=3 are identical, only an example for ReporLevel=3 is provided.

```
Name      Size     Link date/time      Rel  Signed ErrorID Sign date/time      ALG  Cert-
Index
BPXMIDMX  00002218 2021-08-02 14:03:50 0205 No
M1        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M2        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M3        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M41ST     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M4111     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M4112     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
YM1       00000008 2022-09-30 22:19:28 0205 Yes    ERR09
YM2       00000008 2022-10-02 21:19:09 0205 Yes    ERR09
ZM1       00000008 2022-10-13 15:11:32 0205 Yes            2022-10-13 15:11:32 0202 INDEX001


ErrorID   Number    Error explanations
ERR01          6    Signing records lost or incomplete.
ERR09          2    Signature length is invalid.

Algorithm ID        Hash algorithm               Sign algorithm
0202                SHA2-512                     ECDSA-P521

Certificate summary:
Cert-Index:         INDEX001
Subject KeyID:      21CC95D0 8A12F9FE 5AA01598 430EF6A0 8D58DFDE
Cert Fingerprint:   0CED78C4 802B2B9A 3D190F75 8A79F005 87EF2294 69D680A0 C63B2FEE D3120D83

Processing summary of selected primary members:
          Selected                     10
          Processed                    10
          Processed successfully       2
          Processed with error         8

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.
IEW6027E 8 reported load modules have errors.

IEWSIGN exits with return code 8.
```

The following is an explanation of the contents in this example.

The terms in the first table of the example are described as follows:

Size
: The size of the load module's code in bytes.

Link date/time
: When the load module was built. It will be blank if the link time is unavailable.

Rel
: The binder version that linked the load module.

Signed
: Indicates whether the module is signed.

ErrorID
: Indicates whether there is a signature error. Blank means no error. Use this error ID to find an error description.

Sign date/time
: Indicates when the load module was signed.

ALG
: The algorithm of hash and sign. Use this ID to find an algorithm description.

Cert-Index
: An index assigned by IEWSIGN. Use this index to find a certificate displayed in “Certificate summary”, which signed this load module.

The table with heading “ErrorID Number Error explanations” provides the number of each error, along with a brief explanation for the error.

The following table provides a brief explanation and a detailed explanation for each ErrorID:

Table 80. ErrorID explanations

ErrorID	Brief explanation	Detailed explanation
ERR01	Signing records lost or incomplete.	The sign flag in the directory is on, but the signing records are lost or incomplete. This error is usually caused by some tools that are able to read and write regular records in load modules, but will discard the new signing records. For example, IEBCOPY with COPYMOD will cause this error.
ERR02	Subtype of signing record is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR03	Version of signing record is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR04	Flags of the signing record are invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR05	Length of signing record is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR06	Reserved field of signing record is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR07	Signature type is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR08	Signature version is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR09	Signature length is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR10	Signature reserved bytes are invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.

Table 80. ErrorID explanations (continued)

ErrorID	Brief explanation	Detailed explanation
ERR11	Signature algorithm is invalid.	A field in the signing record is invalid. The signing record has been modified by unknown tools.
ERR12	Signature hash is invalid. Load module has been modified.	The hash calculated at the reporting time does not match the hash calculated at the signing time. This error is usually caused by tools that are able to modify load modules. For example, SPZAP will cause this error.
ERR13	Directory entry error. Check the error message.	The directory entry of this module at the time of reporting is different from the directory entry of this module at the time of signing. More information is provided by one or more of the following error messages: IEW6022E, IEW6023E or IEW6024E

The table with the heading "Algorithm ID Hash algorithm Sign algorithm" provides the names of the algorithms used.

"Certificate summary" lists all certificates used to sign the reported load modules.

Return codes for the IEWSIGN utility

The following table lists the return codes, conditions of each return code, and the corresponding message that is issued for each condition.

Table 81. Return codes for the IEWSIGN utility

Return code (decimal)	Conditions of the return code
0	1. Successful completion. No error or warning messages are issued.
4	1. One or more non-load modules are found in INFILE. Message number: IEW6007W 2. One or more overlay load modules are found in INFILE. Message number: IEW6008W 3. One or more zero-text load modules are found in INFILE. Message number: IEW6009W 4. The primary member has been renamed. Message number: IEW6011W
8	1. One or more directory entry errors have been found in INFILE. Message number: IEW6010E 2. When Action=Report, an invalid signing record has been found. Message number: IEW6022E, IEW6023E, IEW6024E, IEW6025E, IEW6026E, IEW6027E 3. RC4LIM or RC8LIM reached. Message number: IEW6014E, IEW6015E 4. A load module has no CESD record or TEXT record, Message number: IEW6031E

Table 81. Return codes for the IEWSIGN utility (continued)

Return code (decimal)	Conditions of the return code
12	1. Invalid parameters. Message number: IEW6001S, IEW6002S. IEW6003S, IEW6028S 2. Necessary DD missed. Message number: IEW6004S 3. Incorrect data set attributes. Message number: IEW6005S, IEW6006S, IEW6017S, IEW6018S, IEW6034S, IEW6035S 4. Syntax error in INCLUDE or EXCLUDE. Message number: IEW6012S 5. No load modules have been selected for processing. Message number: IEW6013S 6. Output error of OUTFILE. Message number: IEW6019S, IEW6020S 7. RACF has not been configured correctly to sign a load module. Message number: IEW6016S, IEW6033S 8. IEWSIGN cannot allocate enough memory. Message number: IEW6021S 9. Invalid blocksize of OUTFILE. Message number: IEW6029S, IEW6030S 10. Language Environment callable service fails. Message number: IEW6032S

Chapter 19. AMATERSE: Pack and unpack a data set

AMATERSE is a service aid program that operates in problem state. You can use AMATERSE to pack a data set before transmitting a copy to another site, typically employing FTP as the transmission mechanism. A complementary unpack service is provided to create a similar data set at the receiving site.

Note: IBM also supports z/OS Problem Documentation Upload Utility (PDUU), which is a utility that sends large amounts of documentation to IBM. AMATERSE is useful for compressing (packing) and unpacking relatively small amounts of service data, but is incompatible with PDUU (output and input), and offers no data transfer capability. For information about PDUU, see Chapter 20, "AMAPDUPL: Problem Documentation Upload Utility," on page 645.

AMATERSE supports Direct Access Storage Device (DASD) and tape data sets:

- Sequential data sets, which can be unpacked by VM TERSE.
- Partitioned data sets (PDS), and partitioned data sets extended (PDSE) that do not contain program objects.
- Large format (DSNTYPE=LARGE) data sets.
- Fixed and variable, blocked and unblocked, spanned and unspanned record formats (RECFM) =F, FB, FBS, V, VB,VBS where the logical record length (LRECL) is less than 32K; and RECFM=VBS where the LRECL may be more than 32K but less than 64K.
- Data sets with records containing ISO/ANSI or machine code printer control characters.
- Placement of data sets into cylinder-managed space is also supported.

Planning for AMATERSE

AMATERSE is an application that prepares diagnostic materials, such as z/OS dumps and traces, for transmission to IBM and independent software vendor sites. When the materials arrive, AMATERSE also provides a means to create similar data sets to support diagnosis of problems.

If you have previously used the TRSMAIN utility (see TRSMAIN Utility (techsupport.services.ibm.com/390/trsmain.html)), note the following changes made to prepare AMATERSE for formal inclusion in z/OS:

- Use AMATERSE as the preferred application program name rather than TRSMAIN. TRSMAIN ships as an alias entry point to AMATERSE.
- Use the replacements for the DDNAMES, which are SYSUT1 and SYSUT2 . When the TRSMAIN entry point of AMATERSE is invoked, DDNAMES INFILE and OUTFILE remain as the defaults.
- AMATERSE is in MIGLIB, a library that is part of the link list. No STEPLIB DDNAME is necessary to invoke AMATERSE.
- In nearly all cases, you can use AMATERSE, the TRSMAIN utility, and VM terse interchangeably. See "Restrictions for AMATERSE" on page 642 for the exceptions to this rule.

Invoking AMATERSE

Figure 220 on page 640 shows an example of the JCL to invoke AMATERSE. Lower case text reflects the data that you must alter.

```
//jobname  JOB ...
//stepname EXEC PGM=AMATERSE,PARM=aaaaa
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=12901)
//SYSUT1   DD DISP=bbb,DSN=your.input.dataset.name
//SYSUT2   DD DISP=ccc,DCB=ddd,DSN=your.output.dataset.name,
//            SPACE=space_parameters
//SYSUT3   DD DISP=ccc,SPACE=space_parameters
```

Figure 220. Example: AMATERSE JCL

Specifying the JCL statements for AMATERSE

If you have previously used the TRSMAIN program to invoke AMATERSE, you can continue using it along with the old DDNAMES. However, if you choose to use AMATERSE instead of TRSMAIN, realize that the DDNAMES are changed: SYSUT1 replaces INFILE and SYSUT2 replaces OUTFILE.

A missing SYSUT1 DD statement results in an RC X'10' error message:

```
RC X'10', AMA522E INPUT DATASET HAS AN UNSUPPORTED DATASET ORGANIZATION
```

A missing SYSUT2 DD statement will result in an RC X'28' error message:

```
RC X'28', AMA518E UNABLE TO OPEN OUTPUT DATASET
```

AMATERSE requires the following JCL statements. The required DD statements are SYSPRINT, SYSUT1 and SYSUT2. SYSUT3 is optional. Replace `aaaaa` in the example with one of the following values:

EXEC
Marks the beginning of the job.

PACK

Compresses records in a data set so that the output is known as the simple format. .

SPACK
Compresses records in a data set so that the output is known as the complex format. The SPACK option is more time-consuming than the PACK option by a factor of about three, but in many cases produces much smaller output.

Note: A data set compressed by either PACK or SPACK should not be modified in any way. If such a data set is modified, the UNPACK routines are unable to reconstruct the original data set.

UNPACK
Reverses the PACK or SPACK operation. If you inadvertently packed a data set multiple times, restore it using the UNPACK function the same multiple number of times.

SYSPRINT statement
This DD defines where all messages from the program are sent. It must be RECFM=FBA and an LRECL between 121 and 133. Any block size that is a legal multiple of the LRECL is supported.

Note: The DCB information does not have to be specified on the DD statement and will default to the correct values.

SYSUT1 statement
This DD defines the data set to be compressed if PACK or SPACK parameter is specified on the EXEC statement. If UNPACK is specified, then it defines the compressed data set to be restored. See "Restrictions for AMATERSE" on page 642 for special considerations.

Note: If TRSMAIN entry point is used, INFILE statement is used instead of SYSUT1.

SYSUT2 statement

This DD defines the data set to receive the compressed output. If you specify PACK or SPACK on the EXEC statement, this is the data set that receives the compressed output. If you specify UNPACK, this is the data set that receives the restored output. See "Allocation considerations" on page 643 and "Space considerations" on page 643.

SYSUT3 statement

This optional DD defines the temporary data set to use when the PACK or UNPACK operation is performed against large PDS data sets. This data set acts as an intermediate form between PDS and PACKED data set. Only the DISP and SPACE parameters are necessary to be supplied by JCL. If the SYSUT3 DD statement is missing, AMATERSE allocates this data set by itself and deletes it automatically after AMATERSE ends.

AMATERSE return codes

When AMATERSE ends, one of the following return codes is placed in general purpose register 15. Refer to other AMA messages issued to debug the problem.

Code

Meaning

0

Successful completion.

4

Error in file operation.

8

Error in file operation.

10

Unsupported data set format.

12

Operation cannot be performed with the specified data set.

16

Invalid input specified.

20

Invalid input specified.

24

Severe error in file operation.

28

Severe error occurred during file open.

32

Invalid output device.

33

Invalid record length.

36

Buffer storage obtain failure.

64

Severe error. Abend with 1111.

99

System or user abend occurred.

Invoking AMATERSE from a problem program

To invoke AMATERSE from a program, specify the following information:

- PACK, SPACK, or UNPACK on the PARM parameter of the EXEC statement
- The DDNAMES of the data sets to be processed by the AMATERSE program, if the calling program is to override the DDNAMES.

Figure 221 on page 642 shows how to invoke AMATERSE using alternate DDNAMES:

```
*        Invoke AMATERSE to perform SPACK processing
             LINK  EP=AMATERSE,PARAM=(PARM,DDNAMES),VL=1
    .
    .
    .
    *        Request SPACK option
    PARM     DS    0H            EXEC PARM= data
    PARMLEN  DC    Y(L'PARMTEXT) Length of data
    PARMTEXT DC    C'SPACK'      AMATERSE processing option
    *
    *        Request MYPRINT, MYSYSUT1, and MYSYSUT2 instead
    *        of SYSPRINT, SYSUT1, and SYSUT2 respectively
    DDNAMES  DS    0H            DDNAME override data
    DDNAMEL  DC    Y(DDNAME9-DDNAMET)  Length of data
    DDNAMTT  DS    0C            DDNAME override list
             DC    5XL8'0'       Not used by AMATERSE
    SYSPRINT DC    CL8'MYPRINT'  Instead of SYSPRINT
             DC    XL8'0'        Not used by AMATERSE
    SYSUT1   DC    CL8'MYSYSUT1' Instead of SYSUT1
    SYSUT2   DC    CL8'MYSYSUT2' Instead of SYSUT2
    SYSUT3   DC    CL8'MYSYSUT3' Instead of SYSUT3
    DDNAME9  DS    0C            End of list
```

Figure 221. Example: AMATERSE JCL from a problem prgram

For a description of the parameter list, see the topic on Invoking Utility Programs from an Application Program in *z/OS DFSMSdfp Utilities*. From that information, AMATERSE supports the *optionsaddr* and *ddnameaddr*, but not *hdingaddr*. AMATERSE also supports DDNAMES SYSPRINT, SYSUT1, SYSUT2, and SYSUT3, but not DDNAMES SYSIN and SYSUT4, which are shown in the "ddname parameter list (DDNMELST)" in that information.

Additional considerations for AMATERSE

When using AMATERSE, certain restrictions apply as well as allocation and space considerations, as well as restrictions to consider when you are using AMATERSE.

Restrictions for AMATERSE

The following restrictions apply to AMATERSE:

- VSAM data sets and direct (DSORG=DA) data sets are not supported.
- Data sets with keys (KEYLEN) are not supported.
- A partitioned data set (PDS) compressed by AMATERSE on MVS cannot be unpacked by VM TERSE. This results in a 1007 or 1009 return code from VM TERSE.
- A PDS must be compressed to a DASD.
- Partitioned data sets extended (PDSE) containing program objects are not supported.
- AMATERSE handles data sets with a LRECL of more than 32K but less than 64K only when RECFM=VBS DASD data sets are processed.
- A data set with the FB record format can be packed and unpacked to a FBS data set. However, during the UNPACK operation, extending a non-empty output data set with DISP=MOD is not possible because this results in a FB data set. An error message is issued for this.
- AMATERSE does not support large block interface (LBI).

Allocation considerations

The data set compressed by AMATERSE (produced by PACK or SPACK) must be of fixed or fixed-blocked record format (RECFM) with a record length (LRECL) of 1024 and any legal block size (BLKSIZE). These values do not have to be specified explicitly on the DD statement.

The data set restored by AMATERSE (produced by UNPACK) must match the original RECFM and LRECL. Leave the DCB information off the DD statement for AMATERSE program to set it up. If it unpacks to an already existing data set then the DCB parameters are checked for compatibility. RECFM must be the same in all cases except for Variable to Undefined and Undefined to Variable. If you specify the DCB parameters to force data that was originally variable (V) format into undefined (U) format, or conversely, a warning message is written and the operation is performed.

Space considerations

When allocating space for the output data set SYSUT2, you must estimate the required size information:

- For the PACK or SPACK option a data set compressed by AMATERSE is expected to be about half the size of the original. Allocate more than expected and use the RLSE (partial release parameter) function of the SPACE value to release the unused portion back to the system.
- For the UNPACK option: If the data set contains random data, allocate three to five times the size of the compressed data set. If the data set contains Listing, Document, or Messages type data, allocate five to ten times the size of the compressed data set.

If there is not enough allocated space, the program issues ABEND X'B37':

```
Not Enough Space Allocated for the Output Data Set
```

Chapter 20. AMAPDUPL: Problem Documentation Upload Utility

The IBM z/OS Problem Documentation Upload Utility (PDUU) is a utility that sends large amounts of documentation in a more efficient manner than sending one large data set to IBM sites. This utility sections the input data set (such as stand-alone dump data set) into smaller data sets that are compressed and sent in parallel using multiple, simultaneous transfer sessions. This results in shorter transmission time for very large data sets. You can also encrypt the data sets. These sessions can send diagnostic documentation to IBM using File Transfer Protocol (FTP) or Secured Hypertext Transfer Protocol (HTTPS).

There are two work buffers for each transfer session (the "A" buffer and the "B" buffer). Each "A" work buffer is filled by copying records from the input data set. When the "A" buffer is full, the sessions are started in parallel. At the same time, each "B" work buffer is filled by copying records from the input data set. When the "B" buffer is full and the transfer of the "A" buffer is complete, transfer of the next "B" buffer starts. This process continues between the "A" and the "B" buffers, until everything in the input data set is sent.

You can have up to 20 transfer sessions running simultaneously, specifiable by the CC_FTP or CC_HTTPS parameter.

For FTP sessions, data is buffered into work data sets. The work data sets are dynamically allocated and can range in size from 1 MB to 9,999 MB. You can experiment to see what works best in your environment, but here are some guidelines:

- Start with three or four parallel FTP sessions. Too many parallel FTP sessions can saturate the network link.
- Use medium size work data sets.

For HTTPS sessions, data is buffered into 31-bit storage. When choosing a WORK_SIZE value, note that you may have limited private storage available (managed on an installation basis) and this number will be used as the size of the buffer.

Each WORK_SIZE buffer sent to IBM results in the creation of a numbered file that IBM uses to recreate the original data set for diagnosis. If the WORK_SIZE is very small in relationship to the input data set, you can end up with too many files on the IBM sites. For example, if you are sending a 100 GB z/OS stand-alone dump and make the work data set size 1 MB, PDUU will attempt to create 100,000 files on the IBM site, which exceeds the IBM limit of 99,999 files. This also causes a lot of delay by starting and stopping the transfer sessions for each file.

If the work buffers are very large in relationship to the input data set size, the amount of overlap time is decreased. When the program first starts, it must fill the "A" work buffer before it starts transmitting any data, which means the copy time is not overlapping with data that needs to be sent. For example, if you were sending a 1 GB dump and you set the work data set size to 1 GB (1,000 MB), there is no overlap between copying the records and sending the work files.

If the input data set is a partitioned data set (PDS/PDSE), PDUU unloads it first into a sequential data set using the IEBCOPY utility.

PDUU typically compresses the input data before it is written to the work buffer; therefore, it is counterproductive to use a tool such as AMATERSE or TRSMAIN to compress the input data set before using PDUU to send it to the IBM site. If a file is tersed, PDUU will not perform further compression. Overall performance of using AMATERSE with PDUU to send the file takes longer than if an untersed file is compressed and sent using PDUU.

Planning to use PDUU

Use PDUU as the primary utility for sending large volumes of documentation, such as stand-alone dumps, to the IBM site.

When using HTTPS mode, PDUU uses virtual storage to buffer requests instead of DASD data sets. The requesting AMAPDUPL job must have access to enough private virtual storage to satisfy the request. Please refer to the discussion of the WORK_SIZE parameter for details.

If you have previously used the MTFTPS stand-alone tool program (before z/OS V1R13), you must understand the following changes made to package the PDUU utility as part of z/OS:

- The PDUU utility name is AMAPDUPL; however, MTFTPS ships as an alias entry point to AMAPDUPL.
- AMAPDUPL resides in SYS1.MIGLIB (which must be a data set in the LNKLST concatenation), so a STEPLIB DDNAME is not necessary to invoke AMAPDUPL.
- AMA messages are described in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.

Prerequisites and restrictions for PDUU

With APAR OA55959, PDUU now supports two transmission protocols: FTP and HTTPS. Both require proper configuration of the z/OS® Communications Server to use the preferred protocol.

For HTTPS mode, specify HTTPS mode using the USE_HTTPS=Y SYSIN parameter. You will need a valid configured key store containing the necessary certificates to access the IBM documentation sites. This can be specified by the HTTPS_KEYRING or HTTPS_KEYFILE parameters.

For FTP mode, see the topic on Transferring files using FTP in *z/OS Communications Server: IP Configuration Guide*. The PDUU uses active FTP mode as the default, unless another mode is requested with the corresponding FTP subcommands defined in FTPCMDS data set.

The PDUU supports the following types of input data sets:

- Members of partitioned data sets (PDS) and partitioned data sets extended (PDSE)
- Large format (DSNTYPE=LARGE) and traditional sequential data sets
- Extended format sequential data sets
- Fixed, variable, and undefined-length, blocked and unblocked, spanned and unspanned record formats (RECFM) = F, FB, FBS, V, VB, VS, VBS, U)
- Data sets with records containing ISO/ANSI or machine code control characters
- Data sets in cylinder-managed space.
- Partitioned data sets (PDS) and partitioned data sets extended (PDSE).

PDUU does not support the following types of input data sets:

- Large block interface (LBI) (no BLKSIZE value).
- VSAM and direct (DSORG=DA) data sets
- Data sets with keys (KEYLEN)
- z/OS UNIX files
- Concatenated data sets of any type

JCL statements for PDUU

The JCL statements for the PDUU are:

SYSPRINT
The data set can be either SYSOUT or a sequential data set. The data set must be RECFM=FB, LRECL=134. For additional details, see "Prerequisites and restrictions for PDUU" on page 646.

SYSUT1

The sequential or partitioned data set to transfer to IBM. For additional details, see "Prerequisites and restrictions for PDUU" on page 646.

SYSUT2

This data set is optional and can be used only when transferring partitioned data sets (PDS/PDSE). It defines a sequential unload data set for IEBCOPY output produced during the unload operation. If the SYSUT2 statement is omitted the unload data set will be allocated dynamically. This parameter can be used if you want to directly control the allocation of the unload data set, for example, to specify a particular volume or certain amount of volume space. For additional details about usage and allocation parameters of the unload data set see the topic on Unloading (Backing up) Data Sets in *z/OS DFSMSdfp Utilities*. See also "Example 9: Using SYSUT2 to allocate an unload data set" on page 655.

SYSIN

A sequential data set that uses the following control statements. The data set must be RECFM=FB, LRECL=80. For additional details, see "Prerequisites and restrictions for PDUU" on page 646.

PDUU is managed through the following SYSIN statements with these guidelines:

- Use an asterisk (*) in the first column of each comment line to indicate comments.
- Keywords must start in column one.
- Use control statements that are in form VERB=OPERAND.
- Mixed case verbs and operands are allowed.
- The operand starts in the column after the equal sign and goes to the first blank column except `TARGET_SYS`, `DIRECTORY`, `CIPHER_KEY`, `ACCOUNT`, `HTTPS_KEYFILE`, `HTTPS_KEYRING`, `HTTPS_KEYSTASH`, `HTTPS_PROXY`, `HTTPS_LOCALIPADDR`, `USERID`, and `PASSWORD`, which can contain blanks.
- Anything after the first blank is ignored except for any operands that can contain blanks. In those cases, do not use blanks from column one to the end of the operand.
- Control statements can be coded on one or more (up to 6) consecutive records. Control statements with operands that allow blanks must not extend beyond column 71, but can continue on the following record in columns 16 through 71. Columns 1 through 15 of the continuation record must be blank. See "Example 10: Using a multiple record control statement in SYSIN" on page 656. Control statements with operands that do not allow blanks can occupy columns 1 through 80.
- When specifying a control statement twice, the last specification is used.

USE_HTTPS

An optional parameter that when specified with a value of 'Y' enables HTTPS mode and indicates that PDUU use the HTTPS protocol to transfer data to IBM.

Omitting this parameter results in PDUU using the default FTP protocol.

TARGET_SYS

The name of the TCP/IP system to transfer the files to. One through 256 characters, dotted decimal format is allowed, no default value, can not contain blanks, and it must be specified.

For FTP mode, if using a proxy server, this should be the name of the proxy server.

You can include additional FTP command parameters on the TARGET_SYS parameter by using the z/OS UNIX specifications as shown in the topic FTP command -- Entering the FTP environment in *z/OS Communications Server: IP User's Guide and Commands*. For example, to trace output (`-d`) and use a specific ftpdata_filename (-f"//'WES.MYFTP.DATA'"):

```
TARGET_SYS=-d -f"//'WES.MYFTP.DATA'" testcase.boulder.ibm.com
```

Use the `-p` parameter to specify an alternate IP stack.

For HTTPS mode, does not allow for the specification of a config file or proxy information through the TARGET_SYS parameter.

USERID
The user ID on the target system that is used to send the files. One through 64 characters, no default value, does not have to be specified, and can contain imbedded blanks.

For FTP mode only, if USERID and PASSWORD are not supplied and NETRCLEVEL=2, the values from the NETRC data set is used for the FTP sessions.

If using a proxy server, this can be the full login to the remote system in the format *userid@remote.system.name*.

PASSWORD
The password for the USERID on the target system. One through 64 characters and the default value is blanks.

For FTP mode only, if using a proxy server, this can be the USERID and PASSWORD for the proxy server in the format *userid@password*.

ACCOUNT
For FTP mode, the account data that is sent when an FTP session is started. One through 64 characters with no default value.

This parameter is ignored for HTTPS mode.

TARGET_DSN
The descriptive portion for the file names for the target system. One through 50 characters, no default value, and it must be specified. It can contain alphanumeric characters and special characters allowed on the target system.

WORK_DSN
For FTP mode, the prefix for the data set names of work files on the sending system. One through 40 characters, no default value, and it must be specified. The work data sets are large format, sequential, data sets and cannot have the compaction attribute.

Note: Because work files are dynamically allocated with large format and do not support compressed format, if you specify data class for work files with the compaction attribute or N, the following message is issued for all work files:

```
IGD17163I COMPRESSION REQUEST NOT HONORED FOR DATA SET
work_file_dsname BECAUSE DATA SET CHARACTERISTICS
DO NOT MEET COMPRESSION CRITERIA, ALLOCATION CONTINUES
```

This parameter is ignored for HTTPS mode.

CC_HTTPS | CC_FTP
The number of parallel FTP transfer sessions to use when transmitting the files. One or two decimal digits, the value must be between one and 20, and the default is two. CC_HTTPS is an alias for CC_FTP.

WORK_SIZE | WORK_DSN_SIZE
The maximum size of the work buffer in megabytes. One through four decimal digits. When unspecified, the default is 100. WORK_SIZE is an alias for WORK_DSN_SIZE.

For FTP mode, two DASD data sets per session (CC_FTP) are dynamically allocated with this size.

For HTTPS mode, virtual storage of the size 2 * CC_HTTPS# * WORK_SIZE# * 1 MB is requested.

When choosing values for WORK_SIZE and CC_HTTPS parameters, be aware that these buffers are allocated in 31-bit storage, and this is limited to significantly less than 2 GB. For example, if you have CC_HTTPS=4 and WORK_SIZE=200, AMAPDUPL will attempt to allocate virtual storage of 1600 MB. This may fail with return code 12 and message AMA761E if requested storage is not available. Also, beware that transfers may begin, but fail due to the web enablement toolkit storage requirements. PDUU will also fail with return code 12 and message AMA761E if the web enablement toolkit is unable to obtain storage for PDUU. Consider lowering the CC_HTTPS or WORK_SIZE parameters in this case.

KEEP_WORK
For FTP mode, the parameter to save the work data sets that are dynamically allocated for each FTP session. If you omit the KEEP_WORK parameter, the program does not save the work data sets. Y is the only value for the KEEP_WORK parameter.

Note: Only specify this parameter when debugging a problem.

This parameter is ignored for HTTPS mode.

DATACLAS
For FTP mode, the data class to use when allocating the work files on the sending system. One through eight characters with no default value.

This parameter is ignored for HTTPS mode.

MGMTCLAS
For FTP mode, the management class to use when allocating the work files on the sending system. One through eight characters with no default value.

This parameter is ignored for HTTPS mode.

STORCLAS
For FTP mode, the storage class to use when allocating the work files on the sending system. One through eight characters with no default value.

This parameter is ignored for HTTPS mode.

DIRECTORY
The directory on the target system where the files will be sent with FTP or HTTPS. One through 32 characters, with no default, can contain blanks, and you must specify the directory.

For HTTPS transfer to www.secure.ecurep.ibm.com specify the destination without any leading or trailing slashes, such as 'DIRECTORY=mvs'.

CASE
The CASE id associated with the file and problem. This field must be 11 numeric or uppercase characters. Do not specify when the PMR statement is specified. Example: TS123456789

PMR
The PMR number with which this file is associated. Do not specify when the CASE statement is specified. This field must be 13 numeric or uppercase characters, specify in the form `xxxxx.yyy.zzz`, and define the variables as:

Table 82. PMR number variables for PDUU

Field	Explanation	Example
XXXXX	PMR Number	34143
YYY	Branch office	055
ZZZ	IBM Country Code	724

CIPHER_KEY
The encryption key to use for 192-bit triple DES encryption. The 24 characters following CIPHER_KEY= are used as the key. The key can include imbedded and/or trailing blanks. For example, `CIPHER_KEY=HERE IS CIPHER KEY IN 24` or `CIPHER_KEY=Shortkey`. If you do not specify CIPHER_KEY=, no encryption is performed. If you encrypt the data set using CIPHER_KEY, you must provide IBM with the encryption key so they can perform problem diagnosis.

Note: If `CIPHER_KEY=` is followed by 24 blanks, the file will be encrypted with a key of 24 blanks.

NO_FTP
A value of 'Y' specifies that PDUU compress, optionally encrypt, separate files into parts, and move the part to a local z/OS Unix Systems Services directory, without FTPing the files. PDUU uses TSO services to send data sets to USS directories. When you specify NO_FTP=Y:

- TARGET_SYS, CC_FTP, USERID, and PASSWORD are not required
- CC_FTP settings are ignored and set to 1
- FTPCMDS DD will be ignored
- The system allocates the SYSTSPRT DD to receive messages from TSO.

Once transferred, you can browse the files and extract them from the z/OS Unix Systems Service directory. The file names used depend on the PMR and TARGET_DSN input:

```
PMR.TARGET_DSN.Tdate.MTFTP.F00002 to ...Fnnnnn
```

The size and number of files you end up with depends on the WORK_DSN_SIZE SYSIN control statement and how well the input file compresses.

The date section is a random string based on the time the job is run, and PDUU creates different output file names if the same file is sent at a different time. In addition there will be a small file with suffix F001 containing the control record with information necessary to recreate the file on the receiving end.

Please package and send all generated files to IBM with the file names created. Use the NO_FTP=Y option if you have a closed data center (with no outside internet access). PDUU then prepares files you can send to IBM service on removable media as desired.

NO_FTP is ignored when USE_HTTPS is specified.

HTTPS_KEYRING
Specify a SAF key ring or PKCS #11 token containing certificates necessary to connect to the HTTPS sessions. Of the form:

- SAF key ring name, specified as userid/keyring
- PKCS #11 token, specified as *TOKEN*/token_name

HTTPS_KEYFILE
Specifies a path and file name of the key data base file created by the System SSL gskkyman utility.

HTTPS_KEYSTASH
Specifies the path and file name of the password stash file created by the System SSL gskkyman utility. This option is required when HTTPS_KEYFILE is sepcified.

HTTPS_PORT
An optional parameter when USE_HTTPS=Y is specified indicating the remote port number to which to connect. The default value is 443.

HTTPS_IPSTACK
An optional parameter when USE_HTTP=Y is specified indicating the local TCP/IP stack name to be used when connecting to the IBM site. 1-8 characters specifications are allowed.

HTTPS_LOCALIPADDR
An optional outgoing IP address from which the connection is to originate.

HTTPS_LOCALPORT
An optional parameter when USE_HTTP=Y is specified indicating the outgoing port number from which the connection is to originate.

HTTPS_PROXY
An optional parameter when USE_HTTP=Y is specified indicating the HTTP proxy to use. Must specify starting with http:// or https://. For example: http://my.proxy.com

HTTPS_PROXYPORT
An optional parameter when USE_HTTP=Y is specified indicating the proxy port to connect to.

HTTPS_PROXYUSERNAME
An optional parameter when USE_HTTP=Y is specified indicating the username to connect to the HTTP proxy. One through 64 characters and can contain embedded blanks. Must specify with HTTPS_PROXYPASSWORD.

HTTPS_PROXYPASSWORD
An optional parameter when USE_HTTP=Y is specified indicating the password to connect to the HTTP proxy. One through 64 characters and can contain embedded blanks. Must specify with HTTPS_PROXYUSERNAME.

HTTPS_VERBOSE
An optional parameter when that when specified with a value of 'Y' and when USE_HTTP=Y indicates to the web enablement toolkit to produce verbose messages for HTTPS transfers. These messages can be helpful in diagnosing connection issues and aiding in IBM problem determination. The HTTPS_VERBOSE_DD can be used to change where these messages are sent.

HTTPS_VERBOSE_DD
An optional parameter to specify a 1-8 character DD name that takes effect when the HTTPS_VERBOSE=Y parameter is used. The DD name indicates where the web enablement toolkit is to place the verbose messages. This must meet the requirements as described in *HTTP/HTTPS enabler options and values*in the HWTH_OPT_VERBOSE_OUTPUT option. The default is SYSPRINT when not specified.

DEBUG
An optional DD statement that gathers debug information such as messages issued to the SYSPRINT data set and the FTP protocol messages. The data set must be RECFM=FB, LRECL=134.

FTPCMDS
For FTP mode, an optional DD statement that provides additional flexibility for traversing firewall or proxy servers. When this DD statement is provided, after the initial USERID and PASSWORD are sent, the specified sequential data set is read by the application and the commands contained in the data set are included as FTP commands. The data set must be RECFM=FB and LRECL=80.

This DD is ignored for HTTPS mode.

Comments and sequence numbers are not allowed. As an example, to supply a userid and password on the hostname system:

```
//FTPCMDS  DD *
USER 'anonymous@testcase.boulder.ibm.com proxyuser'
```

SYSTSPRT
An optional DD statement that shows messages from TSO operations when you have NO_FTP=Y specified. If you receive message AMA778I with a return code of X'04' and function code of X'C', add this DD statement to your JCL to receive messages and rerun if necessary to receive messages.

JCL examples for PDUU

Use the following JCL examples as a guideline for creating your own JCL. Consider storing your user ID and password in a separate concatenated data set. Doing so provides added security because the user ID and password are not directly in the JCL. It is also makes it much easier to change the user ID and password across multiple jobs.

You can use some of the JCL examples as a starting point to traverse a firewall or proxy server. There are very few common characteristics for firewall or proxy servers with local customization. If you are able to traverse the firewall or proxy server with a plain FTP statement, modifications to the parameters `USERID`, `PASSWORD`, `ACCOUNT`, and `TARGET_SYS`, in conjunction with commands in the FTPCMDS data set, the ftp_data file, or both can permit the z/OS PDUU to traverse your firewall or proxy server.

Example 1: Simple FTP connection

The JCL example in Figure 222 on page 652 invokes the AMAPDUPL program to transfer file `H44IPCS.WESSAMP.TRKS055K` to the `testcase.boulder.ibm.com` system as a set of work files stored in `/toibm/mvs` with the shared prefix `TS012345678.wessamp.bigfile`. Each of the three work files is 500 MB.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN    DD  *
USERID=anonymous
PASSWORD=anonymous
TARGET_SYS=testcase.boulder.ibm.com
TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 222. Simple FTP connection

Example 2: FTP connection using a proxy server

In Figure 223 on page 652, the USERID control statement has the format `user@hostname`, where `hostname` is the name of the TCP/IP system to transfer files to, and `user` is the user name on the hostname system. The PASSWORD control statement has format `proxyuser@proxypass`, where `proxyuser` is the user name on the proxy server and `proxypass` is the user password on the proxy server. TARGET_SYS is the name of the TCP/IP proxy server.

```
//FTP      EXEC PGM=AMAPDUPL
//SYSUDUMP  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN     DD *
USERID=anonymous@testcase.boulder.ibm.com
PASSWORD=proxyuser@proxypass
TARGET_SYS=your.proxy.server.name
TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 223. FTP connection using a proxy server

Example 3: FTP connection using a proxy server with proxy user ID

In Figure 224 on page 653, the `USERID` control statement has format `user@proxyuser@hostname`, where `hostname` is the name of the TCP/IP system to transfer the files to, `user` is the user name on the hostname system, and `proxyuser` is the user name on the proxy server. The `PASSWORD` control statement has format `proxyuser@proxypass`, where `proxyuser` is the user name on the proxy server and `proxypass` is the user's password on the proxy server. `TARGET_SYS` is the name of the TCP/IP proxy server.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN DD *
USERID=anonymous@proxyuser@testcase.boulder.ibm.com
PASSWORD=proxyuser@proxypass
TARGET_SYS=proxy.server.name
TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 224. FTP with a proxy user ID

Example 4: Using a proxy server with the FTPCMDS DD statement

In Figure 225 on page 653, the USERID control statement has format proxyuser@hostname, where hostname is the name of the TCP/IP system to transfer the files to, and proxyuser is the user name on the proxy server. The PASSWORD control statement defines the user password on the proxy server. The data set name WES.FTPCMDS.DATA contains an additional user command with an anonymous user name and password on the hostname system.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//FTPCMDS  DD DISP=SHR,DSN=WES.FTPCMDS.DATA
//SYSIN    DD *
USERID=proxyid@testcase.boulder.ibm.com
PASSWORD=proxypass
TARGET_SYS=proxy.server.name
TARGET_DSN=SVCD
WORK_DSN=HLQ.FTPOUT
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 225. FTP using the FTPCMDS DD statement

The example in Figure 226 on page 653 shows the typical format of the FTPCMDS data set.

```
user anonymous pw userid@company.com
```

Figure 226. FTPCMDS data set example

Example 5: Using a proxy server with a port specification on the TARGET_SYS parameter

The example in Figure 227 on page 654 uses a proxy server with a port specification of 2121 on the TARGET_SYS parameter and inline FTPCMDS DD statement. This example is similar to the previous one, the only difference are the FTPCMDS is an in-stream data set and the port specification is included on the TARGET_SYS parameter.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//FTPCMDS  DD *
user anonymous pw userid@company.com
//SYSIN    DD *
USERID=proxyuser@testcase.boulder.ibm.com
PASSWORD=proxypass
TARGET_SYS=proxy.server.name 2121
TARGET_DSN=SVCD
WORK_DSN=HLQ.FTPOUT
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
CIPHER_KEY=PMR99999sad
//
```

Figure 227. FTP specifying port 2121 on TARGET_SYS

Example 6: Forcing PASSIVE mode using the FTPCMDS inline DD statement

Figure 228 on page 654 shows the FTP connection set up from the FTP client to the FTP server using the FTP `locsite fwfriendly` command.

The `FWFriendly` parameter specifies that the FTP client is firewall-friendly. For additional details, see the topic about LOCSIte subcommand--Specify site information to the local host in *z/OS Communications Server: IP User's Guide and Commands*.

Note: When the FTP server has an IPv6 address, data connections are always set up from the FTP client to the FTP server without reference to the FWFriendly setting.

```
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//FTPCMDS DD *
LOCSIte FWFriendly
//SYSIN DD *
USERID=proxyuser@testcase.boulder.ibm.com
PASSWORD=proxypass
TARGET_SYS=proxy.server.name
TARGET_DSN=SVCD
WORK_DSN=HLQ.FTPOUT
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
CIPHER_KEY=PMR99999sad
//
```

Figure 228. FTP forcing PASSIVE mode

Example 7: Using a userid.NETRC data set

The example in Figure 229 on page 655 uses the proxy login and password stored in the userid.NETRC data set (you can submit this as a surrogate job where the userid.NETRC data set is invisible to the job originator). Use of the userid.NETRC data set requires NETRCLEVEL=2, which is set in the FTP.DATA data set. Using the `-f` parameter on TARGET_SYS control statement specifies which FTP.DATA data set to use.

- Find information about the use of the NETRC data set in *z/OS Communications Server: IP User's Guide and Commands*.
- Find information about the use of the FTP.DATA data set in *z/OS Communications Server: IP Configuration Reference*.

```
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN DD *
TARGET_SYS=-f"//'WES.MYFTP.DATA'" testcase.boulder.ibm.com
CASE=TS012345678
DIRECTORY=/toibm/mvs/
TARGET_DSN=wes.gb01tst
work_dsn=wes.ftpout
cc_ftp=03
WORK_DSN_SIZE=500
//

For this instance, the userid.NETRC datas et consists of one line:
machine testcase.boulder.ibm.com login anonymous password ibmusr@ibm.com

The FTP.DATA data set contains:
;**********************************************************************
;*
NETRCLEVEL 2
;*
```

Figure 229. FTP using a userid.NETRC data set

Example 8: Using the DEBUG statement

The example in Figure 230 on page 655 adds the DEBUG DD statement, which creates a data set that contains the message data, as described in "DEBUG" on page 651.

```
//FTP EXEC PGM=AMAPDUPL
//SENDSTP  EXEC PGM=AMAPDUPL
//SYSPRINT DD SYSOUT=*
//DEBUG    DD DSN=PDUU.DEBUG,DISP=(,CATLG),
//            UNIT=SYSALLDA,SPACE=(CYL,(1,1),RLSE)
//SYSUT1   DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN    DD *
USERID=anonymous
PASSWORD=anonymous
TARGET_SYS=testcase.boulder.ibm.com
TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs
CASE=TS012345678
//
```

Figure 230. FTP connection with the DEBUG DD statement

Example 9: Using SYSUT2 to allocate an unload data set

The JCL example in Figure 231 on page 656 invokes the AMAPDUPL program to transfer partitioned data set H44IPCS.PDS.DATA to the testcase.boulder.ibm.com. The optional SYSUT2 statement is used to allocate an unload sequential data set for the IEBCOPY utility invoked by the PDUU.

```
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DSN=H44IPCS.PDS.DATA,DISP=SHR
//SYSUT2  DD DSN=H44IPCS.UNLOAD.DASD,DISP=(NEW,CATLG),
//  DCB=(RECFM=VS),
//  SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA
//SYSIN DD *
USERID=anonymous
PASSWORD=anonymous
TARGET_SYS=testcase.boulder.ibm.com
TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 231. Using SYSUT2 statement for allocating an unload data set

Example 10: Using a multiple record control statement in SYSIN

The following JCL example in Figure 232 on page 656 invokes the AMAPDUPL program with FTP.DATA data set specified by the –f parameter on TARGET_SYS control statement coded on 3 consecutive SYSIN records.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
//FTP EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DSN=H44IPCS.SEQ.DATA,DISP=SHR
//SYSIN DD *
USERID=anonymous
PASSWORD=anonymous
TARGET_SYS=-d -f /u/directoryForFtpData/subdirectoryForFtpData1/subdire
               ctoryForFtpData2/subdirectoryForFtpData3/subdirectoryFor
               FtpData4/ftpDataFile testcase.boulder.ibm.com

TARGET_DSN=wessamp.bigfile
WORK_DSN=wes.ftpout
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
//
```

Figure 232. Using a multiple record control statement

Example 11: Using the NO_FTP option

The following JCL invokes the AMAPDUPL program to compress and separate the file into chunks that can be brought to IBM for documentation. File H44IPCS.WESSAMP.TRKS055K will be compressed and stored into several files at:

```
/u/nickj/pduu/TS012345678.WESSAMP.BIGFILE.Txxxxx.MTFTP.Fnnnnn
```

```
//FTP     EXEC PGM=AMAPDUPL
//SYSPRINT DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN    DD  *
NO_FTP=Y
TARGET_DSN=WESSAMP.BIGFILE
WORK_DSN=WES.FTPOUT
WORK_DSN_SIZE=500
DIRECTORY=/u/nickj/pduu/
CASE=TS012345678
```

Figure 233. Using the NO_FTP option

The following files will be stored on your local systems directory:

```
/u/nickj/pduu/:
 TS012345678.WESSAMP.BIGFILE.TE3246.MTFTP.F00002
 TS012345678.WESSAMP.BIGFILE.TE3246.MTFTP.F00003
 TS012345678.WESSAMP.BIGFILE.TE3246.MTFTP.F00004
 ...
 TS012345678.WESSAMP.BIGFILE.TE3246.MTFTP.F001
```

Example 12: Using a proxy server with multiple FTPCMDS DD statements

The example in Figure 234 on page 657 uses a proxy server with multiple statements in an inline FTPCMDS DD. In this example, FTPCMDS DD runs the USER and PASS commands, and the USERID and PASSWORD SYSIN statements are omitted. The user also specifies PASSIVE mode.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//FTPCMDS  DD *
USER 'anonymous@testcase.boulder.ibm.com proxyuser'
PASS userid@company.com
ACCT proxypassword
LOCSIte FWFriendly
//SYSIN    DD *
TARGET_SYS=proxy.server.name
TARGET_DSN=SVCD
WORK_DSN=HLQ.FTPOUT
CC_FTP=03
WORK_DSN_SIZE=500
DIRECTORY=/toibm/mvs/
CASE=TS012345678
CIPHER_KEY=PMR99999sd
//
```

Figure 234. FTP specifying multiple statements in an inline FTPCMDS DD

Example 13: Simple HTTPS connection to testcase

The JCL example in Figure 235 on page 658 invokes the AMAPDUPL program to transfer file H44IPCS.WESSAMP.TRKS055K to the testcase.boulder.ibm.com system as a set of work files stored in /toibm/mvs with the shared prefix TS012345678.wessamp.bigfile. Each of the three work files will be 100 MB. This connection uses the RACF keyring for user TSOUSER named 'pduu'.

PDS entry TSOUSER.FTPINFO(TESTCASE) contains the USERID and PASSWORD in a RACF protected and encrypted PDS with contents described in Figure 236 on page 658.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//DEBUG    DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN    DD  DISP=SHR,DSN=TSOUSER.FTPINFO(TESTCASE)
//         DD  *
TARGET_SYS=testcase.boulder.ibm.com
TARGET_DSN=wessamp.bigfile
CC_HTTPS=03
WORK_SIZE=100
DIRECTORY=/toibm/mvs/
CASE=TS012345678
USE_HTTPS=Y
HTTPS_KEYRING=TSOUSER/pduu
```

Figure 235. Simple HTTPS connection to testcase

```
USERID=user@domain.com
PASSWORD=userpassword
```

Figure 236. Simple HTTPS connection contents of TSOUSER.FTPINFO(TESTCASE)

Example 14: Simple HTTPS connection with verbose HTTPS messages

The JCL example in Figure 237 on page 658 is similar to "Example 13: Simple HTTPS connection to testcase" on page 657 but puts verbose messages into the HTTPDEBG DD. This connection uses a key database file created by the gskkyman and a matching key database password stored in a key stash.

```
//FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//DEBUG    DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//HTTPDEBG DD DISP=(NEW,CATLG,KEEP),
//            DSN=TSOUSER.HTTPDEBG,DCB=(RECFM=V,DSORG=PS),
//            SPACE=(CYL,(1,1))
//SYSIN    DD  DISP=SHR,DSN=TSOUSER.FTPINFO(TESTCASE)
//         DD  *
TARGET_SYS=testcase.boulder.ibm.com
TARGET_DSN=wessamp.bigfile
CC_FTP=03
WORK_SIZE=50
DIRECTORY=/toibm/mvs/
CASE=TS012345678
USE_HTTPS=Y
HTTPS_KEYFILE=/etc/websrv1/mykeys.kdb
HTTPS_KEYSTASH=/etc/websrv1/mykeys.sth
HTTPS_VERBOSE=Y
HTTPS_VERBOSE_DD=HTTPDEBG
```

Figure 237. FTP specifying multiple statements in an inline FTPCMDS DD

Example 15: Simple HTTPS connection to Ecurep

The JCL example in Figure 238 on page 659 invokes the AMAPDUPL program to transfer file H44IPCS.WESSAMP.TRKS055K to the testcase.boulder.ibm.com system as a set of work files. These will be sent to ecurep and directed to the mvs directories with the shared prefix TS012345678.wessamp.bigfile. Each of the three work files will be 100 MB. This connection uses the RACF keyring for user TSOUSER named 'pduu'.

PDS entry TSOUSER.FTPINFO(TESTCASE) contains the USERID and PASSWORD in a RACF protected and encrypted PDS with contents described in Figure 239 on page 659.

```
///FTP     EXEC PGM=AMAPDUPL
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//DEBUG    DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=H44IPCS.WESSAMP.TRKS055K
//SYSIN    DD  DISP=SHR,DSN=TSOUSER.FTPINFO(ECUREP)
//         DD  *
TARGET_SYS=www.secure.ecurep.ibm.com
TARGET_DSN=wessamp.bigfile
CC_FTP=03
WORK_SIZE=100
DIRECTORY=mvs
CASE=TS012345678
USE_HTTPS=Y
HTTPS_KEYRING=TSOUSER/pduu
```

Figure 238. Simple HTTPS connection to Ecurep

```
USERID=user@domain.com
PASSWORD=userpassword
```

Figure 239. Simple HTTPS connection contents of TSOUSER.FTPINFO(ECUREP)

Return codes for PDUU

Upon completion, PDUU places one of the return codes listed in Table 83 on page 659 in general purpose register (GPR) 15.

Table 83. Return codes for z/OS Problem Documentation Upload Utility

Return Code	Explanation
0	Successful completion
4	Potential successful completion. If not, investigate messages.
8	Invalid parameters in control statement
10	Unsupported data set format
12	Storage obtain failure
16	Required input parameters are missing
20	Invalid input data set specified
24	Severe error occurred during dictionary building
28	Severe error occurred during file open
32	Severe error occurred during compression process
36	Error in FTP or HTTPS operation
40	Severe error during IKJTSOEV operation for NO_FTP=Y invocations, see message AMA777I
44	Severe error during IKJEFTSR operation for NO_FTP=Y invocations, see message AMA778I and SYSTSPRT DD
64	Severe error in file operation
68	Error unloading a PDS or PDSE
69	Failure to attach a subtask
70	Failure in pause element service

Table 83. Return codes for z/OS Problem Documentation Upload Utility (continued)

Return Code	Explanation
71	Failure in BPX1SDD service
98	System or user abend occurred during subtask operation
99	System or user abend occurred

Chapter 21. Dump suppression

The system requests dumps you might not need. To keep from using your system's resources on unneeded dumps, you should suppress them. The reasons for the unneeded dumps and ways to suppress them are:

- **Duplicate dumps:** The system can request a dump for a problem that recurs. The dump written when the problem first occurs can be used for diagnosis; additional dumps are unneeded. Also, sometimes a system can request several dumps for one instance of a problem. A recurring problem may have been diagnosed, but the fix has not yet been incorporated into the system.

  To eliminate the duplicate dumps, use dump analysis and elimination (DAE). See "Using DAE to suppress dumps" on page 661.
- **Dumps for certain abend codes:** For some abend codes, the accompanying messages provide the needed problem data. To eliminate the dumps for these abend codes, use a SLIP command. See "Using a SLIP command to suppress dumps" on page 669.
- **A dump for an abend in an application program:** if the dump is not needed. See "Using an ABEND macro to suppress dumps" on page 670.
- **Dumps the installation decides are not needed:** If you decide that certain dumps are not needed, you can code a routine for an installation exit to suppress these dumps. See "Using installation exit routines to suppress dumps" on page 670.

This topic lists the ways that an expected dump can be suppressed, so that you can determine why you did not receive an intended dump. See "Determining why a dump was suppressed" on page 671.

Using DAE to suppress dumps

Dump analysis and elimination (DAE) suppresses dumps that match a dump you already have. Each time DAE suppresses a duplicate dump, the system does not collect data for the duplicate or write the duplicate to a data set. In this way, DAE can improve dump management by only dumping unique situations and by minimizing the number of dumps.

For more information about the topics described in this section, refer to the following references:

- See *z/OS MVS Diagnosis: Reference* for symptoms and symptom strings.
- See *z/OS MVS Initialization and Tuning Reference* for the ADYSETxx and IEACMD00 parmlib members.
- See *z/OS MVS IPCS Commands* for the VERBEXIT DAEDATA subcommand.
- See *z/OS MVS Planning: Global Resource Serialization* for data set serialization.
- See *z/OS Security Server RACF Command Language Reference* to control access to data sets.

Performing dump suppression

To perform dump suppression, DAE builds a symptom string, if the data for it is available. If the symptom string contains the minimum problem data, DAE uses the symptom string to recognize a duplicate SVC dump or SYSMDUMP dump requested for a software error. When installation parameters request suppression, DAE suppresses the duplicate dump. The following describes DAE processing.

1. **DAE obtains problem data**. DAE receives the data in the system diagnostic work area (SDWA) or from values in a SYMREC parameter on the SDUMP or SDUMPX macro that requested the dump.
   - The ESTAE routine or the functional recovery routine (FRR) of the failing program supplies module-level information, such as the failing load module name and the failing CSECT name.
   - The system supplies system-level data, such as the abend and reason codes, the failing instruction, and the register/PSW difference.

If the failing component does not supply the failing load module name or CSECT name, the system determines the name, if possible. In this case, the name may be IEANUC0x.

2. **DAE forms a symptom string.** DAE adds a descriptive keyword to each field of problem data to form a symptom. DAE forms MVS symptoms, rather than RETAIN symptoms. DAE combines the symptoms for a requested dump into a symptom string.

   The following tables show the required and optional symptoms. SDWA field names are given for the symptoms the failing program must provide to enable dump suppression. The tables have both MVS and RETAIN symptoms so that you can relate the MVS symptoms DAE uses to the RETAIN symptoms you might use to search the RETAIN data base. An MVS symptom string must contain at least five symptoms that are not null. DAE places symptoms into strings in the order shown in the tables.

   Table 84 on page 662 summarizes the required symptoms, which are first and must be present.

   Table 84. Summary of required symptoms

Symptom	SDWA Field	MVS Keyword	RETAIN Keyword
Name of the failing load module	SDWAMODN	MOD/name	RIDS/name#L
Name of the failing CSECT	SDWACSCT	CSECT/name	RIDS/name

   Table 85 on page 662 summarizes the optional symptoms, which must follow the required symptoms. DAE needs at least three of these optional symptoms to make a useful symptom string.

   Table 85. Summary of optional symptoms

Symptom	SDWA Field	MVS Keyword	RETAIN Keyword
Product/component identifier with the component identifier base	SDWACID, SDWACIDB	PIDS/name	PIDS/name
System completion (abend) code		AB/S0hhh	AB/S0hhh
User completion (abend) code		AB/Udddd	AB/Udddd
Recovery routine name	SDWAREXN	REXN/name	RIDS/name#R
Failing instruction area		FI/area	VALU/Harea
PSW/register difference		REGS/hhhhh	REGS/hhhhh
Reason code, accompanying the abend code or from the REASON parameter of the macro that requests the dump		HRC1/nnnn	PRCS/nnnn
Subcomponent or module subfunction	SDWASC	SUB1/name	VALU/Cname

3. **DAE tries to match the symptom string from the dump to a symptom string for a previous dump** of the same type, that is, SVC dump or SYSMDUMP. When DAE finds a match, DAE considers the dump to be a duplicate.

   When DAE is started, it selects active symptom strings to be used to determine which dumps to suppress. An active symptom is one where either the string was created for a unique dump within the last 60 days, or its dump count was updated within the last 60 days.

   The systems in a sysplex can share the DAE data set to suppress duplicate dumps across the sysplex. While each system in a sysplex can use its own DAE data set, IBM recommends that systems in a sysplex share a DAE data set so that:

   - DAE can write a dump on one system and suppress duplicates on other systems in the sysplex.
   - Only one DAE data set is required, rather than a data set for each system.

   See "Defining a DAE data set" on page 666 for more information, including recommended names for the data set.

4. **DAE updates the symptom strings in storage and, when the dump is written to a dump data set, in the DAE data set**, if updating is requested.

   - For a unique symptom string, DAE adds a new record. The record contains the symptom string, the dates of the first and last occurrences, the incidence count for the number of occurrences, and the name of the system that provided the string.
   - For a duplicate symptom string, DAE updates the incidence count for the string, the last-occurrence date, and the name of the last system that found the string.

   If updating is requested, DAE examines the incoming dump requests against captured dumps. If the incoming dump's symptom string matches any dump on the captured dump queue, it is suppressed. Updates are done when the DAE data set is updated.

   In a sysplex, changes to the in-storage strings of other systems are made after the shared DAE data set is updated. If an incident is occurring at about the same time on multiple systems, multiple dumps will be generated — but only one per system. Dumps on other systems are suppressed after one of the dumps is written, the DAE data set updated, and the updates propagated to the other systems.

   If the system with the original dump fails before it writes the captured dump, the dump will not be suppressed the next time it is requested.

5. **DAE suppresses a duplicate dump**, if DAE is enabled for dump suppression.

   Note that, if you specify an ACTION of SVCD, TRDUMP, or NOSUP on a SLIP command, the command overrides DAE suppression and the system writes the dump. Also, dumps requested by the DUMP operator command are not eligible for suppression.

   When DAE does not suppress a dump, the symptom string is in the dump header; you can view it with the IPCS VERBEXIT DAEDATA subcommand. DAE also issues informational messages to indicate why the dump was not suppressed.

   DAE suppresses a dump when all of the following are true:

   - DAE located in the dump the minimum set of symptoms.

     Note: If more then the minimum number of symptoms are passed, ALL symptoms are used for comparison, up to 150 characters.
   - The symptom string for the dump matches a symptom string for a previous dump of the same type.
   - Either of the following is true:
     - The current ADYSETxx parmlib member specifies SUPPRESS for the type of dump being requested and the VRADAE key is present in the SDWA.
     - The current ADYSETxx parmlib member specifies SUPPRESSALL for the type of dump being requested and the VRANODAE key is absent from the SDWA.
   - The complete dump has been successfully written to a DASD data set or a complete captured dump remains un-deleted.

   Table 86 on page 663 shows the effect of the VRADAE and VRANODAE keys on dump suppression when SUPPRESS and SUPPRESSALL keywords are specified in the ADYSETxx parmlib member. For SUPPRESS, the VRANODAE key can be present or absent; the system does not check it. The table assumes that the symptom string from the dump has matched a previous symptom string.

Table 86. VRADAE and VRANODAE keys on dump suppression when SUPPRESS and SUPPRESSALL keywords are specified in ADYSETxx

ADYSETxx Option	VRADAE Key in SDWA	VRANODAE Key in SDWA	Dump Suppressed?
SUPPRESS	Yes	N/A	Yes
SUPPRESS	No	N/A	No
SUPPRESSALL	Yes	No	Yes

ADYSETxx Option	VRADAE Key in SDWA	VRANODAE Key in SDWA	Dump Suppressed?
SUPPRESSALL	No	Yes	No
SUPPRESSALL	No	No	Yes
SUPPRESSALL	Yes	Yes	No

Table 86. VRADAE and VRANODAE keys on dump suppression when SUPPRESS and SUPPRESSALL keywords are specified in ADYSETxx (continued)

The only way to ensure that a dump is **not** suppressed, regardless of the contents of the ADYSETxx parmlib member, is to specify the VRANODAE key in the SDWA.

Managing rapidly recurring dumps

DAE can suppress rapidly recurring dumps automatically and the support staff does not need to be aware when a dump request recurs. However, a surge of dump requests could affect system performance, even though the dumps are suppressed. The surge could go unnoticed for hours. To help the support staff take actions to avoid impact to users, the system can notify you of high-frequency dump requests.

To obtain notification, add a NOTIFY parameter to the SVCDUMP statement on the ADYSETxx parmlib member to establish a threshold for notification. The SVCDUMP statement must also specify UPDATE. The default threshold is 3 dumps requested in 30 minutes for the same symptom string. The notification time is measured from completion or suppression of dumps, rather than from initiation of dumps.

The notification is made by the event notification facility (ENF). You can use an ENF exit to:

- Notify the support staff by a message or a signal to a beeper
- Use automation

Any program can receive the ENF signal. If active, the First Failure Support Technology (FFST) issues a generic alert in response to this ENF signal.

If DAE is stopped and restarted, DAE begins counting dumps again to reach the threshold.

If each system has its own DAE data set, notification is for a system. If the systems of a sysplex share the DAE data set, notification is for the sysplex. For example, with a shared DAE data set, four dumps for the same symptom string on the same or different systems in 25 minutes would cause notification if the ADYSETxx parmlib member contains NOTIFY(4,25).

Note: The system in the sysplex that crosses the notification threshold is the system that does the notify.

Planning for DAE dump suppression

Planning for DAE dump suppression consists of tasks to be done before an initial program load (IPL). The system programmer performs the following tasks:

- Selecting or creating an ADYSETxx parmlib member
- Defining a DAE data set

Selecting or creating an ADYSETxx parmlib member

Select or create an ADYSETxx parmlib member to be used at IPL. IBM supplies three ADYSETxx members:

- **ADYSET00**, which starts DAE and keeps 400 symptom strings in virtual storage. The IBM-supplied ADYSET00 member contains:

  ```
  DAE=START,RECORDS(400),
      SVCDUMP(MATCH,SUPPRESSALL,UPDATE,NOTIFY(3,30)),
      SYSMDUMP(MATCH,UPDATE)
  ```

  ADYSET00 does not suppress SYSMDUMP dumps because installation-provided programs deliberately request them. If desired, change the ADYSETxx member being used to suppress SYSMDUMP dumps.

- **ADYSET01**, which stops DAE processing. The IBM-supplied ADYSET01 member contains:

  ```
  DAE=STOP
  ```

  When using the DAE Display facility's TAKEDUMP (T) action in a sysplex where DAE is active, you must change the contents of ADYSET01 to:

  ```
  DAE=STOP,GLOBALSTOP
  ```

- **ADYSET02**, which contains the same parameters as ADYSET00.

The IBM-supplied IEACMD00 parmlib member issues a SET DAE=00 command, which activates ADYSET00 during IPL. If you do not want DAE to start during IPL, change IEACMD00 to specify SET DAE=01.

For a sysplex, IBM recommends that you use the same ADYSETxx parameter values in each system. To use the same values, use a shared SYS1.PARMLIB. If your installation does not share a SYS1.PARMLIB, make the ADYSETxx and IEACMDxx members in the SYS1.PARMLIB for each system identical. A shared ADYSETxx or identical ADYSETxx members should specify SHARE(DSN) to share the DAE data set.

IBM recommends that the ADYSETxx member specify SUPPRESSALL, which requests that dumps be suppressed even though the component or program did not request dump suppression with a VRADAE key in the system diagnostic work area (SDWA). SUPPRESSALL is useful because it allows more dumps to be eligible for suppression.

In the example shown in Figure 240 on page 665, the systems in the sysplex share a DAE data set, SYS1.DAESHARE, so DAE can suppress a duplicate of a previous dump from any system. This member also specifies SUPPRESSALL.

```
DAE=START,RECORDS(400),
    SVCDUMP(MATCH,SUPPRESSALL,UPDATE,NOTIFY(3,30)),
    SYSMDUMP(MATCH,UPDATE)
    SHARE(DSN,OPTIONS),
    DSN(SYS1.DAESHARE)
```

Figure 240. Example: An ADYSETxx Member for a System in a Sysplex

The ADYSET00 member specifies RECORDS(400). If your system does not suppress a dump when the matching symptom string is in the DAE data set, you might need more than 400 records in storage; the IBM Support Center can advise you.

Changing Parmlib Members to Change DAE Processing: While the system is running, change the DAE data set or parameters for the dumps by creating a new ADYSETxx parmlib member. See “Changing DAE processing in a Sysplex” on page 669 for the operator actions needed to change the parmlib member.

There is another benefit when all the systems in a sysplex are sharing the DAE data set. That is, after DAE is started on each system using an ADYSETxx member which at least contains SHARE(DSN). One operator command can set the DAE values to be the same on all systems. This is accomplished by issuing the SET DAE= command, for an ADYSEYxx member which includes the GLOBAL parameter. ALL systems sharing the DAE data set will be effected.

In Figure 241 on page 665, the following ADYSET04 member changes the DAE data set being used on all systems to SYS1.DAESH2 and changes the dump options on all systems.

```
DAE=START,RECORDS(400),
    SVCDUMP(MATCH,SUPPRESSALL,UPDATE,NOTIFY(3,30)),
    SYSMDUMP(MATCH,UPDATE)
    SHARE(DSN,OPTIONS),
    DSN(SYS1.DAESH2)
    GLOBAL(DSN,OPTIONS)
```

Figure 241. Example: An ADYSETxx Member with GLOBAL

None of the changes made using operator commands are kept across an IPL of a system. At IPL, each system will again use the member specified in IEACMD00 or the COMMNDxx member being used. To make the changes permanently effective, do one of the following:

- Make the changes in ADYSET00 and the default IEACMD00 will start DAE.
- Make the changes in the ADYSETxx member and update a COMMNDxx member to start the ADYSETxx using the SET DAE=xx statement. Then update the appropriate IEASYSxx member to include the CMD=xx statement. Once complete, remove the SET DAE=00 within the IEACMD00 member to ensure that the two commands do not complete out of sequence.

Defining a DAE data set

Define a DAE data set when defining system data sets. When the system is IPLed or if DAE is stopped and restarted, DAE should continue using the DAE data set previously used.

1. **Define the DAE data set in a DD statement.** Use the default name of SYS1.DAE for a single system; use a different name for a DAE data set shared by systems in a sysplex.

   The sample DD statement shown in Figure 242 on page 666 is for a DAE data set used by a single system.

   ```
   //DAE    DD  DSN=SYS1.DAE,DISP=(,CATLG),VOL=(,RETAIN,SER=SG2001),
   //            DCB=(RECFM=FB,LRECL=255,DSORG=PS,BLKSIZE=0),
   //            UNIT=3390,SPACE=(TRK,(6,2))
   ```

   Figure 242. Example: DAE Data Set for Single System

   In a sysplex, each system can have its own DAE data set, but IBM recommends that all systems in a sysplex share a DAE data set.

   The sample DD statement in Figure 243 on page 666 is for a DAE data set shared by the systems in a sysplex. The statement will catalog the DAE data set in the shared master catalog or in the master catalog on each system that uses it.

   ```
   //DAE    DD  DSN=SYS1.DAESHARE,DISP=(,CATLG),VOL=(,RETAIN,SER=SG1055),
   //            DCB=(RECFM=FB,LRECL=255,DSORG=PS,BLKSIZE=0),
   //            UNIT=3390,SPACE=(TRK,(12,2))
   ```

   Figure 243. Example: DAE Data Set Shared by Sysplex Systems

   If you manage your dumps with the hierarchical storage manager (HSM), consider using an HSM purge time of 60 days to correspond to the DAE record aging of 60 days.

2. **Provide DAE data set integrity through a serialization component**, such as global resource serialization.

   For a single system, the DAE data set is a local resource. The default DAE data set, SYS1.DAE, is defined as a local resource in the default global resource serialization resource name list (RNL). If you give the DAE data set another name, add the name to the SYSTEMS exclusion RNL to avoid contention when more than one system uses the same DAE data set name for physically different data sets.

   For systems in a sysplex, the shared DAE data set is a global resource. To make global resource serialization treat it as a global resource, do one of the following:

   - Give the DAE data set a name other than SYS1.DAE. For example, SYS1.DAESHARE.
   - If you use the name SYS1.DAE, delete the DAE data set entry from the default SYSTEMS exclusion RNL. The DAE data set entry is `SYSDSN SYS1.DAE`.

   For information, see *z/OS MVS Planning: Global Resource Serialization*.

3. **Control access to the DAE data set.** On a single system or on all systems sharing the DAE data set in a sysplex, use Resource Access Control Facility (RACF) to control access. Enter a RACF ADDSD command to define a data set profile for the DAE data set.

Accessing the DAE data set

A DAE data set that is used by one system or is shared by systems in a sysplex is accessed by:

- Invoking the IPCS DAE Display panel
- Generating a suppressed dump
- Editing the DAE data set

Invoking the IPCS DAE display panel

For the ways to invoke the panel, see IPCS option 3.5 in the *z/OS MVS IPCS User's Guide*. On the panel, you can:

- View the symptom strings the data set contains by entering:
  - The date of the dump,
  - The last date for the string,
  - The number of times the dump has been requested,
  - And the last system that requested the dump.
- Search the Entry list for symptoms, system names, dates, etc.
- Navigate through the sysplex dump directory (or whatever dump directory is active) for the symptom string.
- View the dump title for a symptom string.

Generating a suppressed dump

You may want to obtain a dump that is being suppressed. Perhaps the first dump was ignored and thrown out, but since then the dump has been requested often enough so that you would like to analyze the dump. Do the following to obtain the suppressed dump through the IPCS TAKEDUMP option:

1. Customize the TSO user ID that will invoke the TAKEDUMP action. Make sure it:
   - has authority to issue an MVS operator SET command and, if DAE is active in a sysplex, the ROUTE command
   - has RACF UPDATE access to the DAE data set.
2. Ensure that the ADYSET01 member(s) contains DAE=STOP (or DAE=STOP,GLOBALSTOP in a sysplex).
3. Check that the active IKJTSOxx member includes the program name ADYOPCMD in the AUTHCMD NAMES section.
4. In a sysplex, the maximum benefit is realized when DAE is started using ADYSETxx members which contain at least SHARE(DSN) — enabling shared data set activities.
5. Use the IPCS DAE dialog Panel to issue action code T (the TAKEDUMP option) on the line showing the symptom string of interest.

To process the TAKEDUMP option of the IPCS DAE dialog, DAE processing is stopped, dialog processing occurs, and DAE processing is restarted on the systems involved. There are some cases where a particular system may end up using different DAE parameters from those it was previously using. Table 87 on page 668 illustrates possible results.

For this discussion there are two systems (SY1 and SY2), and five ADYSETxx members involved. Members ending with G1 and G2 include GLOBAL(DSN,OPTIONS) parameters. Members ending with S1 and S2 have SHARE(DSN,OPTIONS) without GLOBAL options. The center column in Table 87 on page 668 indicates the system where the TSO user is issuing the TAKEDUMP request.

Table 87. Examples of when DAE parameters may change

Start State		TSO	Final State	
SY1	SY2		SY1	SY2
G1	G1	SY1	G1	G1[1]
S1	S2	SY1	S1	S1[2]
S1	S2	SY2	S2	S2[2]
S1	G1	SY1	S1	S1[2]
S1	G2	SY2	G2	G2[2]
G2	S2	SY2	S2	S2[2]
00	*	SY1	00	*[3]

Note:

1. GLOBAL(DSN) systems remain synchronized.
2. When all systems are NOT in GLOBAL(DSN) mode, the system will be started using the member last active on the system where the IPCS TAKEDUMP dialog runs.
3. No change to other systems if all work is done on a system which is not sharing the DAE data set. Here, ADYSET00 contains the default IBM—supplied values.
4. The commands necessary to accomplish the task are issued by the TSO user at their dispatching priority. It is possible that the system may not dispatch the TSO user due to that dispatching priority, and therefore the action may not complete in a timely manner.

The system will generate the next dump for the symptom string. After that dump, DAE resumes suppressing dumps for the symptom string.

Note: Despite specifying action code T, the dump might still be suppressed. See "Determining why a dump was suppressed" on page 671 for the reasons, other than DAE suppression.

Editing the DAE data set

Edit the DAE data set, using Interactive System Productivity Facility (ISPF) edit. For ISPF edit, see *z/OS ISPF Dialog Developer's Guide and Reference*. You must have WRITE access to the DAE data set. Once in ISPF Edit, use the Edit macro ADYUPDAT as described below.

In the edit session, type one of the following on the command line, place the cursor on the symptom string line for the dump, and press ENTER. If the cursor is on the command line, the first symptom string is used. Note that DAE must be stopped before these actions and started again after.

```
ADYUPDAT TAKEDUMP
ADYUPDAT NODUMP
```

ADYUPDAT TAKEDUMP requests that the next dump be generated for this symptom string. SLIP can still suppress the dump.

ADYUPDAT NODUMP undoes the effect of TAKEDUMP, if it was in effect. Otherwise, NODUMP results in no action.

In the edit session, you can also delete every symptom string that has not been updated within a specified number of days. You must SAVE the DAE data set for the deletions to take effect. To request the deletions, enter on the command line:

```
ADYUPDAT CLEANUP nnn
```

nnn
: number of days a record has not been updated for it to be selected for deletion. The default is 60 days.

ADYUPDAT always issues a status message reflecting the outcome of the command.

Stopping, starting, and changing DAE

If an ADYSET00 parmlib member is used and the DAE data set is allocated, DAE starts during IPL. Normally, DAE runs at the same time as the system. However if DUMPSRV is ever cancelled or restarted, DAE restarts as well albeit with dump suppression inactive You must manually reactivate dump suppression using the SET DAE=xx command.

An operator can stop and start DAE with the following steps. One reason to use these steps would be to change to a different ADYSETxx parmlib member with different parameters.

Stopping DAE

You can stop DAE with a SET DAE command that specifies the ADYSET01 parmlib member, which contains a DAE=STOP statement:

```
SET DAE=01
```

Starting DAE

You can start DAE with a SET DAE command that specifies an ADYSETxx parmlib member that contains the DAE=START parameter, such as an installation-provided ADYSET03 parmlib member:

```
SET DAE=03
```

Changing DAE processing in a Sysplex

The operator can change all DAE processing in a sysplex, if desired. For example, the operator can do the following to make all systems in a sysplex use a different ADYSETxx member:

1. Stop DAE processing using the IBM-supplied ADYSET01 member:

   ```
   ROUTE *ALL,SET DAE=01
   ```

   Another way to stop DAE processing on all systems in a sysplex is to specify in the SET DAE command an ADYSETxx member containing a GLOBALSTOP parameter.
2. Start DAE processing using, for example, the ADYSET04 member:

   ```
   ROUTE *ALL,SET DAE=04
   ```

Using a SLIP command to suppress dumps

Some dumps are almost never needed. For example, some abend codes tell the diagnostician enough to solve the problem. For these codes, place SLIP operator commands in an IEASLPxx parmlib member to suppress the unneeded dumps. The IBM-supplied IEASLP00 member contains the SLIP commands to suppress abend dumps that are seldom needed.

Using SLIP to suppress dumps also suppresses message IEA995I, which contains symptom dump information. The system may document the abend in a LOGREC error record.

To suppress dumps for an abend code, specify a SLIP operator command with one of the following ACTION parameters. Place all the SLIP commands in an IEASLPxx parmlib member and activate the IEASLPxx member with the following command in a COMMNDxx or IEACMDxx member that is always used:

```
CMD='SET SLIP=xx'
```

SLIP Parameter
   Dumps Suppressed

ACTION=NODUMP
All dumps

ACTION=NOSVCD
SVC dumps

ACTION=NOSYSA
ABEND SYSABEND dumps

ACTION=NOSYSM
ABEND SYSMDUMP dumps

ACTION=NOSYSU
ABEND SYSUDUMP dumps

For example, to suppress SVC dumps and ABEND SYSMDUMP dumps for abend code X'B37', add the following to IEASLPxx:

```
SLIP SET,COMP=B37,ACTION=(NOSVCD,NOSYSM),END
```

For more information about the following topics, see the following references:

- See *z/OS MVS Initialization and Tuning Reference* for the IEASLPxx member.
- See *z/OS MVS System Commands* for the SLIP operator command.

Using an ABEND macro to suppress dumps

A program can suppress a dump by issuing an ABEND macro without a DUMP parameter. Application programmers should not specify a DUMP parameter when a symptom dump can provide enough information for diagnosis.

See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for more information about the ABEND macro.

Using installation exit routines to suppress dumps

An installation can add installation exit routines, summarized in Table 88 on page 670, to suppress dumps. Use IEAVTABX if you want to suppress abend dumps based on the job name, abend code, or other information in the system diagnostic work area (SDWA). Use IEAVTSEL if you want to discard an SVC or SYSMDUMP dump based on information in the dump header or from DAE. Use JES2 exit 4 or JES3 exit IATUX34 to suppress different types of dumps.

Table 88. Summary of installation exit routines for dump suppression

Exit	Processing	Dump Suppression
IEAVTABX	Before any ABEND dump	Routine(s) can place a return code of 8 in register 15 to suppress the requested dump.
IEAVTSEL	After an SVC dump or ABEND SYSMDUMP dump, if the dump was not suppressed by DAE	Routine(s) can clear the dump data set.
JES2 exit 4 or JES3 IATUX34	For any JCL statement	Can change the DSNAME parameter on a dump DD statement to DUMMY to suppress the dump.

For more information about the following topics, see the following references:

- See *z/OS MVS Installation Exits* for IEAVTABX and IEAVTSEL.
- See *z/OS JES2 Installation Exits* for the JES2 exit 4 routine.
- See *z/OS JES3 Customization*for the JES3 IATUX34 exit routine.

Determining why a dump was suppressed

If an intended dump is missing, use this list to decide why. The list gives reasons why dumps are suppressed, including the ways discussed in this topic. In planning for problem determination, be aware of all of these ways so that your installation does not suppress intended dumps.

- **DAE suppression of dumps.** See "Using DAE to suppress dumps" on page 661.
- **SLIP command that suppresses all dumps for an abend code.** See "Using a SLIP command to suppress dumps" on page 669.
- **An ABEND macro without a DUMP parameter.** See "Using an ABEND macro to suppress dumps" on page 670.
- **An MVS installation exit routine that suppresses the dump.** See "Using installation exit routines to suppress dumps" on page 670.
- **Resource Access Control Facility (RACF) control of programs in an address space to be dumped**: Beginning with RACF 1.8.1, the installation can protect ABEND dumps of programs using the FACILITY class. The protection can keep you from accessing a dump.
- **Dump on another system blocked by SYSDCOND in the PROBDESC area and the IEASDUMP.QUERY routine**. A dump on another system in a sysplex is requested by a DUMP command or SDUMPX macro with a REMOTE parameter. If the area specified by the PROBDESC parameter contains SYSDCOND, the dump on the other system is not written because of either of the following on the other system:
  - No IEASDUMP.QUERY routine exists
  - No IEASDUMP.QUERY routine returns a code of 0
- **Dump suppressed by CHNGDUMP command**. If a CHNGDUMP command specifies NODUMP for SVC dumps:
  - All SVC dumps on the system are suppressed.
  - If a DUMP command or SDUMPX macro includes a REMOTE parameter, the dump on the local system and the dumps on other systems in the sysplex are suppressed.
- **Dump on another system suppressed by CHNGDUMP command on the other system**. If a DUMP command or SDUMPX macro includes a REMOTE parameter and a CHNGDUMP command previously entered on another system in the sysplex specifies NODUMP for SVC dumps, the SVC dump on the other system is suppressed. The dump on the local system is written.

The system can also place the dump in another data set, so that it is not in the original data set specified in a message you received:

- **An installation exit routine at JES2 exit 4 or at JES3 exit IATUX34 can change the dump data set name.**
- **DUMPDS operator command can redirect SVC dump output.** The command can redirect SVC dump output to other SYS1.DUMPxx data sets.

For more information about the following topics, see the following references:

- See *z/OS Security Server RACF Security Administrator's Guide* for the FACILITY class to control access to program dumps.
- See *z/OS MVS System Commands* for the DUMP, DUMPDS, and SLIP commands.
- See *z/OS MVS Installation Exits* for the IEAVTABX and IEAVTSEL exit routines.
- See *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU* for the SDUMPX macro.
- See *z/OS MVS Programming: Authorized Assembler Services Guide* for the IEASDUMP.QUERY routine.
- See *z/OS JES2 Installation Exits* for the JES2 exit 4 routine.
- See *z/OS JES3 Customization*for the JES3 IATUX34 exit routine.

Chapter 22. Messages

The system issues messages to do the following:

- Tell the operator or system programmer of progress and problems in system processing
- Ask the operator to take actions and make decisions
- Tell the application programmer how the system ran the application program and of problems in the application program

The system issues messages from the base control program components and a variety of subsystems, products, and applications. Applications running under the system can also issue their own messages.

Producing messages

You can get the system to produce a message by issuing a macro in any program or by asking an operator to enter a command. The macros and command are:

- LOG operator command to write a message to the SYSLOG and OPERLOG
- WTL macro to write a message to the SYSLOG and OPERLOG

  Note: Use WTO specifying MCSFLAG=HRDCPY instead of using WTL, which provides additional information with the WTO message that is not with the WTL message.
- WTO macro to write a message to the operator
- WTOR macro to write a message to the operator and request a reply

Use the following for related activities:

- DOM macro to delete an operator message or group of messages from the display screen of a console
- REPLY operator command to answer a message
- WRITELOG operator command to start, stop, or print the SYSLOG and to change the output class for the SYSLOG

For additional information, see the following resources:

- See *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for DOM and *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for the WTO, WTOR, and WTL macros.
- See *z/OS MVS System Commands* for the LOG, REPLY, and WRITELOG commands.

Receiving messages

The system issues messages through WTO and WTOR macros to the following locations. Routing codes determine the display and print location of the messages.

- Console
- Extended console
- Hard-copy log
- Job log
- SYSOUT data set

The system issues messages to the SYSLOG and OPERLOG using the WTL macro.

The access methods issue messages directly to one of the following locations:

- Display terminal
- Output data set

Console

Messages sent to a console with master authority are intended for the operators. The system writes in the hard-copy log all messages sent to a console, regardless of whether the message is displayed.

Hard-Copy log

The hard-copy log is a record of all system message traffic:

- Messages to and from all consoles
- Commands and replies that are entered by the operator.

In a dump, these messages appear in the master trace. With JES3, the hard-copy log is written to the SYSLOG, the OPERLOG, or both. With JES2, the hard-copy log is written to the SYSLOG, the OPERLOG, or both; it can also be viewed using a product like System Display and Search Facility (SDSF). For more information abut SDSF, see *z/OS SDSF Operation and Customization*.

System log

The SYSLOG is a SYSOUT data set provided by the job entry subsystem (either JES2 or JES3). SYSOUT data sets are output spool data sets on direct access storage devices (DASD). Use SDSF to view the SYSLOG to check for problems. The SYSLOG consists of the following:

- All messages issued through WTL macros
- All messages entered by LOG operator commands
- Usually, the hard-copy logs
- Any messages routed to the SYSLOG from any system component or program

Job log

Messages sent to the job log are intended for the programmer who submitted a job. Specify the system output class for the job log in the MSGCLASS parameter of the JCL JOB statement.

SYSOUT data set

Messages sent to a SYSOUT data set are intended for a programmer. These messages are issued by an assembler or compiler, the binder and loader, and an application program. To make all messages about a program appear in the same SYSOUT listing, specify the same class for the SYSOUT data set and in the MSGCLASS parameter on the JCL JOB statement.

Receiving symptom dumps

A symptom dump is a system message, either message IEA995I or a numberless message, which provides some basic diagnostic information for diagnosing an abend. Often the symptom dump information can provide enough information to diagnose a problem.

The example in Figure 244 on page 675 shows the symptom dump for an abend X'0C4' with reason code X'4'. This symptom dump shows the following information:

- Active load module ABENDER is at address X'00006FD8'.
- The failing instruction was at offset X'12' in load module ABENDER.
- The address space identifier (ASID) for the failing task was X'000C'.

```
IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=0C4  REASON CODE=00000004
 TIME=16.44.42  SEQ=00057  CPU=0000  ASID=000C
 PSW AT TIME OF ERROR  078D0000   00006FEA  ILC 4  INTC 04
   ACTIVE LOAD MODULE=ABENDER   ADDRESS=00006FD8  OFFSET=00000012
   DATA AT PSW  00006FE4 - 00105020  30381FFF  58E0D00C
   GPR  0-3  FD000008  00005FF8  00000014  00FD6A40
   GPR  4-7  00AEC980  00AFF030  00AC4FF8  FD000000
   GPR  8-11 00AFF1B0  80AD2050  00000000  00AFF030
   GPR 12-15 40006FDE  00005FB0  80FD6A90  00006FD8
 END OF SYMPTOM DUMP
```

Figure 244. Example: Symptom Dump Output

Symptom dumps appear in the following places:

- For SYSUDUMP and SYSABEND ABEND dumps: in message IEA995I, which is routed to the job log.
- For a SYSMDUMP ABEND dump: in message IEA995I in the job log and in the dump header record.
- For an SVC dump: in the dump header record.
- For any dump in a Time Sharing Option/Extensions (TSO/E) environment: displayed on the terminal when requested by the TSO/E PROFILE command with the WTPMSG option.
- In response to a DISPLAY DUMP,ERRDATA operator command, which displays information from SYS1.DUMPxx data sets on direct access.

If the information in a symptom dump is enough for diagnosis, do not provide a DD statement for a dump.

For additional information, see one of the following resources:

- See Chapter 9, “Master trace,” on page 205 for information on master trace.
- See *z/OS MVS JCL Reference* for the JOB statement.
- See Chapter 5, “ABEND dump,” on page 121 for information about the ABEND dump header record
- See Chapter 2, “SVC dump,” on page 7 for information about the SVC dump header record
- See *z/OS TSO/E Command Reference* for the PROFILE command.

Planning message processing for diagnosis

Your installation can change message processing in a number of ways to optimize diagnosis. Your installation can do the following tasks:

- Control message location
- Suppress messages
- Automate message processing
- Not retain action messages
- Suppress the symptom dump message (IEA995I)

This section can help you find the information you need to optimize message processing for your installation.

Controlling message location

An installation can change the following:

- The routing codes for specific messages to control
- On which console to display a message.

Change or specify the routing codes using the following methods:

- A WTO or WTOR macro specifies the routing code for the message that the macro creates.

- A WTO/WTOR installation exit routine changes the routing code for any WTO or WTOR message. This exit routine is the routine named in the USEREXIT parameter in the MPFLSTxx parmlib member or IEAVMXIT.
- The JES3 MSGROUTE initialization statement changes the routing code of JES3 console messages.
- The JES3 CONSOLE initialization statement can control which messages a JES3 console receives.
- Subsystem interface listeners, such as NetView, can change the routing codes.

Suppressing messages

An installation can use the following to suppress messages. Suppressed messages do not appear on a console.

- An MPFLSTxx parmlib member can specify message suppression. A suppressed message does not display on a console, but writes to the hard-copy log.
- A WTO/WTOR installation exit routine can suppress any WTO/WTOR messages or override suppression. This exit routine is the routine named in the USEREXIT parameter in the MPFLSTxx parmlib member or IEAVMXIT.
- The JES2 installation exit routine, Exit 10, can suppress messages issued by the JES2 main task.
- The JES3 installation exit routine, IATUX31, can suppress messages routed to JES3 consoles.
- The CONTROL V operator command can suppress messages by specifying the message levels to display at a console.
- VARY operator command can change the messages received by a console by specifying the routing codes of messages to be displayed.
- The LEVEL keyword on the CONSOLE statement in the CONSOLxx parmlib member also can suppress messages by specifying the message levels to be displayed at a console. The ROUTCODE keyword can specify the routing codes of messages that are displayed at the console.

Handling message floods

A MSGFLDxx parmlib member can specify the criteria required to recognize that a message flooding situation is occurring and the actions to be taken to handle the message flood. The message flood automation policy can prevent flood messages from being:

- prevent flood messages from being displayed on a console
- prevent flood messages from being queued for automation
- prevent flood messages from being written to the SYSLOG or OPERLOG

Messages are written to the SYSLOG or OPERLOG identifying the job or specific message that is causing the message flood; messages are also written to the logs when action is taken against specific messages or messages from that job.

Automating message processing

An MPFLSTxx parmlib member can specify that a message is to be passed to an automation subsystem, such as NetView. The automation subsystem can perform actions that the operator would have performed without operator intervention.

Not retaining action messages

The MPFLSTxx parmlib member can specify not to retain a message using the Action Message Retention Facility (AMRF). An operator cannot recall to the screen action messages that are not retained.

Suppressing symptom dumps (IEA995I)

Table 89 on page 677 lists ways to suppress symptom dumps for ABEND dumps. These ways suppress only message IEA995I; symptom dumps continue to appear in other locations.

Table 89. Suppressing symptom dumps		
Dump Type	**CHNGDUMP Operator Command used to Suppress Symptom Dump**	**Parmlib Member used to Suppress Symptom Dump**
SYSABEND ABEND	`CHNGDUMP SET,SYSABEND,SDATA=(NOSYM)`	IEAABD00
SYSMDUMP ABEND	`CHNGDUMP SET,SYSMDUMP=(NOSYM)`	IEADMR00
SYSUDUMP ABEND	`CHNGDUMP SET,SYSUDUMP,SDATA=(NOSYM)`	IEADMP00

For more information, see one of the following references:

- See *z/OS MVS Planning: Operations* for message flood automation, controlling message display, suppressing messages, AMRF, and automating messages in a sysplex.
- See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for the WTO and WTOR macros.
- See *z/OS MVS Initialization and Tuning Reference* for the MPFLSTxx and CONSOLxx members.
- See *z/OS MVS Installation Exits* for the exit routine named in the USEREXIT parameter and for IEAVMXIT.
- See *z/OS JES3 Initialization and Tuning Reference* for the JES3 initialization statements.
- See *z/OS JES2 Installation Exits* for Exit 10.
- See *z/OS JES3 Customization* for the IATUX31 exit.
- See *z/OS MVS System Commands* for the CONTROL V and the CHNGDUMP operator commands.

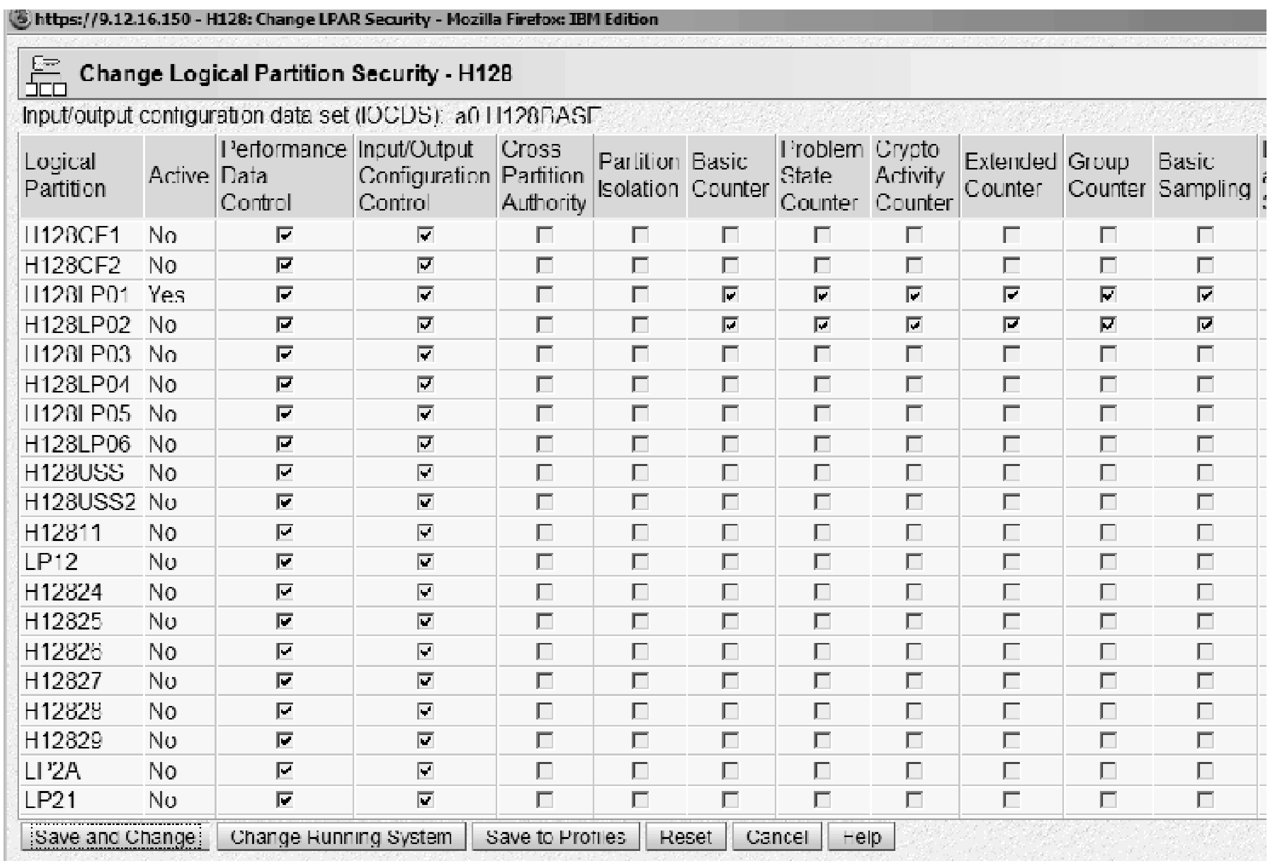
Chapter 23. Hardware Instrumentation Services

Hardware instrumentation services (HIS) is a function that collects hardware event data for IBM System z10 or later machines.

IBM may request that you provide hardware event data for diagnostics. Before you start the HIS data collection, you may first need to authorize the sampling facilities and counter set types you want to use through the support element (SE) console. For each LPAR of interest, you will need to verify each of the following settings:

- Basic counter set authorization control
- Problem state counter set authorization control
- Crypto activity counter set authorization control
- Extended counter set authorization control
- Basic sampling authorization control

The LPAR to be measured may be already activated and running. A dynamic update can be done to enable the CPU Measurement Facility without disrupting this LPAR by selecting the Change LPAR Security icon on the SE or HMC. The Save and Change options on the LPAR security panel will dynamically change the running system and also save the change to the activation profiles. See Figure 245 on page 679 for an example.



Logical Partition	Active	Performance Data Control	Input/Output Configuration Control	Cross Partition Authority	Partition Isolation	Basic Counter	Problem State Counter	Crypto Activity Counter	Extended Counter	Group Counter	Basic Sampling
H128CF1	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128CF2	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128LP01	Yes	☑	☑	☐	☐	☑	☑	☑	☑	☑	☑
H128LP02	No	☑	☑	☐	☐	☑	☑	☑	☑	☑	☑
H128LP03	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128LP04	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128LP05	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128LP06	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128USS	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H128USS2	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12811	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
LP12	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12824	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12825	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12826	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12827	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12828	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
H12829	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
LP2A	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
LP21	No	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

Figure 245. Change LPAR Security panel for active LPAR

For LPARs that have not yet been activated, the CPU Measurement Facility can be enabled using the security tab for the activation profile. See Figure 246 on page 680 for an example.

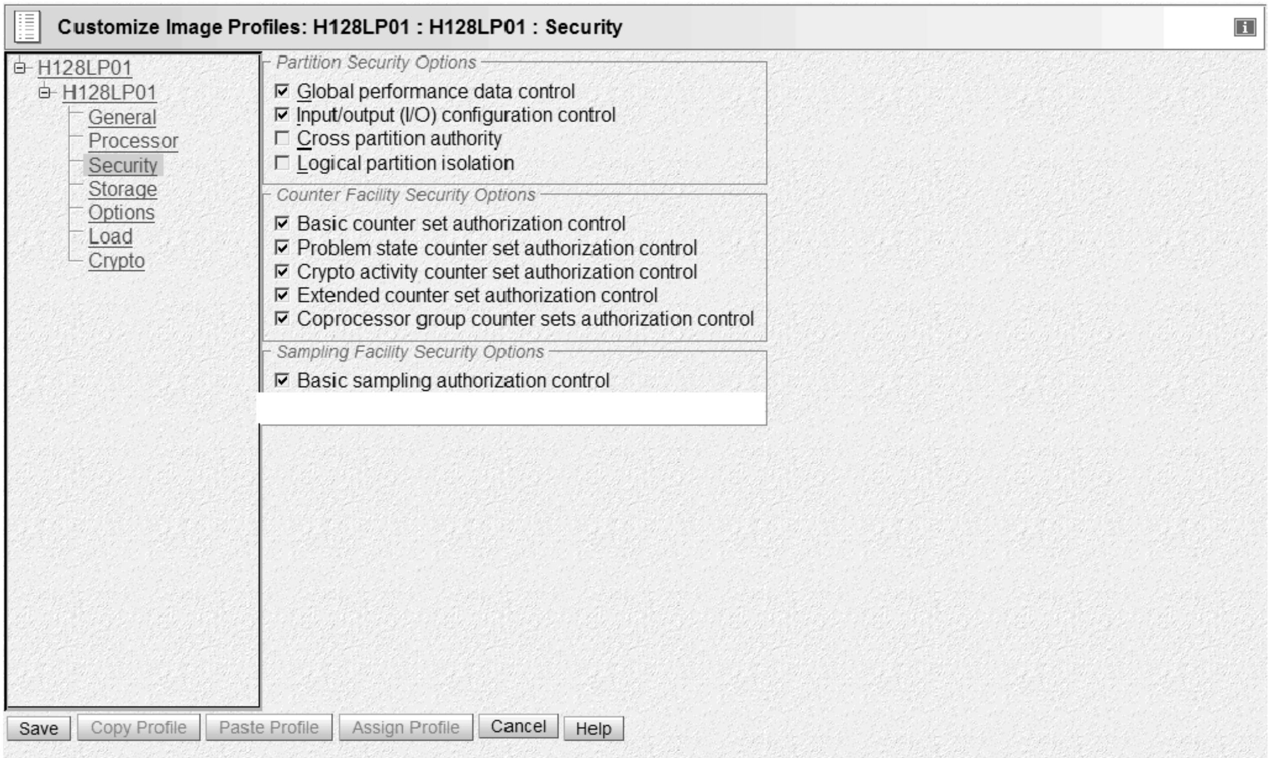


Figure 246. Activation profile security panel for new LPAR

For information about how to set up the authorization of the sampling facilities and counter sets, see *Support Element Operations Guide* for System z10 machine on the Resource Link home page (www.ibm.com/servers/resourcelink).

In addition, with the enhanced-monitor facility hardware released with z196 machines, the HIS function expands into a z/OS software event data collector that will be used by IBM for improved problem analysis. The z/OS event counters are viewed as an additional counter set, but there is no authorization required to use the hardware. For information on the enhanced-monitor facility hardware, see *z/Architecture Principles of Operation*.

Chapter 24. Data Privacy for Diagnostics (DPfD)

Enterprises have a requirement to prevent customer personal or other sensitive information from being exposed to those who have no need to see such data. In the course of data processing, various types of system and application errors can require an installation to send diagnostic data to program vendors for analysis and problem resolution. Diagnostic data on the MVS platform typically takes the form of SVC dumps, Stand-Alone dumps (SADMP), LOGREC data, traces, system and application logs, etc. Dumps have the greatest exposure of containing sensitive data along with the required system and or application data.

Data Privacy for Diagnostics provides facilities for tagging sensitive data and subsequently producing redacted dumps which do not contain the tagged sensitive data. The original dump should be retained for the entire period that problem analysis is being conducted. The redacted dump would be made available to the appropriate program vendors.

To accomplish data tagging by applications, a set of services are provided by the storage management interfaces of the MVS operating system for Independent Software Vendor (ISV) applications and operating system components to use. For more information on tagging storage, see the Tagging 64-bit memory objects for data privacy topic of the Tagging 64-bit memory objects for data privacy in *z/OS MVS Programming: Assembler Services Guide*. Once data has been tagged, a set of services available via Interactive Problem Control System (IPCS) parts may be used to post process the dumps taken on z15 or later processors.

The following functions are being provided by OA57570:

- You may redact any data tagged as sensitive=yes in SVC or stand-alone dumps captured on a z15 or later processor using the sample job SYS1.SAMPLIB(BLSJDPFD)
- One may obtain a report about the pages which were marked as sensitive in a redacted dump using 'SYS1.SBLSCLI0(BLSXREDR)' providing an input dump dataset name, and optionally a filtering ASID.

The Data Privacy for Diagnostics Analyzer is introduced by the solution for OA58114. The Data Privacy for Diagnostics Analyzer provides the facilities to scan and identify data within dumps that may be sensitive personal information (SPI). Because of the complexity of guidelines, requirements and SPI data identification, the Analyzer requires installations to tailor its privacy controls for whatever unique distinctions are necessary to filter out SPI from diagnostic data. Over redaction is possible which can negatively impact problem diagnosis. Most system areas are tagged as not having sensitive data, so it is possible for some SPI to escape redaction. At its core is an application which runs via batch jobs. Those jobs may be tailored through an IPCS dialog, or manually managed by the installation. The details for setup and execution will be found within the Interactive Problem Control System (IPCS) framework which consists of documentation within the IPCS Customization, IPCS Commands and IPCS User's Guide publications

Chapter 25. IEAVBPRT: Validated Boot for z/OS print utility

The IEAVBPRT utility reports the following information after a validated boot IPL:

- Audit records that were created
- Certificate extracts that are being used
- Certificate extracts that were found not to be valid

For an enforce-mode IPL, no more than 1 audit record would be produced because any relevant issue would cause the system to enter a wait state right after building the audit record.

The IEAVBPRT utility provides options to generate a detailed report or a summary.

The same information is also provided by the IEAVBIPC utility within IPCS (VERBEXIT IEAVBIPC).

Invoking the IEAVBPRT utility

Invoke the IEAVBPRT utility as a job step program (such as, `EXEC PGM=IEAVBPRT`). The report output is written according to the SYSPRINT DD statement. IEAVBPRT opens the SYSPRINT DD with the attributes `RECFM=FBA,LRECL=133`.

The following example shows sample JCL for such a job step:

```
//VBPRT1    EXEC PGM=IEAVBPRT,TIME=1440,PARM=parm
//SYSPRINT DD   SYSOUT=A
```

The value of the *parm* parameter can be:

SUMMARY
: Generates a summary report. This is the default value.

DETAIL
: Generates a detailed report.

IEAVBPRT messages

The IEAVBPRT utility (and the IEAVBIPC utility in IPCS) issues the following messages:

IEAVB001I Validated Boot Information
: This is the report header message.

IEAVB003I Audit Information
: This message is followed by all the audit entries.

  Within the audit entry messages, the term *DSNE* refers to a data set name entry. (Audit information is tracked by data set name.) Within those messages, the term *DSNE ModE* refers to a module name entry for a particular data set name entry. (Audit records are typically for a specific module within a specific data set.)

IEAVB004I There are no valid certificates
: No valid certificates were found.

IEAVB005I Valid Certificates
: This message is followed by information about each of the valid certificates.

IEAVB006I No certificates were discarded
: There were no discarded certificates.

IEAVB007I Discarded Certificates
: This message is followed by information about each of the discarded certificates.

IEAVB008I Validated Boot is not in effect

Validated boot is not in effect.

IEAVB009I Unable to access *yyy* at *xxxxxxxx*

This message is issued only by the IEAVBIPC utility.

IEAVB010I Unissued validated boot messages

This message is issued only by the IEAVBIPC utility and is followed by information about each unissued message.

***(Audit mode)* IEAVB011I PLPA page data set was specified. It would not be used if enforce mode.**
***(Enforce mode)* IEAVB011I PLPA page data set was specified. It was not used.**

A PLPA page data set was specified. PLPA page data sets are not used for an enforce-mode IPL.

The enforce-mode form of this message is issued only by the IEAVBIPC utility.

***(Audit mode)* IEAVB012I Not enough storage-class memory to hold LPA. A wait state would result if enforce mode.**
***(Enforce mode)* IEAVB012I Not enough storage-class memory to hold LPA.**

There was insufficient storage-class memory to hold the LPA. This would cause an enforce-mode IPL to enter a wait state.

The enforce-mode form of this message is issued only by the IEAVBIPC utility.

Contents of an IEAVBPRT report

The overall audit information displays one or more of the following lines:

```
There are no valid certificates
Could not retrieve certificate information
Total verification failures: n
Number of DSNEs: n
Number of DSNE ModEs: n
```

The last 2 lines are displayed only if the DETAIL option is in effect.

There might be no data set related audit entries, in which case the following line appears:

```
No dataset information is available
```

An audit entry begins with the following lines:

```
DSN(VOL): dataset_name(volume)
  Total DSN verification failures: n
  Number of DSNE ModEs: n
  [No module information is available]
```

- The "Number of DSNE ModEs" line appears only when DETAIL is in effect.
- The last line is displayed when there are no module name entries.

When DETAIL is not in effect and there is at least one module name entry, a table of module names and reasons appears:

```
Modname  Reason
m        r
```

When DETAIL is in effect and there is at least one module name entry:

```
Modname: m
  Reason: r
  {Key ID: xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx | Key ID: not known}
  Fetch Type: ft
  Number of failures: n
  When first failed: yyyy/mm/dd hh:mm:ss
  Cert Name: cn
  When signed: yyyy/mm/dd hh:mm:ss
  Machine loader error info: xxxxxxxx xxxx
```

- The "Key ID" and "When signed" lines appear only when the module signature is found.
- The "Cert Name" line appears only when a certificate with a matching key ID is found.
- The "Machine loader error info" line appears when there are machine loader errors, for module name IEAIPL00 only, for one of the following reasons:
  - Module was not signed
  - Signature verification failed
  - Machine loader detected error(s)

Within the message text:

m
: The name of the module. When the module name ends with a X'C0' character, that character is displayed as '*'.

r
: One of the following reasons:

  Module was not signed
  : The module is not signed.

  Directory entry not found
  : The directory entry for the module could not be found.

  Directory entry did not match
  : The directory for the module was found but does not match.

  Signature not found
  : No signature record was found for this module.

  Hash algorithm not valid
  : The signature record does not indicate a valid hash algorithm.

  Signature algorithm not valid
  : The signature record does not indicate a valid signature algorithm.

  Hash value not correct
  : The hash value in the signature record does not match the calculated hash value.

  No certificate with matching key ID
  : The key ID in the signature record does not match any verification key available to this LPAR.

  Signature verification failed
  : The signature verification operation did not complete successfully.

  Overlay module
  : This is an overlay module. Signature support is not provided.

  Signature record version not valid
  : The version of the signature record is not valid.

  Machine loader detected error(s)
  : The machine loader detected one or more errors.

ft
: One of the following fetch types:

  IPL
  : Indicates that the fetch is during the early IPL phase.

  Nucleus
  : Indicates that the fetch is for a module that is being used to build the nucleus.

  NIP
  : Indicates that the fetch is for a module during the later IPL phase.

  LPA
  : Indicates that the fetch is for a module that is being placed into PLPA, MLPA, or FLPA.

An entry for a valid certificate contains the following lines:

```
Name: cert_name
    Key ID: xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx
    Successful uses: n
    Valid as of: yyyy/mm/dd hh:mm:ss
    Expiration: yyyy/mm/dd hh:mm:ss
    [Reason: Key is not valid]
```

- The "Key ID", "Valid as of", and "Expiration" lines appear only when DETAIL is requested.
- The "Reason: Key is not valid" line is determined after the system has started using the certificate. If this occurs, correct the certificate.

An entry for a discarded certificate contains the following lines:

```
Name: cert_name
    Reason: r
    KeyID: xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx
    Valid as of: yyyy/mm/dd hh:mm:ss
    Expiration: yyyy/mm/dd hh:mm:ss
```

- The "Key ID", "Valid as of", and "Expiration" lines appear only when DETAIL is requested.

Within the message text:

r

One of the following reasons:

Not valid yet
The certificate is not yet valid.

Expired
The certificate has expired.

Key is not valid
The key is not valid.

Key type is not valid
The key type is not valid.

Key ID length is not valid
The length of the key ID is not valid.

Hash type is not valid
The hash type is not valid.

Hash length is not valid
The length of the hash is not valid.

If any of these reasons occur, correct the certificate.

IEAVBPRT return codes

Table 90. Return codes for the IEAVBPRT utility

Return code (decimal)	Meaning
0	Successful completion. No audit information was found.
2	Successful completion. This was not a validated boot IPL.
4	Successful completion. Some audit information was found.
8	An invalid parameter was specified.
12	An invalid SYSPRINT data set was specified.

Examples

1. The following example shows a DETAIL entry for a module (within an entry for a data set):

```
Modname: IEAIPL00
  Reason: Module was not signed
```

```
  Fetch Type: IPL
  Number of failures: 1
  When first failed: 2022/10/19 13:15:07
  Machine loader error info: 12000000 3400
```

2. The following example shows a partial DETAIL entry for a data set and module:

```
IEAVB003I Audit Information
  Total verification failures: 1909
  Number of DSNEs: 7
  Number of DSNE ModEs: 1754

  DSN(VOL): SUPER.CSV.LOAD.PDS.HUGE.SIGNED(D16PK8)
    Total DSN verification failures: 1
    Number of DSNE ModEs: 1

    Modname: GM64
      Reason: No certificate with matching key ID
      Fetch Type: LPA
      Number of failures: 1
      When first failed: 2022/10/26 17:51:50
      Key ID: 21CC95D0_8A12F9FE_5AA01598_430EF6A0_8D58DFDE
      When signed: 2022/10/26 17:46:19
```

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Index

Special Characters

A

B

D

E

F

H

I

J

K

L

M

N

O

T

U

V

W

X

Z

Product Number: 5650-ZOS

GA32-0905-50